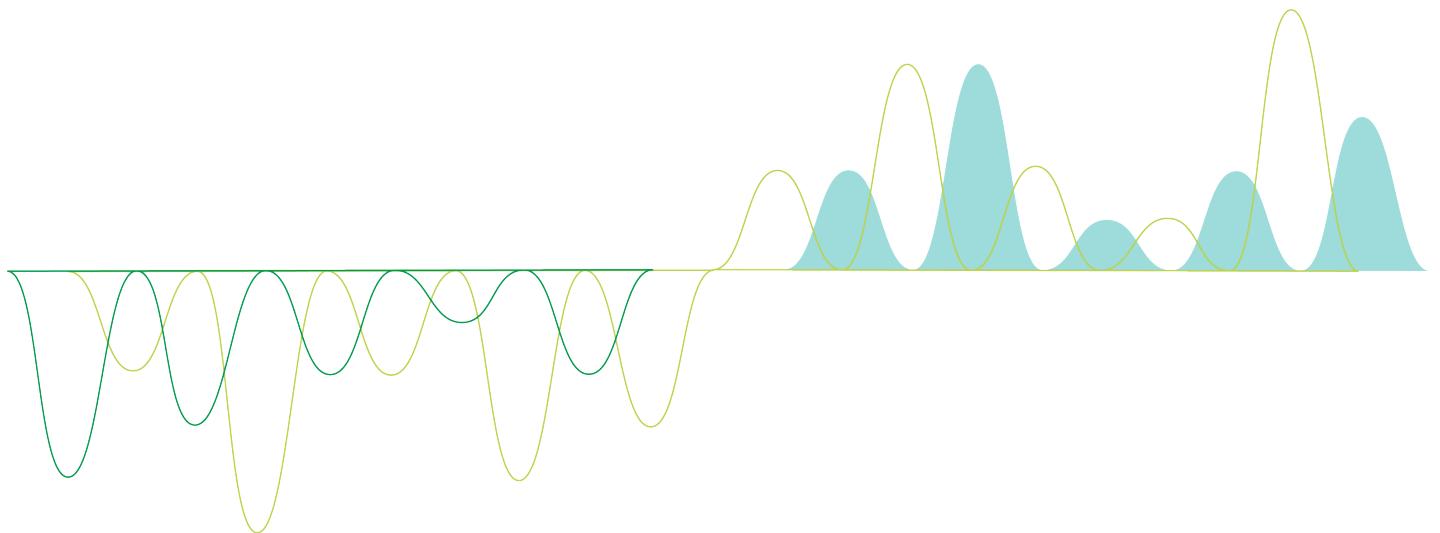


System requirements for Qlik Sense

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 System requirements for Qlik Sense Enterprise	4
1.1 Qlik Sense Enterprise on Windows	4
1.2 Qlik Sense Enterprise SaaS	8
1.3 Qlik Sense Mobile Client Managed app	9
1.4 Qlik Sense Desktop	9
1.5 Qlik DataTransfer	10
2 Supported browsers	11
2.1 Supported Microsoft Windows browsers	11
2.2 Supported Apple macOS browsers	11
2.3 iOS/iPadOS	11
2.4 Android	12

1 System requirements for Qlik Sense Enterprise

This section lists the requirements that must be fulfilled by the target system in order to successfully install and run Qlik Sense.

1.1 Qlik Sense Enterprise on Windows

Qlik Sense Enterprise on Windows requirements

Item	Requirements
Platforms	<ul style="list-style-type: none">Microsoft Windows Server 2012 R2Microsoft Windows Server 2016Microsoft Windows Server 2019Microsoft Windows Server 2022 <p>For development and testing purposes only:</p> <ul style="list-style-type: none">Microsoft Windows 10 (64-bit version only)Microsoft Windows 11 <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <i>These operating systems are supported by Qlik Sense. Third-party software may require service packs to be installed.</i></div>
Processors (CPUs)	<p>Multi-core x64 compatible processors</p> <p>Advanced Vector Extensions (AVX) support</p> <p>We recommend that you use at least 4 cores per node in a Qlik Analytics Platform deployment.</p>
Memory	<p>8 GB minimum (depending on data volumes, more may be required)</p> <p>Qlik Sense is an in-memory analysis technology. The memory requirements for the Qlik Sense products are directly related to the amount of data being analyzed.</p>
Disk space	5.0 GB total required to install
Disk share	SMB or NFS

1 System requirements for Qlik Sense Enterprise

Item	Requirements
Storage	<ul style="list-style-type: none">A network file share is required for the storage to be accessible by all servers in the site. In case of a single-server deployment, local disk storage may be sufficient.Sufficient storage is required for the volume of apps and content used in the deployment. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p> <i>Qlik periodically runs network file share performance tests on Qlik Sense using WinShare, and FreeNAS with SMB 3.0. For more information on network file share solutions, contact your Qlik representative.</i></p></div>
Security	<ul style="list-style-type: none">Microsoft Active DirectoryMicrosoft Windows Integrated AuthenticationThird-party security
WebSockets	Web browsers and infrastructure components (such as proxies and routers) must support WebSockets.
.NET framework	4.8
PowerShell	4.0 or higher <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p> <i>When installing or upgrading Qlik Sense Enterprise Client-Managed, several unsigned PowerShell scripts are executed. If your company has a policy only allowing the execution of signed scripts, you will have to bypass this policy during installation or upgrade. See Set-ExecutionPolicy for more information on the PowerShell execution policy.</i></p></div>

1 System requirements for Qlik Sense Enterprise

Item	Requirements
Repository database	<p>PostgreSQL:</p> <ul style="list-style-type: none">• 13.x (not included in the installer)• 12.x (included in the installer)• 11.x (not included in the installer)• 9.6x (for upgrade scenarios from previous versions only, not supported in February 2023 and later) <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p> <i>If you originally installed Qlik Sense before May 2021, upgrading Qlik Sense from such a version will not upgrade the bundled PostgreSQL database. You must manually upgrade the bundled PostgreSQL database running on the 9.6 version to a newer version.</i> <i>You can use the Qlik PostgreSQL Installer to upgrade your bundled PostgreSQL database from 9.6 to the 12.5 version.</i></p></div> <p>PostgreSQL is included in the Qlik Sense setup by default. However, you can also download and install it manually. The PostgreSQL installer will automatically install Microsoft Visual C++ 2015-2019 Redistributable (x64), which is required with PostgreSQL 12.x.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p> <i>The version of PostgreSQL 12.x installed with Qlik Sense does not include pgAdmin tools. You can download and install them manually if required.</i></p></div> <p>PostgreSQL is an open source object-relational database management system. It is released under the PostgreSQL license, which is a free and open source software license.</p>
Internet protocol	<ul style="list-style-type: none">• IPv4• IPv6• Dual stack (IPv4 and IPv6)
Network	The configured hostname must resolve to an IP address on the host machine.

1 System requirements for Qlik Sense Enterprise

Item	Requirements
Qlik Management Console (QMC), supported browsers	<p>The following browsers are supported for accessing the QMC.</p> <p>Supported Microsoft Windows browsers:</p> <ul style="list-style-type: none">• Microsoft Edge (only for Microsoft Windows 10)• Google Chrome• Mozilla Firefox (requires hardware acceleration, not supported in virtual environments) <p>CefSharp embedded browser v55 or later (CefSharp allows you to embed the Chromium open source browser inside .Net apps)</p> <p>Supported Apple macOS browsers:</p> <ul style="list-style-type: none">• Apple Safari 13 or later• Google Chrome• Mozilla Firefox (requires hardware acceleration, not supported in virtual environments)
QMC, minimum screen resolution	Desktops, laptops, and Apple Mac: 1024x768 No mobile or small screen support.
QlikView compatibility	It is not possible to install Qlik Sense on a machine with QlikView Server already installed.
Insight Advisor Chat	<p>Natural Language Processing (NLP) support for Insight Advisor requires a CPU that supports Advanced Vector Extensions (AVX) instructions. To find out if your CPU supports AVX, download Coreinfo v3.5 from Microsoft to view your CPU and memory topology.</p> <p>Coreinfo v3.5 - Dump information on system CPU and memory topology Copyright (C) 2008-2020 Mark Russinovich Sysinternals - www.sysinternals.com</p> <p>...</p> <pre>Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz Intel64 Family 6 Model 158 Stepping 13, GenuineIntel Microcode signature: 000000CA HTT * Hyperthreading enabled HYPERVISOR * Hypervisor is present ... AES * Supports AES extensions AVX * Supports AVX instruction extensions FMA * Supports FMA extensions using YMM state ... Logical Processor to Group Map: ***** Group 0</pre>

1 System requirements for Qlik Sense Enterprise



We do not recommend that you install Qlik Sense on domain controller machines, as group policies may prevent Qlik Sense from getting access to required services.



License activations request access to the Qlik Licensing Service. Open port 443 and allow outbound calls to license.qlikcloud.com.

Use of a proxy is supported. For more information about setting up a proxy service in Windows, see [Configuring a proxy for Qlik Licensing Service communication in Qlik Sense Enterprise on Windows](#).

1.2 Qlik Sense Enterprise SaaS

Qlik Sense Enterprise SaaS requirements

Maximum app size (in memory)	5 GB You can evaluate memory allocation of an app in the cloud hub. <i>To monitor your in-memory app size and memory usage over time, use the App Analyzer for Qlik SaaS.</i> <ul style="list-style-type: none">• This app is provided as-is and is not supported by Qlik Support.• Always use the latest version of the app.• Qlik does not collect any information when using the App Analyzer for Qlik SaaS.
Total cloud storage	*Unlimited
Maximum concurrent reloads	*Unlimited
Maximum reloads per day	*Unlimited
WebSockets	Web browsers and infrastructure components (such as proxies and routers) must support WebSockets.



* Subject to restrictions described in the Qlik Sense License Metrics. You can find this document at [Qlik Product Terms](#).



When distributing to Qlik Sense SaaS, your Qlik Sense Enterprise on Windows deployment must be either the current version or one of the previous two releases (starting from the June 2018 release).

1.3 Qlik Sense Mobile Client Managed app

Qlik Sense Mobile Client Managed client managed requirements

Qlik Sense Mobile Client Managed client managed device compatibility	<ul style="list-style-type: none">• 64-bit CPU architecture (ARM)• RAM: 2 GB or more (Dependent on data size)• Screen size: 720x1280 HDPI (267) or better
Qlik Sense Mobile Client Managed client managed compatibility with Qlik Sense	Qlik Sense February 2020 and later releases
Qlik Sense Mobile Client Managed client managed Apple support	<ul style="list-style-type: none">• iOS 14 or later• iPadOS 14 or later
Qlik Sense Mobile Client Managed client managed Android support	Android 10 or later

1.4 Qlik Sense Desktop

To successfully install and run Qlik Sense Desktop, the requirements listed in this section must be fulfilled.

Qlik Sense Desktop requirements

Operating system	Microsoft Windows 10 (64-bit version only) Microsoft Windows 11
Processors (CPUs)	Intel Core 2 Duo or higher recommended. Advanced Vector Extensions (AVX) support.
Memory	4 GB minimum (depending on data volumes, more may be required).  <i>Qlik Sense uses an in-memory analysis technology. The memory requirements are directly related to the amount of data being analyzed.</i>
Disk space	5.0 GB total required to install
.NET framework	4.8 or higher
Security	Local admin privileges needed to install.

1 System requirements for Qlik Sense Enterprise

Minimum screen resolution	<ul style="list-style-type: none">• Desktops, laptops and tablets: 1024x768• Small screens: 320x568
Browser support	<ul style="list-style-type: none">• Microsoft Edge• Google Chrome• Mozilla Firefox
	<p> <i>By default, Qlik Sense Desktop runs in a window of its own. But you can also open it in a web browser.</i></p> <p> <i>Mozilla Firefox requires hardware acceleration, not supported in virtual environments.</i></p>

1.5 Qlik DataTransfer

Qlik DataTransfer requirements

Platforms	<ul style="list-style-type: none">• Microsoft Windows Server 2012 R2• Microsoft Windows Server 2016• Microsoft Windows Server 2019• Microsoft Windows Server 2022 <p>For development and testing purposes only:</p> <ul style="list-style-type: none">• Microsoft Windows 10 (64-bit version only)
Processors (CPUs)	Multi-core x64 compatible processors. We recommend a minimum of 4 cores.
Memory	8 GB minimum The memory requirements for the Qlik Sense products are directly related to the amount of data being analyzed.
Disk space	2 GB minimum
Storage	Sufficient storage is required for the volume of apps and content used in the deployment.
PowerShell	5.1 or higher
TLS	1.2 or higher

2 Supported browsers

Qlik Sense is designed to work on the platform and web browser combinations described in this section, using default browser settings.

Each Qlik Sense release is tested for compatibility with the latest publicly available browser versions. Due to the frequency of browser version updates, Qlik does not include specific browser version numbers in the system requirements.

Each Qlik Sense release is compatible with and supported on the latest iOS versions that are publicly available at the time of the Qlik Sense release. Due to the frequency of iOS version updates, Qlik does not include specific iOS version numbers in the system requirements.



Minimum screen resolution for desktops and laptops is 1024x768; tablets is 1024x768; small screens is 320x568.

2.1 Supported Microsoft Windows browsers

The following browsers can be used on supported Microsoft Windows and Microsoft Windows Server machines to access the Qlik Management Console (QMC) and the hub:

- Microsoft Edge
- Google Chrome
- Mozilla Firefox (requires hardware acceleration, not supported in virtual environments)

CefSharp embedded browser v55 or later (CefSharp allows you to embed the Chromium open source browser inside .Net apps)

2.2 Supported Apple macOS browsers

The following browsers can be used on supported Apple macOS computers to access the Qlik Management Console (QMC) and the hub:

- Apple Safari (the last 3 major versions)
- Google Chrome
- Mozilla Firefox (requires hardware acceleration, not supported in virtual environments)

2.3 iOS/iPadOS

The following browsers can be used on supported devices (script editing is not supported on tablet devices):

- Apple Safari (the last 3 major versions)
- VMware browser (using AirWatch per-app VPN)
- MobileIron Web@Work (using MobileIron Tunnel)

- BlackBerry Access
- Microsoft Edge

2.4 Android

The following browsers can be used on supported devices (script editing is not supported on tablet devices):

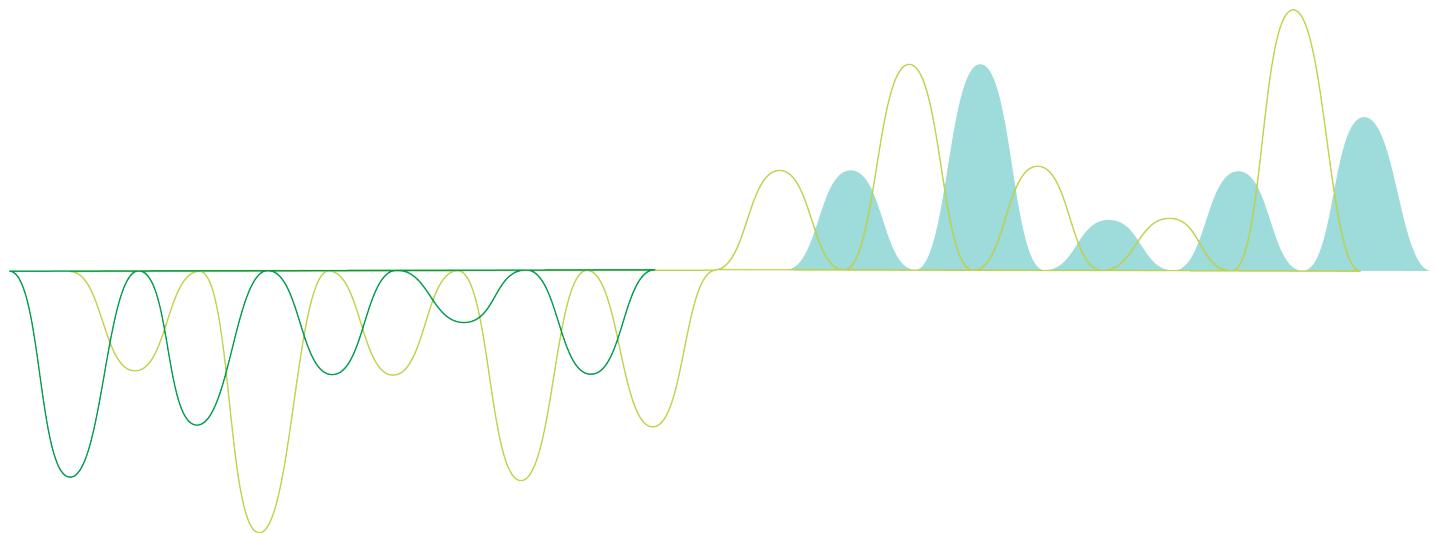
- Google Chrome
- BlackBerry Access
- Microsoft Edge

News since Qlik Sense 1.0

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 What's new in Qlik Sense May 2023	14
1.1 Visualizations and dashboards	14
New customization options for filter panes	14
Styling improvements to charts	14
Line charts can display larger datasets	14
Add background colors and images to charts	15
1.2 Advanced authoring	15
New chart functions for custom app navigation	15
1.3 Connectors	15
New Google Analytics 4 connector	15
2 What's new in Qlik Sense February 2023	16
2.1 Augmented analytics	16
Add natural language insights to sheets	16
2.2 Visualizations and dashboards	16
New styling panel for visualizations	16
Scatter plots can display larger datasets	16
New sheet grid and sheet background options	16
Show gradual changes with chart animations	17
2.3 Connectors	17
New ServiceNow ODBC Connector	17
New Amazon S3 connectors with enhanced security	17
3 What's new in Qlik Sense November 2022	18
3.1 Augmented analytics	18
Auto-generate an analysis using Insight Advisor Analysis Types	18
Example questions in Insight Advisor	18
Insight Advisor Search	18
Improvement to Insight Advisor Chat's support for follow-up questions	18
3.2 Visualizations and dashboards	18
Custom tooltips for KPIs	19
New sub-toolbar	19
New distribution functions from Cephes library	19
Font styling with map charts	19
3.3 Connectors	19
ODBC database connector performance increase	19
Update to MongoDB connector	19
Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed	19
3.4 Data and platform	19
Improvements to navigation in the hub	20
3.5 Documentation improvements and additions	20
Updates to scripting topics	20
4 What's new in Qlik Sense August 2022	21
4.1 Augmented analytics	21
Insight Advisor is no longer available with Qlik Sense Desktop	21
Chart level scripting	21
4.2 Visualizations and dashboards	21
Styling panel for bar and pie charts	21

Contents

New chart and script functions to support JSON	21
New inner and outer set expressions in set analysis	21
4.3 Connectors	22
Office 365 SharePoint Metadata Connector supports new tables	22
Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed	22
4.4 Documentation improvements and additions	22
Scripting help improvements	22
5 What's new in Qlik Sense May 2022	23
5.1 Augmented analytics	23
Business logic fine grain controls in Insight Advisor	23
5.2 Visualizations and dashboards	23
New actions for button object	23
Label coloring with map chart	23
Favorites in the hub	23
Scatterplot now supports regression lines	23
KPI background color and new icons	24
Uploading apps with GeoAnalytics extensions to Qlik Cloud	24
5.3 Connectors	24
Databricks ODBC Connector has OAuth support and improved interface	24
Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed	24
Support for uploading Oracle Wallets	24
Self-service task management—Create, edit, and delete reload tasks in the hub	24
6 What's new in Qlik Sense February 2022	25
6.1 Augmented analytics	25
Period over period dashboards in Insight Advisor	25
Insight Advisor supports dollar-sign expansion	25
Business logic fine grain controls in Insight Advisor	25
New business logic options for customization in Insight Advisor	25
6.2 Visualizations and dashboards	25
Improvements to the grid chart	26
Improvements to managing variables	26
6.3 Connectors	26
Qlik Sense connector for Google Drive	26
Google Drive and Spreadsheets connector	26
Snowflake connectivity enhancements	26
Support for AWS IAM credentials authorization with Amazon Redshift	26
6.4 Data and platform	26
Formatted export to Excel	26
Add self-service task management	26
6.5 Documentation improvements and additions	27
Scripting help improvements	27
Chart expressions and load scripts	27
7 What's new in Qlik Sense November 2021	28
7.1 Augmented analytics	28
Custom chart periods	28
DPS period over period analysis	28
Consistency in accessing fields and master items across Insight Advisor and Insight Advisor	28

Contents

Chat	28
Create custom responses to question	28
7.2 Visualizations	28
New host for satellite tiles	28
Add charts in tooltips	29
Accessibility	29
Layered labels	29
All labels visible in line chart	29
Sizing and positioning of images	29
Edit master items from the property panel	29
Define default tab in container object	29
7.3 Connectors	30
Snowflake connectivity enhancements	30
7.4 Platform	30
Partial reloads for scheduled reloads	30
NVDA and Chrome screen reader support	30
Improvements to the Qlik Sense Enterprise on Windows installer	30
Show license information in QMC when using a signed key	30
Allow for SCRAM encryption	30
8 What's new in Qlik Sense August 2021	31
8.1 Augmented analytics	31
Mutual Information Key Driver Analysis	31
8.2 Visualizations	31
Bars on secondary Axis	31
Color per measure	31
Dark mode base map	31
Image by URL in point layer	31
Images in straight table	31
9 What's new in Qlik Sense May 2021	32
9.1 Augmented analytics	32
Insight Advisor Chat improvements	32
Insight Advisor Search	32
9.2 Visualizations	32
Video player	32
Active chart and grid dimensions	32
New and improved combo chart	32
9.3 Connectors	32
New Databricks ODBC Connector	32
New Azure Synapse Connector	33
Presto Kerberos support	33
9.4 Mobile	33
Support for Microsoft Edge	33
9.5 Platform and administration	33
Self-service task management	33
Enhanced ODAG performance	33
10 What's new in Qlik Sense February 2021	34
10.1 Augmented analytics	34

Contents

Business logic	34
Business logic tutorial	34
Normalization for K-means clustering	34
Insight Advisor Chat	34
10.2 Visualizations	34
10.3 Connectivity	34
10.4 Qlik Engine	34
10.5 Documentation improvements and additions	35
Improvements for data security and Section Access	35
11 What's new in Qlik Sense November 2020	36
11.1 Augmented analytics	36
Business logic	36
Insight Advisor Chat	36
K-means Clustering - Expanded Functionality	36
Insight Advisor visualization alternatives	36
11.2 Visual analytics and usability improvements	36
Images in Custom Tooltips	36
Copy sheets between apps	37
Copy measures between objects	37
Other chart improvements	37
Usability look and feel	37
11.3 Platform	37
11.4 Mobile	37
12 What's new in Qlik Sense September 2020	38
12.1 Augmented intelligence	38
New search-based visual analysis chart types	38
Search-based visual analysis on mobile devices	38
Advanced analytics calculation - K-means clustering	38
12.2 Visual analytics improvements	38
Improved reference lines	38
Animator control	38
Other chart improvements	38
12.3 Administration improvements	39
App distribution	39
QMC status page	39
Self-service hub improvements	39
12.4 Other improvements	39
Support for NFS protocol	39
Data literacy built-in	39
13 What's new in Qlik Sense June 2020	40
13.1 Visualizations and dashboards	40
Sparkline chart	40
New bullet chart	40
Table customization	40
Org chart enhancements	40
Filter pane enhancements	40

Contents

Number formatting	40
Custom Tooltip enhancements	40
13.2 Data management, connectivity and integration	41
Dynamic views and ODAG improvements	41
Qlik Sense Desktop authentication against SaaS	41
13.3 Administration improvements	41
Qlik Management Console improvements	41
Licensing improvements	41
14 What's new in Qlik Sense April 2020	42
14.1 Visualizations and dashboards	42
Visualizations	42
Dashboards and applications	42
Themes and styling	43
14.2 Advanced authoring	43
15 What's new in Qlik Sense February 2020	44
15.1 Augmented Intelligence	44
Improved Natural Language Processing (NLP)	44
Natural language support	44
Extended chart support in Insights	44
15.2 Visualizations and dashboards	44
Visualization	44
Dashboards and applications	45
Themes and styling	45
15.3 Data management, connectivity and integration	46
QVD Catalog Browsing within Qlik Sense via Qlik Catalog	46
Connectors	46
Back end improvements	46
16 What's new in Qlik Sense November 2019	47
16.1 Visualizations and Mapping	47
Table styling	47
Improved Accumulation in Bar chart, Line chart, Combo chart, and Table	47
Mekko chart	47
Exclude map layers from auto-zoom	47
Trellis container improvements, an enhancement to the visualization bundle	47
16.2 Qlik Sense Mobile Client Managed	47
Support for offline access to apps with Qlik Sense Mobile for Android	47
17 What's new in Qlik Sense September 2019	48
17.1 Advanced Authoring	48
Default Landing Sheets	48
Increased Search Limit on Filter Panes	48
Expression Promotion	48
17.2 Visualizations and Mapping	48
Location finder	48
Variance Waterfall Chart	48
Pie chart styling	48
17.3 Qlik Sense Mobile Client Managed	49

Contents

Support for MobileIron using Web@work	49
Support for iPadOS	49
17.4 Qlik Connectors	49
Salesforce connector	49
Snowflake connectivity	49
18 What's new in Qlik Sense June 2019?	50
18.1 Advanced Authoring	50
Copy value to clipboard	50
Grid layout on mobile devices	50
Native container object	50
18.2 Visualizations and Mapping	50
New charts for the Visualization Bundle	50
Improved bar chart	50
Exploration menu for maps	51
18.3 Qlik Sense Mobile Client Managed	51
Push notifications to Qlik Sense Mobile Client Managed	51
18.4 Multi-Cloud	51
18.5 Qlik Connectors	51
19 What's new in Qlik Sense April 2019?	52
19.1 Augmented Intelligence	52
Associative insights	52
19.2 Visualizations and Mapping	52
Maps enhancements	52
Visualization Bundle	52
19.3 Multi-Cloud	52
19.4 Qlik Sense Mobile Client Managed	53
Qlik Sense Mobile for BlackBerry	53
19.5 Qlik Connectors	54
20 What's new in Qlik Sense February 2019?	55
20.1 Usability Improvements	55
Single page application flow	55
20.2 Advanced Authoring	55
Dollar-sign expansion preview	55
20.3 Visualizations and Mapping	55
Visualization Bundle	55
Dashboard Bundle	55
Support for WMS in map background layers	55
20.4 Qlik Sense Mobile Client Managed	56
20.5 Qlik Connectors	56
Integrated Single Sign On Support (SSO)	56
21 What's new in Qlik Sense November 2018?	57
21.1 Advancements to Augmented intelligence	57
Precedent based learning	57
Insight advisor light authoring	57
21.2 New advanced authoring features	57
Alternate states	57

Contents

Exposed set analysis	57
Set expression improvements	57
Single selection in fields	57
Fully supported Dashboard Extension bundle [#1]	57
21.3 New ways to collaborate	58
Sharable Qlik Sense chart links	58
21.4 Advancements in Visualizations and Mapping	58
New map chart layer	58
Improved pie chart	58
Outline opacity setting	58
Further advancements in tile map services	58
21.5 Improved Management	59
Import Export App enhancements	59
Multi-Cloud Developments	59
21.6 Accessibility	59
22 What's new in Qlik Sense September 2018?	60
22.1 Augmented intelligence	60
Insight advisor	60
22.2 Advanced authoring	60
Improvements to sheet control	60
App customization	60
Expression editor enhancements	60
Better control in visualizations	60
22.3 Visualizations and mapping	60
New map layer	60
Map improvements	61
22.4 Management	61
Improvements to back end work flow	61
22.5 Mobile	61
22.6 Qlik connectors	61
23 What's new in Qlik Sense June 2018?	62
23.1 Create, Discover, Collaborate	62
Insight advisor	62
Accelerated self-service	62
Advanced authoring	62
Filtering data from files	62
Visualizations and mapping	62
Improved app management from the hub	63
Accessibility	63
23.2 Deploy and Administer	63
Multi-Cloud	63
23.3 Mobile	63
Support for BlackBerry Access browser	63
23.4 Qlik Connectors	64
24 What's new in Qlik Sense April 2018?	65
24.1 Create, Discover, Collaborate	65

Contents

Assisted data visualization with Qlik Sense chart suggestions	65
Publishing an app from the hub	65
Improvements based on customers feedback	65
Maps visualizations improvements	65
Keyboard navigation support for Qlik Sense hub	65
Linking Qlik Sense Mobile Client Managed to third-party applications	65
24.2 Deploy	66
Deployment improvement	66
Per-app VPN mode for Qlik Sense Mobile Client Managed	66
24.3 Administer	66
Allocations for new license types	66
Analytic connections improvement	66
Enable anonymous users to export data	66
25 What's new in Qlik Sense February 2018?	67
25.1 Create, Discover, Collaborate	67
Dynamically generated queries to web sources in scripts	67
Loading a table from an analytic connection	67
Simplify data preparation with recommended associations	67
Styling an app with custom themes	67
On-demand App Generation	67
Qlik Sense Mobile Client Managed	67
Updated ODBC connectors	67
25.2 Deploy, Administer	68
SSO with Microsoft SQL Server	68
26 What's new in Qlik Sense November 2017?	69
26.1 Create, Discover, Collaborate	69
Keyboard navigation in Qlik Sense apps	69
Details dialog in Data manager	69
Add data manually	69
Additional functions for calculated fields	69
New ODBC connectors	69
26.2 Deploy, Administer	69
SAML single logout	69
27 What's new in Qlik Sense September 2017?	70
27.1 Create, Discover, Collaborate	70
Sequential operations in Data manager	70
Recent colors in color picker	70
Navigation and usability improvements to Data manager	70
New visualization: Waterfall chart	70
Ease-of-use enhancements to on-demand apps	70
27.2 Deploy, Administer	70
Qlik Sense Mobile Client Managed app	70
Centralized logging	71
28 What's new in Qlik Sense June 2017?	72
28.1 Create, Discover, Collaborate	72
On-demand apps	72
New visualizations	72

Contents

Synchronizing scripted tables in Data manager	72
Data profiling cards	72
Concatenation in Data manager	73
Colors and dimensions	73
28.2 Administer	73
Single sign-on to Cloudera Impala	73
Three new Monitoring apps	73
Analytic connections	73
28.3 History	73
29 What's new in Qlik Sense 3.2?	74
29.1 Create, Discover, Collaborate	74
Calendar measures	74
Colors and measures	74
29.2 Deploy	74
Shared persistence	74
29.3 Administer	74
Desktop authentication	74
QlikView converter	74
30 What's new in Qlik Sense 3.1?	75
30.1 Create, Discover, Collaborate	75
City and country recognition when loading data	75
Scroll alignment	75
Default app theme	75
Drag and drop coloring	75
Navigation	75
Search Qlik DataMarket	75
Salesforce Connector supports primary key chunking	75
Filtering data in the database connectors	75
30.2 Administer	75
Qlik Management Console	75
31 What's new in Qlik Sense 3.0?	76
31.1 Create, Discover, Collaborate	76
Qlik connectors installed with Qlik Sense	76
Managing table associations in Data manager	76
Single sign-on to SAP HANA	76
New Qlik DataMarket packages	76
New multiple-table structure for Qlik DataMarket data sets	76
Publishing apps	77
New language support	77
App styling	77
Smart search now includes visual search	77
Enhance your apps with widgets	77
Time-aware charts	77
Shared content in the Qlik Sense hub	77
Additional changes	77
31.2 Deploy	77
IPv6	77

Contents

Qlik Deployment Console	77
31.3 Administer	78
Monitoring apps in QMC updated	78
32 What's new in Qlik Sense 2.2?	79
32.1 Create, Discover, Collaborate	79
Data manager	79
Alternative dimensions and measures	79
Export data from pivot tables and other charts	79
Qlik DataMarket	79
Data storytelling	79
32.2 Deploy	79
Qlik Sense Proxy Service metrics	79
Qlik Sense Printing Service logging	80
Qlik Deployment Console system requirements	80
Cloning sites	80
Qlik Sense setup files stored in the S3 bucket	80
32.3 Administer	80
New license option	80
Limit resource consumption by apps	80
Redesign of the audit page in the Qlik Management Console	80
Operations Monitor app in QMC updated	80
33 What's new in Qlik Sense 2.1?	81
33.1 Create	81
Managing data	81
Creating apps and visualizations	81
Managing images	81
33.2 Discover	82
Interacting with visualizations	82
33.3 Collaborate	82
Data storytelling	82
33.4 Deploy	82
Planning Qlik Sense deployments	82
Deploying Qlik Sense sites in cloud computing environments	82
33.5 Administer	82
Monitoring a Qlik Sense site	82
34 What's new in Qlik Sense 2.0?	83
34.1 Working with Qlik Sense	83
Managing data	83
Creating apps and visualizations	83
Discovering and analyzing	84
Managing apps	84
Sharing and collaborating	84
Using data storytelling	84
Qlik Sense Desktop	85
34.2 Deploying Qlik Sense	85
Planning Qlik Sense deployments	85
Installing and upgrading	85

Contents

Deploying Qlik Sense sites in cloud computing environments	86
34.3 Administering Qlik Sense	86
Managing a Qlik Sense site	86
Monitoring a Qlik Sense site	87
Troubleshooting Qlik Sense using logs	87
35 What's new in Qlik Sense 1.1?	88
35.1 Working with Qlik Sense	88
Creating apps and visualizations	88
Using data storytelling	88
35.2 Deploying Qlik Sense	89
Installation and setup	89
35.3 Administering Qlik Sense	89
Managing a Qlik Sense site	89
Monitoring a Qlik Sense site	89
Managing Qlik Sense sites in cloud computing environments	90

1 What's new in Qlik Sense May 2023

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

1.1 Visualizations and dashboards

New customization options for filter panes

App developers now have new ways to customize the appearance and functionality of filter panes. The listbox for each field or master dimension in a filter pane can be customized individually with multiple new properties. The following options are available:

- Hide the displayed title of the field.
- Remove search functionality for the field, or switch to **Wildcard** mode (inserts * characters around string).
- Compact view for optimizing space between values.
- Checkbox mode for alternative selection method.
- Histogram view to display the frequency of each value in the data.
- Grid layout with custom ordering and display options.

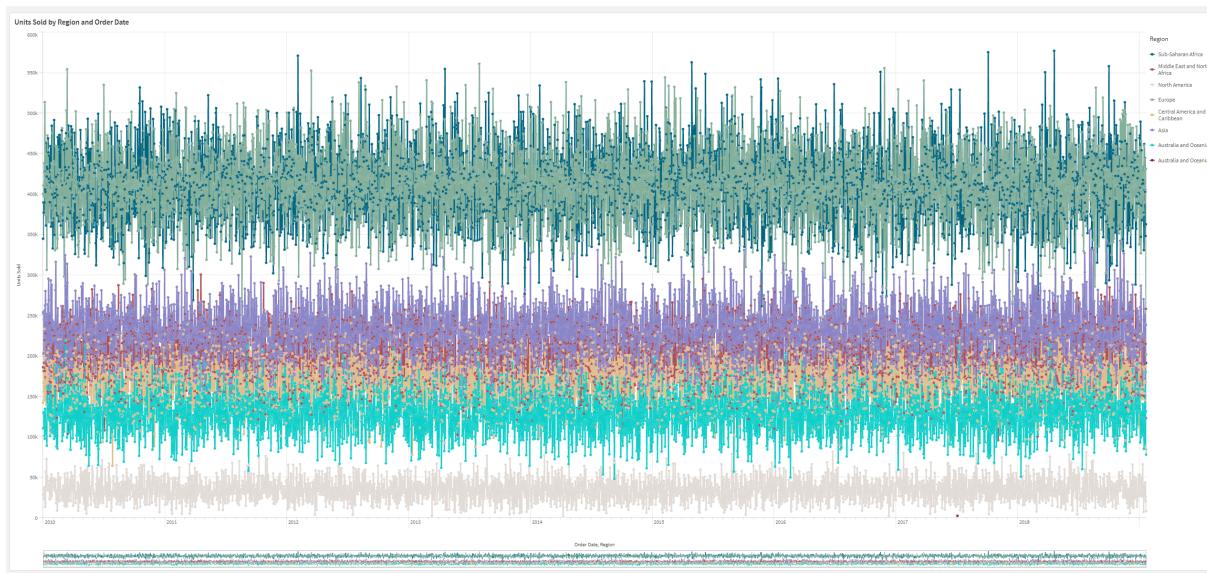
Styling improvements to charts

New styling improvements are available for several Qlik Sense charts, giving app developers more customization options when building visualizations, allowing them to conform to company, department, or personal style standards and preferences.

Line charts can display larger datasets

App developers can now control the number of visible points and lines displayed in line charts. The maximum for visible points is 50,000. The maximum for visible lines is 1,000. These options are available for line charts with a continuous dimension axis.

A line chart with a large data set. This chart is set to display a maximum of 50 lines and 20,000 data points.



Add background colors and images to charts

App developers can now add a custom background color or image to straight tables, pivot charts, pie charts, and bar charts. Any image in the media library can be used as a background. You can choose a single background color or color by expression.

1.2 Advanced authoring

New chart functions for custom app navigation

The new chart functions `objectId()` and `inobject()` enable creation of guided applications with custom navigation functionality. For example, you can use the functions in conditional expressions to create colored buttons to navigate between sheets in an app.

The functions give access to the hierarchy of objects in a Qlik Sense app. `objectId()` will return the ID of the object in which the expression is evaluated. The function takes one optional string parameter, which can specify the type of the object you're looking for. `objectId('sheet')` will return the ID for the sheet of the expression. `inobject(ID)` is true when evaluated inside an object with the specified ID.

1.3 Connectors

New Google Analytics 4 connector

The Google Analytics 4 connector extracts traffic and engagement metrics from your Google Analytics 4-enabled websites and apps and loads that data into your Qlik Sense apps.

Google is replacing Universal Analytics with Google Analytics 4. For more information about this change, see [Google Analytics Help](#).

2 What's new in Qlik Sense February 2023

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

2.1 Augmented analytics

Add natural language insights to sheets

NL Insights is a new Dashboard bundle control that displays natural language insights about selected dimensions and measures. This allows app developers to effectively share insights about data with their app consumers. Insights includes analyses based on the data selected. App developers can remove unwanted analyses if needed.

2.2 Visualizations and dashboards

New styling panel for visualizations

App developers can now style the font used for titles, subtitles, and footnotes, giving them more options for customizing their visualizations. A new General tab provides options to change the font type, size, color, and emphasis. The Chart tab controls styling of rows, scrollbars, and custom headers.

Scatter plots can display larger datasets

You can now control the number of visible bubbles displayed in scatter plots you create. Scatter plots can be set to display up to 50,000 data points. If there are more than 5,000 visible bubbles, then bubble labels and out of bound bubbles are not shown.

New sheet grid and sheet background options

Sheets can now have custom background images and colors, allowing you to divide your sheet into sections, add images from your media library, and more. You can set a background color using the color picker, or by expression.

App developers now have more control over object placement and sheet grid size to support advanced formatting for presentations.

Show gradual changes with chart animations

Chart animations are the gradual transitions in a visualization from the old view to the new view when data has been changed, for example, after a selection has been made. Animations can be turned on or off under app settings, in unpublished apps.

Animations can be controlled in the following chart types:

- Bar charts
- Bullet charts
- Combo charts
- Line charts
- Pie charts
- Scatter plots
- Funnel charts (Visualization bundle)
- Grid charts (Visualization bundle)
- Sankey charts (Visualization bundle)

2.3 Connectors

New ServiceNow ODBC Connector

You can now access data stored in ServiceNow. The new ServiceNow connector allows you to include ServiceNow data, such as incidents, requests, and case records within your analysis. The new connector includes security options such as OAuth or SSL to ensure that only authorized users can access this data.

New Amazon S3 connectors with enhanced security

Qlik Cloud has two new connectors: Amazon S3 Metadata V2 and Amazon S3 V2 Web storage provider. These connectors are more secure because they use the Amazon S3 API to access your Amazon S3 metadata, such as the names of files and subfolders in your Amazon S3 bucket. Previous versions of these connectors used the REST API. For now, the initial release of the Amazon S3 connectors V2 and the existing Amazon S3 connectors have the same capabilities. However, new capabilities will only be added to the V2 connectors going forward.

At some point in the future, the previous Amazon S3 connectors will no longer be supported.

3 What's new in Qlik Sense November 2022

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

3.1 Augmented analytics

Auto-generate an analysis using Insight Advisor Analysis Types

Let Insight Advisor auto-generate your analysis, complete with visualizations, narrative readouts, even entire sheets. Choose the type of analysis from a variety of available options, such as comparison, ranking, trending and more. Select your data with guidance from Insight Advisor, and then let Insight Advisor generate the rest. Edit the analysis and add it to dashboards for further exploration. With Analysis Types, Insight Advisor supports multiple paths to insight including auto-analysis when you select fields, natural language search, and now auto-generating the analysis when you select an analysis type, for a more directive approach.

Example questions in Insight Advisor

App creators can now provide example questions in Insight Advisor within the business logic layer of an app. This allows them to customize the questions to their specific analytic content, and thereby help users learn how to use natural language. When users ask questions, the examples appear in a drop-down menu from which users can select, modify, and submit the question. In Insight Advisor Chat, example questions from across your apps will be available.

Insight Advisor Search

Insight Advisor Search provides an improved Insight Advisor experience in-app, allowing users to easily auto-generate the most relevant analyses and insights from their data. A new **Discovery** button opens Insight Advisor, where users can select fields and generate insights without having to navigate away from their current sheet. The new **Ask Insight Advisor** search box is now visible at all times within dashboards, allowing users to easily generate insights using natural language processing. Auto-generated visualizations and analyses can be refined and added directly to dashboards for further exploration.

Improvement to Insight Advisor Chat's support for follow-up questions

Insight Advisor Chat improves support for follow-up questions with the **Explore this further** option. The option takes you directly to search-based discovery within apps for deeper analysis. When you ask follow up questions to an original question and select **Explore this further**, the app opens with the selections applied for the full line of questioning, allowing you to continue visually where you left off conversationally.

3.2 Visualizations and dashboards

Custom tooltips for KPIs

App developers can add custom tooltips to KPIs. Previously, tooltips could not be added to objects without dimensions.

New sub-toolbar

Navigate quickly and use the tools that you need in the redesigned navigation bar and toolbar in Qlik Sense.

New distribution functions from Cephes library

The following distribution functions from the Cephes library have been added, allowing you to perform more advanced statistical analysis of simulations.

List of distribution functions

These distribution functions can be used in both the data load script and in chart expressions.

Font styling with map charts

App developers can now style the font used for titles, subtitles, and footnotes in map charts, giving them more options for customizing their visualizations.

You also now have the option to customize font properties (family, size, and color) for labels in several types of map layers. This applies to point, area, chart, and line layers.

3.3 Connectors

ODBC database connector performance increase

ODBC data sources have increased performance when working with larger datasets. All new connections automatically use the Bulk Reader feature.

Instead of loading data row by row, the Bulk Reader works with larger portions of data in the iterations within a load. This can result in faster load times for larger datasets. To activate this feature in existing connections, open the connection properties window by selecting **Edit** and then click **Save**. No other connection properties need to be changed.

Update to MongoDB connector

The MongoDB connector in Qlik Sense Client-Managed has been improved with new security features. You can now use SCRAM-SHA-256 and LDAP authentication options when defining your MongoDB connection.

Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed

The following Qlik Web Connectors are now available in Qlik Sense Enterprise on Windows without the need for Qlik Web Connectors to be separately installed: Facebook Insights, Google Ads, Google Calendar, Google Search Console, Outlook 365, Mailbox IMAP, Microsoft Dynamics CRM V2, OData, Qualtrics, SMTP, SurveyMonkey, and YouTube Analytics.

3.4 Data and platform

Improvements to navigation in the hub

Certain options in the Qlik Sense Enterprise hub have been re-arranged for a better user experience. The user profile icon has been moved to the top right corner in the toolbar. Clicking on this icon will open up the global menu with the following options:

- **Profile:** Click the icon at the top of the menu to view your user ID and directory.
- **Client authentication:** A Qlik Sense administrator can allow users to authenticate their client against Qlik Sense. This item is not enabled by default. For more information, see [Configuring client authentication](#).
- **Dev Hub:** To learn more, see  [Dev Hub](#).
- **About:** Access your license agreement, version information, and third-party software information.
- **Help:** Redirects to Qlik help documentation for your Qlik Sense version.
- **Privacy Policy:** Learn more about how Qlik manages privacy in its products.
- **Log in:** If you are using Qlik Sense as an anonymous user, you can log into your account.

This option will only be available if your administrator has allowed users to use the Qlik Sense deployment anonymously. For more information, see [Anonymous authentication](#).

- **Log out:** If you have logged into your account, use this option to log out.

3.5 Documentation improvements and additions

Updates to scripting topics

Scripting topics covering date and time functions, number interpretation, and scripting prefixes have been improved. Qlik Cloud Help has added new examples to the documentation.

List of updated help topics

To test sample load scripts and chart expressions, see:

4 What's new in Qlik Sense August 2022

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

4.1 Augmented analytics

Insight Advisor is no longer available with Qlik Sense Desktop

Insight Advisor, including business logic, is no longer supported with Qlik Sense Desktop. In November 2022, Insight Advisor will be upgraded to a new experience. This will only be available on Qlik Sense Enterprise on Windows. Users who want to continue using Insight Advisor and Business Logic on Qlik Sense Desktop should not upgrade to August 2022.

Chart level scripting

Chart level scripting is a powerful feature that allows you to modify the dynamic dataset behind a chart using a subset of the Qlik scripting language, with techniques such as variables and loops. You can add or modify rows and columns that were not in the original dataset. This allows for calculations in chart expressions that were previously not possible, such as simulations or goal-seeking.

4.2 Visualizations and dashboards

Styling panel for bar and pie charts

App developers can now style the font used for titles, subtitles, and footnotes in bar and pie charts, giving them more options for customizing their visualizations. A new General tab provides options to change the font type, size, color, and emphasis. The Chart tab controls styling of the bars in the bar chart and the styling of slices in pie charts.

New chart and script functions to support JSON

New functions have been introduced to support the testing, validation, and modification of JSON (JavaScript Object Notation) data: [IsJson](#), [JsonGet](#), [JsonSet](#).

New inner and outer set expressions in set analysis

Outer set expressions simplify governance of Qlik Sense applications. It is particularly helpful when working with complex master measures that come in different flavors of one base measure.

You can now use set expressions to tweak the base master measure, for example {<Year={2021}>} [Master Measure].

4.3 Connectors

Office 365 SharePoint Metadata Connector supports new tables

The Office 365 SharePoint Metadata connector has been updated to support access to Microsoft SharePoint lists and views. The following new tables can now be loaded in the Data manager or the Data load editor: **Views** and **ItemsFromList**.

Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed

The following Qlik Web Connectors are now available in Qlik Sense Enterprise on Windows in the same way as in Qlik Sense SaaS, without the need for Qlik Web Connectors to be separately installed.

4.4 Documentation improvements and additions

Scripting help improvements

Scripting topics covering date and time functions, number interpretation, and scripting prefixes have been improved. Qlik Sense Help has added new examples to the documentation.

5 What's new in Qlik Sense May 2022

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

5.1 Augmented analytics

Business logic fine grain controls in Insight Advisor

Business Logic allows you to define the default grain for a calendar period, such as a yearly, quarterly or monthly basis. When you create behaviors such as default calendar periods, you can now specify whether to use or ignore the grain for a particular analysis, providing more advanced fine grain controls.

5.2 Visualizations and dashboards

New actions for button object

New actions available with the button object improve the workflow for on-demand apps and dynamic views. App developers can use the button to make interfacing with and transitioning to a new app more user friendly.

Label coloring with map chart

New map settings include options for label coloring, allowing you to customize how labels contrast with the base map or colored areas. Set your map to automatically adjust the label coloring depending on the base map, or select your preferred coloring.

Favorites in the hub

Mark your private or any published apps as favorite for easy access in the hub. Favorite apps are marked with a ★ icon and display in the new **Favorites** section. This section is visible in the hub only when at least one app is marked as a favorite.

Scatterplot now supports regression lines

The redesigned scatterplot chart provides greater flexibility and detail. Scatterplot now supports regression lines, including average, linear, exponential, and logarithmic, as well as second, third, and fourth polynomial. Customize the regression line settings including color and type, vertical or horizontal fit, and show formula to quickly see the relationship between two variables.

KPI background color and new icons

Now you can add background colors to KPI charts, giving you more flexibility when customizing your visualizations. An extensive array of new icons has also been added to the existing icon set.

Uploading apps with GeoAnalytics extensions to Qlik Cloud

To help customers who are transitioning to Qlik Cloud, there is a 12-month grace period for apps that use GeoAnalytics extensions and have been uploaded to Qlik Cloud from other versions of Qlik Sense. Following the grace period, you can use the map chart in Qlik Cloud, which is faster, easier to use, prints better and includes more features.

5.3 Connectors

Databricks ODBC Connector has OAuth support and improved interface

The Databricks Connector now supports OAuth 2.0 authentication with Databricks on Azure. The support for OAuth also allows single sign-on (SSO) to Databricks when using an identity provider.

The Databricks connection interface has been simplified to include the database properties that apply only to this connector.

Qlik Web Connectors included in Qlik Sense Enterprise Client-Managed

The Qlik Web Storage Provider Connectors are now available in Qlik Sense Enterprise on Windows in the same way as in Qlik Sense SaaS. The corresponding metadata connectors are also integrated, without the need for Qlik Web Connectors to be separately installed. These connectors allow you to connect to file-based data stored on a web storage provider, either by browsing for folders and files directly in the interface, or using the separate metadata connectors listing the structures and objects in tables. The web storage provider platforms supported are Amazon S3, Azure Storage, Dropbox, Google Cloud Storage, Google Drive, Office 365 SharePoint, and OneDrive.

Support for uploading Oracle Wallets

The Qlik Oracle Connector now provides additional security capabilities with support for uploading an Oracle Wallet file. Oracle Wallets are containers that store authentication credentials, private keys, certificates, and more. This allows organizations to easily enforce security rules based on defined user privileges contained in an Oracle Wallet. The Qlik Oracle Connector can now access Oracle Wallets through a TLS-encrypted communication channel.

Self-service task management—Create, edit, and delete reload tasks in the hub

New capabilities in the hub allow users with the appropriate permissions to create, edit, and delete reload tasks and scheduled triggers.

This feature is released under the feature flag "HUB_CREATE_EDIT_DELETE_TASK", and is disabled (set to false) by default.

6 What's new in Qlik Sense February 2022

This section provides Qlik Sense business users, analytic creators, and data integrators a summary of the features and improvements available in Qlik Sense Enterprise on Windows.



Qlik Sense administrators should review the [What's New](#) section in the Qlik Sense for Administrators documentation set.

Qlik Sense developers should review the [What's New](#) section in the Qlik Sense for Developers documentation set.

6.1 Augmented analytics

Period over period dashboards in Insight Advisor

Insight Advisor now returns more robust period over period analyses in the form of entire dashboards, when users select or search for applicable fields. Instead of a simple chart, you get one or more charts and KPIs, along with a filter pane, allowing you to make selections and further explore comparative performance.

Insight Advisor supports dollar-sign expansion

Insight Advisor can now recognize expressions contained in variables and use them when generating analytics. This allows users who manage expressions in variables to generate more relevant insights.

Business logic fine grain controls in Insight Advisor

Business logic allows you to define the default grain for a calendar period, such as a yearly, quarterly or monthly basis. When you create behaviors such as default calendar periods, you can now specify whether to use or ignore the grain for a particular analysis, providing more advanced fine grain controls.

New business logic options for customization in Insight Advisor

New parameters have been added to business logic that help refine the analysis generated by Insight Advisor. Options include setting trend direction and sort order for a measure, specifying favorite types of analyses to be used by the system, and specifying the overall aggregation type for complex expressions.

- Favorable trends: Sets whether the desired trend for a measure is to increase or decrease.
- Favorite: Identifies a measure of interest for Insight Advisor so that Insight Advisor will use the measure more often when generating visualizations without user queries or selections.
- Overall aggregation: Helps Insight Advisor determine which aggregation to use for queries that involve master measures with complex expressions where the aggregation is not clear from the outset.

6.2 Visualizations and dashboards

Improvements to the grid chart

Improvements to the grid chart now let you show labels for each data point in the chart. Labels identify the value of the measure for the data point. In preparation for the deprecation of the Heatmap extension in this release, the grid chart now includes two layouts, **Standard** and the new **Heatmap** option. The heatmap layout adds the functionality of a heatmap chart to the grid chart. As of this release, the Heatmap chart extension will no longer be supported.

Improvements to managing variables

A top request through Ideation, the updated, flexible **Variables** dialog simplifies the management of chart variables. Displaying all variable elements, including name, description, value, and tags, it allows you to add, search, and duplicate variables. You can even delete multiple (up to 20) variables simultaneously. These capabilities alleviate manual work and help teams work more efficiently.

6.3 Connectors

Qlik Sense connector for Google Drive

Qlik Sense connector for Google Drive is included with Qlik Sense Enterprise Client-Managed. Now you can access data stored in Google Drive and load it directly into your Qlik Sense app.

[Qlik Web Storage Provider Connectors](#)

Google Drive and Spreadsheets connector

The Qlik connector for Google Drive and Spreadsheets Metadata is now included with Qlik Sense Enterprise Client-Managed.

Snowflake connectivity enhancements

The Snowflake connector now supports the use of key pair authentication to allow you to use enhanced security measures when accessing Snowflake from Qlik Sense.

Support for AWS IAM credentials authorization with Amazon Redshift

New authentication support with the Amazon Redshift connector allows you to use AWS Identity and Access Management (IAM) credentials authorization. This will allow organizations to assign role-based access privileges in addition to designating access rights individually.

6.4 Data and platform

Formatted export to Excel

Formatting options are now supported when exporting straight tables to Excel. Exported tables now include totals and styling as they appear in Qlik Sense.

Add self-service task management

Users with the appropriate permissions can now easily start, stop, and view reload tasks in the hub.

6.5 Documentation improvements and additions

Scripting help improvements

Set expressions tutorial

Learn how to build set expressions for set analysis.

Chart expressions and load scripts

Many new examples of functions used in chart expressions and load scripts.

7 What's new in Qlik Sense November 2021

7.1 Augmented analytics

Custom chart periods

Users can now customize the analysis period used by Insight Advisor. Previously, users would need permissions to change the business logic. Now, users can make these changes on demand, improving their ability to get the insights they need quickly.

[Creating visualizations with Insight Advisor](#)

DPS period over period analysis

Users can now compare a change in a measure of the current period versus the previous period by using the period over period analysis. The analysis is applied to a line chart which lets users visualize how a measure differs between two periods of analysis.

[Creating visualizations with Insight Advisor](#)

Consistency in accessing fields and master items across Insight Advisor and Insight Advisor Chat

What users can search for and access with the Measure and Dimension buttons with Insight Advisor Chat now depends on if the app is published and if there is a logical model applied to your app.

[Making apps available in Insight Advisor Chat](#)

Create custom responses to question

Custom analyses allow you to create custom responses for specific phrases in Insight Advisor. You can define the analysis that will be returned, such as comparison, ranking, and clustering, using specified input fields. This provides improved control over auto-generated analytics without having to define complex rules.

[Adding custom analyses to Insight Advisor](#)

7.2 Visualizations

New host for satellite tiles

Due to API changes at our satellite tile provider, the host that is used for fetching satellite background data will change from services.arcgisonline.com to ibasemaps-api.arcgis.com. .

The switch took place on December 7, 2021 and should not be noticeable to most users. However, customers who restrict domains with their firewall will need to allow this new domain after the transition. If your maps are not functioning as expected, contact your Qlik administrator. They might need to allow this new service.

Add charts in tooltips

This release includes further customization of tooltips with the ability to embed a master visualization inside the tooltip. This allows you to visualize an overview first then drill down directly in the chart for details. The embedded chart inherits the state with dimension value in focus selected.

[Creating a custom tooltip](#)

Accessibility

Keyboard navigation in view data mode

Straight tables in view data mode have improved keyboard navigation.

[View data mode](#)

Layered labels

This release introduces layered labels, giving you greater control and flexibility over the presentation of graphs. In addition to label options like auto, horizontal, and tilted, you can now layer your labels on the X-axis. Layering staggers your labels to make better use of space and provide more room for your graph.

[Changing the appearance of a visualization: X-axis and Y-axis](#)

All labels visible in line chart

The line chart labels have been enhanced with a new **All** option, which forces data labels to always remain visible. Additionally, the **Auto** option, which automatically shows labels based on the space available, has been improved to display more labels.

[Line chart properties: Appearance](#)

Sizing and positioning of images

You can now add images by URL to rows in straight tables for added context and insight.

[Table properties: Data](#)

Edit master items from the property panel

You can now edit master visualizations, master dimensions, and master measures directly from the property panel.

[Editing a master visualization](#)

[Editing a master dimension](#)

[Editing a master measure](#)

Define default tab in container object

The container object lets you add visualizations in a limited space. By default, the first tab is displayed when you view a container. To improve the user experience, you can now set the default tab when creating or editing a container.

[Creating a container](#)

7.3 Connectors

Snowflake connectivity enhancements

You can now authenticate using key pairing. It is also possible to override the default role in the connection dialog, and specify any role you have access to.

[Create a Snowflake connection](#)

7.4 Platform

Partial reloads for scheduled reloads

You can now use the partial reload option for scheduled reloads in the QMC. Partial reloads have several benefits compared to full reloads:

- Faster, because only data recently changed needs to be loaded. With large data sets the difference is significant.
- Less memory is consumed, because less data is loaded.
- More reliable, because queries to source data run faster, reducing the risk of having network problems.

[Editing tasks](#)

NVDA and Chrome screen reader support

The supported configuration is now NVDA screen reader for Microsoft Windows and Google Chrome.

[Using Qlik Sense with a screen reader](#)

Improvements to the Qlik Sense Enterprise on Windows installer

The installation flow has been improved. It now also supports:

- Setting the listening port during installation.
- Configuring the QRS connection pool size.

[Installing Qlik Sense Enterprise on Windows on a single node](#)

[Installing Qlik Sense in a multi-node site](#)

Show license information in QMC when using a signed key

With a signed license key, license information can now be viewed in the QMC.

[Site license](#)

Allow for SCRAM encryption

Adding support for SCRAM encryption as part of the QRS.

[Database security](#)

8 What's new in Qlik Sense August 2021

8.1 Augmented analytics

Mutual Information Key Driver Analysis

Insight Advisor can now analyze and score the mutual information between fields. This provides a measure of the influence that one field has on another, allowing you to understand which fields are drivers of another target field.

You can also now calculate mutual information between targets and drivers in charts with the MutualInfo function. This allows analysis such as pair-wise mutual information analysis and driver breakdown by value.

8.2 Visualizations

Bars on secondary Axis

Users can now add bars on the secondary axis in a combo chart.

Color per measure

In a combo chart, you can configure each measure to have its own color setting, either a single color or color by expression.

Dark mode base map

A fourth base map type with a dark theme has been added to Map chart options. The dark base map with neutral colors puts the focus on the content and makes light-colored feature layers stand out.

Image by URL in point layer

You can now add an image to a point layer map from a URL. This is in addition to adding images located in the Qlik Cloud media library.

Images in straight table

You can now add URL-based images to straight tables.

9 What's new in Qlik Sense May 2021

9.1 Augmented analytics

Insight Advisor Chat improvements

In this release, we have made several enhancements to Insight Advisor Chat including improved narratives in question responses, a refined UI for question parsing, and more intelligent app selections when responding to questions.

Insight Advisor Search

We have made several improvements to our search-based analysis within Qlik Sense apps, including:

- Analysis of change over time – Insight Advisor will now generate multiple charts for a single analysis type, such as responding with analysis of data between two date periods through a combination of charts.
- Natural language generation (NLG) – Visual charts generated by Insight Advisor Search now have a new option to view narrative insights and interpretations of the data.

9.2 Visualizations

Video player

A new capability to embed and play videos directly in Qlik Sense apps.

Active chart and grid dimensions

Enhanced capabilities added to grid dimensions for your visualizations, which helps better identify chart location and associated dimensions on the design canvas. There is a new indicator on the bottom right side of an object which contains these added dimensions and coordinates when moving or resizing chart objects on your design canvas.

New and improved combo chart

An improved combo chart is now available, with enhanced functionality including support for vertical presentation, reference lines based on measures and dimensions, and line and bar styling.

9.3 Connectors

New Databricks ODBC Connector

To support the growing popularity of Databricks, a Qlik strategic partner, Qlik Sense now includes a pre-configured connector for Databricks, including the new Spark 3.0 engine. Utilizing the Databricks SQL Analytics service, the Databricks Connector enables Qlik Sense users to easily include data in Databricks to gain a comprehensive view of their business as well as generate new insights and data-driven actions.

New Azure Synapse Connector

Qlik continues to expand the number of data sources that users can directly access by introducing a new connector for Azure Synapse Analytics.

Presto Kerberos support

You can now authenticate the Presto Connector with Kerberos kinit authentication.

9.4 Mobile

Support for Microsoft Edge

Support for Microsoft Edge mobile browser on iOS and Android.

9.5 Platform and administration

Self-service task management

Users now have ability to create and edit tasks as part their hub experience for improved management and scheduling.

Enhanced ODAG performance

ODAG performance enhanced by increasing the maximum concurrent ODAG requests from 10 to 50, making it easier for large user groups to leverage ODAG collectively.

10 What's new in Qlik Sense February 2021

10.1 Augmented analytics

Business logic

Default Analysis Periods: Enables users to specify a default period for analysis when using Insight Advisor, for natural language questions that use terms such as "Month", "Quarter" or "Year".

Business logic tutorial

There is a new tutorial showing how to improve search-based analysis in Insight Advisor using business logic. By customizing the logical model and adding vocabulary, you can improve the experience for search-based analysis.

Normalization for K-means clustering

Enhanced functionality in the K-means clustering function that enables the normalization of data to be selected and controlled by the user.

Insight Advisor Chat

Users now can take advantage of the multi-lingual natural language capabilities running on Qlik Cloud, as a remote service.

10.2 Visualizations

- Grid Chart: A new type of visualization that provides a highly effective means of visualizing measures across two dimensions.
- Hidden sheets: App creators can now show or hide sheets based on conditions, Enables enabling them to target different user groups with the same app by tailoring the experience.
- Master measures in expressions: Allows users to reference master measures within expressions, improving productivity and governance.

10.3 Connectivity

- New ODBC connector for Amazon Athena.
- Updated Teradata ODBC Connector, adding support for multi-domain Single Sign-On.

10.4 Qlik Engine

- Merge command in script: A new "Merge" command in Qlik script that allows changed data to be loaded into a Qlik application without having to reload the entire data model.

- Note: This is only the engine functionality and for commercial use it will need additional functionality to allow "partial reload" to be exposed for scheduled reloads vs. via API.

10.5 Documentation improvements and additions

Improvements for data security and Section Access

Documentation for managing data security with Section Access has been revamped with a new organizational structure to highlight the levels of security through the use of data reduction. A clear explanation of Section Access concepts and more authorization script examples have been added.

11 What's new in Qlik Sense November 2020

11.1 Augmented analytics

Business logic

A robust business logic layer provides the ability to create business rules and metadata to customize and guide the behavior of Insight Advisor when generating insights and interacting conversationally with users. It includes the ability to logically group fields, classify data, specify default behaviors, define preferred relationships, and more. Users can create calendar periods to define how measures should be filtered, aggregated, and compared based on preferred time frames. Natural language processing can also be customized, including defining vocabulary rules and synonyms for more natural interaction.

Insight Advisor Chat

Insight Advisor Chat is our next-generation, fully conversational analytics experience native to Qlik Sense, available in the Qlik Sense hub. It is driven by our cognitive engine and uses natural-language processing and generation (NLP and NLG) to understand user intent and generate both narrative and visual responses to questions. It works across Qlik Sense apps and allows people to transition directly to in-app, search-based visual analysis for deeper exploration. Released in September on SaaS, it is now offered as a licensed, value-added product for Qlik Sense on Windows, replacing Qlik Insight Bot. The Windows version supports integrations with Microsoft Teams and Slack.

K-means Clustering - Expanded Functionality

K-means clustering on-board the Qlik Engine was released in September. This function allows data points to be grouped into clusters based on similarity. It's a highly useful function for customer segmentation, fraud detection, and many other use cases.

In this release, we have improved our KMeans functions with auto-clustering support. When a user sets 0 for the number of clusters, an optimal number of clusters for that dataset is calculated. This enhancement builds on Qlik's advanced clustering capabilities.

Insight Advisor visualization alternatives

Insight Advisor now offers alternative visualization options for results generated in search-based analysis, helping to ensure your data is displayed in the most useful and meaningful way. In addition, users will have more chart types to choose from when they view alternatives within Insight Advisor analysis.

11.2 Visual analytics and usability improvements

Images in Custom Tooltips

Users can now load custom images in tooltips for more styling options. They can feature an image from the media library or reference one by a URL. This offers further flexibility and enhanced options for app development, as images are a great way to provide additional context when hovering over an object.

Copy sheets between apps

Users can now copy sheets between apps to improve productivity for power users and application developers. This is a highly requested feature from our customer base, as it drives faster insights across the organization.

Copy measures between objects

Another feature to speed app development, users can quickly copy measures between charts. This is a small, but powerful feature enhancement that drives faster insights within an organization.

Other chart improvements

- The ability to hide disclaimers in charts; a user can now decide whether the disclaimer for additional data points should be visible or not.
- Increased measure limit in the Waterfall chart, which now supports 50 measures instead of 15.
- Extended functionality in table mini charts, including `others`, and `null`.

Usability look and feel

A new modern Qlik Sense theme for better-looking apps. It introduces a modern look and feel with clean styling options for color and font selections, ensuring apps have a fresh appeal by default and are consistent with the Qlik brand.

11.3 Platform

Extended app distribution from Qlik Sense Enterprise Client-Managed to SaaS spaces:

- New "Test Connection" button to easily test cloud connections.
- Use of local bearer token when creating new deployment in multi-cloud setup console.
- Improved navigation for app distribution status and policies, now within a cloud distribution section.
- New link to multi-cloud setup console in QMC.

11.4 Mobile

Qlik Sense Mobile Client Managed updates to support the latest mobile OS releases (iOS14 and Android 11) for Qlik Sense Mobile Client Managed and Qlik Sense Mobile for BlackBerry.

12 What's new in Qlik Sense September 2020

12.1 Augmented intelligence

New search-based visual analysis chart types

When generating insights in search-based visual analysis, Insight Advisor will now auto-generate several new types of visualizations. These include:

- Cluster Chart – Shows clusters using the new k-means clustering function.
- Correlation Chart – Shows correlations using the correlation function.
- Control Chart – Shows how a process changes over time.

Search-based visual analysis on mobile devices

Search-based visual insight generation is now available on handheld mobile devices. This provides a more intuitive means of creating new analytics and insights on small form factors, as opposed to traditional authoring. It is available on handheld devices through the browser-based user interface.

Advanced analytics calculation - K-means clustering

With this release, advanced analytics calculations on-board the Qlik Associative Engine are being introduced. The first function available is k-means clustering - allowing data points to be grouped into clusters based on similarity. This is a highly useful function for customer segmentation, fraud detection, and many other use cases.

12.2 Visual analytics improvements

Improved reference lines

Bar and line charts now include dimension based (vertical) reference lines with formatted labels, useful for enhancing charts with added detail along a time axis (such as start and end dates of important events)

Animator control

The Animator control gives users the ability to animate changing data over time by auto-selecting values in any dimension and playing them in a loop, with or without aggregation.

Other chart improvements

- Number formatting of master measures
- Turn on and off borders in containers
- Custom sorting in Sankey Chart
- Frequency counts in filter pane
- WMS (web map service) layer opacity
- Hover icons toggle

12.3 Administration improvements

App distribution

QSEoW now supports app distribution from a multi-node deployment into a Qlik Sense Enterprise SaaS tenant.

QMC status page

The status page offers a better way to understand and monitor app distributions, such as showing the user a link to their Multi-Cloud Setup Console.

Self-service hub improvements

Table view in the hub presents a scalable and easy to use app manager. The following features are included:

- Columns for name, description, owner, published (when applicable), Data last reloaded (when applicable), details
- Sort by clicking on the column headers
- Keyboard shortcut just like grid (Ctrl + g) and list view (Ctrl + Shift + l)
- Navigate with keyboard
- Support for Qlik-supported screen readers

12.4 Other improvements

Support for NFS protocol

When installing Qlik Sense Enterprise on Windows, you can connect to a file store through the Network File System (NFS) protocol. With NFS support, you can use a Linux-based file store, for example, when using Google Cloud Platform.

Data literacy built-in

Qlik has focused on our vision for a data-literate world. As part of this mission, and keeping first-time users in mind, new capabilities have been added within Qlik Sense to assist and educate users about their data. For first-time users, these capabilities help explain data concepts unique to Qlik to expedite their onboarding and shorten the time to insights, including more context for chart creation and additional explanation for dimensions/measures.

13 What's new in Qlik Sense June 2020

13.1 Visualizations and dashboards

Sparkline chart

In a straight table, you can now add a mini chart to a table cell. This mini chart, called a Sparkline chart, is a small line chart that lets you measure across a second dimension.

New bullet chart

The new bullet chart is a custom-built gauge for displaying KPIs. The new bullet chart has extended customization and improved functionality. The new bullet chart replaces the bullet chart that was previously included as part of the visualization bundle.

Table customization

When using either the straight table or the pivot table, you can now set the size of the scrollbar, and you can highlight rows on hover.

Org chart enhancements

It is now easier to interact with Org charts with improvements such as a home button, border color options, and automatic resizing.

Filter pane enhancements

When using a filter pane, text can now be aligned left, right, or center, and users can copy values to their local clipboard.

Number formatting

You can now customize the color range when using color by measure or color by expression when using bar, line, and pie charts.

Custom Tooltip enhancements

Improved custom tooltip options:

- Tooltip support for line charts
- More information in tooltips for all charts
- Dynamic titles with expressions
- Support for expression modifiers: `ALL` and `TOTAL`
- Option to hide default rows

13.2 Data management, connectivity and integration

Dynamic views and ODAG improvements

Dynamic views now provide more information during refresh and have received significant performance improvements. The scripting required for dynamic views and ODAG has also been greatly simplified, and the amount of code needed for passing selections and making queries is now minimal.

Qlik Sense Desktop authentication against SaaS

You can now authenticate your Qlik Sense Desktop against SaaS editions, either Qlik Sense Business or Qlik Sense Enterprise SaaS. Now, any user with creation rights on any edition of Qlik Sense can unlock Qlik Sense Desktop for local and offline users.

[Authenticating against Qlik Sense Cloud](#)

13.3 Administration improvements

Qlik Management Console improvements

Improved accessibility by exposing custom filter capabilities on the Qlik Management Console landing page, which enables the user to save table views as custom filters, providing the ability to easily select how to best present a table.

Licensing improvements

Qlik now offers license proxy authentication to simplify license management for customers. License proxy authentication provides a secure way of managing license authentication between QlikView and Qlik Sense deployments, and the Qlik Licensing Service. Qlik Licensing Service provides significant benefit by removing the need for customers to oversee the governance and compliance requirements.

14 What's new in Qlik Sense April 2020

14.1 Visualizations and dashboards

This release brings many new visualization features and styling options to Qlik Sense, letting you create more advanced and customized analytics. We have added an organization chart (Org chart) to the visualization bundle, and you can now add a relative modifier to bar charts, combo charts, line charts, and tables. In addition, we have made improvements to tables, pivot tables, KPIs, maps, and custom tooltips, and we have enhanced app navigation.

Visualizations

Organization chart

We have enhanced our visualization bundle by adding a new chart to represent organizations. The org chart lets you create organization charts from data with a tree structure. You can navigate through the hierarchy by expanding and collapsing the tree.

Search in tables

In addition to the standard search by a URL link, you can now search by a link label in a table. There is also the option to disengage native scroll on touch to enable first column freeze.

Pivot table improvements

There are two improvements to pivot tables:

- You can now save your pivot table layout state. This lets you store expanded or collapsed nodes as a bookmark. This improvement is ideal for sharing views of large pivot tables with many nodes.
- You can now sort by first measure. This sorting method precedes all other sorting. This lets you sort categories differently, for example, sorting category by sales rather than by name.

Map chart improvements

We have made several enhancements to map layer properties. Now, you have number formatting options in the map chart for color and size by measure. We have increased the limit of items on point and area map layers to 50,000.

Relative modifier (percentages)

The relative modifier provides you with a quick way to display the relative percentage of a measure, without having to write expression syntax. You can use it to see the impact of specific selections, relative to the selection, relative to the total, or relative to other fields. This is available as a setting is available in bar charts, combo charts, line charts, and tables.

Dashboards and applications

Tooltip customization

You can now choose to exclude default measures in tooltips. You can also hide all or part of the tooltips from a visualization.

Sheet trigger

This feature lets you add actions that are triggered when a user opens a sheet. With this feature, you can enable more sophisticated navigation functionality. Actions include setting selections, field states, bookmarks, and variables.

Global Navigation

Global navigation provides you with a new approach to top-level navigation through drop-down lists in the Qlik Sense client.

[Tabbed navigation](#)

Themes and styling

KPI font layout

You have new options when choosing the layout behavior for fonts in KPIs. You can choose a fixed layout (same text size always), fluid layout (text size depends on box size), or responsive layout (adapts to the size of the box and the length of text).

14.2 Advanced authoring

App distribution into spaces

In a multi-cloud deployment, you have additional controls over content distribution policies. You can now send staged apps from a Qlik Sense Enterprise on Windows environment directly to a managed space in Qlik Sense Enterprise SaaS, for broader consumption across the organization. These apps are also refreshed in the managed space after they are reloaded in the Windows environment.

[Publishing to cloud hubs](#)

Share bookmarks

To improve collaboration, we have added a new way to share bookmarks in published apps. You can copy a link to a community bookmark, which can then be shared in other channels such as email or Slack.

Secure Scheduled reloads

You can now run secure scheduled reloads through the QMC, which improves protection and governance around data access. This includes single sign-on support across multiple domains.

15 What's new in Qlik Sense February 2020

15.1 Augmented Intelligence

With this release, we have made several improvements to our visual insight generation capabilities in Qlik Sense, driven by the Qlik cognitive engine. We have added support for radar charts, allowing our cognitive engine to automatically generate this type of visualization when applicable criteria is met. We have also improved pattern detection for natural language processing (NLP), as well as made natural language search available in Qlik Sense Enterprise on Windows. These capabilities help improve the quality and relevance of AI-generated insight suggestions in Qlik Sense.

Improved Natural Language Processing (NLP)

We have improved NLP in the Qlik cognitive engine with better pattern detection, driving more relevant insight suggestions.

Natural language support

Natural language support in Insights is enabled for new installs of Qlik Sense Enterprise on Windows. It is possible to enable it when upgrading an existing install, but this requires the server SSL certificates to be replaced. Also, we have improved Natural Language Insights with better pattern detection across all platforms.

Extended chart support in Insights

Support is added for Radar chart in Insights. Users will see this when there are 2 dimensions with 12 or fewer values, and a measure. The distinct ratio between the values on the first dimension must be below 0.3.

15.2 Visualizations and dashboards

This release brings many new visualization features and styling options to Qlik Sense, allowing users to create more advanced and customized analytics. We have added trendlines, indicators in tables, moving average and difference modifiers, custom tooltips, and a duplicate function for measures – all driven by customer feedback. New styling options include dimension, header and cell options for pivot tables, a new action button with improved styling, line options for the line chart, custom sorting in the funnel chart, and font family support in custom themes. And we have made improvements to mapping functionality and presentation. With these features, we have once again expanded the quality and power of the visualization and dashboarding capabilities in Qlik Sense.

Visualization

Trend lines

This release features new statistical capabilities to easily add a trend line to both the bar and the line chart. Trend line options include average, linear, exponential, logarithmic, power and polynomial of second, third, and fourth degree.

Table trend indicators

The straight table is being enhanced with graphical elements. The user can add conditional icons similar to the KPI object. Table indicators are useful for presenting trends and warning information and highlighting essential information.

Moving average and difference

Two new modifiers are added to the bar chart, line chart, table, and combo chart. The new modifiers support full moving average and difference in steps, for instance rolling 12 months average. The modifier concept provides a quick and easy way to add another level to any expression. The generated expressions are available for inspection copy and edit.

Custom map point symbols

In addition to standard shapes the user can now add any image in the point layer of the Map chart. The point layer also supports the rotation of images. Having images as map symbols is highly requested by customers, typical use cases are separating types of assets or using logos on the map.

Funnel chart enhancements

A new custom sort option is available for funnel charts. This feature is valuable when a user wants to model a less strict funnel.

Dashboards and applications

Improved action button

We have improved our action button with more styling options, including background color and image, label font size and color and border color, width, and radius. This was highly requested by our customers.

[Button](#)

Custom tooltips

Custom tooltips are available for the bar chart, combo chart, map chart, pie chart, scatter plot, and the treemap. You can tailor the tooltip by adding any measure. The measure can be number formatted. This was highly requested by our customers.

Map layer selection toggle

The content creator can now decide which layers should be selectable or not. This is convenient for layers holding background information.

Condition background layers

Map background layers can be switched on or off using a Show conditions option.

Duplicate dimensions and measures

You can now create a master dimension based on a dimension created in a visualization.

Themes and styling

Pivot table styling

Styling includes header font size and color, cell font size and color, text alignment and color of dimension values. Styling improves readability and gives users the power to create and customize the look and feel of Qlik Sense tables.

Line chart styling

The line chart has been improved with a number of new styling options. The lines can be individually styled per measure or for the whole chart.

- Line thickness
- Line style: solid or dashed
- Line curve: smooth or straight
- Vertical presentation

Extended font family selection

The new theme setting allows developers to set the font family for text in Qlik Sense charts. The ability to style apps with custom fonts has been requested by many customers.

15.3 Data management, connectivity and integration

With this release, we expanded the integration between Qlik Sense and Qlik Catalog, as well as overall Qlik Sense connectivity.

QVD Catalog Browsing within Qlik Sense via Qlik Catalog

Since the June 2019 release, Qlik Catalog has had the ability to ingest, profile, refine, catalog and publish data stored within QVD files. That was supplemented by the introduction of a new offering in September 2019 that is solely focused on helping Qlik customers organize and find the QVD files – Qlik Catalog for QVDs. We have now additionally strengthened the integration between Qlik Sense and Qlik Catalog by introducing a new capability in Qlik Sense February 2020. For customers that have Qlik Sense and either Qlik Catalog product, Qlik Sense users will now be able to view the QVD catalog from within the Qlik Sense data manager. This now allows any Qlik Sense user to take advantage of Qlik Catalog within leaving their familiar Qlik Sense environment.

- Easily find and select QVDs from within Qlik Sense data manager. No need to navigate complex folder hierarchies or rely on file naming conventions.
- Take advantage of Qlik Catalog capabilities without leaving Qlik Sense.

Connectors

Google Big Query improvements

Support added for Google Big Query Storage API. Service authentication option is now supported for Google Big Query connector.

Back end improvements

- Significant performance improvements on load balance data reload tasks.
- Upgraded SAML2 component to fulfill the latest security enhancements.

16 What's new in Qlik Sense November 2019

16.1 Visualizations and Mapping

Table styling

This feature gives the app developer more control over the style, look and feel when creating tables, including:

- Header font size and color
- Cell font size and color

Improved Accumulation in Bar chart, Line chart, Combo chart, and Table

- Short cut to accumulation on measures in UI with check box option
- Accumulation can be made over the whole dimension or in a finite number of steps

Mekko chart

A new native chart also known by other names such as Marimekko chart, Mosaic plot or Mondrian diagram. This chart is commonly used in finance and marketing use cases.

Useful for showing how big the share is of different categories, e.g. sales per company, channel and combined; instead of having four regular bar charts, the Mekko can capture all the aspects.

[Mekko chart](#)

Exclude map layers from auto-zoom

The ability to exclude map layers from auto zoom gives more layout control in the map chart. When building map applications, it is sometimes important to exclude layers from zoom, typically background layers such as grid layers, background/cad drawings, etc. The default behavior is to zoom out so all layers are visible.

Trellis container improvements, an enhancement to the visualization bundle

- You can now use 2 dimensions to create a grid with one dimension in each axis. You can also use calculated dimensions.
- You can show a border to highlight the individual trellis charts, and adjust width, color and style.
- Extended advanced mode support enables trellising of more complex charts.

16.2 Qlik Sense Mobile Client Managed

Support for offline access to apps with Qlik Sense Mobile for Android

With Qlik Sense Mobile November 2019 you can download Sense apps for offline access on Android devices.

17 What's new in Qlik Sense September 2019

17.1 Advanced Authoring

Default Landing Sheets

Building upon the ability to apply predefined selections when a user opens a Qlik Sense application, Qlik Sense September 2019 enables developers to choose a landing sheet to direct the user to a specific sheet. This action can be combined with predefined selections to provide the desired focal point in the analysis process for users, which enhances the Guided Analytics experience in Qlik Sense.

[Setting a default bookmark to create an app landing page](#)

Increased Search Limit on Filter Panes

The limit of search values on a filter pane has been increased to 5,000 characters, allowing more values to be selected in bulk. Previously, search values on a filter pane had a limit of 255 characters, which prevented users from copying and pasting a large number of distinct values to apply selections on a field.

Expression Promotion

This feature provides a fast and efficient way to promote a dimension or measure created in an object to a master item. With this capability, application creators have a better workflow when creating and testing expressions that will be later used across the entire application, increasing productivity and driving reuse.

17.2 Visualizations and Mapping

Location finder

The map chart now features a new option to show your location. A new button allows users to fetch their location from the device being used when the button is clicked, it then adds an icon to the map, and zooms to the user's current location. Users can now perform location-based search using the circle select, which delivers value in mobile and field use cases. Due to security reasons, the Qlik Sense Mobile Client Managed app for iOS currently does not support this feature. This functionality can however be accessed through the mobile browser on iOS.

Variance Waterfall Chart

The Variance Waterfall is a new chart in the visualization bundle that provides variance analysis between two measures shown over the different values of a dimension. This type of chart is typically used in financial actual versus forecast analysis. This highly requested chart type provides a more robust financial charting option and fills an important gap for financial analysis.

Pie chart styling

This release features new styling options for the pie chart in the same fashion as the bar chart. These include the ability to tune the pie chart outline width, color, corner radius and inner radius, which gives the user a variety of options to improve the style, look, and feel of the chart. This helps customers moving from QlikView

to Qlik Sense and reduces the need for mashup and extension development.

17.3 Qlik Sense Mobile Client Managed

Support for MobileIron using Web@work

Users can now use Qlik Sense within a MobileIron environment using Web@work, Safari and Chrome browsers, expanding access within secure MDM environments.

Support for iPadOS

Qlik Sense Mobile Client Managed now supports the new iPadOS.



Support for iPadOS is dependent on the timing of the iPadOS release.

[System requirements](#)

17.4 Qlik Connectors

Salesforce connector

Salesforce connector is now available on all Qlik Sense editions:

- Added to Qlik Sense Business, Qlik Sense Enterprise SaaS and Qlik Sense Enterprise on Kubernetes.
- Updated SFDC connector to stay up to date with the latest API changes.

Snowflake connectivity

Integrated Snowflake connectivity on all Qlik Sense editions and deployment options except for Qlik Sense Cloud Business.

18 What's new in Qlik Sense June 2019?

18.1 Advanced Authoring

Copy value to clipboard

With Qlik Sense June 2019 you can now copy values to clipboard from tables. This improvement will reduce several steps and increase productivity when reusing values from a Qlik Sense application.

Grid layout on mobile devices

Qlik Sense app developers can choose the rendering layout for sheets on small screens, such as tablets and mobile phones. List view remains the default option, and only shows the visualization titles. The new Grid view shows a small preview of your visualizations, and reduces the amount of white space. This gives developers more flexibility and control over how apps designed for mobile are displayed.

[Changing the small screen layout](#)

Native container object

The container object extension was first released as part of the Dashboard bundle, and was very well received for the flexibility it adds to a Qlik Sense application sheet space. The Qlik Sense June 2019 release brings the first ever release of a native Container object, providing the existing functionality from the extension, while adding better integration capabilities with other authoring actions. This first release of the native container object will not support Qlik NPrinting, but this is planned for future releases.

With the release of the native container object, current applications using the existing extension objects will continue to work as before but the two Container Object extensions available in the Dashboard bundle will no longer be listed as new object options for applications.

18.2 Visualizations and Mapping

New charts for the Visualization Bundle

- Trellis container

Makes small multiples of a master visualization. Great chart for showing trends across several dimensions.

- P&L pivot chart

A pivot table with customization options for easier creation of financial reports, such as profit and loss statements.

Improved bar chart

- Labels on stacked bars.
- Improved tooltip for stacked bars now displays positive and negative totals.
- Cut bars also for stacked bars when bars go outside of the measure range.

- You can create a chart with measures only, without dimension.
- Styling options for the number of bars: auto, all or a fixed number.
- Styling options for stroke color, width and bar width.
- Toggle mini chart for all scales, not just for continuous axis.

Exploration menu for maps

The exploration menu for maps allows users to toggle layers and labels, adjust object size, and also change the base map image, map language, legend and zoom configuration.

18.3 Qlik Sense Mobile Client Managed

Push notifications to Qlik Sense Mobile Client Managed

With the June release, Qlik Sense Mobile Client Managed for iOS and Android introduce support for native mobile push notifications. Administrators can push system notifications to their users from the Qlik Management Console. Notifications can be sent to all users, or to specific subsets of users, and can include a link into a dashboard. System notifications can be useful to inform users about updates to dashboard, or to announce system events.

System notifications and System notification policies features are available only on Qlik Sense Enterprise on Windows installations licensed with a signed key

[Managing system notifications](#)

[Managing system notification policies](#)

18.4 Multi-Cloud

The following capabilities are now available for Qlik Sense Enterprise SaaS(SaaS) and Qlik Sense Enterprise on Kubernetes deployments:

- Insight advisor precedent based learning.
- Use of IdP groups for access control
- Teradata connector
- Download a single Visualization as data to Excel
- Support for RedHat Openshift in Qlik Sense Enterprise on Kubernetes. This features is not available in Qlik Sense Enterprise SaaS

18.5 Qlik Connectors

- single sign-on (SSO) support in Windows.

19 What's new in Qlik Sense April 2019?

19.1 Augmented Intelligence

Associative insights

Offering powerful new Augmented Intelligence (AI) insights that reveal what is hidden in your data. This feature is driven by the Qlik cognitive engine and Qlik associative engine working together.

Associative insights helps users discover and understand the significance of values in the data that are not selected. Look for **Associative insights** in the selections tool.

19.2 Visualizations and Mapping

We have introduced Bar & Area Chart and Bullet Chart in the Visualization Bundle, and a new capability to set default map backgrounds for improved customization.

Maps enhancements

It is now possible to set a default map background for all new map charts. Enabling further customization of backgrounds, now users can choose to set their own predefined background instead of using the Qlik standard background. This feature provides better support for handling geographic considerations such as disputed regions with politically sensitive boundaries.

In addition, you will find the following feature enhancements for mapping:

- The ability to include new lines in labels, with an improved look of labels in background maps (also added to Qlik GeoAnalytics).
- New size legends for map point layers.
- The capability to insert images in map background layers. This is suitable for simple images such as drawings, floor plans, etc.

Visualization Bundle

Two new charts in the visualization bundle

- Bar & Area chart: a bar chart with transition effects, connectors and labels on tucked bars.
- Bullet chart: a bar chart ideally used for tracking towards a goal, showing target, or actual and bands for good/bad/average.

19.3 Multi-Cloud

With Qlik Sense April 2019 we have introduced a standalone SaaS deployment option for Qlik Sense Enterprise (QSE), with the ability to create, reload, and consume Qlik Sense apps entirely on Qlik's hosted cloud. This release also enables parity between Qlik Sense Enterprise on Windows and Qlik Sense Enterprise on

Kubernetes for adding, updating, and removing themes and extensions. With Qlik Sense April 2019, all deployment options for Qlik Sense Enterprise, which includes Qlik Cloud (SaaS), Kubernetes, and Windows , are available independently and work together as part of a multi-cloud deployment.

The following capabilities are now available for Qlik Sense Enterprise deployed on Qlik Cloud (SaaS) and Kubernetes:

- Create apps
- Connect to data sources
- Store/Use QVDs / Files
- Schedule app reloads
- Team Co-development of apps
- Personal and shared spaces
- Role-based access control
- Share sheets/bookmarks

[Granting access to sheets, bookmarks, and stories](#)

- Upload / export apps
- Console for management

[Management console](#)

- License integration
- Bundled extensions
- Custom extensions (only on Qlik Sense Enterprise on Kubernetes)
- Mashups (only on Qlik Sense Enterprise on Kubernetes)
- ODAG
- API (reload only)
- Usage auditing

[Events](#)

- Link to QlikView documents (available both in Qlik Cloud and Qlik Sense Enterprise on Kubernetes)

19.4 Qlik Sense Mobile Client Managed

Qlik Sense Mobile for BlackBerry

Qlik Sense Mobile for BlackBerry is the new Qlik Sense Mobile Client Managed application built for the BlackBerry Dynamics EMM platform, supporting advanced management and security for BYOD (bring your own device) environments. It allows BlackBerry Dynamics EMM users to access Qlik Sense in a containerized mobile app, with end to end secure communication, including SSO and at-rest encryption. Qlik Sense Mobile for BlackBerry enables administrators to govern Qlik Sense Mobile Client Managed deployments across the organization by setting specific policies and managing app distribution.

[Qlik Sense Mobile for BlackBerry](#)

19.5 Qlik Connectors

- Support for Enterprise Data Sources in a SaaS environment with ODBC Drivers in Qlik Cloud and Multi-Cloud environments.
- Extended connectivity reload capabilities with Basis Scheduled reloads for Enterprise SaaS.

20 What's new in Qlik Sense February 2019?

20.1 Usability Improvements

Single page application flow

Qlik Sense now has a single page application flow. You can move between Data, Analysis, and Story spaces without reloading the page. This creates fluid and natural navigation for end users and developers, with fewer clicks and faster time to insights.

20.2 Advanced Authoring

Dollar-sign expansion preview

The expression editor now provides a way to evaluate the results of calculations with dollar-sign expansions. Developers who use variables can see how these work in the context of an entire expression. This reduces the chances of having the wrong syntax in nested expressions.

Targets.WebHelpOnly">[Expression editor](#)

20.3 Visualizations and Mapping

Visualization Bundle

Visualization bundle is a set of new charts:

- Funnel chart: shows progression of a measure through stages.
- Sankey chart: displays a measure as flow, and how the measure is divided into different categories in one or more stages.
- Radar chart: shows a measure spread on polar chart with two category dimensions, one for the axes and one for areas.
- Heat map chart: shows a matrix of color values with two dimensions and a measure.
- Multi KPI: shows measures with more options and customizations than the standard KPI object.
- Word cloud chart: highlights the most common occurrences in a text dimension.
- Network chart: visualizes a graph with dimensions for nodes and parents and measures for link values.

Dashboard Bundle

One extension added to the existing bundle:

- Share button: creates sharable app links with current sheet and selection.

Support for WMS in map background layers

Map charts can now use maps from third-party WMS servers as background layers. WMS background layers are added through a wizard interface, making configuring the WMS background easy.

[Maps](#)

20.4 Qlik Sense Mobile Client Managed

- The Qlik Sense Mobile Client Managed app is now available for devices running Android OS, allowing to consume Qlik Sense apps online.
- You can now open and consume Qlik Sense mashups using the Qlik Sense Mobile Client Managed app.
- Microsoft Intune EMM solution is now supported for accessing Qlik Sense Enterprise from a mobile device.

20.5 Qlik Connectors

Integrated Single Sign On Support (SSO)

Three drivers in the Qlik ODBC Connector Package now have beta-level support for single sign-on (SSO) (MS SQL Server, Apache Hive and Cloudera Impala).

21 What's new in Qlik Sense November 2018?

21.1 Advancements to Augmented intelligence

Precedent based learning

The time has come for us to unveil Qlik Sense November 2018. We are introducing Machine Learning with precedent/application learning capabilities, leveraging information from users' interactions with the product to feed the cognitive engine for smarter insights and results.

Insight advisor light authoring

Users can now do light authoring with insight advisor. This capability lets users change the suggested visualizations and analytics offered by the cognitive engine, giving greater flexibility and control when discovering insights.

21.2 New advanced authoring features

Alternate states

We have exposed the alternate states functionality that was previously only available through API calls. Alternate states let you create visualizations for comparative analysis based on a state other than the default selection state. Alternate states are a new type of master item, and once created, they can be applied on sheets or visualization objects using the new alternate state option, located under Appearance. With this feature, you can make different selections on the same dimension and compare them in a single visualization, or in two or more visualizations side by side. You can also assign an alternate state to an expression, using set analysis. This allows for complex comparative analysis.

Exposed set analysis

This feature makes it easier for developers to create expressions with the correct set analysis syntax.

Set expression improvements

Set expression in the expression editor dialog has been improved.

Single selection in fields

This capability supports guided analytics use cases in which a single selection will change the analysis flow. The new Qlik Sense field settings allow developers to enable the "always one selected value" for a particular field. The ability to always have one selected value can be used in applications that require better control for conditions in expressions, localization requirements, and other guided analytics use cases.

Fully supported Dashboard Extension bundle [#1]

Qlik Sense November 2018 is introducing extension bundling. This Dashboard extension bundle can be installed alongside Qlik Sense and are fully supported by Qlik. Users can choose to opt out of this bundle option. The objects included in this bundle were previously available as popular extensions on Qlik Branch.

Now as part of the Qlik offering, Qlik will maintain and ensure the quality and performance of these objects upon version upgrades. These extension objects are fully functional, but do not maintain other product standards such as accessibility, multi-language, and RTL.

The Dashboard bundle includes:

- Date range picker: allows users to quickly and easily select dates and ranges.
- Navigation button: lets users quickly navigate to sheets, stories, and websites. Can also trigger actions like selections and set variables.
- On-Demand reporting: generation of Qlik NPrinting reports is now possible from inside apps.
- Tabbed container: ability to switch between visualizations with tabs.
- Show/hide container: similar to tabbed containers, but now with show conditions.
- Variable input: users can set values with buttons, drop-downs, sliders, and input boxes.

[Dashboard bundle](#)

21.3 New ways to collaborate

Sharable Qlik Sense chart links

You can now share chart links with custom selection states. This means you can share initial insights and gather further analysis quickly.

This feature is currently available only in Qlik Cloud and Qlik Sense Enterprise for elastic deployments.

21.4 Advancements in Visualizations and Mapping

New map chart layer

A new map chart layer is now available for displaying pie or bar charts on top of a map to illustrate distribution of values of multiple types.

Improved pie chart

Pie charts now support a second measure to display multiple values in the outer radius. This visualization is also known as rose chart.

Outline opacity setting

Outline opacity setting can now be applied with a slider for increased readability, particularly with small features.

Further advancements in tile map services

Further advancements in tile map services, such as Bing Maps, are now supported by the map chart background layer.

21.5 Improved Management

Import Export App enhancements

Import Export App enhancements gives you the ability to import and export apps with or without data out of the QMC.

Multi-Cloud Developments

Qlik's multi-cloud offering now offers simplified configuration options using self-signed JSON Web Tokens (JWTs). This means components can be connected without requiring specific features of an IdP. In addition, Qlik Cloud can now integrate with ADFS as the identify providers allowing customers with Active Directory infrastructure to authenticate their users.

21.6 Accessibility

Accessible list boxes, keyboard navigation, and screen reader capabilities for list box access.

22 What's new in Qlik Sense September 2018?

22.1 Augmented intelligence

Insight advisor

The insight advisor has been extended to all users of Qlik Sense apps, including consumers of published apps. Users can search and generate insights on master items.

22.2 Advanced authoring

Improvements to sheet control

- You have improved control over Qlik Sense when using devices that support touch and mouse input events.
- App developers can set a default bookmark for an app. The selections are applied when the app is opened.

App customization

App developers can disable responsive layout for sheets and set a custom size in pixels. Your dashboard will be presented to users exactly as you create it. This feature is not supported in mobile device mode.

Expression editor enhancements

- Direct links are provided to the Qlik Sense help page from expression functions.
- Improved categorization of functions.
- Improved search makes it easier to find field names, functions and variables.

Better control in visualizations

- You can show or hide columns in a pivot table based on a formula.
- You can customize master measure items with color scales or gradients. This is a convenient alternative to custom color expressions.

22.3 Visualizations and mapping

New map layer

Density layer is a multi-color gradient map background where the color intensity depends on the weight and closeness of points. This layer is useful, for example, for mapping data such as crime statistics and house values on a neighborhood level.

Map improvements

- Adaptive pixel zoom and pan to optimize performance when browsing dense maps.
- Default colors for added layers. Each new layer has an independent color picked from palette.
- KML files with geographic line data can now be loaded and rendered.
- Field labels for size and width make legends and pop-ups easier to read.

22.4 Management

Improvements to back end work flow

Multiple apps can be moved between streams in the QMC. This builds on the functionality released in the Qlik Sense Enterprise April 2018, where single apps could be moved between streams.

Open source front end framework upgraded to version 1.6.9, which provides better stability and compatibility.

22.5 Mobile

Access to Qlik Sense Enterprise is now supported in AirWatch EMM (Enterprise Mobile Management) environments using the Safari, Chrome or VMware browser.

22.6 Qlik connectors

We continue to expand connectivity options.

- Native connectivity added to MS Azure QSL DB, with tested and supported integration.
- The JIRA connector is released to GA from beta.
[JIRA](#)
- Enhanced security for LDAP authentication, which is built into the Qlik ODBC Connector. This improves customer experience and time to value, with industry recognized encryption and authentication standards.

23 What's new in Qlik Sense June 2018?

23.1 Create, Discover, Collaborate

Insight advisor

Insight advisor is an AI capability that suggests the most relevant insights and visualizations for users to consider and explore. Insight advisor is powered by Qlik cognitive engine, which auto-generates and prioritizes relevant insights and analytics based on Qlik's proprietary algorithms. Creators of apps can use Insight advisor to find insights in the data they load into Qlik Sense and evaluate these insights for the overall data set, dimensions in the data, or search criteria to target specific areas. Insights are contextually aware and work with Qlik's associative engine.

Accelerated self-service

- Alternative chart suggestions: Qlik Cognitive engine now provides a number of alternative charts in the property panel to complement the initial recommendation.
- Switch dimensions and measures: you can now switch dimensions and measures in the property panel with a simple drag and drop.
- You can now quickly change between common aggregation methods using a drop-down menu in the property panel.

Advanced authoring

- Qlik Sense sheets can now be extended vertically to a scrollable mode.
- Grid size can now be changed in sheets with existing objects.

New capabilities in the Expression Editor

Qlik Sense Expression editor now has improved usability:

- **Fields:** create statistical aggregation functions based on field data. Two check boxes allow you to independently insert **Distinct** and **Total** clauses in statistical aggregation functions.
- **Functions:** use this control to enter general Qlik Sense functions into an expression.
- **Variables:** use this control to insert variables into an expression. When a variable is selected, a preview of its definition and value appears.

Filtering data from files

With Qlik Sense June 2018 you can create filtering conditions for the field content in your files. These filters are automatically applied into the script when loading data in Data manager.

Visualizations and mapping

- Line layers are now available in the multi-layer map chart.
- You can now show and hide specific columns in a straight table using expression conditions.
- The mini graph used for scrolling can now be turned off for line charts, bar charts, and combo charts.

Improved app management from the hub

Qlik Sense June 2018 brings new and improved capabilities to how you manage apps from the hub:

- You can now republish an app that you have already published, from the hub. Create a duplicate of your published app first, make your changes, and then republish it back to the same stream as the published app. The stream will be automatically selected from the system based on the app name.
- A new dedicated section for published apps has been added to the hub. Within this section you can still create an app duplicate and then use the new Republishing an app from the hub capability to publish your changes back to your users.
- The app owner can now approve community sheets of a published app and add them to the list of base sheets. The app owner can also decide to unapprove base sheets to the **Community** section.

Accessibility

To further improve Qlik Sense accessibility, the following features have been implemented:

- Screen reader tags
- Keyboard navigation for tables
- Flip from chart to table for value inspection and selection

Accessible Qlik Help site

The following accessibility improvements have been implemented for the entire Qlik Help site, help.qlik.com:

- Users can now navigate and interact with the help site using keyboard keys and shortcuts.
- UI elements, text, search on, and images support zooming, resizing and screen magnifiers.
- Text, icons and images are compliant with accessibility standards for contrast and readability.

23.2 Deploy and Administer

Multi-Cloud

With Qlik Sense June 2018, you can distribute apps developed in Qlik Sense Enterprise on Windows to Qlik Cloud and Qlik Sense Enterprise for elastic deployments. Apps are consumed in the new cloud hub. Qlik Sense Enterprise on Windows and the cloud environment are managed using a single license and sign-on.

For an appropriate license and detailed information, please contact your Qlik representative or Qlik Support.

23.3 Mobile

Support for BlackBerry Access browser.

With Qlik Sense June 2018, you can now access Qlik Sense and consume apps from a mobile device that uses a BlackBerry Access browser. A BlackBerry Dynamics deployment must be set up by an administrator for your users. For information about how to set up and use BlackBerry Access to connect to Qlik Sense, see the support article *Connecting to Qlik Sense using BlackBerry Access*.

23.4 Qlik Connectors

With Qlik Sense June 2018, five new connectors have been added to the ODBC connectors package. All connectors listed here are updated to the latest version of OpenSSL.

- **Presto:** the PrestoDB Connector enables you to create connections that query all the data sources in an environment that have been configured with Presto.

The following connectors are available in beta version:

- **Apache Drill:** the Apache Drill Connector provides access to non-relational data stores.
- **Apache Phoenix:** the Apache Phoenix Connector provides access to relational data stores.
- **Apache Spark:** the Apache Spark Connector is used for direct SQL and HiveQL access to Apache Hadoop/Spark distributions.
- **MongoDB:** the MongoDB Connector gives access to MongoDB, which is a NoSQL database program.

24 What's new in Qlik Sense April 2018?

24.1 Create, Discover, Collaborate

Assisted data visualization with Qlik Sense chart suggestions

Qlik Sense chart suggestions make it easier to create a visualization by allowing you to simply drag and drop fields onto your sheets. Chart suggestions are created using the Cognitive Engine in Qlik, which leverages insights from the data loaded and combines them with best practices for data visualization.

Publishing an app from the hub

In Qlik Sense April 2018 you can publish an app that you have created to any stream for which you have publish access. If you have published an app to a stream, you can move your app between the streams for which you have permission to publish.

Improvements based on customers feedback

Qlik Sense April 2018 introduces a number of improvements based on customer feedback :

- Grid size of an app sheet can now be customized with three different sizes: small, medium or large.
- You can now set custom abbreviations in the load script. For example, you can choose to use Billions instead of G.
- When clicking the Edit button of a linked visualization, a new shortcut redirects you to editing the Master visualization item.
- New keyboard shortcuts for selection back/forward have been added.

Maps visualizations improvements

Qlik Sense April 2018 features significant improvements to the built-in maps visualization:

- Support for multiple layers.
- Labels for point layers and area layers.
- Quick look up of countries, divisions, cities, postal code areas.
- Higher fixed upper limit of number of objects.
- Circle select with distance measure.
- Drill down support.
- Layer control, zoom limit and draw order.
- English or local name in the background map.

Keyboard navigation support for Qlik Sense hub

To improve accessibility, Qlik Sense hub now supports keyboard navigation and shortcuts.

Linking Qlik Sense Mobile Client Managed to third-party applications

Qlik Sense Mobile Client Managed can now interact with third party mobile applications through a custom generated URL (deep link). The link can be embedded within the third party mobile application, with

appropriate selections and filters. Clicking the link opens the app in Qlik Sense Mobile Client Managed with the filters and selections that were applied during original presentation. As a result, user experience is improved and context is provided when interacting with the app.

24.2 Deploy

Deployment improvement

From Qlik Sense April 2018 it is no longer necessary to use port 4244 as the authentication port. If you are using SSL to protect your environment, you can use port 443 as an external facing port for the Qlik Sense Proxy service (QPS).

Per-app VPN mode for Qlik Sense Mobile Client Managed

Qlik Sense Mobile Client Managed now works in a per-app VPN mode with the appropriate VMware Workspace ONE tunnel components. This helps secure network traffic between Qlik Sense Mobile Client Managed and Qlik Sense Enterprise deployed behind a corporate firewall. With this addition, it is no longer necessary for device network traffic to go through a VPN. Only Qlik Sense Mobile apps are routed through the VPN, reducing load on the VPN server.

24.3 Administer

Allocations for new license types

Customers who have purchased Qlik Sense with support for the new Professional and Analyzer license types can now configure the allocation of these licenses in the QMC.

Analytic connections improvement

With Qlik Sense April 2018, Qlik Sense Enterprise administrators now have the ability to add, remove, and reconfigure Analytic connections without additional restarts of any service becoming required for the Analytic connections to function. Administrators now can also stop and start any services in any order without impacting the Analytic connection functionality. Advanced Qlik Sense Desktop users can now develop and use Analytic connections even when the SSE server implementing the Analytic connection is started after the Qlik associative engine.

Enable anonymous users to export data

From Qlik Sense April 2018 anonymous users can print and export data.

25 What's new in Qlik Sense February 2018?

25.1 Create, Discover, Collaborate

Dynamically generated queries to web sources in scripts

You can now generate the URL dynamically before retrieving data using the webfile connector in the data load script. This opens up a range of new possibilities for querying web hosted files.

[Loading files from web resources](#)

Loading a table from an analytic connection

You can now return a full table in a single request to an analytic connection from the data load script, using the new **Extension** clause of **Load**. This improves performance drastically when loading data from an analytic connection during data reload.

[Load](#)

Simplify data preparation with recommended associations

You can now use recommended associations to see possible associations between tables. This facilitates experimentation before applying changes to the data model .

[Managing data associations](#)

Styling an app with custom themes

You can now use custom themes to style an app on a global or granular basis:

- Change colors of background and individual chart elements.
- Define color palettes and color gradients.
- Specify font sizes and font color.

[Styling an app](#)

On-demand App Generation

You can now create On-demand App Generation (ODAG) solutions with support for anonymous usage. This adds capabilities for external facing websites and OEM scenarios.

[Managing big data with on-demand apps](#)

Qlik Sense Mobile Client Managed

You can now use Qlik Sense Mobile Client Managed on your iPhone. The app features the full QIX engine that delivers Qlik's patented associative technology on all supported Apple mobile devices online and offline.

[Qlik Sense Mobile app](#)

Updated ODBC connectors

The Qlik ODBC Connector Package includes updated connectors:

- Oracle
[Oracle](#)
- PostgreSQL
[PostgreSQL](#)

25.2 Deploy, Administer

SSO with Microsoft SQL Server

You can now create a single connection to Microsoft SQL Server that can be shared across a number of different users. Each user is only able to see tables and values in SQL Server as defined by the database security rules.

[Configuring single sign \(SSO\) with Microsoft SQL Server \(MS SQL Server\)](#)

26 What's new in Qlik Sense November 2017?

26.1 Create, Discover, Collaborate

Keyboard navigation in Qlik Sense apps

You can now navigate using your keyboard in Qlik Sense apps. Keyboard navigation is supported both within the app overview page and within the Qlik Sense toolbar.

[Keyboard navigation and shortcuts in Qlik Sense](#)

Details dialog in Data manager

You can now view the operations and transformations performed on tables and fields using the **Details** dialog. **Details** displays the current operations and transformations made to the selected table or field, in the order they are applied in the generated data load script. This enables you to easily see the source of a table or field, the current changes that have been made, and the sequence in which the changes have been applied.

[Viewing table and field transformation details in Data manager](#)

Add data manually

You can now manually enter data in **Add data** in **Data manager**. **Manual entry** in **Add data** enables you to enter data into a table editor and then add it as a table in **Data manager**.

[Adding data manually in Qlik Sense](#)

Additional functions for calculated fields

Additional functions are now available when you are creating calculated fields in **Data manager**.

[Using calculated fields](#)

New ODBC connectors

The Qlik ODBC Connector Package includes two new connectors: Amazon Redshift and Google BigQuery.

26.2 Deploy, Administer

SAML single logout

With SAML single sign-on (SSO), you only have to log in once to access several web sites. There is then a potential risk that one or more sessions are not properly closed. By using SAML single logout you eliminate that risk.

27 What's new in Qlik Sense September 2017?

27.1 Create, Discover, Collaborate

Sequential operations in Data manager

Improvements to **Data manager** enable you to perform sequential transformations on your tables and fields. Where previously you were limited in what transformations you could perform together, such as being able to apply a single data profiling card transformation on a single field, you can now use the data profiling cards, concatenation, calculated fields, and unpivoting data together.

[Managing data in the app with Data manager](#)

Recent colors in color picker

The color picker in Qlik Sense now displays the five most recent colors selected in Qlik Sense.

Navigation and usability improvements to Data manager

Enhancements have been made to Data manager to improve the user experience, including repositioning and restyling the Add data buttons and redesigning the app overview page when no data is loaded.

New visualization: Waterfall chart

You can now illustrate how an initial value is affected by intermediate positive and negative value with the new waterfall chart. For example, you can show the positive and negative contributions of different accounts in an income statement.

Ease-of-use enhancements to on-demand apps

Users now have access to more information and have more control over generated on-demand apps. The navigation point panel now displays the number of rows selected and constraints on individual fields and the selection status related to each constraint. Users also have control over the naming of generated on-demand apps, and they can reload and regenerate on-demand apps based on current selections. They can also copy and regenerate an on-demand app to preserve its current state while creating a version of the app with new selections.

27.2 Deploy, Administer

Qlik Sense Mobile Client Managed app

The Qlik Sense Mobile Client Managed app allows you to securely connect to your Qlik Sense Enterprise deployment from your supported mobile device. The Qlik Sense Mobile Client Managed app can be deployed and managed using either Enterprise Mobile Management (EMM) software, or Apple Developer Enterprise Program tools.

You can download the new Qlik Sense Mobile Client Managed app to your supported iOS device, and then connect to a Qlik Sense Enterprise server. You can download Qlik Sense apps, and then view those apps when you are not connected to a server.

Centralized logging

With the introduction of shared persistence, all nodes now have direct access to a common database and file system. The Qlik Logging Service centralizes the logging by collecting all the messages and inserting them into a PostgreSQL database. This feature is optional but is enabled by default.

28 What's new in Qlik Sense June 2017?

28.1 Create, Discover, Collaborate

On-demand apps

On-demand apps enable you to load and analyze very large volumes of data, so-called big data. On-demand apps provide aggregate views of big data stores and allow you to identify relevant subsets of the data to load for detailed analysis. On-demand apps are created from specially designed selection apps and template apps.

[Managing big data with on-demand apps](#)

New visualizations

You can now use these additional chart types to visualize distribution and range of your data.

The box plot is suitable for comparing range and distribution for groups of numerical data. Data is visualized by a box with whiskers, and a center line in the middle.

The distribution plot is suitable for comparing range and distribution for groups of numerical data. Data is plotted as value points along an axis.

The histogram is suitable for visualizing distribution of numerical data over a continuous interval, or a certain time period. The data is divided into bins.

Synchronizing scripted tables in **Data manager**

You can now synchronize your scripted tables in **Data manager**, enabling you to use the tools available in **Data manager** with your scripted tables.

[Synchronizing scripted tables in Data manager](#)

Data profiling cards

You can now view summaries of table field data and transform that data in **Data manager** using the data profiling cards.

The **Summary** card enables you to view a summary of data in a table's field. In addition, the **Summary** card enables you to view different possible data interpretations, such as viewing the field's data as a dimension or measure, enabling different potential transformation options.

The **Replace** card enables you to select one or more values from a field and replace them with another value.

The **Set nulls** card enables you to select values from a table field and then manually set them as null.

The **Order** card enables you to apply a custom order to values in a dimension field.

The **Split** card enables you to split content from a field into multiple fields.

The **Bucket** card enables the grouping of measure field data into ranges, creating a new field with the specified groupings.

Concatenation in **Data manager**

You can now manually concatenate tables in **Data manager**.

Colors and dimensions

You can now assign colors to master dimensions, ensure that the same colors are used for your dimensions across visualizations.

You can also now assign individual colors to a dimension's values, ensuring that individual values are colored consistently across visualizations when coloring by dimension.

28.2 Administer

Single sign-on to Cloudera Impala

You can now set up SSO connections to Cloudera Impala.

You set up single sign-on by establishing a trusted connection to Cloudera Impala, and then setting the ODBC connection to use the Qlik Sense credentials.

Three new Monitoring apps

To scale with deployment size and expanding log history, and to meet greater needs for Qlik Sense monitoring, the Monitoring apps have been refactored into three new smaller, more specialized apps.

Analytic connections

With the analytic connections you can configure a server side extension to extend the Qlik Sense expression library and support calls to third-party engines.

28.3 History

This PDF contains all the news and updates since the first release of Qlik Sense.

29 What's new in Qlik Sense 3.2?

29.1 Create, Discover, Collaborate

Calendar measures

You can now create calendar measures to analyze data over relative time ranges. For example, you can use calendar measures to compare year-to-date sales figures with figures from the same period the previous year.

Colors and measures

You can select colors for charts based on measures. You can also specify a single color from a full color palette.

29.2 Deploy

Shared persistence

You can now deploy a multi-node site with shared persistence. This means that nodes share a single repository database and a single network folder for the application files. You can either share the central node repository, or set up a high availability database cluster as the repository database. This allows for higher volumes of changes, such as reloads, and removes delays caused by synchronization.

29.3 Administer

Desktop authentication

Qlik Sense Desktop users can now authenticate against their Qlik Sense Enterprise server. The Qlik Sense Enterprise administrator configures an authentication link in the Qlik Management Console, and distributes it to users.

 [Starting Qlik Sense Desktop](#)

 [Configuring Qlik Sense Desktop authentication](#)

QlikView converter

The QlikView converter simplifies the work associated with converting elements from a QlikView document into master items in a Qlik Sense app. Visualizations, expressions, dimensions, and variables can be selected for conversion. The tool is available from the Dev Hub.

30 What's new in Qlik Sense 3.1?

30.1 Create, Discover, Collaborate

City and country recognition when loading data

Geographical data (points and area polygons) are now created automatically when you load data containing names of recognized cities and countries. This enables you to quickly create a map visualization of your data without loading geographical data separately.

Scroll alignment

You can now set the position of the chart scroll bar to start at the end of the data.

Default app theme

You can now change the default app theme. The new Qlik - **Standard** theme adjusts the padding and spacing around objects as well as provides designated spaces for titles. New and existing apps use the Qlik - **Classic** theme by default.

Drag and drop coloring

You can now drag and drop any field, dimension, or measure from the assets panel onto a visualization to change the color.

Navigation

You can now choose to show or hide the navigation menu in the hub.

Search Qlik DataMarket

Qlik DataMarket now includes a search facility that allows users to search for terms and phrases in DataMarket's packages, categories, and data sets.

Salesforce Connector supports primary key chunking

The Qlik Salesforce Connector now supports primary key (PK) chunking when data is loaded in Bulk mode.

Filtering data in the database connectors

The database connectors in the Qlik ODBC Connectors Package installed with Qlik Sense allow subsets of data to be selected by filtering for specific data in database records.

30.2 Administer

Qlik Management Console

The QMC now has an engine setting: **Create search index during reload**, which improves the first search experience for a user.

31 What's new in Qlik Sense 3.0?

31.1 Create, Discover, Collaborate

Qlik connectors installed with Qlik Sense

Qlik connectors that previously had to be installed separately are now installed automatically with Qlik Sense.

- The Qlik REST Connector 1.1 enables Qlik Sense to efficiently load data into a Qlik Sense app from a REST data source. The QlikREST Connector is a generic connector. That is, it is not tailored to a specific REST data source.
- The Qlik Salesforce Connector 14.0 enables Qlik Sense to efficiently load data into a Qlik Sense app from a Salesforce.com data set. Salesforce.com data is available to users with a Salesforce.com account and current access credentials.
- The Qlik database connectors in the ODBC Connectors Package 1.1 enable Qlik Sense to efficiently load data into a Qlik Sense app from databases accessed through supported ODBC (Open Database Connectivity) drivers. When using one of the database connectors in the Qlik ODBC Connectors Package, you do not need to create a DSN connection before connecting to the ODBC database.

Managing table associations in Data manager

The Associations view in Data manager has a new and improved user interface, with bubbles representing the tables in the data model. You can associate your data more easily according to recommendations.

[Managing data associations](#)

Single sign-on to SAP HANA

You can now set up SSO connections to SAP HANA.

You set up single sign-on by establishing a trusted connection to SAP HANA, and then setting the ODBC connection to use the Qlik Sense credentials.

New Qlik DataMarket packages

Qlik DataMarket provides new premium data packages:

- Historical stock prices from major stock exchanges
- Financial data from companies worldwide
- Population indicators for India's states and districts
- Population of Canada by provinces or territories

New multiple-table structure for Qlik DataMarket data sets

A multiple-table structure increases the efficiency with which data is loaded, and it can improve the data associations.

When data is loaded from a Qlik DataMarket data set, it is allocated to multiple individual tables. These tables are associated by generated key fields. Measures and time periods from the data set are consolidated in one table that is assigned the name of the data set. Dimension fields are allocated to individual tables.

[Making associations in Qlik DataMarket](#)

Publishing apps

You can now publish your apps from Qlik Sense. In previous versions, you could only publish of apps from the Qlik Management Console.

[Publishing an app](#)

New language support

Qlik Sense is now available in four new languages: Polish, Turkish, Korean and Traditional Chinese.

Apps now support bidirectional reading order, for languages such as Arabic and Hebrew.

App styling

You can now apply styling to your app to customize the app based on your company standards.

Smart search now includes visual search

You can now search visualizations as well as data items. Search results are returned as a gallery of visualizations in which the search terms are found. Click on a visualization to go directly to the sheet it comes from.

Enhance your apps with widgets

To enhance the appearance and behavior of your apps, you can now create and use a new type of custom object, the widget. Libraries of widgets appear in the assets panel alongside visualization extensions. Widgets are simpler than visualization extensions to build. Typically, widgets are customized KPI objects, simple chart-style visualizations, tables, and sheet navigation objects.

Time-aware charts

You can now use a continuous scale on the x-axis in a line chart to get a accurate view of time-based data.

Shared content in the Qlik Sense hub

QlikView documents can now be shared from QlikView to the Qlik Sense hub.

Qlik NPrinting reports can now be distributed to the Qlik Sense hub.

Additional changes

Geopoints, which you use in map visualizations, can be created automatically from latitude and longitude data.

You can now open a dialog with user information from the hub.

You can add and attach multiple data files to your app in one go, using drag and drop.

31.2 Deploy

IPv6

Qlik Sense now supports IPv6.

Qlik Deployment Console

The Qlik Deployment Console (QDC) is not supported in Qlik Sense 3.1.

31.3 Administer

Monitoring apps in QMC updated

The Monitoring apps now include alternative dimensions and measures for increased customization.

The Operations Monitor app includes basic metadata about users, tasks, apps, and app objects.

The License Monitor app reports token usage compared with allocated and available tokens and shows token usage by stream in addition to app. Basic metadata about users and apps is also available in the app.

32 What's new in Qlik Sense 2.2?

32.1 Create, Discover, Collaborate

Data manager

The data manager user interface is improved, and several features have been added:

- You can now split tables that have been concatenated.
- You can rename tables and fields.
- You can add calculated fields to a table. A calculated field uses an expression to define the result of the field. You can use functions, fields, and operators in the expression.
- You can change the display format of date and timestamp fields.
- Fields containing dates and timestamps are expanded with date attributes that you can use in visualizations and expressions.

Alternative dimensions and measures

You can now add alternative dimensions and measures to some visualizations, using the property panel. The alternative dimensions and measures are then easily accessible and you can quickly change the data in your visualization. The visual exploration menu (available while analyzing visualizations) also supports this.

Export data from pivot tables and other charts

You can now export data from pivot tables, stacked bar and line charts, and treemaps.

Qlik DataMarket

New DataMarket packages are available for global weather, currencies, and stock markets.

The interface for selecting data has been improved.

- Licensed and free data sets are separated into clearly marked categories.
- Hierarchical data sets are structured to facilitate selection.

Data storytelling

Data storytelling has been improved.

- You can now take a snapshot of a visualization when you hover over the visualization.
- You can now choose to create an annotation when you take a snapshot of a visualization. The annotation is displayed in the snapshot library and helps you to distinguish between your snapshots.
- When you add a snapshot to a story slide, the snapshot now snaps to grid.
- When you play a story, a tool tip now appears when you hover over data points.

32.2 Deploy

Qlik Sense Proxy Service metrics

A new metric, `PrintingLoadBalancingDecisions`, is available for the Qlik Sense Proxy Service (QPS).

Qlik Sense Printing Service logging

The folders and files used by the Qlik Sense Printing Service (QPR) for logging have been updated.

Qlik Deployment Console system requirements

Microsoft Windows 10 is now a supported operating system for the Qlik Deployment Console (QDC).

Cloning sites

The Qlik Deployment Console (QDC) can be used to clone entire Qlik Sense sites.

Qlik Sense setup files stored in the S3 bucket

The Amazon Web Services (AWS) plug-in uses Amazon Simple Storage Service (S3) to cache the Qlik Sense setup files, so that they do not have to be loaded for each new site or node.

32.3 Administer

New license option

The capacity-based license gives you the flexibility to configure the number of cores to use on a CPU.

Limit resource consumption by apps

The Qlik Sense Engine Service now includes settings for limiting the amount of resources (memory or time) that can be consumed by apps.

Redesign of the audit page in the Qlik Management Console

The audit overview page has an enhanced table of rules for security, sync, and license.

Operations Monitor app in QMC updated

The Operations Monitor app now incorporates the newly added Printing (Export) logging. A new Export Overview sheet is included in the app.

33 What's new in Qlik Sense 2.1?

Here are the highlights of the new and updated features in Qlik Sense 2.1:

Attach data files	You can now upload and attach data files to an app on a server.
Variables overview	You can now get an overview of all the variables in an unpublished app, and create, edit, and delete variables.
Media library	You can view all images in an app, and upload new images.
Visual exploration	You can now edit some visualization properties through the new exploration menu.
Editable range selection	It is now possible to input exact values and do more precise selections.
Exporting a story	A story can be exported as a PowerPoint presentation.
Repository Snapshot Manager	The Repository Snapshot Manager (RSM) can be used to automate the backup and restore procedures.

33.1 Create

Managing data

Attach data files

You can now upload data files to an app on a server. The file is attached to the app, and you can load and select data to use in the app.

Creating apps and visualizations

Variables overview

In the variables overview, you can now get an overview of all variables in an unpublished app. From the overview, you can create, edit, and delete variables.

Auto-complete and color coding in the expression editor

When typing in the expression editor, you now get an auto-complete list of matching fields, variables, and functions to select from. The added color coding helps you to see where fields, variables, and functions are used in the expression.

Managing images

Media library

You can now find all the images you can use in the app in the media library.

Upload image

You can now upload images to the media library.

33.2 Discover

Interacting with visualizations

Visual exploration

The new exploration menu makes it possible to change some properties while analyzing. The menu is accessible when viewing a visualization in full-screen mode. It is available for line chart, bar chart, pie chart, and scatter plot. It is a simplified version of the properties panel, with the purpose of making data exploration faster and more easily accessible.

Editable range selection

It is now possible to input exact values and make more precise selections.

33.3 Collaborate

Data storytelling

Exporting a story

A story can be exported as a PowerPoint presentation.

33.4 Deploy

Planning Qlik Sense deployments

Backing up and restoring a site

The Repository Snapshot Manager (RSM) can be used to automate the backup and restore procedures.

Deploying Qlik Sense sites in cloud computing environments

Qlik Deployment Console system requirements

Added Microsoft Windows 10 as a supported platform.

33.5 Administer

Monitoring a Qlik Sense site

- The Monitoring apps now handle incomplete or malformed logs without failing during reload.
- The Monitoring apps now generate simple logs for tracking their own reload history.
- The QMC change history contains a more complete and detailed list of changes tracked by the Monitoring apps.
- The *Operations Monitor's Errors and Warnings* table is updated to allow more timely access to other log entries associated with an error or warning.
- The *Operations Monitor* reports aborted reload tasks as well as successful and failed reloads.
- The *Operations Monitor* library includes two new heat map visualizations that can be used on customized sheets for performance monitoring.

34 What's new in Qlik Sense 2.0?

Here are the highlights of the new and updated features in Qlik Sense 2.0:

Qlik DataMarket	The DataMarket service enables you to add data from external sources directly within Qlik Sense. Data from a wide range of frequently-updated sources is available.
Data manager	A new tool for smart loading of data. The data manager uses a new, visual data profiling and modeling capability to make it easier to integrate data from multiple sources without the need to learn a scripting language.
App creation and visualization enhancements	Enhancements and improvements to visualizations include: improved map functionality and map data usage; smoother use of dimensions in pivot tables; calculation conditions can be added to most visualizations; scatter plots can use compressed data; various user interface enhancements.
Export	You can export and print visualizations and sheets as images or to PDF.
Improved search	Faster response times and smarter search behavior.

34.1 Working with Qlik Sense

Managing data

Data manager

You can now add data to your app from a multitude of data sources using the **Data manager**. This tool means you don't need to learn a scripting language to get your data into your apps. As you load, you will also get assistance with creating data relationships based on data profiling. **Data manager** also lets you edit your data selections and add or remove fields easily.

Qlik DataMarket

You can now add data from external sources with Qlik DataMarket. Qlik DataMarket offers an extensive collection of up-to-date and ready-to-use data from external sources accessible directly within Qlik Sense. Qlik DataMarket provides current and historical weather and demographic data, currency exchange rates, as well as economic and societal data.

Creating apps and visualizations

Map visualizations

You can now select to show/hide excluded values and zero values for map visualizations. A map will automatically scale when you resize the window. Improvements have been made to map functionality and design.

Selections in pivot tables

You can now select dimension values directly in the pivot table, without first having to open the dimension drop-down list.

Calculation condition

You can add a calculation condition to most objects (not **Filter pane** or **Text & image**). A calculation condition is set as an expression in the properties panel. The object is calculated only when the expression is fulfilled.

Compressed data in scatter plots

The scatter plot object now provides an overview of the density distribution in large data sets. You can zoom, pan and select data to narrow down the data set, while the context is preserved and shown in a mini chart.

User interface enhancements

The following enhancements have been implemented:

- The setting for conditional colors in KPI objects has been improved and now also updates in real time.
- The gauge visualization now has an arrow indicator for values that are outside of the defined range.
- Column content in table visualizations can now be aligned left, right, or automatically. You can change the setting in the properties panel.
- Improved functionality when making selections in range legends.
- Styling updates to tables and pivot tables.
- You can now convert KPI objects to/from other object types.

Discovering and analyzing

Smart search

The improved search function in Qlik Sense has faster response times, supports searches within selections and is able to search for dimensions. The user interface has been improved and mobile support has been added.

Managing apps

Duplicate apps

You can now duplicate any app you have access to on the hub. You can then work on your own copy, without changing the original, for example.

Delete apps

You can now delete any unpublished app from the hub.

Sharing and collaborating

Exporting a sheet

In Qlik Sense, you can export an entire sheet as a PDF file and print it.

Exporting a visualization

You can export a visualization as an image or to a PDF file.

Using data storytelling

Replacing snapshots on slides

When replacing a snapshot on a slide, you can now choose to navigate to the sheet and visualization from which the snapshot originated.

Showing slide numbers

When playing a story, you can now choose to display the number of the slide you are on and the total number of slides in the story.

Qlik Sense Desktop

Move apps

When you save an app that you have created in Qlik Sense Desktop, the images included in the app are bundled together with the rest of the contents of the app. This makes it easier to share an app with another person or move the app to another computer.

34.2 Deploying Qlik Sense

Planning Qlik Sense deployments

SAML

Qlik Sense supports the use of SAML V2.0 for user authentication.

Port added

Port 4239 has been added.

Qlik Sense Printing Service

Added a service that manages all export-related actions in Qlik Sense.

Qlik Sense Service Dispatcher

Added the Qlik Sense Service Dispatcher (QSD), which is a service controller that is used to launch and manage other Qlik Sense services. The following services are launched by the QSD: Migration Service, Data Profiling Service, and Chart Sharing Service.

New logging framework

Introduced a new logging framework. The legacy logging framework is still available in Qlik Sense, but it is now referred to as tracing.

Guidelines for deploying multi-node sites

Updated guidelines for how to deploy multi-node sites.

Added multi-node deployment scenario that includes a development node

Added a new multi-node deployment scenario that includes information on how to set up a development node.

Added multi-node deployment scenario for geographically dispersed sites

Added a multi-node deployment scenario for geographically dispersed sites.

Backing up and restoring a site

Improved the descriptions of how to backup and restore Qlik Sense sites.

Installing and upgrading

Installing

Installation is now optimized for Central and Rim nodes. Individual feature installation is dependent on the node type selected. Rim nodes can be installed as either Proxy, Engine, Proxy and Engine, or Scheduler where everything that is needed will be installed.

Modifying

The Rim node type can be switched to another type: Proxy, Engine, or Scheduler. Individual features can no longer be added or removed.

Uninstalling

Options have been added so you can remove all data and certificates during Qlik Sense uninstallation.

Updating

An update option is now available when Qlik Sense has been previously installed and a service release (version x.x.x) of the software is available to update the installation. A service release primarily includes software updates and fixes which are then applied to the existing version. Updates are installed without the need to remove earlier updates or the baseline program itself.

Uninstalling an update

Updates can also be uninstalled, in which case the updated version will be reverted back to either the previous service release version or major release version of Qlik Sense, whichever is applicable.

Deploying Qlik Sense sites in cloud computing environments

New user interface

The Sites view in the Qlik Deployment Console (QDC) has a new user interface that makes it easier to manage Qlik Sense sites.

Platform support

The Microsoft Windows Server 2012 R2 platform is now supported for the Qlik Deployment Console (QDC).

.NET framework version

The .NET framework version supported by the QDC is now version 4.5.2.

Amazon Simple Storage Service (S3) bucket

The instructions on how to copy files to the S3 bucket have been removed because the files are now copied automatically.

Firewall rule

A firewall rule, WINRM, has been added in the Security group rules for the Amazon Web Service (AWS) plug-in.

Timeout setting

A timeout setting, Time to wait for a free machine, has been added. The setting is used when the VMware vSphere plug-in is used with a machine pool.

34.3 Administering Qlik Sense

Managing a Qlik Sense site

New tables

New, improved tables have been implemented throughout the QMC. You can now select which columns to display and adjust the width of each column.

Search

You can now perform searches in almost all tables in the QMC. Search criteria can be arranged into subgroups and combined with column filtering.

Download of script log for reload tasks

From the QMC, you can now download script log files for reload tasks.

Monitoring a Qlik Sense site

The Qlik Sense monitoring apps have been redesigned with improved navigation and cleaner layout to take advantage of refinements in Qlik Sense logging that provide more relevant and concise data.

Operations Monitor

The *Operations Monitor* provides a more comprehensive and comprehensible view of reload tasks, user sessions and app usage. New Key Performance Indicators (KPI) make it easy to identify conditions and trends. New sheets and links between sheets make it easier to navigate and locate relevant data.

Troubleshooting Qlik Sense using logs

New help section

New help section that describes how to troubleshoot Qlik Sense using the logs produced by the Qlik Sense services.

35 What's new in Qlik Sense 1.1?

Here are some of the highlights of the new and updated features in the release of Qlik Sense 1.1.

- New chart types: pivot table and KPI
- Date & time fields in visualizations
- Snapshots of maps
- Guidance after installation

35.1 Working with Qlik Sense

Creating apps and visualizations

Pivot table

You can use pivot tables to dynamically change the presentation of the data and focus on areas of interest.

KPI

You use the KPI to track performance. The KPI shows a main value, and, optionally, a complementary value.

Open apps without data

You can open an app without loading its data. This is useful, for example, when there are large amounts of data that would take a long time to load.

Sheet thumbnails

You can change the thumbnail for a sheet to make it easier to find in the app overview.

Duplicate sheets from the global menu

Using the global menu in the app overview or the sheet view, you can duplicate a sheet, even if you are editing or analyzing the sheet.

Edit sheets from shortcut menu

You can start editing a sheet by right-clicking the sheet in the app overview or in the sheet navigator.

Additional sheet details

The details for a sheet include information about when the sheet was published and updated, and by whom.

Date & time fields

You can now use date & time fields in visualizations. These fields are derived fields, which are defined by a calendar template in the data load script, and generated when the script is run. They appear in the assets panel.

Using data storytelling

Duplicate a slide

You can now duplicate a slide in a story.

Snapshots of maps

You can now take snapshots of maps.

Change the resize behavior of a snapshot

You can now change the resize behavior (keep aspect ratio) of a snapshot. When you unlock the snapshot you make it freely-resizable, which uses a progressive disclosure technique.

Story thumbnails

You can change the thumbnail for a story to make it easier to find it in the app overview.

Play stories from the shortcut menu

You can start playing a story by right-clicking/long-touching the story in the app overview or in the story navigator.

Additional story details

The details for a story include information about when the story was published and updated, and by who.

35.2 Deploying Qlik Sense

Installation and setup

Guidance after installation

Online help is now available to guide you through setting up your system when installation is completed.

35.3 Administering Qlik Sense

Managing a Qlik Sense site

Qlik Management Console user interface

The user interface theme in the QMC has been updated with better contrast and improved visual hierarchies. The action bar layout and style have been improved. There is also more table header information, and improved help documentation.

Virtual proxies and load balancing

Virtual proxies are now a separate resource and can be accessed directly from the QMC start page. Load balancing is now done on the virtual proxy level.

Monitoring a Qlik Sense site

Empty charts and tables

Explanatory text has been added to both the Operations Monitor and License Monitor chart and table visualizations when they are empty. The text explains why no data is presented.

Libraries for customizing monitoring apps

Both the Operations Monitor and License Monitor now contain libraries that provide access to the dimensions and measures used for their visualizations. You can use these dimensions and measures to create additional visualizations for your particular environment.

The libraries also include additional chart visualizations that can be added to custom sheets.

QMC Change Log sheet (Operations Monitor)

You can track changes made to the QMC settings on the new QMC Change Log sheet.

Reference lines (Operations Monitor)

Reference lines have been added to the 24-hour Summary and Performance History charts to indicate thresholds and 28-day averages for server CPU and RAM usage.

License usage by app

You can track the number of licenses and tokens used by individual apps on the new Usage by App sheet. The sheet shows the apps for which both login and user access passes are being used. The usage values are the passes that have been used, not the number of passes allocated.

Managing Qlik Sense sites in cloud computing environments

Local user group

A local user group is used to authorize the Qlik Deployment Console (QDC) users.

New ports for communication with QDC

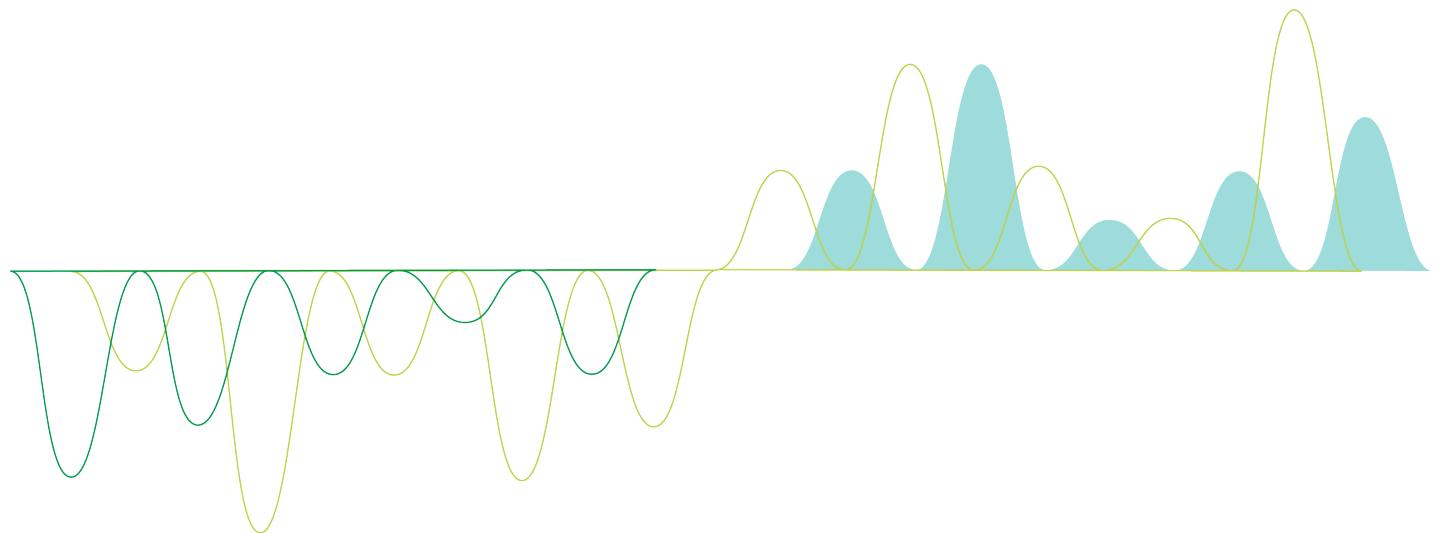
New ports for http, https and SignalRPort are used by the QDC.

Create apps and visualizations

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 About this document	7
2 Creating apps	8
2.1 Foundations	8
Data manager and Data load editor	8
Measures	8
Dimensions	8
2.2 Structure and visuals	8
Sheets	8
Bookmarks	8
Stories	9
2.3 Structuring an app using sheets	9
Overview	9
Creating a new sheet	10
Changing the title and description of a sheet	10
Setting a show condition for a sheet	11
Changing the sheet thumbnail	11
Changing the grid sizing of a sheet	12
Customizing sheet size	14
Changing the small screen layout	14
Extending the sheet area	15
Adding actions to sheets	16
Adding a background color or image	16
Copying, replacing and moving items on sheets	17
Duplicating a sheet	17
Adding actions to sheets	18
2.4 Managing apps	20
Creating an app	20
On-demand apps	20
Styling an app	25
Reloading app data	31
Managing app reload tasks	31
Converting a QlikView document into a Qlik Sense app	38
Manually converting a QlikView document into a Qlik Sense app	38
Changing the title and description of an app	42
Changing the thumbnail of an app	43
Duplicating an app	43
Making apps available in Insight Advisor Chat	44
Turning on chart level scripting	45
Turning off Insight Advisor	45
Deleting an app	45
Uploading image files to media library	46
Deleting image files from media library	47
2.5 Troubleshooting - Creating apps	48
Images are not included in an app that has been moved from one Qlik Sense environment to another	48
Images are not included in an app that has been moved from one Qlik Sense Desktop installation to another	49

Contents

The image I want to use does not seem to work	49
Using Insight Advisor impacts system performance	49
I cannot find Reload when I right-click on an app	50
I cannot manage my app reload tasks in the hub	50
Thumbnails are not included when copying a sheet	50
2.6 Optimizing app performance	50
App complexity	51
App details	51
Monitoring your app	52
Large data volumes	53
Data model performance	53
Sheet performance	56
3 Visualizations	60
3.1 Understand the data sources of your visualizations	60
3.2 Select visualization types that align with your purpose	61
3.3 Update visualizations to improve how data is displayed	61
3.4 Data assets in visualizations	61
Data assets	61
Expressions	62
Data types in visualizations	63
Fields	64
Dimensions	74
Measures	77
Reusing assets with master items	86
Using expressions in visualizations	122
Using chart level scripting in visualizations	133
Searching in the assets panel	134
Designing visualizations with Direct Discovery	135
3.5 Best practices for choosing visualization types	136
Viewing comparisons	136
Viewing relationships	137
Viewing compositions	137
Viewing distributions	137
Viewing performances	138
Viewing data	138
Viewing geography	139
What if no standard chart suits my purpose?	139
3.6 Visualizations	139
Creating a visualization	139
Reusing a visualization	140
Which visualizations are available?	140
Bar chart	142
Box plot	155
Bullet chart	159
Combo chart	162
Distribution plot	168
Filter pane	171

Contents

Gauge	176
Histogram	178
KPI	181
Line chart	184
Map chart	187
Mekko chart	244
Pie chart	248
Pivot table	252
Scatter plot	262
Table	268
Text & image	284
Treemap	287
Waterfall chart	292
Button	297
Container	302
Reference lines	307
Custom tooltips	310
Null values in visualizations	313
Dashboard bundle	313
Visualization bundle	331
Changing a measure/label color	356
Changing background color	356
Alignment	359
Number formatting	359
Format pattern	360
Segment/card	360
Items per row	361
Borders	361
Value/Label layout and formatting	361
3.7 Creating and editing visualizations	406
Creating visualizations	407
Editing visualizations	408
Best practices for designing visualizations	409
Creating visualizations with Insight Advisor	412
Creating visualizations using Insight Advisor chart suggestions	434
Guidelines for visualizations, fields, and naming	436
Using alternate states for comparative analysis	442
Creating a visualization using a custom object	446
Copying a visualization from an existing visualization	447
Creating time-aware charts	448
Changing the data of a visualization	449
Changing the appearance of a visualization	454
Examples	469
RGB	472
ARGB	472
HSL	472
Color keywords	472
Qlik Sense color functions	473

Contents

Example 1: Coloring by a dimension in the visualization	478
Example 2: Coloring by a dimension not included in the visualization	479
Example 1: Color by expression in a table	480
Example 2: Color by expression in a chart	481
Converting a visualization to another kind of visualization	482
Embedding a visualization or a sheet in a web page	483
3.8 Troubleshooting - Creating visualizations	485
I cannot find the fields in the assets panel	485
My chart is not sorted correctly	485
My calendar measures display incorrect aggregations in visualizations	486
There are no time ranges to select in Create calendar measures	486
My date field selected for calendar measures does not use the correct calendar	486
I cannot edit a variable value	487
The map is placing the locations in my location field incorrectly	487
No map is displayed	487
Error message: The data contains invalid geometries that could not be shown on the map.	
Review your data for errors and try again.	488
Error message: The following locations could not be found: <locations>. Review the values in your data and try again.	488
Error message: The following locations could not be located: <locations>. Review the values in your data and try again.	488
Error message: The following locations had more than one result: <locations>. Set a custom scope to clarify which locations to display.	489
Error message: Some lines could not be shown because of invalid data in the width expression. Review your data for errors and try again.	489
Error message: Some density points could no be shown because of invalid data in the weight expression. Review your data for errors and try again.	489
I added an image background layer and cannot see my image	489

1 About this document

Visualizations are used to present the data that is loaded into the app. The selections you make in the app are reflected in all associated visualizations on all sheets.

Read and learn how to create and customize sheets and visualizations in your app. You will also learn about creating reusable master items, and about expressions in visualizations.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

You find the online help, additional guides and much more at help.qlik.com/sense.

2 Creating apps

Qlik Sense apps contain data, and use visualizations to explore that data. Make discoveries by applying selections to visualizations.

The foundation of an app is the data model and load script. Measures and dimensions are reusable data items used to build charts. Sheets and stories display and organize your visualizations. Bookmarks are an easy way to save a specific selection state on a sheet.

Whoever creates an app is automatically designated as its owner. An app can be re-used, modified, and shared with others, depending on access rights. Different actions can be carried out depending on whether the app is published or not.



The .qvf file format is a proprietary format.

2.1 Foundations

Data manager and Data load editor

The Data manager lets you quickly add and transform data. You can also associate data tables. The Data load editor uses scripts to connect to a data source and retrieve the data.

Measures

Measures are calculations used in visualizations. Measures are created from an expression composed of aggregation functions, such as **Sum** or **Max**, combined with one or several fields.

Dimensions

Dimensions are fields used in visualizations. Dimensions determine how the data in a visualization is grouped. For example: total sales per country or number of products per supplier.

2.2 Structure and visuals

Sheets

Sheets contain data visualizations, such as charts and tables. You create structure in your app by grouping visualizations on sheets. For example, you could have a sheet for North American sales, and a different sheet for Europe.

Bookmarks

Bookmarks are shortcuts to a certain set of selections and chart expansions.

Stories

Stories are based on snapshots of visualizations. You present your data by creating a story that guides you to new insights by combining snapshots of data at specific times and selection states.



In Qlik Sense Desktop, apps are typically stored in <user>|Documents|Qlik|Sense|Apps, as <Appname>.qvf, where <Appname> is the name of the app. If you rename an app in Qlik Sense Desktop, the .qvf is not updated.

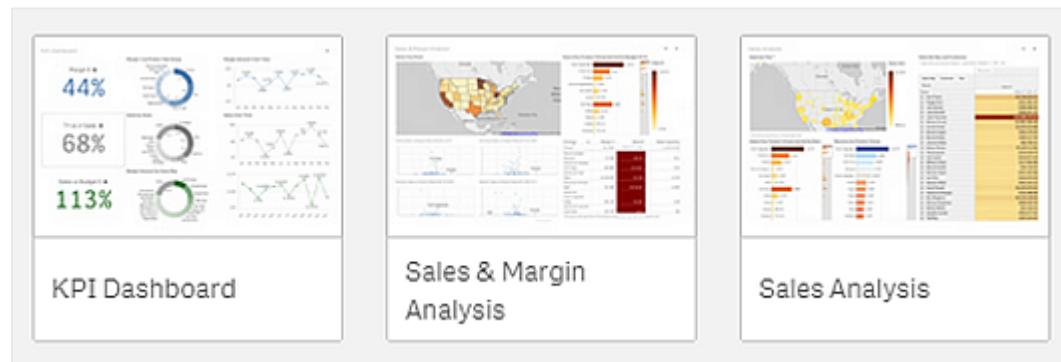
2.3 Structuring an app using sheets

Sheets structure your ideas and purpose of your app. When you create a new app, it is good practice to first build a structure of empty sheets, where each sheet represents an idea or a goal. This gives a good overview of the app.

Overview

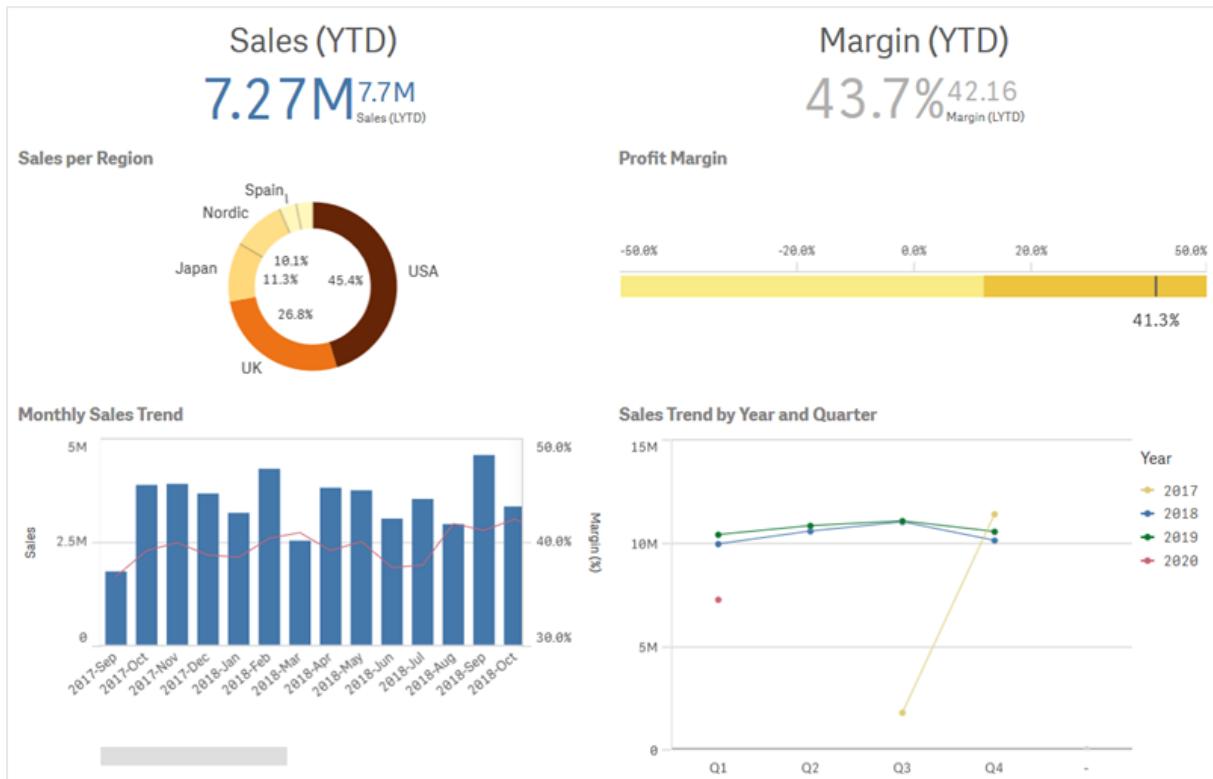
For example, you are creating an overview of your company's key metrics, sales, and margins by state, region, and product. Instead of having all this information in one place, you could structure it by having one sheet for each purpose.

Each sheet has a purpose and an idea of its own.



A sheet is where charts and tables for data visualization are placed. An app can include several sheets. The selections that you make affect visualizations, regardless of which sheets they are on.

An example of a sheet with boxes on the left to select and filter out the data to be presented in the visualizations on the right.



Creating a new sheet

You can create a new sheet from the app overview or from the sheet navigator.



You access the sheet navigator from the sheet view by clicking in the toolbar.

Do the following:

- From the app overview, click to view the sheets.
- Click or **Create new sheet**.
- Give your sheet a title and add a description.
- Click outside the text area to save the title and description.

A new sheet is created.

Changing the title and description of a sheet

You can change the title and description of your sheets. You can either use a fixed sheet title, or a dynamic sheet title based on an expression.

Do the following:

1. In the app overview, click  to view the sheets.
2. Do one of the following:
 - If you are in grid view, , click the sheet title followed by clicking .
 - If you are in list view, , click .
3. Edit **Title** and **Description**.
4. Click outside the text area.

The changes you made are saved.



You can also change a sheet's title and description in the Sheet properties panel.

Using a dynamic sheet title

You can set a dynamic sheet title based on an expression in the **Title expression** property of the sheet properties panel. You can use any valid chart expression.

If you set a dynamic sheet title, the fixed title (**Title**) is not used.

Setting a show condition for a sheet

You can set a condition in **Show condition** to show or hide a sheet in an app depending on if an expression evaluates as true or false. The sheet is only shown if the expression evaluates as true. Hiding a sheet does not hide or exclude data in an app.

For example, you could create a sheet that is only available if certain values are present in your data.

Typically, you should use an if function.

Changing the sheet thumbnail

You can replace the default thumbnail, to make it easier to distinguish between sheets in the app overview and in the sheet navigator. You can use one of the default images, or an image of your own.

Note the following:

- The following formats are supported: .png, .jpg, .jpeg, and .gif.
- The optimal aspect ratio of a thumbnail is 8:5 (width:height).
- You can only add or change the thumbnail of an unpublished sheet.

Do the following:

1. In the app overview, click  to view the sheets.
2. Do one of the following:
 - If you are in grid view, , click the sheet title followed by clicking .
 - If you are in list view, , click .

3. Click  on the default thumbnail.
The **Media library** opens.
4. Click on a folder in the media library, for example **In app** or **Default**.
5. Select the image you want to use as a thumbnail for the sheet and click **Insert**.
6. Click  to stop editing.

The image you selected is now used as a thumbnail for the sheet, and is visible in the sheet navigator and in the app overview.



You can also change a sheet's thumbnail in the sheet navigator at the top right or in the Sheet properties panel.

The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

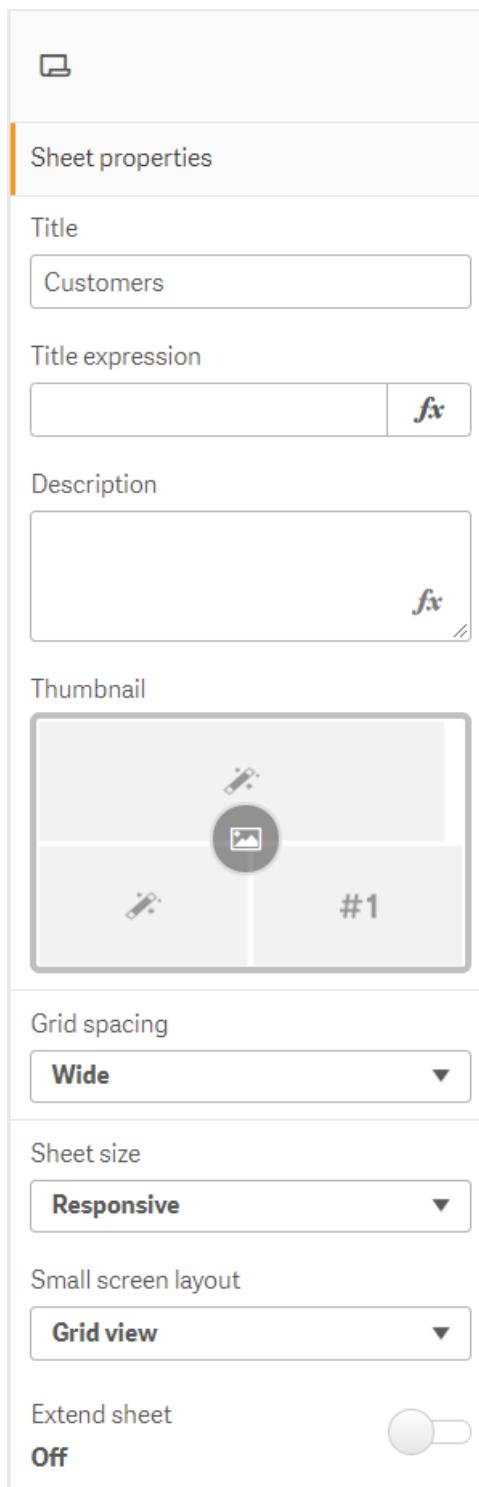
For Qlik Sense Desktop: You can place images in the following folder on your computer:

C:\Users\<user>\Documents\Qlik\Sense\Content\Default. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.

Changing the grid sizing of a sheet

You can adjust the grid sizing of the sheet to be able to fit more visualizations on a sheet, or have more control over how visualizations are positioned. The grid displays when you are adding, moving, or resizing a visualization.

Grid size can be changed in the Sheet properties pane



Do the following:

1. Go to the **Grid spacing** setting under Sheet properties.
2. Change spacing from **Wide** (the default option) to **Medium**, **Narrow**, or **Custom**.

3. If you choose Custom, a slider will appear. Use the slider to change the grid size. The higher the number, the more narrow the grid spacing is.

Customizing sheet size

By default, Qlik Sense uses a responsive layout for sheets that adjusts the sheet to the dimensions of the user's screen. You can set a custom width and height for the sheet if you want to use a non-responsive layout for your sheets. This ensures that your dashboard will be presented to users exactly as you created them, with no adjustments for responsiveness. You can set the height and width of a sheet anywhere between 300 pixels and 4,000 pixels.

If the PDF download of your sheet appears overly zoomed in, switching the **Sheet size** from **Responsive** to **Custom** is recommended.



*Changing the **Sheet size** from **Responsive** to **Custom** may impact the interactive experience for business users. If report output formatting is particularly important, consider creating a report-specific app for better control over themes and layout.*

When a sheet is using a custom size, you can change the grid spacing, but you cannot use **Extend sheet** to change the size.

For **Responsive** sheets that are not extended, the exported PDF of the sheet will be:

- Landscape orientation: 1680 by 1120 pixels
- Portrait orientation: 1120 by 1680 pixels

If you are using a **Custom** sheet size, the PDF output of the sheet will keep the custom pixel values for width and height.

Changing the **Sheet size** from **Responsive** to **Custom** may impact the clarity of the PDF output. If you are manually exporting a custom-sized sheet as a PDF, the best practice is:

- Portrait orientation: values of 1680 by 1120 pixels, or a ratio of 1:1.5
- Landscape orientation: values of 1120 by 1680 pixels, or a ratio of 1.5:1

Extended sheets and custom-sized sheets may have a lower-resolution PDF output if the sheet is too large to clearly display on a single PDF page.

Do the following:

1. Change the **Sheet size** setting from **Responsive** to **Custom**.
2. Enter a width for the sheet in pixels.
3. Enter a height for the sheet in pixels.

Changing the small screen layout

By default, sheets are displayed in List view when viewed on small screens. You can change them to display as a grid. Sheets must be set to responsive layout to display as grid on small screens.

A sheet viewed on a small screen in List view (left) and Grid view (right)

The image shows two side-by-side views of a 'Cars dashboard' sheet. On the left, under 'List view', there are four input fields: 'Name', 'Year', 'Country', and 'Cylinders'. Below them is a large visualization titled 'Avg Price' showing the value '4,81k'. At the bottom is a button labeled 'Car development analysis' with a right-pointing arrow. On the right, under 'Grid view', there is a search bar at the top. Below it is a table with four columns: 'Name', 'Year', 'Country', and 'Cylinders'. Under each column are three summary values: 'Avg Price' (4,81k), 'Avg Acceleration, 0-60' (16,21), and 'Avg Weight' (2,68k). Below the table is a chart titled 'Price development' showing price trends for France and Germany from 1978 to 1982. At the bottom is a table titled 'Totals' with two rows: 'AMC Concord' (USA, 80) and 'AMC Concord D1' (USA, 78). A button at the bottom right says 'Car development analysis grid' with a left-pointing arrow.

Grid view shows a small preview of your visualizations. List view only shows the visualization titles.

Do the following:

1. Under **Sheet properties**, go to **Small screen layout**.
2. From the drop-down, select **Grid view**.

Extending the sheet area

You can extend the area of a sheet vertically if you need to add more visualizations than what can fit on a screen.



Downloaded extended sheets may have a lower quality PDF output if the sheet is too large to clearly display on a single PDF page.

Note the following:

- You need to activate **Extend sheet** under Sheet properties.
- Each extension adds 50% of the original sheet height.
- If you have visualizations that scroll, this can interfere with scrolling of an extended sheet. You should leave some empty space that allows for scrolling when you design the sheet.

You can extend the sheet in two different ways:

Do the following:

- Drag an object to the bottom of the sheet and drop it on the drop zone that appears.
- Toggle **Extend sheet** under **Sheet properties**.

The sheet is now expanded by 50% of the original height. You can scroll vertically in the sheet to access all content.

Adding actions to sheets

You can add actions to sheets that trigger when users navigate to those sheets. For example, you could automatically clear selections in the app when users navigate to a particular sheet.

For more information, see *Adding actions to sheets (page 18)*.

Adding a background color or image

You can customize the background of a sheet. You can chose a specific background color, or color by expression. Any image in the media library can be set as a background. The image can be positioned and resized so it does not fill the entire background.

The following formats are supported: .png, .jpg, .jpeg, and .gif. If you use a .gif, it can be animated or static.

Do the following:

1. Open the sheet in Edit mode.
2. Under **Sheet properties**, click **Styling**.
3. Set **Background** to **Custom**. The default is **Auto**.
4. **Background color:**
 - **Single color:** Choose a color using the color picker.
 - **By expression:** Set a color using a user-defined expression. For more information, see *Examples (page 469)*.
5. **Background image:** Select **Image from media library**. The default is **None**.
6. Click  on the default thumbnail.
The **Media library** opens.
7. Click on a folder in the media library, for example **In app** or **Default**. You can also choose to **Upload media**, if the image you want is not already in the media library.
8. Select the image you want to use and click **Insert**.
9. Use the drop down to change the size of your image:
 - Original size
 - Always fit
 - Fit to width
 - Fit to height
 - Stretch to fit
 - Always fill

10. Under **Position**, you can change the alignment of the image. For example, the image can be centered on the sheet, or in the top-right corner.

The image and background color you selected is now visible on the sheet.

Sheet in Edit mode, with a background image set on the left



Copying, replacing and moving items on sheets

You can copy, replace, and move items on a sheet and between sheets. You can do this in the following ways:

- Using the edit bar on the sheet (and).
- Using the edit bar on the sheet (and).
- By right-clicking and selecting **Cut**, **Copy** and **Paste**.
- With the keyboard shortcuts Ctrl+C, Ctrl+X and Ctrl+V.

Duplicating a sheet

You can duplicate any sheet, regardless of whether it is a sheet that belongs to the app or a sheet you have created yourself. The purpose of duplicating sheets is to save time by reusing content, and to allow you to modify the duplicate so that it fits your needs better. A duplicated sheet contains the same visualizations as the original sheet, and is linked to the same master items. The duplicated sheet is a standalone sheet with no connection to the original sheet. Duplicated sheets appear under **My sheets** in app overview and in the sheet navigator.

You can duplicate a sheet in the following ways:

- Click **Duplicate** in the shortcut menu of a sheet in the app overview or in the sheet navigator.
- Click **Duplicate sheet** in the global menu in the app overview or in sheet view.

Adding actions to sheets

You can set actions to sheets that trigger when users navigate to the sheet. Sheet actions are useful when you want to assist app users with predefined selection or selection controls when they navigate to a sheet. For example, a sheet could automatically apply a bookmark or clear all selections when users navigate to the sheet.

Sheet actions can be configured in the properties panel. You can add multiple actions to a sheet. The actions are performed in the order they are listed under **Actions**. You can change the order of an action by dragging it.

For a visual demo about adding actions to sheets, see [Adding actions to sheets](#).



All expressions used in sheet actions are evaluated before the actions are performed. For example, you could not use an expression in an action that selects results from a previous action as the expression evaluates before the selection is made by the action.

Sheet actions are not triggered if the sheet is an embedded sheet or in a mashup.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
2. Click **Actions** in the properties panel of the sheet.
3. Click **Add action**.
4. Select the action you want to use.
For some actions you need to provide details for the action. For example, for the **Select values in a field** action, you need to select a field, and which value to select in the field.
5. Optionally, after **Label**, enter a name for the action.

Available sheet actions

You can add one or more actions to be performed when users navigate to the sheet. For some actions, you need to provide details for the action.

Apply bookmark

You can apply the selection that is defined in a bookmark that you choose.

Clear all selections

You can clear all selections in all states in the app. You can optionally overwrite locked selections.

Clear selections in other fields

You can clear selections from all fields except the one you specify. You can optionally overwrite locked selections.

Move forwards in your selections

You can move one step forwards in your selection history.

Move backwards in your selections

You can move one step backwards in your selection history.

Clear selections in field

You can clear all selections from a field that you specify.

Lock all selections

You can lock all selections in the app.

Lock a specific field

You can lock selections in a field that you specify.

Unlock all selections

You can unlock all selections in the app.

Unlock a specific field

You can unlock selections in a field that you specify.

Select all values in a field

You can select all values in a field that you specify. You can optionally overwrite locked selections.

Select values in a field

You can select a list of values in a field that you specify. Separate the values to select with a semi colon. You can optionally overwrite locked selections.



It is not possible to use fields with date, timestamp, or money data type.

Select values matching search criteria

You can select all values that match the search results from a search criteria that you specify. You need to specify the search criteria as a string. You can optionally overwrite locked selections.

- If you want to use an expression, you need to enclose it in single quotes, for example, ='=Sum([Sales Amount]) > 200000'.
- If you want to search for a partial string, you need to use wild cards (*, ?, ^). If you do not use wild cards, only strings that match exactly are selected.

Select alternatives

Select all alternative values in a field that you specify. You can optionally overwrite locked selections.

Select excluded

Select all excluded values in a field that you specify. You can optionally overwrite locked selections.

Select possible values in a field

Select all possible values in a field that you specify. You can optionally overwrite locked selections.

Toggle field selection

You can set the button to toggle between the current selection, and a selection that adds selections defined by a search string. You can use wild cards in the search string. If you want to define a list of values you need to use the format $(A|B)$, where A and B are values to select.

Set variable value

You can assign a value to a variable.

2.4 Managing apps

Once you have created and built an app with the sheets and visualizations you want it to have, you may want to fine-tune it to make it easy and efficient to use, not only for yourself but also for other people.

The following actions are available to app owners:

- Apply app styling (logo and header)
- Create reusable master items (visualizations, dimensions and measures)
- Add bookmarks to keep track of important and interesting data selections and connections
- Making apps available in Insight Advisor Chat
- Turn off Insight Advisor
- Change the app's title and description, and also add a thumbnail to it
- Reload app data
- Manage app reload tasks
- Publish your own apps
- Move your own published apps between streams

Creating an app

The first thing you need to do when building an app is to create an empty placeholder for it. You create the app placeholder from the hub.

Do the following:

1. Click **Create new app** in the hub.
2. Give your app a name.
3. Click **Create**.
The app is created.
4. Click **Open app**.
The app opens in the app overview.

The next step is to add data to the new app.

On-demand apps

On-demand apps enable you to load and analyze big data sources in Qlik Sense.

Trying to analyze an entire big data store at one time is highly inefficient. Nevertheless, to make representative visualizations, all the data must be discoverable. Qlik Sense on-demand apps give users aggregate views of big data stores and allow them to identify and load relevant subsets of the data for detailed analysis.

On-demand apps are made up of several building blocks or components, and some of those components are built by users with advanced scripting skills.

Generating an on-demand app

You generate an on-demand app when you have selected a manageable subset of data using an on-demand selection app. Any selection app that you are working with will contain one or more navigation links for generating on-demand apps in the **App navigation** bar.

The on-demand apps shown on the **App navigation** bar have completion indicators that start to turn green as you make selections in the selection app. Each on-demand app on the app navigation bar has a limit on the amount of data it can contain. When selections are made in the selection app, the completion indicator shows when the amount of data selected is within the bounds set for the on-demand app.

Once an on-demand app's indicator turns completely green, you can generate that app with the currently selected data. You can also choose to open a previously generated instance of that app. Every on-demand app in the app navigation bar can be generated multiple times, and those generated apps remain accessible. When the maximum number of apps has been generated, you must delete an existing app before you can generate a new on-demand app. On-demand apps also may have an expiration time after which they are automatically deleted.



Anonymous users can only generate on-demand apps that are published automatically. Because anonymous users can only use published apps, they cannot use an on-demand app unless it is published automatically when it is generated. If an anonymous user attempts to generate an on-demand app that is not set for automatic publication, a message displays indicating that the user cannot generate an app from that particular on-demand app navigation point.

The maximum number of apps and the retention time are set on the on-demand app navigation link. The app navigation link is one of the building blocks of on-demand apps, and it is usually added by the creator of the selection app.

Do the following:

1. Open an on-demand selection app.
2. Select from the visualization objects in the selection app.
3. When the completion indicator on an on-demand app in the **App navigation** bar turns completely green, click the on-demand app.

You can select an on-demand app in the **App navigation** bar to open its generation panel. There you can click the to see the **Constraint** (maximum number of records allowed) and the number of records currently selected. You can also see the number of values selected for each field and any constraints on the fields. When on-demand apps are created, constraints can be placed on individual fields. For example, a field for Year might be limited so that no more than two values can be selected.

When the number of records (**Row count**) currently selected is less than or equal to the maximum number of records allowed, the completion indicator turns completely green. An app cannot be generated, however, until all the constraints have been met. If the row count is within the constraint but one or more of the fields have not met the requirements of their constraints, the **Generate new app** button will not be enabled.



*If you select **Generate new app** when constraints panel is open, you will not see the generated app. Click the **i** to close the panel, and you will see the new app listed if it generated successfully.*

4. Click the **Generate new app** button to create a new instance of the on-demand app with the data currently selected.

The new instance of the app is generated and appears in the generation panel above the **Generate new app** button.

To see more about the generated app, open its detail panel.

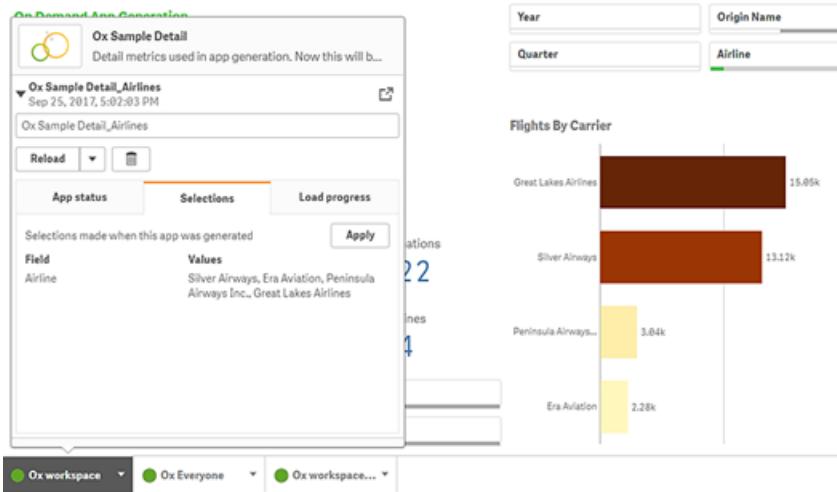
The screenshot shows the 'Ox Sample Detail' application interface. At the top, there's a placeholder for a thumbnail image with the text 'Ox Sample Detail'. Below it, a message says 'Detail metrics used in app generation. Now this will b...'. A section titled 'Ox Sample Detail_Airlines' is expanded, showing the date 'Sep 25, 2017, 4:32:13 PM'. There's a 'Reload' button and a trash can icon. Below this is a navigation bar with tabs: 'App status' (selected), 'Selections' (highlighted with an orange border), and 'Load progress'. Under 'Selections', it says 'Selections made when this app was generated' and has an 'Apply' button. A table lists 'Field' and 'Values': 'Year' is 2015, 2014; 'Fare Class Name' is Restricted Business Class, Restricted First Class.

In this view, you can also rename the on-demand app. By default, on-demand apps are assigned the name of the navigation point from which they are generated, and the user's name is appended. For example, when the name of the navigation point is "Ox Sample Detail," the default name of the generated on-demand app would be "Ox Sample Detail_John-Doe" for user "John Doe." In the illustration above, the name of the on-demand app has been changed to "Ox Sample Detail_Airlines." You can rename an on-demand app even if it has been published.



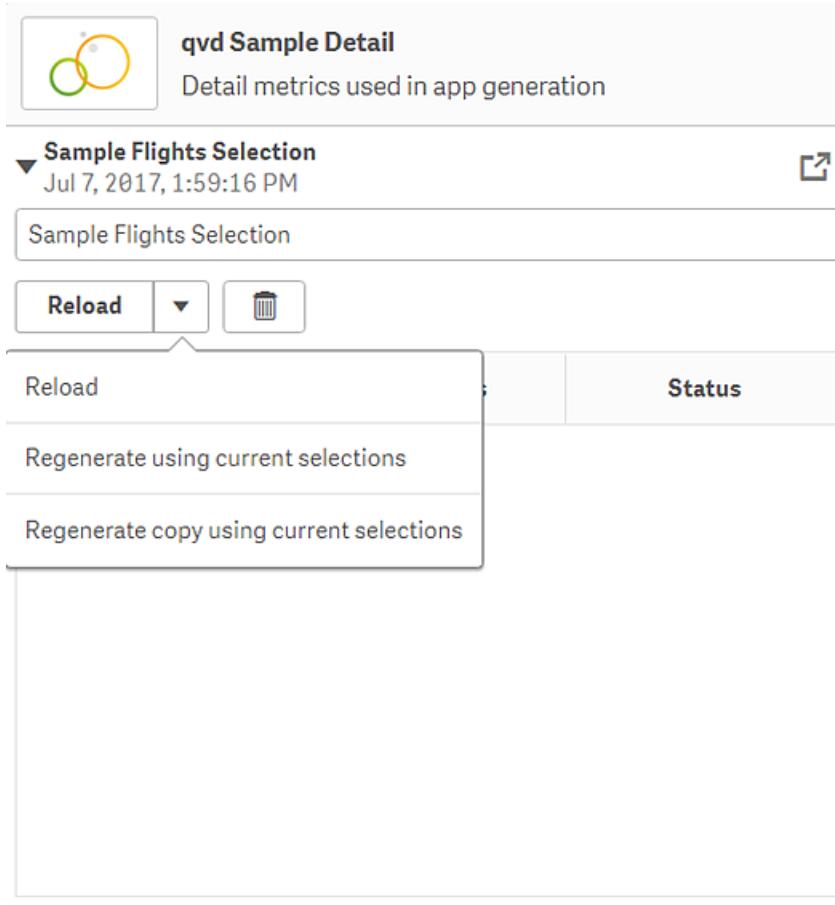
On-demand apps generated by anonymous users are given default names indicating that they were generated by an anonymous user rather than a registered user. Anonymous users can change the names of generated apps just as registered users can change the names of their apps.

The **Apply** button applies the selections listed on the generated on-demand app's **Selections** tab to the selection app.



- Select **Open app** from the **•••>** menu to open the generated app.

You can also reload data and regenerate an on-demand app. The **Reload** menu is available when the detail panel is open:



The selections on the **Reload** menu work as follows:

- **Reload:** reloads the data based on the current selections that have been made within the on-demand app.
- **Regenerate using current selections:** regenerates the on-demand app using the current selections that have been made within the selection app.



*The **Apply** button on the **Selections** tab applies the on-demand app's selections to the selection app. The on-demand app's selections are those listed as **Selections made when this app was generated**. For more information, see [the illustration above](#).*

*This is the reverse of what is done when **Regenerate using current selections** is used.*

- **Regenerate copy using the current selections:** generates a new on-demand app using the current selections that have been made within the selection app. The previously generated on-demand app remains in its current state.

Exploring a published on-demand app

Most users who use on-demand apps to explore big data sets access them as published apps. There are two points at which most users will use on-demand apps:

1. In a published on-demand selection app, where the user selects data and generates an on-demand app from the **App navigation** bar.
2. In a stream where a generated on-demand app is published.

Published on-demand apps have a preselected subset of data from a very large data source. You can explore that data through the visualization objects in the on-demand app. In that way, on-demand apps are like apps created with data from any other source.

You cannot change them after they have been published, though like other published apps, you can add private sheets and stories if you have the correct access rights. You work with published on-demand apps the same way you work in other published apps.

Styling an app

You can apply styling to your app to customize the app based on your company standards. The selected styling will be applied to all sheets in the app. Once the app has been published, you cannot change the app styling.

[Styling an app](#)

The following styling options are available:

- Changing the direction of the characters in strings of text or numbers.
- Changing the default app theme.
- Disabling the hover menu on all visualizations.
- Changing the background color for the sheet titles.
- Changing the sheet title font color.
- Adding and aligning an image, such as a logo.



More styling can be applied to an app by a developer through custom theme extensions.

Custom styling on individual objects override app styling.

Opening app options

You can open app options from anywhere in an unpublished app.

Do the following:

1. In an unpublished app, click and then click to open app options.
2. Click to close app options.



App options are not available on a mobile device.



As of Qlik Sense June 2018, app options is not available on a published apps even when having custom security rules defined with action on .

Changing the reading order

Do the following:

1. In an unpublished app, click  and then click  to open app options.
2. Under **Appearance > Right-to-left**, select **On** or **Off**.



If a visualization is shared using Qlik Sense Charts, changing the reading order in the app will also affect the reading order in the shared chart.

Changing the default app theme

You can apply one of the default Qlik themes or any custom theme that you have created and installed.

The default Qlik themes are:

- **Sense Horizon** - This is the default theme when you create a new app.
- **Sense Classic** - Provides a more compact view of objects, and limits the space between them.
- **Sense Focus** - Adjusts the padding and spacing around objects as well as provides designated spaces for titles.
- **Sense Breeze** - Based on **Sense Focus** but with different color settings.

Do the following:

1. In an unpublished app, click  and then click  to open app options.
2. Select the theme to apply from the **App Theme** drop-down menu.



You cannot change themes in published apps.

Custom themes

You can also create custom themes based on your company standards. With themes you can precisely style an app by changing the colors, adding images and backgrounds as well as specifying the fonts, font sizes, font weights and font styles on a global or granular basis throughout your app. You can also define color palettes and customize the specifications for margins, padding and spacing.

 For more information, see [Sense for Developers: Custom themes](#).



Custom themes are supported when exporting sheets and charts in PDF format. This applies to manually exported PDFs, as well as to PDFs exported automatically using subscriptions or via Qlik Reporting Service. However, a PDF generated from an app that uses a custom theme may look different than seen in the app. For more information, see [Custom theme JSON properties](#).

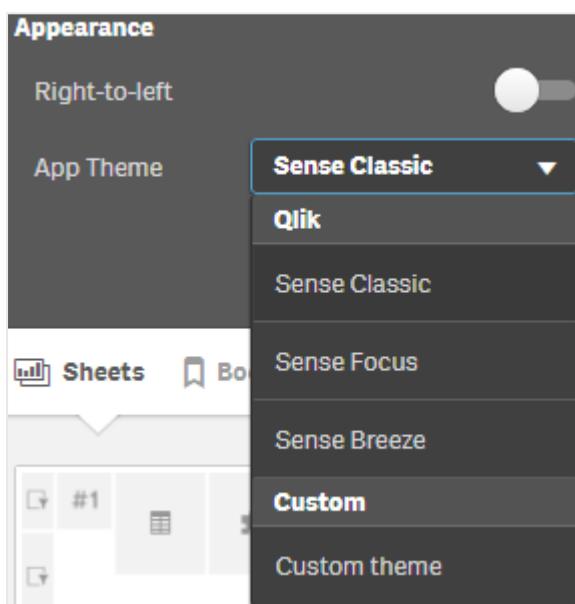


Custom themes are not dependent on the app. This means that if you for example change the colors defined in a custom theme these will be updated in all apps using the theme, even if the app is published.

When you have created a custom theme, you store it in Qlik Sense as an extension. The JSON file and any additional resources, such as CSS files, are zipped and imported as an extension in the Qlik Management Console (QMC). This enables that security rules can be added in the QMC for controlling which themes specific users have access to.

Custom themes appear under **Custom** in the **App Theme** drop-down menu.

App theme drop-down menu



Turning off the hover menu

You can choose to turn off the hover menu that appears on visualizations when users place their cursor over them. This will affect all visualizations in the app.

Do the following:

1. In an unpublished app, click **...** and then click **⚙** to open app options.
2. Click **Turn off hover menu for visualizations**.

You can turn off the hover menu for a single visualization in edit mode.

Do the following:

1. In edit mode, select the visualization.
2. In the properties panel, go to **Appearance > General**.
3. Select **Turn off hover menu**.

Changing the chart animations

Chart animations are the gradual transitions in a visualization from the old view to the new view when data has been changed, after for example a selection has been made.

Chart animations can be turned off in the app settings. They are available for the following chart types:

- Bar charts
- Bullet charts
- Combo charts
- Line charts
- Pie charts
- Scatter plots
- Funnel charts (Visualization bundle)
- Grid charts (Visualization bundle)
- Sankey charts (Visualization bundle)

Do the following:

1. In an unpublished app, click  and then click  to open app options.
2. Under **Appearance > Chart animations**, select **On** or **Off**.

Changing sheet title colors

The sheet title background color can be set to a solid color or a gradient of colors by selecting two colors. The sheet title font can only be set to a solid color.



If you are using a mobile device, the sheet title background is displayed as one solid color even if two colors has been set.

When choosing colors, you have the following options:

- Choose a color from the default color palette.
- Set a Hex color by typing 6 characters in the # input field.
- Click the palette to show more color options:
 - Click the color wheel to select color.
 - Drag the slider to change the color saturation.

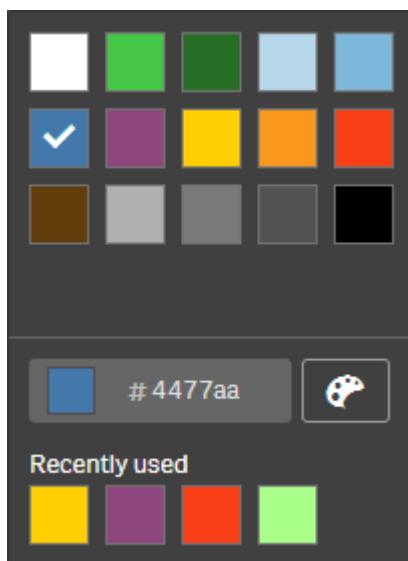
Choosing a color from the default color palette

Do the following:

1. Click ▼ in a color drop-down.
The dialog opens and displays the default colors.
2. Click one of the colors in the palette.
The color is selected.
3. Click outside the dialog.
The dialog is closed.

Now you have set a color by selecting in the default color palette.

Color dialog with the default color palette and a blue color selected.



Typing a Hex color

Do the following:

1. Click ▼ in a color drop-down.
The dialog opens and displays the default colors.
2. Type 6 characters in the Hex input field: #.
The color is selected in the palette.
3. Click outside the dialog.
The dialog is closed.

Now you have set a color by typing the 6 hexadecimal digits.

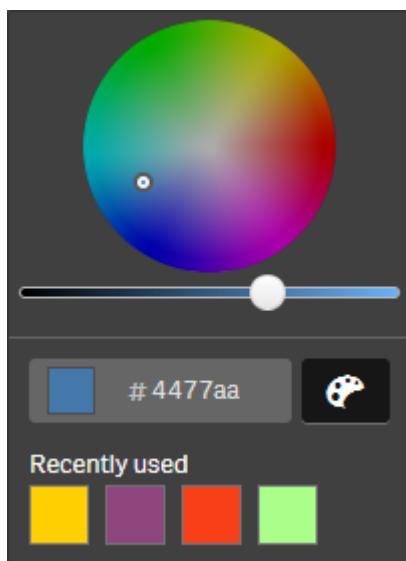
Using the advanced color options

Do the following:

1. Click ▼ in a color drop-down.
The dialog opens and displays the default colors.
2. Click  at the bottom of the dialog.
The dialog changes and displays the advanced options.
3. Do one of the following:
 - Click in the color wheel.
The color changes and the Hex color code updates accordingly.
 - Drag the slider.
The saturation changes and the Hex color code updates accordingly.Either way a color is selected.
4. Click outside the dialog.
The dialog is closed.

Now you have set a color by using the color wheel and/or the slider.

Color dialog with the advanced options and a blue color selected.



Adding an image

You can add an image to the sheet title, such as a logo. The following formats are supported: .png, .jpg, .jpeg, and .gif.

Do the following:

1. Click the image placeholder next to **Image**.
The **Media library** opens.

2. Click on a folder in the media library (for example, **In app** or **Default**).
3. Select the image that you want to add to the sheet title.
A preview of the image is shown.
4. Click **Insert**.
The image is added.

Now you have added an image to the sheet title.

Reloading app data

Apps in the hub do not automatically update when their data sources are updated. Reloading an app updates it with the latest data from the app data sources. The first time you reload an app, a reload task is automatically created with a once-only trigger.

Do the following:

- Right-click the app and select **Reload**.



*Manage the new reload task in the hub by right-clicking the app and selecting **Manage reload tasks**. For example, you can view, stop, or start the reload task. For more information about managing and scheduling reload tasks in the hub, see [Managing app reload tasks \(page 31\)](#). Administrators can also schedule app reloads in QMC. For more information, see [Creating reload tasks](#).*

App reload permissions

To reload an app in the hub, you require the following access rights set by an administrator in the QMC:

- Update access to the app.
- Read access to the HubSection_Task resource filter.
- HubAdmin role or Read and Create access to the ReloadTask resource filter.

Managing app reload tasks

View and manage app reload tasks from the hub. Manually start reload tasks or create new reload tasks with scheduled triggers to execute the reload automatically. You can also edit and delete reload tasks.

App reload tasks that are created by an administrator in QMC also display in the hub. To view the reload tasks for an app in the hub, right-click the app and select **Manage reload tasks**.



You must have read access to HubSectionTask and either the HubAdmin role or read access to reload tasks in the hub, set in QMC, to manage app reload tasks in the hub.

Manage reload tasks displays the following information and options:

- **Name:** Task name.
- **Enabled:** If the reload task is enabled or disabled. Disabled tasks cannot be managed or enabled from the hub.
- **Status:** Displays the current status of the task. You can click  to view a summary of the latest task execution. You can also click **Download script log** to download the script log.
- **Last execution:** Displays when the task was last executed.
- **Next execution:** Displays when the task is next scheduled to run.
- **Actions:** Management option for the task. You can click  to start tasks or click  to stop tasks. Click  to edit a task or  to delete a task.
- **Create task:** Add a new app reload task and set task properties, including creating scheduled triggers.

You can click  to refresh the task view.

Creating app reload tasks

Create an app reload task in the hub. When executed, the reload task fully reloads the data in an app from the source. Any old data is discarded. App reload tasks must include a scheduled trigger to execute the reload task automatically. The trigger determines the number of times the task will run (from one to infinity) and the frequency of the reload (once, daily, weekly, or monthly). For example, you can create a trigger to reload task to run 56 times, an app on a weekly basis. Create scheduled triggers for your tasks to reload app data at a specific time, date and frequency.

For information about creating app reload tasks in QMC, see [Creating reload tasks](#).



You must have read access to HubSectionTask and be assigned the HubAdmin role, set in QMC, to create app reload tasks in the hub.

Do the following:

1. In the hub, right-click the app that you want to create a reload task for, and then click **Manage reload tasks**.
2. Click **Create task**.
3. Enter a name for the task in the **Name** field.
4. Under **Execution**, review the default settings and edit any of the following properties:
 - a. Select or clear **Enabled** to enable or disable the task. The task is **Enabled ✓** by default.
 - b. **Task session timeout (minutes):** The maximum period of time before a task is aborted if the session times out. The default setting is 1440 minutes.
 - c. **Max retries:** The maximum number of times the scheduler will try to rerun a failed task. The default setting is 0.
5. Click **Create**.
6. In the **Tasks** list, under **Actions**, click  to open the task properties.
7. Click **Create scheduled trigger** and select the properties that you want to set for the trigger.

Scheduled triggers

Scheduled trigger properties in the hub

Property	Description
Trigger name	Name of the trigger. Mandatory.
Enabled	Status of the trigger. When selected, the trigger is active.
Time zone	The time zone of your operating system, at the time you create the trigger. When you save a trigger, the settings are kept, and if you move to a different time zone, the original values are still displayed. If you want to change the time zone and start time of a trigger, you need to do that manually.  <i>For a trigger that was created before the introduction of the time zone setting, all times and dates are by default presented in Coordinated Universal Time (UTC).</i>
Daylight saving time	Way to account for daylight saving time. Observe daylight saving time: This option takes daylight saving time (DST) into account. If DST is in use in the selected time zone, the execution time and date are adjusted accordingly. Permanent standard time: This option does not take DST into account. If DST is in use in the selected time zone, the execution time and date are not adjusted. Permanent daylight saving time: This option takes DST into account. If a time zone uses DST, execution time and date are always according to DST, even during periods when DST is not in use.  <i>For time zones not using DST, always select Permanent standard time.</i>
Example:	
<p>You created a trigger for an event at 10:00 AM, while you were working in Ottawa, Canada, in January. The time zone is (GMT-0500) Eastern Time (US & Canada) and DST is used between March and November.</p> <p>If you select Observe daylight saving time, a trigger set to start at 10:00 will always start at 10:00.</p> <p>If you select Permanent standard time, a trigger set to run at 10:00 will run at 10:00 in the winter but at 09:00 in the summer.</p> <p>If you select Permanent daylight saving time, a trigger set to run at 10:00 will run at 11:00 in the winter and at 10:00 in the summer.</p>	

Property	Description
Start	Start time and date: <ul style="list-style-type: none"> Start time: (hh:mm) Start date: (YYYY-MM-DD)
End	End time and date: <ul style="list-style-type: none"> End time: (hh:mm) End date: (YYYY-MM-DD) <p>Select Infinite to create a trigger with no end date.</p>
Schedule	Frequency of the trigger: <ul style="list-style-type: none"> Once. Hourly. Time period between executions of the trigger. Edit Repeat after each by typing the values for: <ul style="list-style-type: none"> hour(s) (default is 1) minute(s) (default is 0) Daily. Time period between executions of the trigger. Type a value for Every day(s) (default is 1). For example, type 2 to repeat the trigger every second day. Weekly. Time period between executions of the trigger: <ul style="list-style-type: none"> Type a value for Every week(s) (default is 1). Select one or more days under On these weekdays to determine which days the trigger is repeated (on the weeks you have specified). For example, type 3 and select Mon to repeat the trigger on Mondays every third week. Monthly. Select one or more days under On these days to define the days when the trigger is repeated every month.



If you have selected **Monthly** and want to be sure that a trigger is repeated every month, you need to select a day no later than the 28th.

- Click **Create**, and then click **Apply** to add the trigger to the reload task.



After adding a trigger for a reload task, click **Refresh task list** in the notification message or click in the task list to see the next execution time for the reload task in task list.

Editing app reload tasks

Edit app reload tasks in the hub. For example, create a new scheduled trigger, update the task name or execution details, or edit the properties of a scheduled trigger.



You must have read access to HubSectionTask and be assigned the HubAdmin role, set in QMC, to edit app reload tasks in the hub.

Do the following:

1. In the hub, right-click the app with the reload task that you want to edit, and then click **Manage reload tasks**.
2. In the **Tasks** list, under **Actions**, click to open the task properties.
3. Edit the task properties.
 - a. You can change the task name in the **Name** field.
 - b. You can change the **Execution** properties.
 - Select or clear **Enabled** to enable or disable the task. The task is **Enabled✓** by default.
 - **Task session timeout (minutes)**: The maximum period of time before a task is aborted if the session times out. The default setting is 1440 minutes.
 - **Max retries**: The maximum number of times the scheduler will try to rerun a failed task. The default setting is 0.
 - c. Create a new trigger. Select **Create scheduled trigger**, set the properties (see descriptions below) for the new trigger, and then click **Create** to add the trigger to the reload task.
 - d. Edit a scheduled trigger. In the **Triggers** list, under **Actions**, click . Set the properties (see descriptions below) and click **OK** to save your changes.

Scheduled Trigger

Scheduled trigger properties in the hub

Property	Description
Trigger name	Name of the trigger. Mandatory.
Enabled	Status of the trigger. When selected, the trigger is active.

Property	Description
Time zone	<p>The time zone of your operating system, at the time you create the trigger. When you save a trigger, the settings are kept, and if you move to a different time zone, the original values are still displayed. If you want to change the time zone and start time of a trigger, you need to do that manually.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>For a trigger that was created before the introduction of the time zone setting, all times and dates are by default presented in Coordinated Universal Time (UTC).</i> </div>
Daylight saving time	<p>Way to account for daylight saving time.</p> <p>Observe daylight saving time: This option takes daylight saving time (DST) into account. If DST is in use in the selected time zone, the execution time and date are adjusted accordingly.</p> <p>Permanent standard time: This option does not take DST into account. If DST is in use in the selected time zone, the execution time and date are not adjusted.</p> <p>Permanent daylight saving time: This option takes DST into account. If a time zone uses DST, execution time and date are always according to DST, even during periods when DST is not in use.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>For time zones not using DST, always select Permanent standard time.</i> </div> <p>Example:</p> <p>You created a trigger for an event at 10:00 AM, while you were working in Ottawa, Canada, in January. The time zone is (GMT-0500) Eastern Time (US & Canada) and DST is used between March and November.</p> <p>If you select Observe daylight saving time, a trigger set to start at 10:00 will always start at 10:00.</p> <p>If you select Permanent standard time, a trigger set to run at 10:00 will run at 10:00 in the winter but at 09:00 in the summer.</p> <p>If you select Permanent daylight saving time, a trigger set to run at 10:00 will run at 11:00 in the winter and at 10:00 in the summer.</p>
Start	<p>Start time and date:</p> <ul style="list-style-type: none"> • Start time: (hh:mm) • Start date: (YYYY-MM-DD)

Property	Description
End	<p>End time and date:</p> <ul style="list-style-type: none"> • End time: (hh:mm) • End date: (YYYY-MM-DD) <p>Select Infinite to create a trigger with no end date.</p>
Schedule	<p>Frequency of the trigger:</p> <ul style="list-style-type: none"> • Once. • Hourly. Time period between executions of the trigger. Edit Repeat after each by typing the values for: <ul style="list-style-type: none"> • hour(s) (default is 1) • minute(s) (default is 0) • Daily. Time period between executions of the trigger. Type a value for Every day(s) (default is 1). For example, type 2 to repeat the trigger every second day. • Weekly. Time period between executions of the trigger: <ul style="list-style-type: none"> • Type a value for Every week(s) (default is 1). • Select one or more days under On these weekdays to determine which days the trigger is repeated (on the weeks you have specified). For example, type 3 and select Mon to repeat the trigger on Mondays every third week. • Monthly. Select one or more days under On these days to define the days when the trigger is repeated every month.



If you have selected **Monthly** and want to be sure that a trigger is repeated every month, you need to select a day no later than the 28th.

4. Click **Apply** to save your changes.



If you modified a task trigger, click **Refresh task list** in the notification message. You can also click  to refresh the task list.

Deleting app reload tasks

You must have the appropriate permissions to delete app reload tasks in the hub.



By default, the HubAdmin role cannot delete app reload tasks. Administrators can modify the permissions in QMC for the HubAdmin role to allow this role to delete tasks, or set the following permissions for a user, set in QMC, to delete app reload tasks in the hub: read permission for HubSection_Task, read permission for ReloadTask_, and delete permission for ReloadTask_*.*

Do the following:

1. In the hub, right-click the app with the reload task that you want to delete, and then click **Manage reload tasks**.
2. In the **Tasks** list, under **Actions**, click .

Converting a QlikView document into a Qlik Sense app

If you have a QlikView document, you can reuse part of your work in Qlik Sense. The load script and data model can be used to create an app in Qlik Sense. Visualizations, dimensions and measures, however, have to be created in Qlik Sense.

After converting the document to an app, you may need to adapt the load script to be able to reload the script in Qlik Sense. In some cases, you may need to make some changes in the QlikView script before you convert the document.

Qlik Sense provides a tool to help convert a QlikView document (QVW file) to a Qlik Sense app. The tool converts visualizations, dimensions, measures, and variables. You must have access to the Dev Hub To use the QlikView to Qlik Sense converter.

Instructions for how to use the converter tool are provided on the Qlik Sense Developer site.

 For more information, see [QlikView converter](#).

If you do not have access to the Dev Hub, you can convert a QlikView document manually.



We recommend that you make changes only in a duplicate of any app that you convert, so that your changes are not overwritten by a later conversion.

Manually converting a QlikView document into a Qlik Sense app

You can manually convert a QlikView document (QVW file) to a Qlik Sense app. If you have access to the Dev Hub, you can use the QlikView to Qlik Sense converter tool to simplify the process.

The QlikView to Qlik Sense converter is described on the Qlik Sense Developer site.

Prerequisites

- You need to have Qlik Sense Desktop installed to be able to convert a QlikView document into a Qlik Sense app.

- Your computer must have sufficient memory, at least 32 GB, when converting a large QlikView document.
- If the QlikView document contains hidden script or uses section access, you need to have full access to the hidden script and section access of the document.

Before converting the document

The **Always One Selected Value** property for QlikView list boxes or multi boxes must be disabled in order to clear selections in the Qlik Sense app. If your QlikView document contains hidden script or uses section access, you need to adapt the document before you convert it.

Converting hidden script tabs

When a QlikView document with hidden script is converted, the hidden script part will be disregarded and not included in the Qlik Sense script. If you want to convert the entire script, do the following before converting the document. This requires that you have access to the hidden script in the QlikView document.

Do the following:

1. Open the document in QlikView **Script Editor**.
2. Copy the code from the hidden script tabs and paste it into regular script tabs.
3. Save the document.



It is not possible to hide script code in Qlik Sense.

Removing section access code

A QlikView document with section access cannot be imported to Qlik Sense, as there are differences in format and supported functionality. You can still convert the document manually.

Do the following:

1. Remove the section access code in QlikView **Script Editor** before converting the document. This requires that you have full access to the script of the document.
2. Convert the document into an app.
3. Reapply section access in the Qlik Sense app. There are some differences that you may need to take account for:
 - User authentication is changed. The USERID field is now used to authenticate all users, and the NTNAME and PASSWORD fields are not supported anymore.
 - If you have access to the script, but not to the data, you can open the app without data and edit the script, including the section access.
 - Section access is applied using strict exclusion in Qlik Sense. This means that you can only see data that you have been specifically granted access to.

QlikView variables excluded from export

A number of variables are not exported from a QlikView document (QVW) as they are only relevant for QlikView or is handled differently in Qlik Sense.

The following variables are not exported from a QlikView document:

- All variables starting with CD
- All variables starting with FLOPPY
- QvPath
- QvRoot
- QvWorkPath
- QvWorkRoot
- WinPath
- WinRoot
- ErrorMode
- StripComments
- ScriptErrorCount
- ScriptError
- ThousandSep
- DecimalSep
- MoneyThousandSep
- MoneyDecimalSep
- MoneyFormat
- TimeFormat
- DateFormat
- TimestampFormat
- MonthNames
- DayNames
- ScriptErrorDetails
- ScriptErrorList
- OpenUrlTimeout
- HidePrefix
- FirstWeekDay
- BrokenWeeks
- ReferenceDay
- FirstMonthOfYear
- CollationLocale
- LongMonthNames
- LongDayNames

Converting the document into an app

Do the following:

1. Copy the QlikView document (qvw format) to the directory where your apps are located.
This is usually <user>|Documents|Qlik|Sense|Apps, but the location of the Apps directory depends on where you installed Qlik Sense Desktop.
2. Start Qlik Sense Desktop.
You now see the QlikView document as an app in the hub, with the name ending with (qvw).
3. Click on the app to open it.

The app is saved into the Qlik Sense format (qvf file) in the folder where your QlikView document (qvw file) was stored. Also, the QlikView document file (qvw) is removed from the folder and automatically converted into a backup file (qvw.backup) stored here: <user>|Documents|Qlik|Sense|AppsBackup.

You have now migrated a QlikView document into a Qlik Sense app. The app contains the data model, including the loaded data, and the data load script.

You can use the data model to build visualizations, copy it or import it into your server environment through the Qlik Management Console (QMC), but you need to adapt the load script before you can reload the data model.



You can also drag and drop a qvw file from a folder onto the Qlik Sense Desktop hub, to open it as a Qlik Sense app. If the Qlik Sense app (qvf file) becomes stored in another folder than <user>|Documents|Qlik|Sense|Apps, move it to the Apps folder to make it available from the hub.

Adapting the data load script in Qlik Sense

Qlik Sense and QlikView data load scripts are compatible in general, but there are some differences that you may need to take account for and adapt the script in the data load editor before you can reload data. You can adapt the script directly in Qlik Sense Desktop, or you can import the app into Qlik Sense and then adapt the script.

Change file path references to data connections

QlikView uses absolute or relative file paths which are not supported in Qlik Sense standard mode, so you need to use folder data connections to point to file locations instead.

Do the following:

1. Create folder data connections for all file paths that are used in the data load script.
You need to check statements and functions that refer to files. **LOAD** statements are the most common ones, but there is a complete list available.
2. Replace all references to absolute or relative file paths in the script with **lib://** references to the data connections you created.

Examples	
Original script in QlikView	Adapted Qlik Sense script
<code>LOAD * FROM [C:\data\Tutorials source\Sales rep.csv];</code>	<code>LOAD * FROM [lib://Tutorials source/sales rep.csv];</code> In this case, the Tutorials source folder data connection should be pointing to <code>C:\data\Tutorials source</code> , or where the data is located if you have moved the app to another computer or a Qlik Sense server.
<code>Filesize('C:\data\Tutorials source\sales rep.csv')</code>	<code>Filesize('lib://Tutorials source/sales rep.csv')</code>
<code>for each Dir in dirlist ('C:\data\Tutorials source*')</code>	<code>for each Dir in dirlist ('lib://Tutorials source/*')</code>

Handling features that are not supported or recommended

There are some features in QlikView that are not supported or recommended in Qlik Sense, for example:

- Input fields
- Message boxes
- **Bundle** and **Info** load
- **ALL** qualifier

We recommend that you check your script code against these lists of statements and functions that are not supported or recommended, and adapt the code according to recommendations.

Installing custom connectors

If your QlikView document uses custom connectors to access data, you should be able to use them to load data in Qlik Sense as well without changes in the script. This requires that the same connector is installed on the Qlik Sense computer.

If you want to make any changes in the data selection by the custom connector, you need to install a version of the custom connector adapted for Qlik Sense.

Changing the title and description of an app

You can change the title and description of your apps. When creating a new app, the name of the app is used as its title. When you change the title, the name of the app is not changed.

Do the following:

1. In the app overview, click  in the app details area.
2. Edit **Title** and **Description**.
3. Click  to stop editing.

The changes you made are saved.



You can only change the title and description of an unpublished app.



You can open or close the app details area by clicking the app name in the navigation bar.

Changing the thumbnail of an app

You can replace the default thumbnail of an app with another thumbnail, to make it easier to distinguish between apps in the hub. You can use one of the default images, or an image of your own.

Do the following:

1. In the app overview, click in the app details area.
2. Click on the default thumbnail.
The **Media library** opens.
3. Click on a folder in the media library, for example **In app** or **Default**.
4. Select the image you want to use as a thumbnail for the app and click **Insert**.
5. Click to stop editing.

The image you selected is now used as a thumbnail for the app.



The optimal aspect ratio of a thumbnail is 8:5 (width:height).

The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

For Qlik Sense Desktop: You can place images in the following folder on your computer:

C:\Users\<user>\Documents\Qlik\Sense\Content\Default. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.



You can only change the thumbnail of an unpublished app.



You can open or close the app details area by clicking the app name in the navigation bar.

Duplicating an app

You can duplicate an existing app, to create a copy to develop further. You can only duplicate an app that you have created yourself unless your administrator has assigned you a security role with duplication enabled.

If you have an admin role, giving you the administration rights needed, you can create duplicates of apps from the QMC.



When you duplicate a published app, only the base sheets and stories will be included in the copy.

Do the following:

- In the hub, right-click the app you want to duplicate and select **Duplicate**.
A toast notification is displayed for a while, at the bottom of the screen, when the duplication starts. A new toast notification will be displayed when the app has been duplicated and give you the possibility to update the app list. If the duplication fails, an error message will be displayed.

A copy of the app is created under **Work**. You can click the app to open it and start adapting it according to your preferences.



Because of how the synchronization of data works in multi-node sites, apps containing images may display broken thumbnails or images inside the apps if opened right after being duplicated or imported. The broken images are restored when the synchronization is complete. To check if the images have been restored, refresh the browser window.

Making apps available in Insight Advisor Chat

You can make your apps available in Insight Advisor Chat in the hub.

When your app is available, users with access to that app can search Insight Advisor in your app with Insight Advisor Chat.



*Apps that use Section Access require additional configuration for service users to be available in Insight Advisor Chat. If you have sensitive information in app names, field names, or master item names, these may be exposed by making apps using Section Access available for Insight Advisor Chat. App suggestions for queries include app in streams to which users have access. These may include apps to which users do not have access in an app's Section Access. Selecting these apps will do nothing, however. When clicking **Dimensions** or **Measures** to view the available items from an app using Section Access, users may see items to which they do not have access. Clicking on these items will not provide any data to the users, however.*

App content availability in Insight Advisor Chat

What users can search for and access with the **Measure** and **Dimension** buttons with Insight Advisor Chat depends on if the app is published and if there is a logical model applied to your app.

For your own unpublished apps, you can search for fields and master items from the app. For published apps, users can search for only master items. If you have business logic applied to an unpublished or published app, then users can search for available fields and master items based on the logical model.

Clicking the **Measure** or **Dimension** button (or entering show measure or show dimension) will show the corresponding master items. If the app has no master items, fields related to measures or dimensions are shown instead. Fields or master items hidden in the logical model will not appear when the button is clicked, but can be searched for in a query.

Turning on Insight Advisor Chat

Do the following:

1. In your app, click the app name in the navigation bar.
2. In the app details area, click .
3. Turn on **Insight Advisor in hub**.

Turning on chart level scripting

By default, chart level scripting is disabled in apps. You can turn it on in app details.

Do the following:

1. In your app, click the app name in the navigation bar.
2. In the app details area, click .
3. Turn on **Chart scripting**.

Turning off Insight Advisor

By default, Insight Advisor Search and Insight Advisor Analysis are enabled in apps. You can turn off Insight Advisor to prevent users from accessing these features.

Do the following:

1. In your app, click the app name in the navigation bar.
2. In the app details area, click .
3. Turn off **Insight Advisor in this app**.

Deleting an app

You can delete an app that you no longer need.



From the hub, you can only delete apps that you have created and that have not yet been published.

If you have published an app, the published version is locked and can only be deleted from the Qlik Management Console. The published version is a duplicate of the original app. The original version of the app can be deleted from your personal work in the hub.

Do the following:

1. Right-click the app in the hub and select **Delete**.
A confirmation dialog opens.
2. Click **Delete**.
A toast notification is displayed for a while, at the bottom of the screen, when the app is being deleted.
If the deletion fails, an error message will be displayed.



You can delete a published app from the Qlik Management Console.

If an app is being deleted by someone else from the stream you are currently viewing, a toast notification will be displayed and give you the possibility to update the app list.

Uploading image files to media library

The media library contains the images you can use in your app: in text and image visualizations, on story slides, and as thumbnails for apps, sheets, and stories.

You can upload images to the media library. Because of limitations in the web browsers supported by Qlik Sense, it is recommended to keep the height and width as well as the file size of the images as small as possible. The maximum file size is 5 MB.

The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

For Qlik Sense Desktop: You can place images in the following folder on your computer:

C:\Users\<user>\Documents\Qlik\Sense\Content\Default. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.

You can open the media library in several ways. Do one of the following:

- In the app overview, click in the app details area and click on the thumbnail.
- If you are editing a sheet, double-click the text & image visualization to open the editing toolbar and click .
- In storytelling view, click in the toolbar and then select an image.

The **Media library** dialog opens and now you can upload images.

Do the following:

1. Select **Upload media**.
2. Do one of the following:
 - Drop one or more image files onto the designated area.
 - Click the designated area to open the upload dialog, browse and select images, and click **Open**.

An upload indicator is displayed while the image file is being uploaded. You can cancel an ongoing upload by clicking on  on the image.



You cannot upload a file if its file name already exists in the media library.

Now you have added images to the media library. Click the image and select  to insert the image.

Deleting image files from media library

You can delete images from the media library in Qlik Sense.

You can open the media library in several ways. Do one of the following:

- In the app overview, click  in the app details area and click  on the thumbnail.
- If you are editing a sheet, double-click the text & image visualization to open the editing toolbar and click .
- In storytelling view, click  in the story tools panel and drag **Image** onto the slide and then double-click inside the image placeholder.

The **Media library** dialog opens and now you can delete images.

Delete images

Do the following:

1. Select the **In app** folder.
2. Select the file you want to delete.
3. Click .

A confirmation dialog is displayed.

4. Click **Delete**.

Now you have deleted images from the media library.



*You can also right-click on an image file and select **Delete**.*



For Qlik Sense: You need to use the Qlik Management Console to delete images from the default folder, or other folders that were created from the Qlik Management Console.

Delete images using Qlik Sense Desktop

You can delete images from the default folder by removing the files from this location:
`<user>|Documents|Qlik|Sense|Content|Default.`

The images in the **In app** folder are bundled images, saved in the qvf file together with the rest of the contents of the app. If a bundled image is no longer used in the app, the image will be deleted from the qvf file when saving the app.

2.5 Troubleshooting - Creating apps

This section describes problems that can occur when creating apps and visualizations in Qlik Sense.

Images are not included in an app that has been moved from one Qlik Sense environment to another

Possible cause

You have created an app in Qlik Sense and moved the app to another Qlik Sense environment.

When you move apps between Qlik Sense environments, images may not have been moved automatically. The images have to be handled manually, in different ways depending on between which environments the app has been moved.

Proposed action

Import the images from the Qlik Sense environment where the app was created to the target location using the Qlik Management Console.

Images are not included in an app that has been imported from Qlik Sense Desktop to Qlik Sense

Possible cause

You have created an app in Qlik Sense Desktop and imported the app to Qlik Sense.

When you move apps between Qlik Sense environments, images may not have been moved automatically. The images have to be handled manually, in different ways depending on between which environments the app has been moved.

Proposed action

Import the images from the images folder of the Qlik Sense Desktop app using the Qlik Management Console.



Images are included automatically only when you move an app from one Qlik Sense Desktop installation to another.

The default location of the images in Qlik Sense Desktop is <user>|Documents|Qlik|Sense|Content|Default.



If you have organized images in subfolders in the Content|Default folder in Qlik Sense Desktop, these have to be added manually to the app and its sheets, stories and text & image objects after importing the images.

Images are not included in an app that has been moved from one Qlik Sense Desktop installation to another

Possible cause

You have moved an app between Qlik Sense Desktop installations.

When you move apps between Qlik Sense environments, images may not have been moved automatically. The images have to be handled manually, in different ways depending on between which environments the app has been moved.

Proposed action

Do the following:

- Copy the images from the PC where the app was created and paste into the images folder of the target location.

The default location of the images in Qlik Sense Desktop is <user>|Documents|Qlik|Sense|Content|Default.

The image I want to use does not seem to work

Images are part of apps as thumbnails of the apps, sheets and stories, in the text & image object and in story slides.

Possible cause

You are using an image in a format that is not supported.

Proposed action

Do the following:

- Convert the image to one of the supported formats (png, jpg, jpeg or gif).

Using Insight Advisor impacts system performance

Insight Advisor uses master items as fields in recommended charts.

Possible cause

The charts generated by the Insights are choosing fields that end up with costly charts.

Proposed action

Do the following:

- Change what fields are exposed to published app users.

I cannot find **Reload** when I right-click on an app

I want to reload my app, but I do not see **Reload** when I right-click on my app.

Possible cause

You do not have one or more of the following permissions set in QMC:

- Update access to the app.
- Read access to the HubSection_Task resource filter.
- HubAdmin role or Read and Create access to the ReloadTask resource filter.

Proposed action

Contact a QMC administrator to be assigned the HubAdmin role or to get create and read access to the ReloadTask resource filter.

I cannot manage my app reload tasks in the hub

I want to view my app reload tasks in the hub, but I do not see **Manage reload tasks** when I right-click an app.

Possible cause

The security rule HubSectionTask is disabled or you do not have read access to the HubSection_Task resource filter. Alternatively, you may not have read access to app reload tasks in the hub configured in QMC.

Proposed action

Contact a QMC administrator to get read access to HubSection_Task and either read access to app reload tasks in the hub or to be assigned the HubAdmin role. The HubAdmin role, by default, has read access to app reload tasks in the hub.

Thumbnails are not included when copying a sheet

Possible cause

You have copied a sheet from another app. Thumbnails are stored as app resources and are not included when you copy a sheet. Copying sheets within an app works fine.

Proposed action

You can copy an entire app and use that as starting point. This will include all resources, such as images.

2.6 Optimizing app performance

App performance can be improved with reduced app size, simplified data models, and strategic use of set analysis. This section will help you avoid performance issues by pointing out areas where performance can be impacted and how you can evaluate and monitor app performance.

App complexity

These are loose categories that can help diagnose issues. The most complex apps have the lowest performance.

Simple apps:

- Do not include complex set analysis or If() statements.
- Do not include large tables.
- Have a simple data model.
- Contain simple calculations.
- May have large data volumes.

Moderate apps:

- Have a data model with many tables, but follow best practices.
- Use set analysis and several If() statements.
- Have large or wide tables on sheets (15 columns or more).

Complex apps:

- Have a very complex data model.
- Connect to large data volumes.
- Contain complex calculations, charts, and tables.

App details

You need to consider your hardware environment in relation to app size, because it affects the performance of your Qlik Sense deployment. For example, if you do not optimize your apps, they may require more hardware resources.

Monitoring app size will help you:

- Understand current performance.
- Understand the performance impact of deploying a new app.
- Understand the performance impact of modifying an existing app.
- Resolve performance issues.
- Plan for future growth.

Qlik provides tools that can help you assess your apps. For more information, see: [Performance and scalability in Qlik Sense Enterprise](#).

These are the basic app elements that can affect performance:

App details that can affect performance

Feature	Description
App disk size (MB)	You can find app size in the QMC. Go to Apps , and open the Column selector on the right hand side next to Actions . Click the radio button next to File size (MB) . If you are using Qlik Sense Desktop, you can find app size in Windows Explorer. The default folder is %USERPROFILE%\Documents\Qlik\Sense\Apps. The Apps folder lists all app names and file sizes.
App size in RAM (GB)	<p>You can determine an app's base RAM footprint by:</p> <ol style="list-style-type: none"> 1. Restarting the Qlik Sense server. 2. Noting the current RAM usage. 3. Opening up the Qlik Sense app. 4. Recording the difference in RAM. <p>You can use the <i>App Metadata Analyzer</i> to find this metric if you are using Qlik Sense June 2018 or later.</p>
App total rows (M)	You can use system fields to calculate total rows. Create a KPI with the measure <i>Sum(\$Rows)</i> .
App total fields	You can use system fields to calculate total fields. Create a KPI with the measure <i>Sum(\$Fields)</i> .
App total tables	You can use system fields to calculate total tables. Create a KPI with the measure <i>Count(DISTINCT \$Table)</i> .

Monitoring your app

The Qlik Management Console (QMC) provides apps for monitoring system performance and usage on Qlik Sense Enterprise on Windows:

- The *Operations Monitor* app provides information about hardware utilization, such as server memory and CPU usage, active users, and reload task activity. It also provides summary and detailed information about errors, warnings, and log activities in the Qlik Sense server environment.
- The *License Monitor* app tracks license usage, and facilitates monitoring changes to license allocation.
- The *Log Monitor* app presents nearly all log data available and enables trend analysis and troubleshooting.
- The *Sessions Monitor* app shows log data about usage of apps.
- The *Reloads Monitor* app presents detailed information about reload data, both from the QMC and apps open in the hub.
- The *Sense System Performance Analyzer* app displays Qlik Sense performance across all nodes.
- The *Sense Connector Logs Analyzer* app provides insights into usage and errors of specific Qlik connectors.
- The *App Metadata Analyzer* app provides a holistic view of all your Qlik Sense apps, including granular level detail of an apps data model and its resource utilization.

Large data volumes

You can employ these architecture strategies when you are connecting to large data volumes.

Segmentation

You can segment QVDs by dimensions, such as time frame, region or aggregation level. For example, you can have:

- A QVD that contains data from the two most recent years.
- A QVD that contains historical data further than two years.
- A QVD that contains all data aggregated on a higher level. For example, per month instead of date, or per country instead of individual customers.
- One large QVD with all the data, which is only used by a small subset of users.

You can segment the apps in a similar way. Smaller apps will address the analytical needs of most users. This saves memory.

You can also have multiple apps focused on different regions. This way, users will not open an app with data that they are not interested in or do not have rights to access. Data that is not accessible via section access still affects memory.

On-Demand App Generation (ODAG)

Qlik Sense on-demand apps give users aggregate views of big data stores. They can identify and load relevant subsets of the data for detailed analysis.

From a user perspective, there are two apps:

1. A shopping cart with aggregated data.
2. An empty template app used to display detail.

The user makes selections in the shopping cart app. Once a threshold has been met, a custom LOAD script is created which populates the template app with the requested details.

Data model performance

These are indicators that can impact data model performance. Each one is a best practice that will improve app usability.

Data model performance best practices

Action	Description
Synthetic keys removed	Qlik Sense creates synthetic keys when two or more data tables have two or more fields in common. This may mean that there is an error in the script or the data model.

Action	Description
Circular references removed from data model	<p>Circular references occur when two fields have more than one association. Qlik Sense will attempt to resolve these by changing the connection to one of the tables.</p> <p>However, all circular reference warnings should be resolved.</p>
Appropriate granularity of data	<p>You should only load data that is necessary. For example: a group of users only need data divided by week, month, and year. You can either load in the aggregated data or aggregate the data within the load script to save memory. If a user does need to visualize data at a lower level of granularity, you can use ODAG or document chaining.</p>
QVDs used where possible	<p>A QVD is a file containing a table of data exported from Qlik Sense. This file format is optimized for speed when reading data from a script, but is still very compact.</p> <p>Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.</p>
QVD files optimized on load	<p>QVD files can be read in two modes: standard (fast) and optimized (faster). The selected mode is determined automatically by the script engine.</p> <p>There are some limitations regarding optimized loads. It is possible to rename fields, but any of these operations will result in a standard load:</p> <ul style="list-style-type: none"> • Any transformations on the fields that are loaded. • Using a where clause causing Qlik Sense to unpack the records. • Using Map on a field that is loaded.
Incremental loads leveraged	<p>If your app connects to a large amount of data from databases that are continuously updated, reloading the entire data set can be time consuming. Instead, you should use incremental load to retrieve new or changed records from the database.</p>

Action	Description
Snowflake model consolidated	If you have a snowflake data model, you may be able to reduce the number of data tables by joining some of them using the Join prefix or other mapping. This is especially important for large fact tables. A good rule of thumb is to have only one large table.
Tables that have a small number of fields are denormalized	If you have two tables with few fields, it may improve performance to join them. .
Denormalized lookup (leaf) tables with mapping loads	You should not use the Join prefix if you only need to add one field from a table to another. You should use the ApplyMap lookup function.
Time stamps removed or decoupled from date field	Date fields can fill up space when the timestamp is present as the string representation is larger, and the number of distinct values is larger. If the precision is not necessary for your analysis, you can round the timestamp to e.g. the nearest hour using <i>Timestamp(Floor(YourTimestamp,1/24))</i> or remove the time component completely using <i>Date(Floor(YourTimestamp))</i> . If you want the timestamp, you can decouple it from the date itself. You can use the same Floor() function, and then create a new field with the extracted time by using something along the lines of: <i>Time(Frac(YourTimestamp))</i> .
Unnecessary fields removed from data model	You should only load necessary fields in your data model. Avoid using Load * and SELECT. Make sure you keep: <ul style="list-style-type: none"> • Fields that are necessary for your analysis. • Fields that are actually being used in the app.

Action	Description
Link tables avoided when dealing with high data volumes	You should use link tables where possible. However, if you are dealing with large data volumes, concatenated tables can outperform link tables.
Concatenated dimensions broken to new fields	You should break apart concatenated dimensions into separate fields. This reduces the number of unique occurrences of values in your fields. This is similar to how timestamps can be optimized.
AutoNumber used where possible	You can create an optimized load by loading your data from a QVD file first, and then using the AutoNumber statement to convert values to symbol keys.
Data islands avoided	Data islands can be useful, but they usually affect performance. If you are creating islands for selection values, use variables.
QVDs are stored based on incremental timeframes	You should store QVD in segments, such as monthly. These smaller monthly QVD can then support many different apps that might not need all of the data.

Sheet performance

These are best practices that will improve performance of sheets and visualizations.

Sheet performance best practices

Action	Description
The If() function is avoided where possible	If the If() function is used inside an aggregation function, it will operate at the record level and be evaluated many times. For example, if you have 1000 records in an aggregation, an If() condition will be evaluated 1000 times. This could cascade rapidly if you nest statements. You should use set analysis instead. A set analysis filter is applied before the aggregation, resulting in a faster response. These responses can also be cached via set analysis, where If() cannot. You could also consider other functions and modifications to the data model.

Action	Description
Fields from different tables inside an aggregation table are avoided where possible.	<p>When an aggregation is evaluated, the calculation runs through two steps:</p> <ol style="list-style-type: none"> 1. The first step is finding the relevant combinations on which to make the calculation. This step is single-threaded. 2. The second step is performing the calculation. This step is multi-threaded. <p>The single-threaded part can affect performance considerably. One example is if you have multiple fields inside the aggregation, for example, Sum (Quantity*ListPrice). If Quantity is in the fact table, and ListPrice is in the master products table, the engine first needs to join the two tables to find the combinations before it can start to sum the product. The joining is the single-threaded part, and the summing is multi-threaded. If both fields are found in the same table, no join is necessary, and the aggregation is evaluated considerably faster.</p>
Aggr() and nested Aggr() functions are used minimally	The Aggr() function greatly affects performance. Incorrect use can give inaccurate results. For example, in a table with dimensions that vary from the dimensions within the Aggr() function.
Set analysis is used where possible	You can use set analysis to define a set of data values that is different from the normal set defined by the current selections.
String comparisons avoided where possible	String comparisons are not as efficient as set analysis. For example, you should avoid Match(), MixMatch(), WildMatch(), and Pick(). Create flags in the script or use set analysis instead.

Action	Description
Calculation conditions are used on objects containing intensive calculations	You may have visualizations with many records when there are no selections. As a best practice, add calculation conditions to objects so that they only render after certain selections have been made. This stops the creation of very large hypercubes. For example: <code>GetSelectedCount([Country])=1 OR GetPossibleCount([Country])=1</code> . In this scenario, the visualization will not render unless the user selects a single country, or makes other selections where only a single country is possible.
Measures are pre-calculated in the script where possible	Any measure that is at the lowest level of granularity of the data model should be calculated in the script. For example, if in the same record in a table you have Sales and Cost, you could derive the margin by calculating <code>Sales - Cost AS Margin</code> . You can also aggregate other values in advance if you know that they will not vary based on selection, or that are bound to a different level of granularity.
Tables have less than 15 columns and have calculation conditions	A table with 15 columns could be considered wide. If your tables consist of many records, you should use calculated conditions on the table object so that it only renders after certain selections or criteria have been met. If your table is very wide, consider: <ul style="list-style-type: none"> • Creating multiple smaller tables that show conditionally. • Using methods to conditionally show columns. • Keeping your tables limited to only what fields are necessary for your analysis.

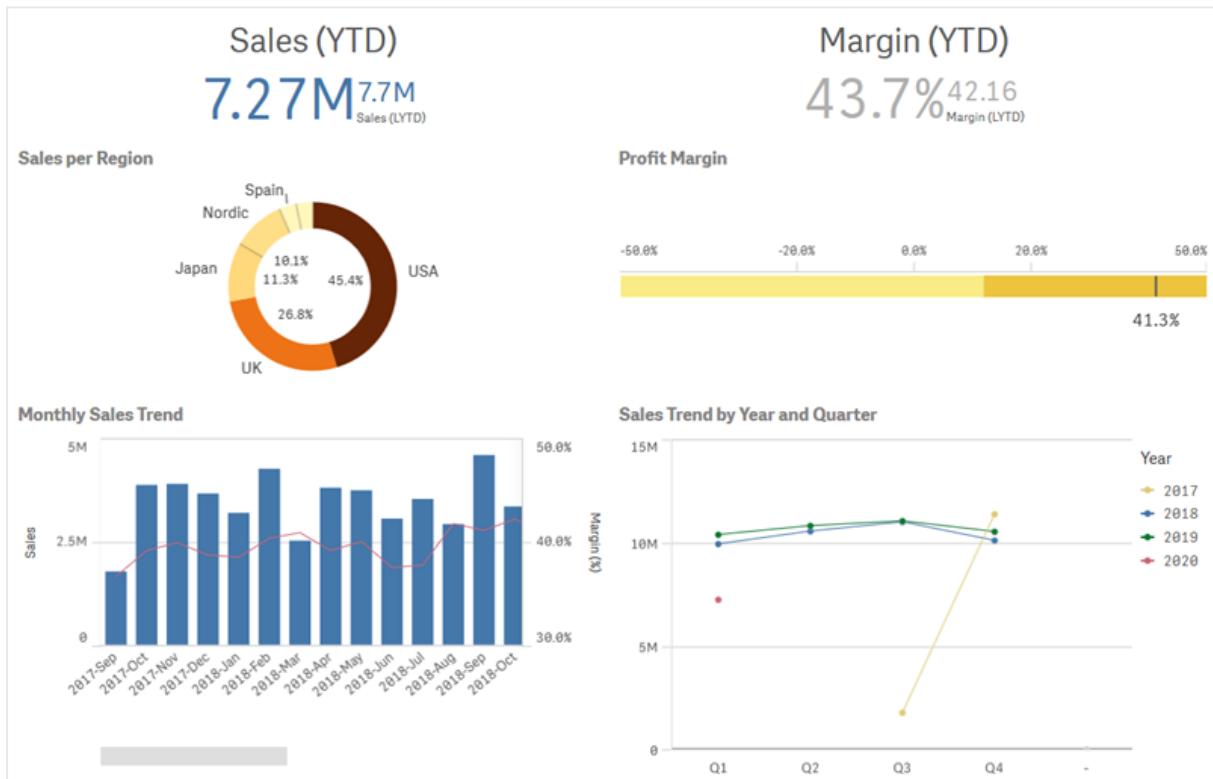
Action	Description
Sheets do not have an excessive number of objects	Objects are calculated when the user navigates to the sheet. Every time a user makes a selection on that sheet, each object will be recalculated if that current state does not exist in the cache. If you have a sheet with many charts, the user will have to wait for every object to calculate on nearly every selection. This puts significant load on the engine. As a best practice, follow the Dashboard/Analysis/Reporting (DAR) concept to develop a clean and minimal app.
Numeric flags are leveraged in the script for use in set analysis	Set analysis with flags can be more efficient than using string comparisons or multiplication.
Master items or variables used for expressions	Master items enable drag and drop of governed metrics and guarantee that expressions will be cached. For example, <i>Sum(Sales)</i> is different from <i>SUM(Sales)</i> . Expressions are cached on spelling and case, and need to match verbatim in order to be reused.

3 Visualizations

Visualizations let you present data so that your app's users can interpret and explore it. For example, a bar chart that compares sales numbers for different regions, or a table with precise values for the same data. Good visualizations help you quickly and accurately interpret displayed data.

Visualizations are easy to add and customize. They can take the form of charts, such as bar charts, pie charts, tables, gauges, or treemaps. Each chart type has unique functionality. In the Qlik Sense Dev Hub, you can create custom objects. Qlik Sense automatically highlights items associated with your selections so you can drill-down and filter.

A sheet with four different visualizations



3.1 Understand the data sources of your visualizations

You need to understand your data in order to design an effective visualization with a clear purpose. Ask yourself the following:

- What kind of data is it? Nominal, ordinal, interval, or ratio data?
- How do different parts of the data relate to each other?
- Can you organize the data in a way that makes it easy for you to create your visualizations?

- What do you want to communicate with your data?
- Are there dimensions or measures you will want to reuse in multiple visualizations?

You also need to understand your data in terms of fields, dimensions, and measures. These affect how your data is used in your visualizations.

3.2 Select visualization types that align with your purpose

Each visualization type has a specific goal. You need to think about the purpose of your visualization, and pick a visualization type that lets you explore your data for that purpose effectively.

For example: You want to show how a measure, quarterly sales, behaves over time. You should create a line chart, because one of its strengths is displaying how measure values change over time.

If you are unsure of what visualization types to use with your data, Qlik Sense offers two methods of creating visualizations with assistance:

- You can use Insight Advisor to let Qlik Sense analyze your data and generate visualizations based on your searches or selections. You can then choose to add these visualizations to your sheets.
- You can also create visualizations using chart suggestions by dragging a field onto the sheet from the assets panel and then dragging additional fields that you want in the visualization onto the first field. Qlik Sense then creates a suggested visualization based on the fields selected for the visualization.

3.3 Update visualizations to improve how data is displayed

As you create your visualizations, you can modify them to improve how data is displayed and highlighted within the sheet. You can also change the data, or switch the visualization type.

3.4 Data assets in visualizations

Visualizations use data in many different ways. How your data is comprised or created impacts your visualizations. Primarily, your data assets become dimensions and measures in your visualizations, defining the categories in your visualizations and the measurements of those categories. A field can be used to group data, or it can be transformed with an aggregate function to provide a measurement in data categories.

The types of data you have in your tables and fields also impacts whether they can be used as dimensions or measures, as well as what sorting options are most effective. For example, quantitative data and qualitative data have different recommended uses when they are used as either dimensions or measures.

In addition to providing the data to display, data assets can be used to control what data is displayed and how it is presented. For example, you can color a visualization using a dimension or measure not present in the visualization. For more information, see *Changing the appearance of a visualization* (page 454).

The assets panel contains the different data sources you can use in your visualizations.

Data assets

The following data assets are available when creating visualizations:

- Fields
- Measures
- Dimensions
- Master items

Fields

Fields hold the data loaded into Qlik Sense. Fields contain one or more values and correspond to columns in a database table. The field data can be qualitative or quantitative.

When creating visualizations, you use fields to create your dimensions and measures. You can also use fields in different ways when you add visualizations to your app. Some visualizations, such as tables, can present fields in an unmodified state.

Some fields require extra considerations, such as date or time fields.

For more information, see *Fields (page 64)*.

Measures

Measures are the data that you want to show. Measures are created from an expression composed of aggregation functions, such as **Sum** or **Max**, combined with one or several fields.

For more information, see *Measures (page 77)*.

Dimensions

Dimensions determine how the data in a visualization is grouped. For example: total sales per country or number of products per supplier. Dimensions display the distinct values from the field selected as a dimension. Dimensions can also be calculated using an expression.

For more information, see *Dimensions (page 74)*.

Master items

Master items are dimensions, measures, or visualizations that can be reused in other visualizations and sheets in your app. Updating a master item updates every instance of it. This means you could have the same measure in 5 visualizations, and they would all update whenever the master item is changed.

Master items also have more design options available. You can, for example, assign colors to a master dimension's distinct values so that the distinct values are consistent across visualizations.

Master items also include special dimensions such as drill-down dimensions and calendar measures.

For more information, see *Reusing assets with master items (page 86)*.

Expressions

An expression is a combination of functions, fields, and mathematical operators (+ * / =). Expressions are used to process data in the app in order to produce a result that can be seen in a visualization

Expressions are used primarily used to create measures. They can also be used to build calculated dimensions, or to set properties within different visualizations. For example, you can use expressions to define range limits for gauges, or reference lines for bar charts.

For more information, see *Using expressions in visualizations (page 122)*.

Data types in visualizations

Different types of data have different properties; certain data may work better as dimensions, and some as measures. Similarly, as dimensions or measures, certain kinds of data may work better as a dimension in some visualizations better than others, or as a measure with certain aggregation functions.

The data in your fields can be quantitative or qualitative. Quantitative data values are measured numerically on an ascending scale. Quantitative data can be ratios or intervals:

- **Ratio:** Ratios are quantitative data that you can perform arithmetic operations on, such as cost or age. For example, you can sum sales values for the month to get totals.
- **Interval:** Intervals are quantitative data that you cannot perform arithmetic operations on. For example, you cannot calculate a sum of temperatures during the week, but you can calculate the average temperature per day, and the high/low for each day.

Qualitative data can not be measured numerically, but can be described through language. Qualitative data can be nominal or ordinal:

- **Nominal:** Fields with nominal data have distinct qualitative values, but without a set order. For example, product names or customer names are nominal data, as they have distinct values, but do not have a required order.
- **Ordinal:** Fields with ordinal data have qualitative values that have a ranked or positioned value. Ordinal data should be sorted by its order as opposed to alphabetically. For example, low, medium, high are ordinal values. Small, medium, and large are also ordinal values.

The following table contains a general overview of recommended visualization types and aggregation functions for data types. These recommendations should not be considered absolute.

Visualization recommendations for data types as measures

Data type	Recommended aggregation functions	Non-recommended aggregation functions
Nominal	Count	Average Median Sum
Ordinal	Count	Average
	Median	Sum

Data type	Recommended aggregation functions	Non-recommended aggregation functions
Ratio	Count Average Median	Sum
Interval	Count Average Median Sum	-

Fields

Fields hold the data that is used in Qlik Sense. Fields can be thought of as the data loaded from the load script.

Fields contain one or more values, called field values, and at the basic level, correspond to columns in a database table, but can also exist in more than one table. Field values consists of numeric or alphanumeric data. When loaded from the load script, fields can be represented as a table visualization.

Example of data in a load script:

```
Temp:  
LOAD * inline [  
Customer Product UnitSales UnitPrice  
Imagine Film 4 16  
Imagine Film 10 15  
Imagine Shutter 9 9  
PhotoInc Shutter 5 10  
PhotoInc Lens 2 20  
PhotoInc Magnifier 4 25  
Gallery Film 8 15  
Gallery Lens 7 19  
] (delimiter is ' ');
```

The fields represented in a data model table after having loaded the data:



The same fields as columns in a table visualization on a sheet:

Customer	Product	UnitPrice	UnitSales
Gallery	Film	15	8
Gallery	Lens	19	7
Imagine	Film	15	10
Imagine	Film	16	4
Imagine	Shutter	9	9
PhotoInc	Lens	20	2
PhotoInc	Magnifier	25	4
PhotoInc	Shutter	10	5

Date & time fields

If you are working with fields containing date or timestamp information in your app, you can define a number of related attributes of a date, for example, year or week, and use them in your visualization.

Creating date fields in **Data manager**

Date fields are created automatically for all data fields recognized as a date or a timestamp when you use **Add data** with data profiling enabled to build your data model in **Data manager**, or when you click **Load data** in **Data manager**.



*Date fields created in **Data manager** are automatically added to autoCalendar.*

If the date or timestamp field is not recognized automatically, you can adjust the input format in the **Data manager** table editor. You can also set the display format to use in visualizations.

Which date & time fields are automatically recognized ?

Date & timestamp fields will be recognized automatically based on your system locale settings. Additionally, the following formats are recognized:

- M/D/YYYY h:mm
- D/M/YYYY h:mm TT
- M/D/YYYY
- D/MM/YYYY
- YYYYMMDD
- YYYYMMDDhhmmss
- YYYYMMDDhhmmss.fff
- YYYYMMDDhhmmssK
- YYYY-MM-DD
- YYYY-MM-DDThh:mm:ss
- YYYY-MM-DD-Thh:mm:ss.fff
- YYYY-MM-DD-Thh:mm:ssK

Date & time formats

Format specifier	Description
YYYY	Year
M, MM	Month
D, DD	Day
hh	Hour
mm	Minute
ss	Second
fff	Millisecond
TT	AM or PM
K	Timezone
T	Divider between date and time. T can not be replaced with another character.

Creating date & time fields in the data load script

If you use the data load editor to build your data model, you need to create a calendar template where you define which fields to derive in the data load script. The derived date & time fields will be generated when the script is run and data is reloaded.

Using date & time fields in your app

Date & time fields in visualizations

All date or timestamp fields in the assets panel **Fields** tab are marked with  and you can expand them to use the generated date & time fields. You can use them in visualizations, just like any other data field.

Date & time fields in dimensions

You can also use date & time fields when you create a dimension. The date & time fields are listed under the field that they have been generated from.

Date & time fields in expressions

You can use date & time fields in all expressions, for example when you create a measure. The date & time fields are named according to:

[field name].autoCalendar.[date & time field].

- [field name] is the name of the data field that was used to generate date & time fields.
- [date & time field] is the date & time field you want to use, for example, Year.

Example:

Date.autoCalendar.Year

Date & time fields in calendar measures

Calendar measures use date & time fields created in autoCalendar. Each of these date & time fields is calculated by a set analysis expression that determines whether or not data falls within the time-to-date period, or if dates are within a defined relative position to the current date. These date & time fields are relative and return results based on the current date. You can use these fields independently of calendar measures.

Calendar field date & time fields are formatted as follows:

[field name].autoCalendar.[date & time field]=[value]

- [field name] is the name of the date field used to generate date & time fields.
- [date & time field] is the name of the date & time field used, for example, InYTD.
- [value] is the value for the date & time field's set analysis expression and determines which dates are included.

Example:

`Date.autoCalendar.YearsAgo={1}`

The following are the available date & time fields with sample values:

InYTD

This date & time field determines whether or not dates are within the year-to-date range or outside the year-to-date range.

InYTD

Example	Result
<code>Date.autoCalendar.InYTD={0}</code>	Returns all dates that fall in the year-to-date time range. For example, if the current date was the 54th day of the year, the dates within the first 54 days of every year in the date field would be included.
<code>Date.autoCalendar.InYTD={1}</code>	Returns all dates outside the year-to-date time range. For example, if the current date was the 54th day of the year, all the dates after the first 54 days of every year in the date field would be included.

YearsAgo

This date & time field determines whether or not dates are from a specific year relative to the current date.

YearsAgo

Example	Result
<code>Date.autoCalendar.YearsAgo={0}</code>	Returns all dates from this year.
<code>Date.autoCalendar.YearsAgo={1}</code>	Returns all dates from last year.
<code>Date.autoCalendar.YearsAgo={8}</code>	Returns all dates from eight years ago.

InQTD

This date & time field determines whether or not dates are within the quarter-to-date range or outside the quarter-to-date range, relative to the current date.

InQTD

Example	Result
Date.autoCalendar.InQTD={0}	Returns all dates from all quarters that are within the quarter-to-date range. For example, if the current date was the 14th day of Quarter 1, the first 14 days of every quarter in the date field would be included.
Date.autoCalendar.InQTD={1}	Returns all dates from all quarters that are outside the current quarter-to-date range. For example, if the current date was the 14th day of Quarter 1, all dates after the first 14 days of every quarter in the date field would be included.

QuartersAgo

This date & time field determines whether or not dates are from a specific quarter relative to the current date.

QuartersAgo

Example	Result
Date.autoCalendar.QuartersAgo={0}	Returns all dates from the current quarter.
Date.autoCalendar.QuartersAgo={1}	Returns all dates from the last quarter.
Date.autoCalendar.QuartersAgo={8}	Returns all dates from eight quarters ago.

QuarterRelNo

This date & time field determines whether or not dates are from a specific quarter relative to the current date.

QuarterRelNo

Example	Result
Date.autoCalendar.QuarterRelNo={0}	Returns all dates from each instance of the current quarter. For example, if the current date was in Quarter 4, all dates from each Quarter 4 in the date field would be included.
Date.autoCalendar.QuarterRelNo={3}	Returns all dates from each instance of the quarter two quarters prior to the current quarter. For example, if the current date was in Quarter 4, all dates from each Quarter 1 in the date field would be included

InMTD

This date & time field determines whether or not dates are within the month-to-date range or outside the month-to-date range, relative to the current date.

InMTD

Example	Result
Date.autoCalendar.InMTD={0}	Returns all dates from all months that are within the current month-to-date range. For example, if the date was November 15, 2016, dates from the first 15 days of every month in the date field would be included.
Date.autoCalendar.InMTD={1}	Returns all dates from all months that are outside the current month-to-date range. For example, if the date was November 15, 2016, dates after the first 15 days to the end of the month of every month in the date field would be included.

MonthsAgo

This date & time field determines whether or not dates are from a specific month relative to the current date.

MonthsAgo

Example	Result
Date.autoCalendar.MonthAgo={0}	Returns all dates from the current month.
Date.autoCalendar.MonthAgo={1}	Returns all dates from the last month.
Date.autoCalendar.MonthAgo={8}	Returns all dates from eight months ago.

MonthRelNo

This date & time field determines whether or not dates are from a specific month relative to the current date.

MonthRelNo

Example	Result
Date.autoCalendar.MonthRelNo={0}	Returns all dates from each instance of the current month. For example, if the current month was June, all dates in every instance of June in the date field would be included.
Date.autoCalendar.MonthRelNo={1}	Returns all dates from the each instance of the previous month. For example, if the current month was June, all dates in every instance of May in the date field would be included.

InWTD

This date & time field determines whether or not dates are within the week-to-date range or outside the week-to-date range, relative to the current date.

InWTD

Example	Result
Date.autoCalendar.InWTD={0}	Returns all dates from all weeks that are within the current week-to-date range. For example, if the current date was the third day of a week, dates from the first three days of every week in the date field would be included.
Date.autoCalendar.InWTD={1}	Returns all dates from all months that are outside the current month-to-date range. For example, if the current date was the third day of a week, dates from the last four days of the every week in the date field would be included.

WeeksAgo

This date & time field determines whether or not dates are from a specific week relative to the current week.

WeeksAgo

Example	Result
Date.autoCalendar.WeeksAgo={0}	Returns all dates from the current week.
Date.autoCalendar.WeeksAgo={8}	Returns all dates from eight weeks ago.

WeekRelNo

This date & time field determines whether or not dates are from a specific week relative to the current date.

WeekRelNo

Example	Result
Date.autoCalendar.WeekRelNo={0}	Returns all dates from each instance of the current week. For example, if the current week was the second week of the year, dates from the second week of every year in the date field would be included.
Date.autoCalendar.WeekRelNo={1}	Returns all dates from each instance of the previous week. For example, if the current week was the second week of the year, dates from the first week of every year in the date field would be included.

Always one selected value

You can select the **Always one selected value** for a field in an app. The value specifies that one, and only one, value is always selected for a field. You can use this value to specify, for example, that one currency is always selected for a field.



Consider the following when using the **Always one selected value**:

- You can use the **Always one selected value** on one or more fields.
- You cannot clear the **Always one selected value** setting when viewing a sheet. You can, however, clear the setting by editing the sheet in your workspace.
- When you publish an app with the **Always one selected value** selected, the value is applied to all sheets in the app. The app consumer cannot clear the value.
- When you duplicate an app, the **Always one selected value** is duplicated with the app.
- When you export an app without data, the **Always one selected value** is not exported with the app.
- The **Always one selected value** is not supported for derived fields.

You can use the value with a default bookmark in to highlight specific selections for your app consumer.

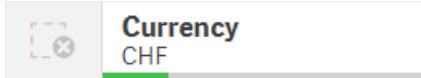
Set the Always one selected value

Do the following:

1. Open a sheet in an app.
2. Click **Edit sheet**.
3. Click **Fields**.
4. Right-click a field, and then click **Field settings**.

5. In the window that opens, select the **Always one selected value** check box, and then save.
6. Click  **Done** to finish editing. The value appears as a tab in the top toolbar. Unlike a bookmark, there is no option to clear the selection.

The Always one selected value in the toolbar



The **Always one selected value** defaults to the first entry in the field. In the example above, the value is applied to **Currency**. **CHF** is the currency that is selected by default. You or your app consumer can change this selection, in this case to a different currency.

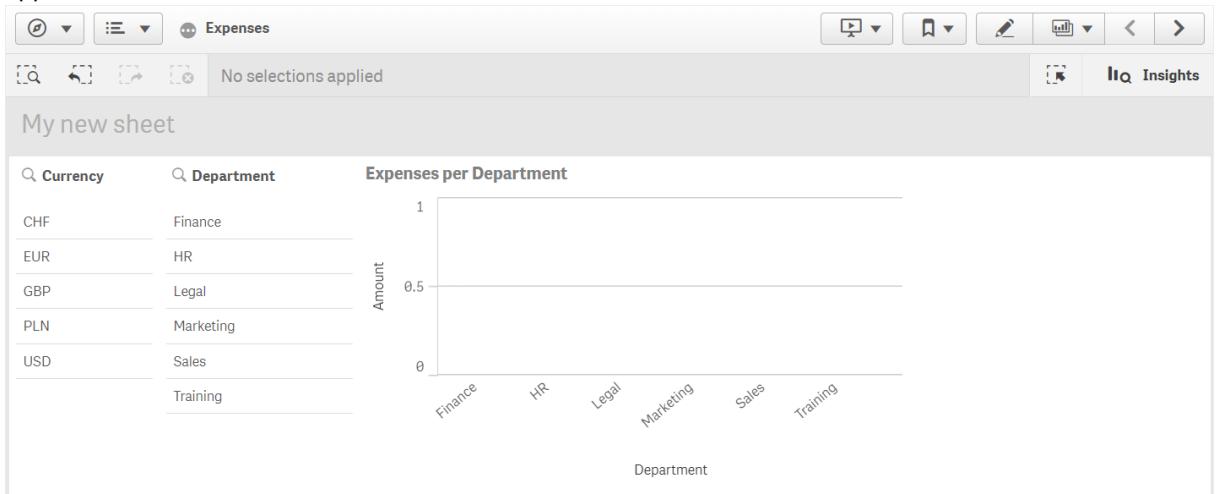
7. To test that the **Always one selected value** is working properly, close and then reopen the app. The value should be shown in the top toolbar.

To remove the **Always one selected value** for a field, right-click the field in **Edit** mode, click **Field settings**, and deselect the **Always one selected value** check box.

Always one selected value and default bookmark example: Controlling selections to guide app consumers to insights

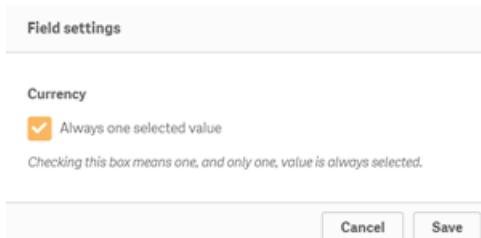
In this example, we set **Always one selected value** for a field. We then specify a default bookmark for the app. By specifying a default bookmark, and enabling **Always one selected value**, we control what our app consumers see when they open our published app.

1. Here is a sheet in our app before the **Always one selected value** and a default bookmark are applied. This should be the sheet that you want app consumers to see as a landing page when they open the app.



For the **Amount** measure in the chart above, we used this expression: `=sum(DISTINCT {<Currency=>} Amount_LOCAL)* RATE`

2. To ensure that one, and only one, value is always selected for the **Currency** field, we edit the sheet, click **Fields**, and then right-click **Currency**.
3. We then select the **Always one selected value** check box and save.



4. When we close **Edit** mode for the app, the **Always one selected value** value appears as a tab in the top toolbar. The value defaults to the first entry in the field, in this case the value is **CHF**.



5. However, we want to show expenses in **Euro** when the app is opened. So we select **Euro** from the **Currency** field, and then click **Bookmarks** to create a new bookmark for **Currency("EUR")**. We then right-click the **Currency("EUR")** bookmark, and click **Set as default bookmark**.

Now, when we open the app, or publish the app for an app consumer, the app opens with one value selected, and the default bookmark is shown. The app is opened on the sheet of the bookmark instead of the app overview.



Dimensions

Dimensions determine how the data in a visualization is grouped. For example: total sales per country or number of products per supplier. You typically find a dimension as the slices in a pie chart or on the x-axis of a bar chart with vertical bars.

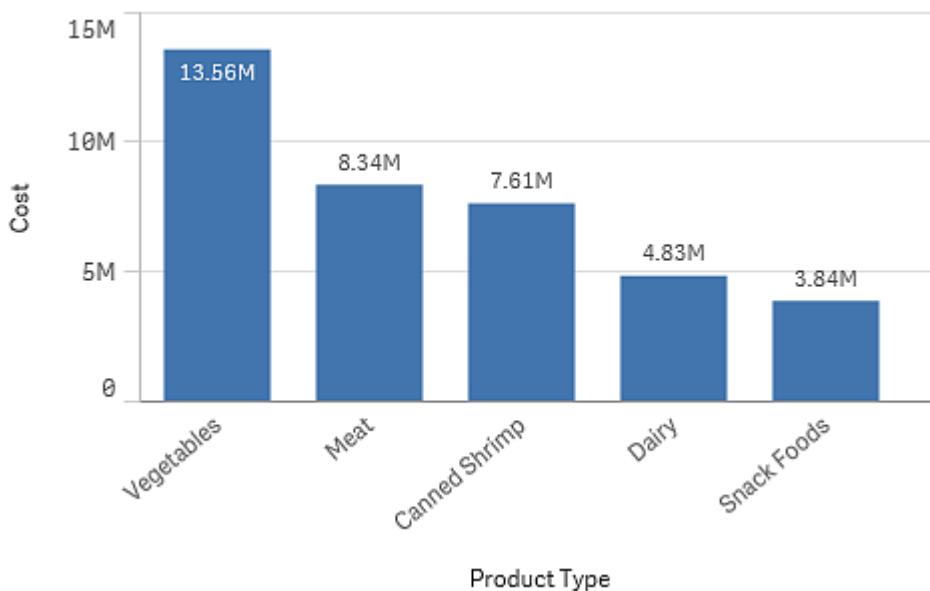
Dimensions are created from fields in the data model tables.

Example:

Product Type is a field in the *Product* table that is loaded into the app. The values of this field are the different types that products are grouped into.

You can, for example, create a bar chart to visualize the cost of each type, by adding the *Product Type* dimension to the chart. To complete the visualization, you must add a measure (in this case Cost), which is grouped by the *Product Type* dimension.

Bar chart with the dimension Product Type and measure Cost.



Field groups as dimensions

One main difference between Qlik Sense and many other database viewers and online analytical processing tools (OLAP systems), is that in Qlik Sense, you do not need to predefined any hierarchies in the input data. The unique internal logic of Qlik Sense gives you the complete freedom to access any field as a full dimension in any order you like.

For most purposes, the built-in functionality is fully satisfactory, but in some situations, a predefined hierarchy can help you to display data more efficiently. In Qlik Sense, you can achieve this by defining hierachic groups of fields as drill-down dimensions.

Any fields or calculated dimensions can be grouped together.

Drill-down groups

When several fields form a natural hierarchy, it can make sense to create a drill-down group.

Example 1:

Organization: Company, Department, Employee

Example 2:

Geography: Continent, Country, State, City

When you use a drill-down group as a dimension in a chart, the chart uses the first field in the group's list of fields that has more than one possible value. If the currently made selections cause the field to have only one possible value, the next field in the list is used instead, provided that it has more than one possible value. If no field in the list has more than one possible value, the last field is used anyway.

In the first example above, *Company* will be used as chart dimension until a single company is selected. The chart will then show *Department*. If a single department is selected, the chart will switch to *Employee*.

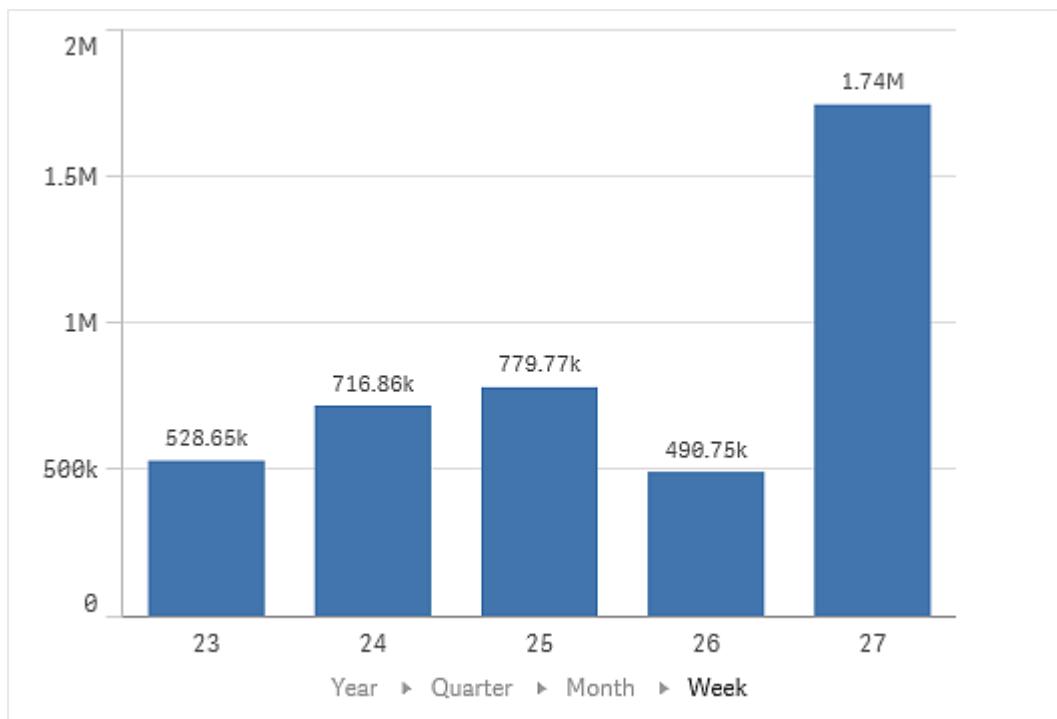
As selections are reverted, so that more than one value becomes possible in the upper fields of the group's field list, the chart is automatically drilled back up.

Drill-up

The drill-up function is available in bar charts, pie charts, and line charts. Other visualizations reflect the changes made in the charts, but cannot themselves be used to drill up through the different dimensions.

When you drill down in a dimension group, breadcrumbs provide links back to the previous dimensions. Click the dimension that you want to drill up to.

In the following bar chart, the breadcrumbs *Year > Quarter > Month* enable drilling up.



Calculated dimensions

You can use expressions to create calculated dimensions. A calculated dimension consists of an expression involving one or more fields. All standard functions may be used.



For performance reasons, it is recommended to perform all calculations in the data load editor. When dimensions are calculated in the chart, Qlik Sense first calculates the dimension values, and then aggregates the measures for these calculated values, which affects the performance more than calculations in the load script.

There are cases when calculated dimensions are powerful in data analysis, for example, if you want to generate the dimensions values during analysis, when dimension values are dependent on the selections.

Calculated dimensions are also useful if you want to modify a field.

Once you have created a calculated dimension, you can use it as any other dimension.

Example:

You have a field called Calendar Month that includes each of the months of the year. In your app, you want include a table that shows the sales for each of the first 6 months of the year. For the rest of the months, you want to see a total. You can use an expression to create this calculated dimension.

Syntax:

```
If ([Calendar Month] <7, [Calendar Month], 'Rest')
```

Measures

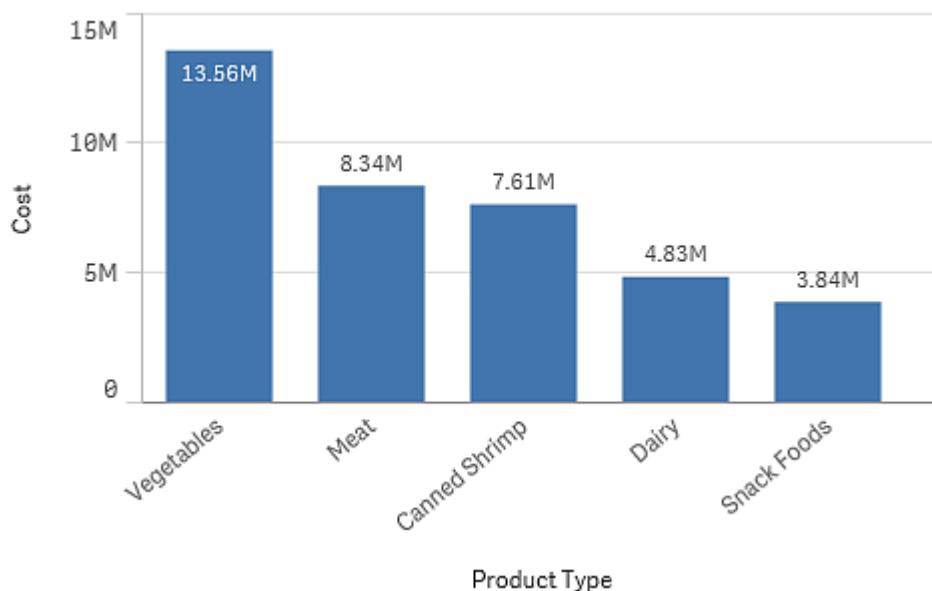
Measures are calculations used in visualizations, typically represented on the y-axis of a bar chart or a column in a table. Measures are created from an expression composed of aggregation functions, such as **Sum** or **Max**, combined with one or several fields.

A measure must have a name, and may also be supplied with descriptive data such as description and tags.

Example:

You can, for example, create a bar chart to visualize the cost of each type, by adding the *Product Type* dimension to the chart, and the measure *Cost*, which is made from the expression **Sum(Cost)**, that is the result of the calculation of the aggregation function **Sum** over the field **Cost**. The results are grouped by the *Product Type* dimension.

Bar chart with the dimension Product Type and the measure Cost.



Modifiers

Measures are calculations based on fields, for example **Sum(Cost)**. You can use modifiers to change how the measure is calculated over the available dimensions.

For example, you can have the values of a measure accumulate over one or two dimensions, or you can calculate the average of your measure over a specific number of steps.

Which visualizations have modifiers

Visualizations comparison

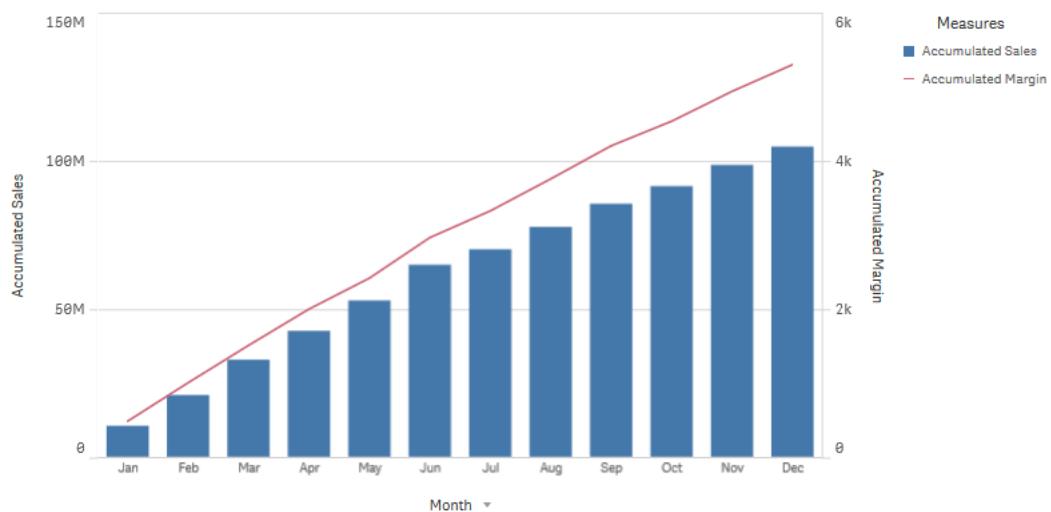
Visualization	Accumulation	Difference	Moving average	Relative numbers
Bar chart	Yes	Yes	Yes	Yes
Combo chart	Yes	Yes	Yes	Yes
Line chart	Yes	Yes	Yes	Yes
Table	Yes	Yes	Yes	Yes

Accumulation

The accumulation modifier allows you to accumulate the values of a measure over one or two dimensions. Accumulating values makes it easy to visualize how the effect of the measure builds up over a dimension.

In the following combo chart, the bars and lines accumulate over time.

A *combo chart* where the line shows the accumulated profit margin and the bars show the accumulated sales figures.



Syntax:

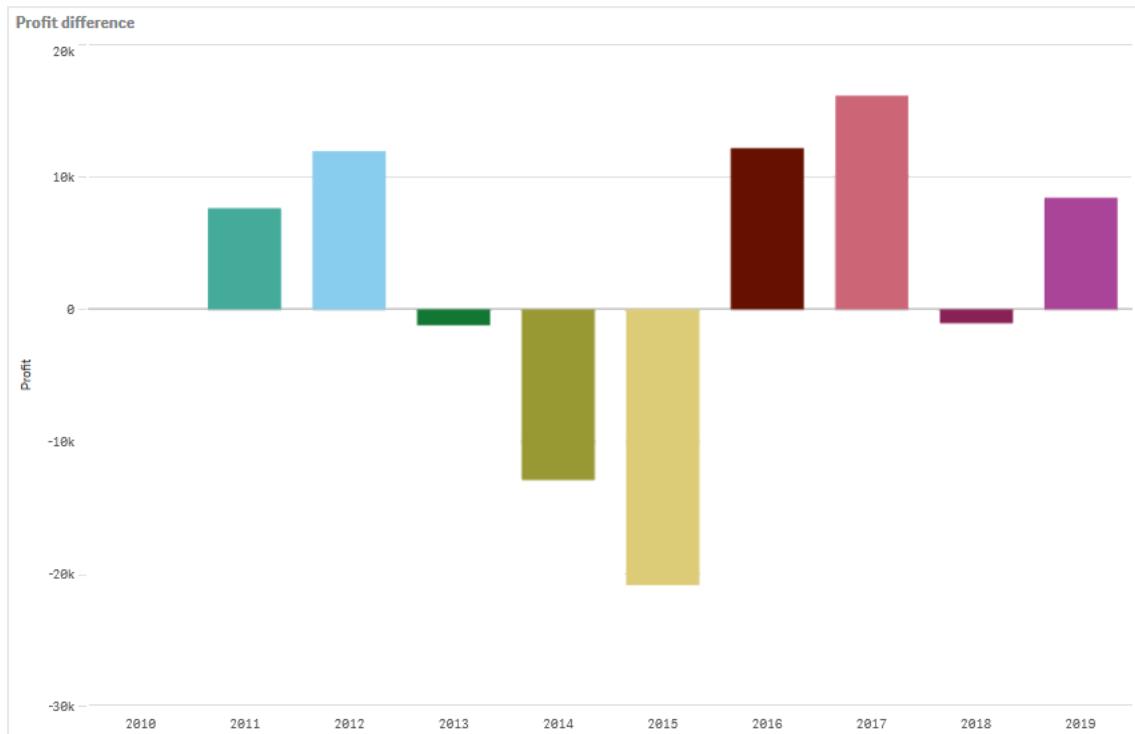
```
RangeSum(Above { $M, 0, Steps })
```

Difference

The difference modifier allows you to visualize the difference between consecutive values of a measure over one or two dimensions. The difference modifier is useful when you want to visualize the change in direction of grouped data.

In the following bar chart, any drops in yearly profits over a 10 year period appear as negative bars.

A *bar chart* showing the profit differences from one year to the next.



Syntax:

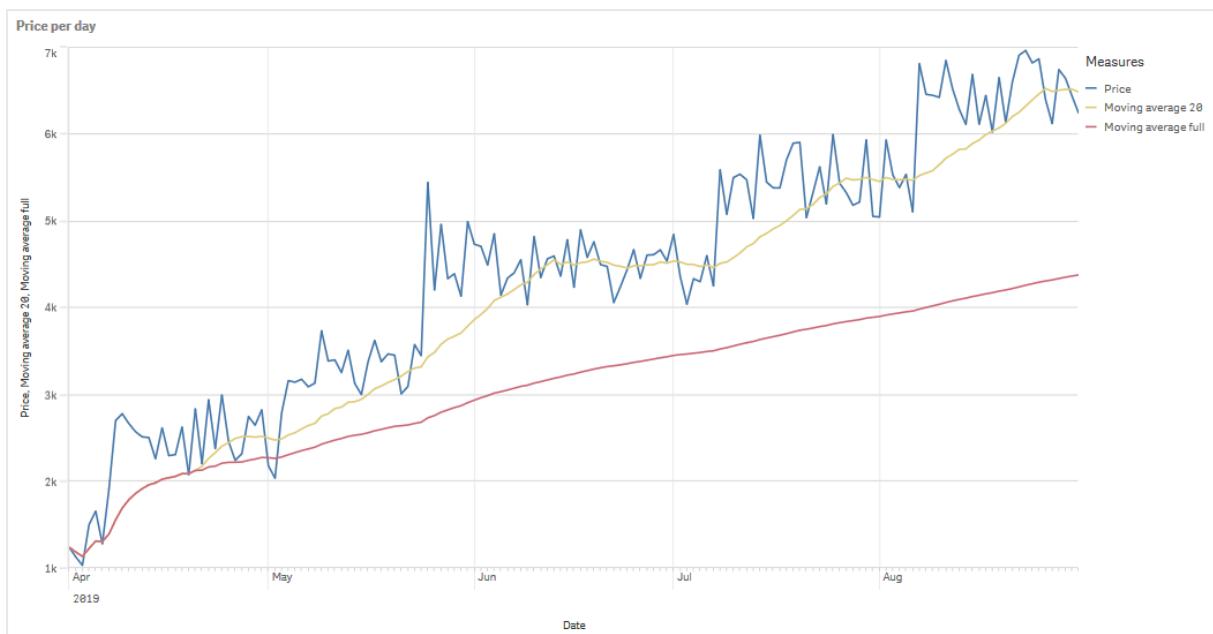
\$M – Above (\$M)

Moving average

The moving average modifier allows you to see the average values of a measure over a specific period. You can use it to filter out the action from short-term value fluctuations. You can change the number of steps over which the averaging takes place, depending on how strongly you want your modifier to follow the changes in your data. A moving average is commonly used with time series data to highlight longer-term trends or cycles.

In the following line chart moving averages with two difference ranges are shown, one with a 20 step range, and one with a full range.

A line chart showing the price of a product over a five month period.



Syntax:

```
RangeAvg(Above ($M, 0, Steps))
```

Relative numbers

The relative numbers modifier allows you to see relative percentages. You can use it to see the impact of specific selections, relative to the selection, relative to the total, or relative to other fields. You can change the basis upon which the relative number is calculated.

In the following table a column with sales of each year of a specific selection, and three columns with relative numbers are shown, one relative to current selection, one relative to total sales for all years, and one relative to the sales of each year.

A table showing different sales percentages relative to current selection, relative to total sales, and relative to each year's sales.

Sales comparison table				
Year	Sales	Percentage of this selection	Compared to total sales from all years	Compared to other products in the same year
Totals	\$ 20,520,054	100%	20%	20%
2012	\$ 8,296,002	40%	8%	21%
2013	\$ 7,602,738	37%	7%	18%
2014	\$ 4,621,314	23%	4%	21%

The following is an example of the syntax of the modifier relative to the total selection.

Syntax:

```
$M / Sum( total Aggr( $M, dim ) )
```

Trend lines

A trend line is a visual representation of the direction of values over a period of time. Trend lines show trends over time, by visualizing the direction of values and how fast the values change.

Which visualizations have trend lines

Visualization	Trend lines
Bar chart	Yes
Line chart	Yes

When to use trend lines

A single trend line can be added to a chart to smooth out fluctuations in data, and to show any trends more clearly.

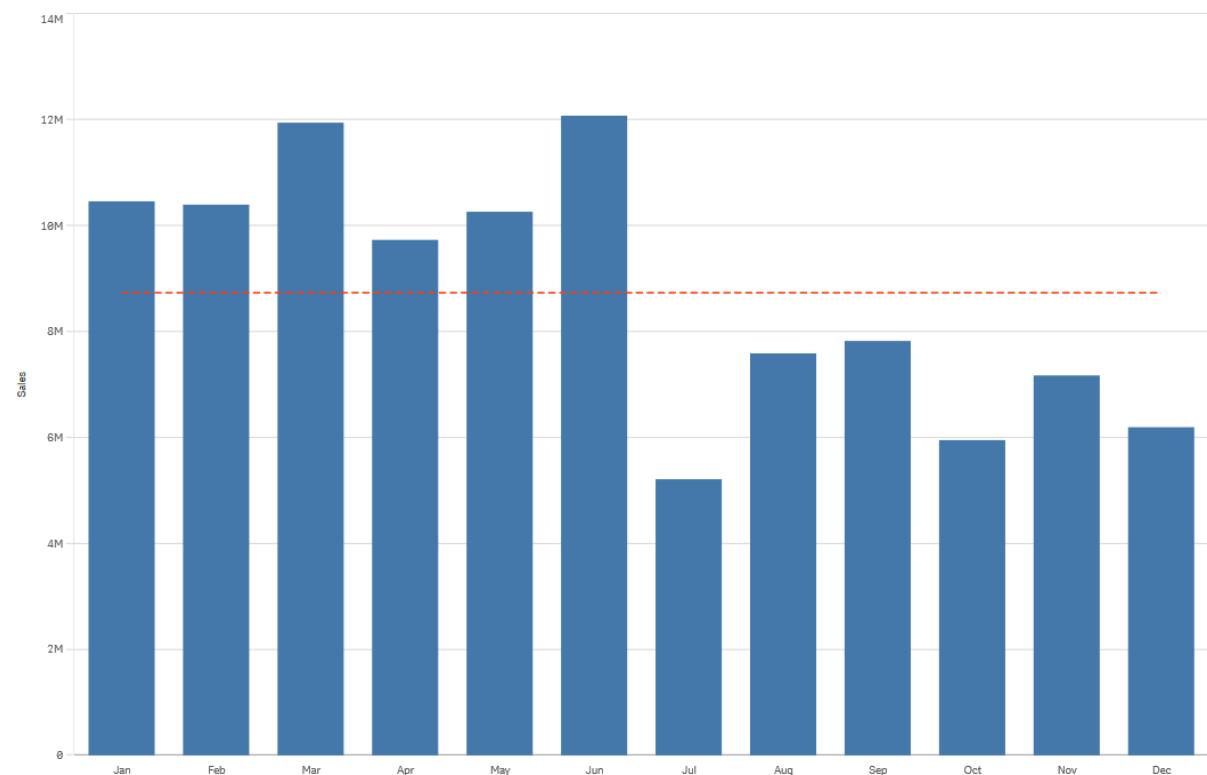
More than one trend line can be added to a chart, showing different types of trends or different values.

Types of trend lines

Average

An average trend line shows the average value of the data, for the time period being analyzed.

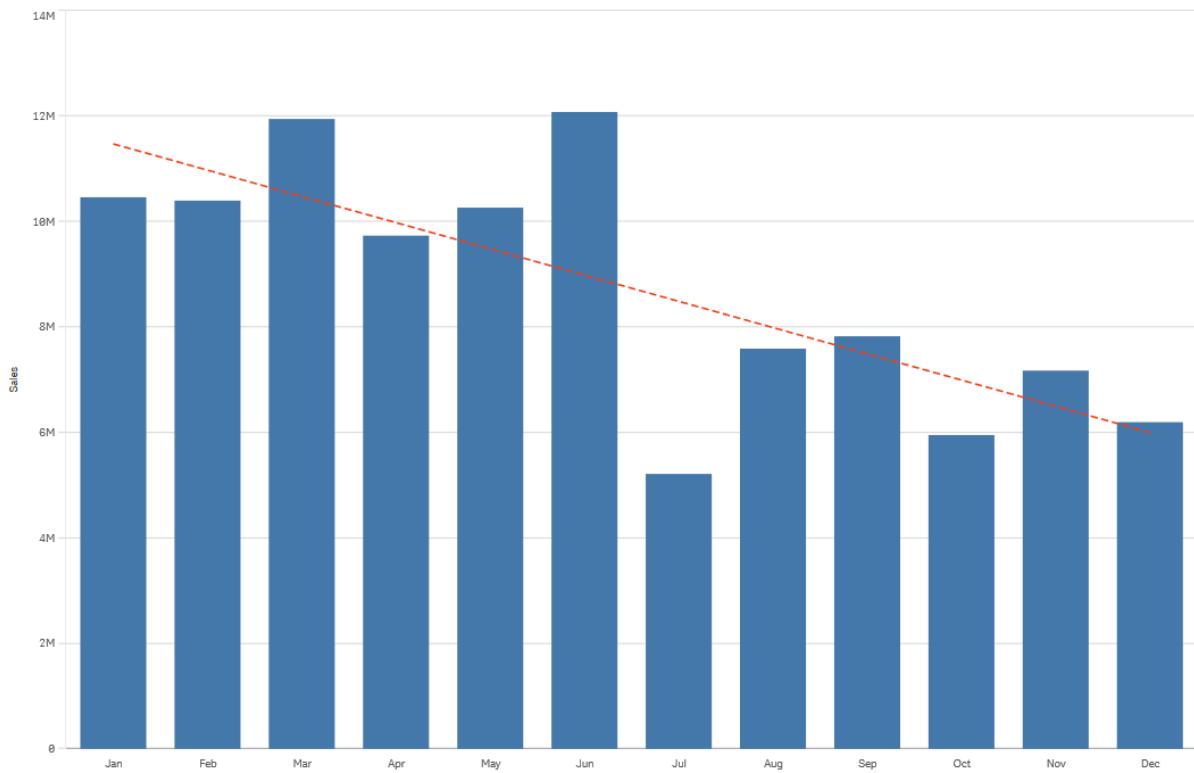
A bar chart showing sales per month. An average trend line is shown with a dashed red line.



Linear

A linear trend line shows increase or decrease of values at a steady rate. Linear trend lines are usually used with simple linear data sets.

A bar chart showing sales per month. A linear trend line is shown with a dashed red line.

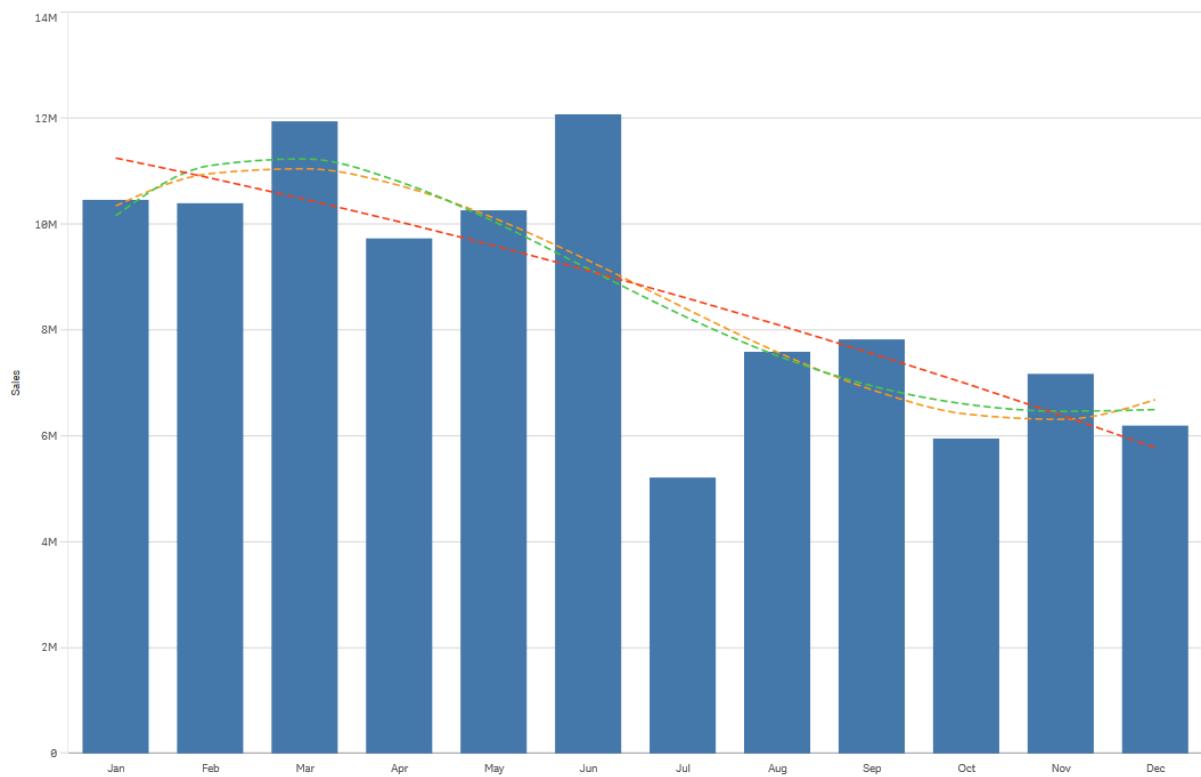


Polynomial (second, third, fourth degree)

A polynomial trend line is a curved line used on fluctuating data.

The number of data fluctuations can determine the order of the polynomial. A second degree polynomial trend line has one hill or valley, a third degree polynomial trend line has up to two hills or valleys, and a fourth degree polynomial has up to three hills or valleys.

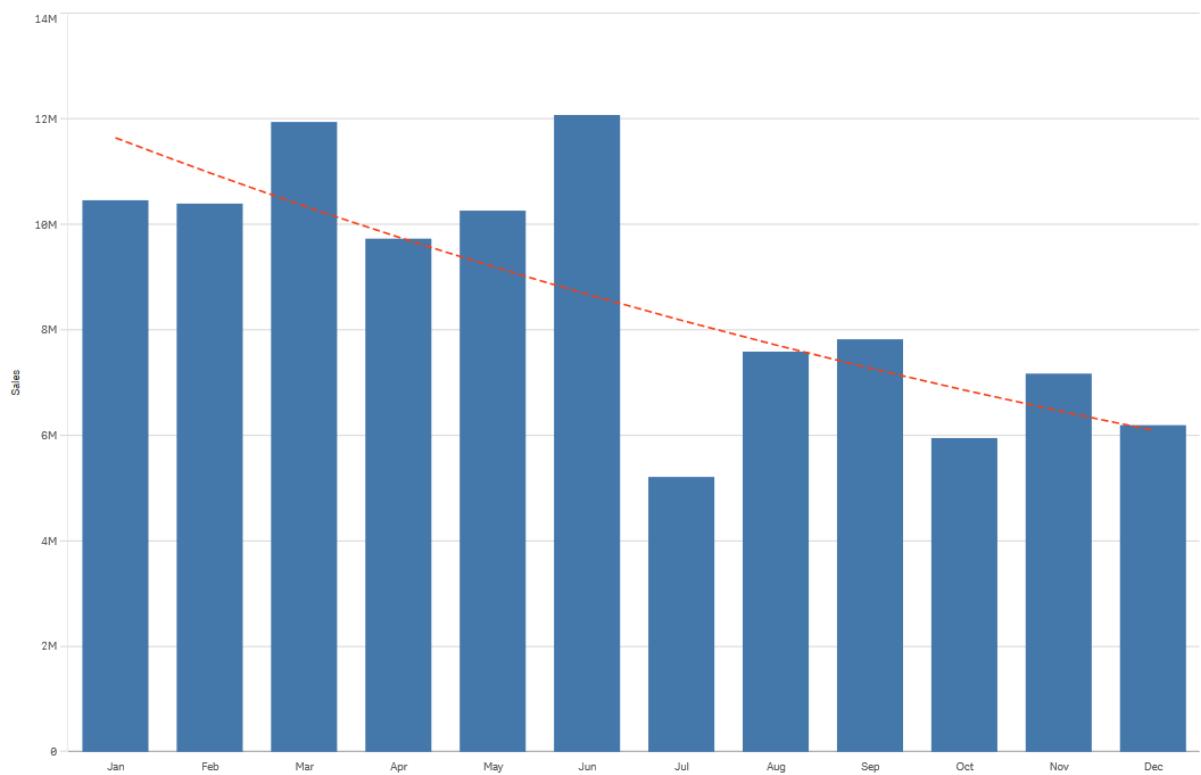
A bar chart showing sales per month. Polynomial trend lines of second, third, and fourth degree are shown with dashed red, yellow, and green lines respectively.



Exponential

An exponential trend line is a curved line used when data values rise or fall at increasingly higher rates.

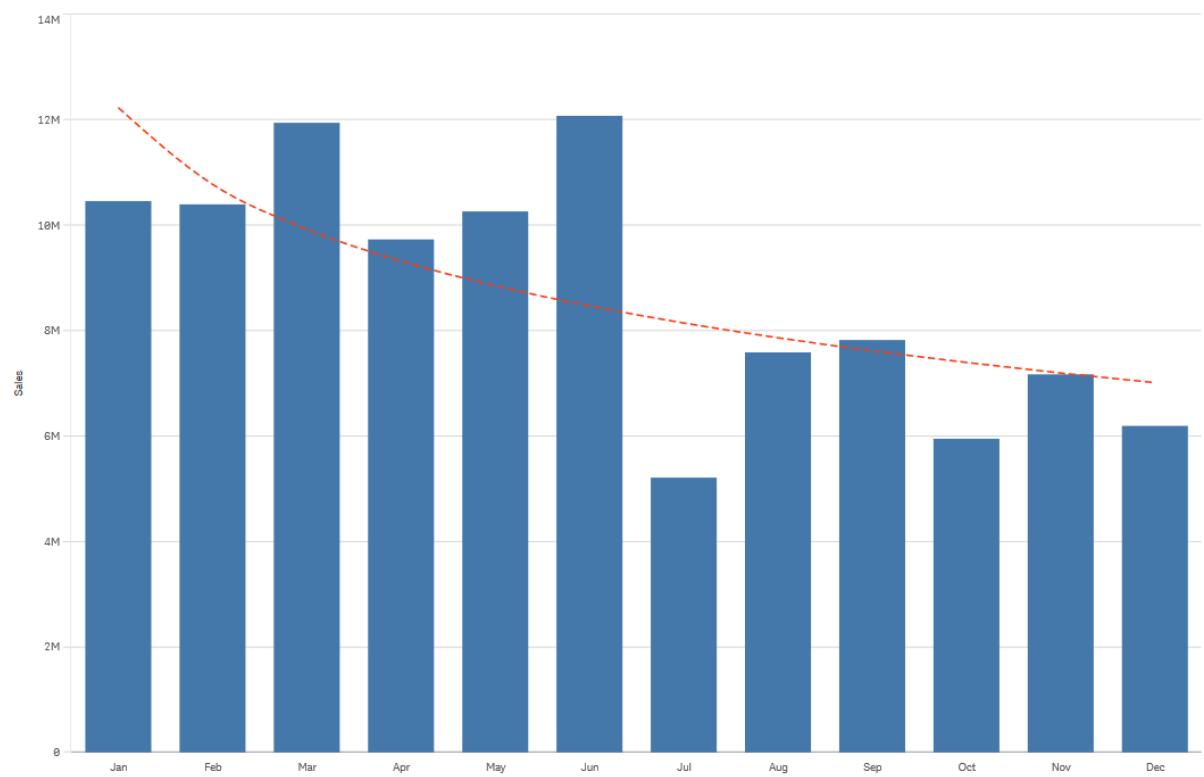
A bar chart showing sales per month. An exponential trend line is shown with a dashed red line.



Logarithmic

A logarithmic trend line is a curved line used when the rate of change in data increases or decreases quickly, before leveling out.

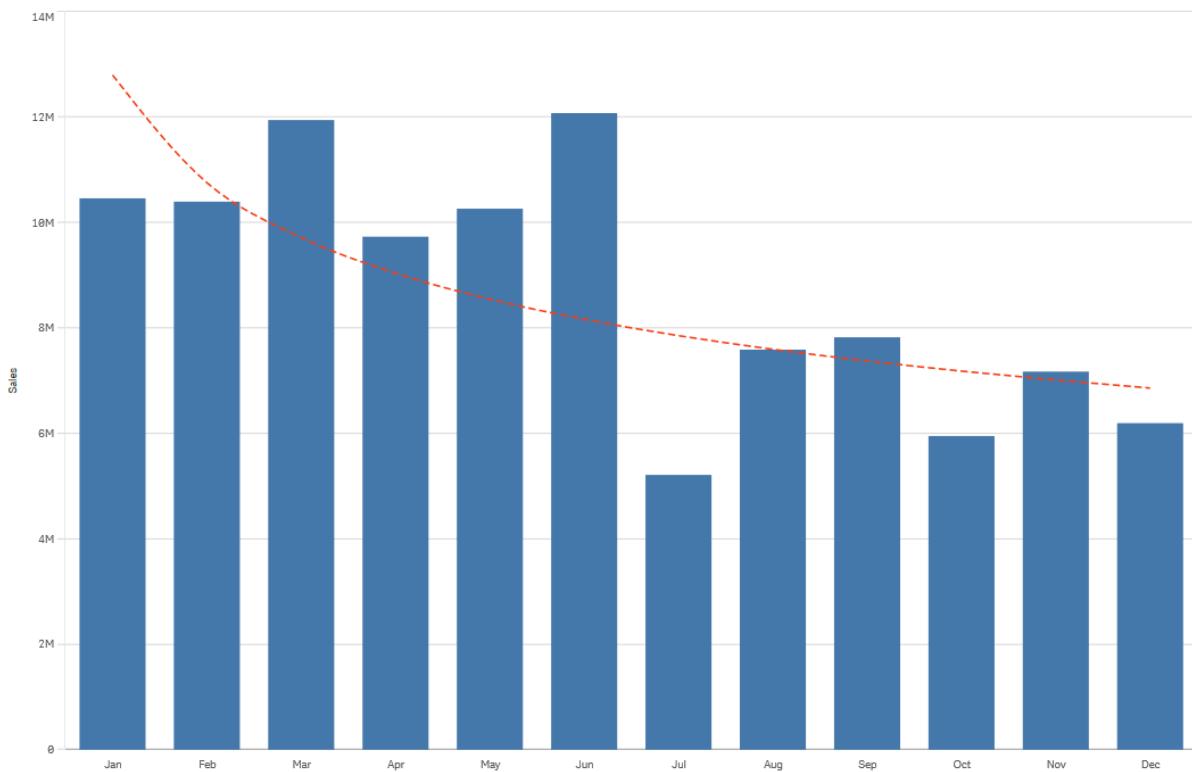
A bar chart showing sales per month. A logarithmic trend line is shown with a dashed red line.



Power

A power trend line is a curved line used with data sets that compare measurements that increase at specific rates.

A bar chart showing sales per month. A power trend line is shown with a dotted red line.



Reusing assets with master items

When you create and build your visualizations, you can save assets to reuse in other visualizations and on other sheets. You can save visualizations, dimensions and measures, as master items in the assets panel.

When your app gets published these master items will be available to others as ready-to-use visualizations, dimensions and measures.

Any updates you make to the master item are applied everywhere the master item is used. For example, you could use a master measure in as many of your visualizations as you like while only having to update it in a single instance to update all instances of the measure in your visualizations.

One of the purposes with creating and maintaining master items is for other users to explore their own ways and directions in the data, on top of what you have provided in the app as pre-made sheets with visualizations. The users will be able to create their own visualizations with your pre-made master dimensions and master measures, for example.

Reusing visualizations with a master visualization

You can create a master visualization to be able to reuse it. Users of a published app will have access to the master visualizations, but will not be able to modify them.



*You can only create master visualizations when you are working with an unpublished app. Visualizations with **Chart suggestions** enabled cannot be made into a master visualization.*

Do the following:

1. While editing a sheet, drag a visualization from the sheet to the master items.
If you have given the visualization a title, this is automatically added as the name of the master visualization.
2. Add a name, or change the name if you want to.
3. Type a description for the visualization (optional).
4. Add tags (optional).
5. Click **Add**.

The visualization is now saved to the master items tab.



*You can also add a visualization to the master items by right-clicking it on the sheet, and selecting **Add to master items**.*

Editing a master visualization

When you update a visualization in the master items, the changes will be reflected in all instances of the master visualization.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click to display the master items.
3. Click the visualization that you want to edit.
The preview opens.
4. Click at the bottom of the preview.
If the visualization is used on a sheet, a dialog is displayed to inform you that any changes to the master visualization will be applied to all its instances on the sheets.
5. Click **OK**.
The visualization opens for editing.



*You can also right-click the master item, and then click **Edit**.*

6. Make the changes you want, and click **Done** in the upper right corner of the visualization to finish editing.

The visualization is updated and reflected in all its instances.



*You can also edit a master visualization by selecting a linked visualization on a sheet, and clicking **Edit** in the properties panel. You must be in sheet edit mode.*

Reusing dimensions with master dimensions

When you are working with an unpublished app, you can create master dimensions so that they can be reused. Users of a published app will have access to the master dimensions, but will not be able to modify them.

You can create a master dimension in different ways.

Creating a master dimension from a field

When you are working with an unpublished app, you can create master dimensions so that they can be reused. You can create a master dimension from the **Fields** section of the assets panel.

Do the following:

1. Click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to select the fields tab.
3. Click the field you want to use to create a dimension.
The preview opens.
4. Click  at the bottom of the preview.
The **Create new dimensions** dialog opens with the field you selected. The field name is also used as the name of the dimension.
5. Click **Fields**.
6. Right-click the field you want to use as a dimension and click **Create dimension**.
The **Create new dimensions** dialog opens with the field you selected. The field name is also used as the name of the dimension.
7. Select if the dimension is to be single or drill-down.
8. Edit the name if you want to.
9. Type a description for the dimension (optional).
10. If you want to specify a color, click ▾ in the color drop down and select a color through one of the following methods:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field: #.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
11. Add tags (optional).
12. Click **Create**.
13. Click **Done** to close the dialog.

The dimension is now saved in the **Dimensions** category in the master items, and you can use it in visualizations.



You can quickly add several dimensions as master items by clicking **Add dimension** after adding each dimension. Click **Done** when you have finished.



Direct Discovery fields are indicated by in the **Fields** section of the assets panel.

Creating a master dimension from the assets panel

Do the following:

1. Click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click to select the master items tab.
3. Click the **Dimensions** heading to expand the category.
4. Click **Create new**.
The **Create new dimensions** dialog opens.
5. Select if the dimension is to be single or drill-down.
6. Click a field on the left-hand side to select it.
The name of the field is automatically added as the name of the dimension.
7. Change the name if you want to.
8. Type a description for the dimension (optional).
9. If you want to specify a color, click in the color drop down and select a color through one of the following methods:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field: #.
 - Click at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
10. Add tags (optional).
11. Click **Create**.
12. Click **Done** to close the dialog.

The dimension is now saved in the **Dimensions** category in the master items, and you can use it in visualizations.



You can quickly add several dimensions as master items by clicking **Add dimension** after adding each dimension. Click **Done** when you have finished.

Creating a drill-down dimension

When adding a dimension, you can select between creating a single or a drill-down dimension.

The following description explains how to create a drill-down group from the **Create new dimensions** dialog.



If selections cause the current drill-down dimension field to have only one possible value, the next field in the list is used instead.

Do the following:

1. Select **Drill-down** as dimension type.
2. Click at least two fields from the fields list on the left-hand side to insert them as the referenced fields.



You can filter which table to select fields from in the drop-down list.



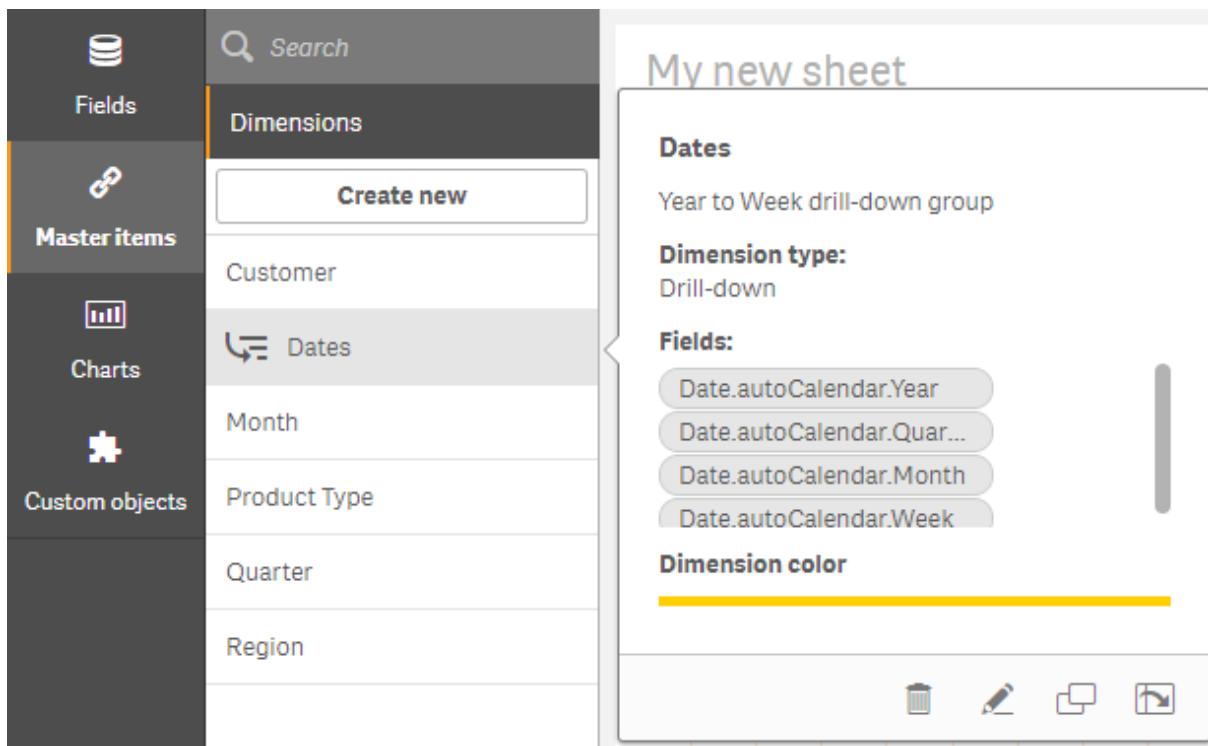
You can rearrange the order of the fields you have selected by dragging them to new positions in the list of selected fields.

3. Type a name for the dimension.
4. Type a description for the dimension (optional).
5. If you want to specify a color, click ▾ in the color drop down and select a color through one of the following methods:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field: #.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
6. Add tags (optional).
7. Click **Create**.
8. Click **Done** to close the dialog.

The drill-down dimension is now saved in the **Dimensions** category among the master items.

When you click the dimension in the panel on the left-hand side, the preview displays the dimension type and which fields are included in the drill-down dimension.

The preview displays the dimension type and which fields are included in the drill-down dimension.



Creating a calculated dimension

You can create a calculated dimension from the **Master items** tab in the assets panel. The expression editor opens from the **Create new dimensions** dialog.

Do the following:

1. Click the **Dimensions** heading on the **Master items** tab to expand the category.
2. Click **Create new**.
The **Create new dimensions** dialog opens.
3. Click **fx** in the **Field** text box to open the **Add expression** dialog.

You will now be able to add expressions in different ways, depending on your preferences and of different levels of complexity.



*You can also add an expression by typing directly into the **Field** text box, but you will then not be provided with syntax highlighting and syntax check.*

Using a common function

1. Select a field from the drop-down list.
2. Click the aggregation function you want to use.
3. Click **Insert** to insert the function and the field into the expression editor.



You can insert just a function or just a field by selecting only one of them.

4. Click **Apply** to close the **Add expression** dialog.
Continue by adding the descriptive data for the dimension.

Adding an expression by typing

1. Type the expression directly in the expression editor.
The field names you use in the expression are checked, and the syntax of the expression is validated.



As you type in the expression editor, the expression is validated continuously. If there is an error, you see a hint about what is incorrect. Additional error information may be available by clicking the icon next to the hint.

Each line in the expression editor is numbered and syntax highlighting is used.

2. Click **Apply** to close the **Add expression** dialog.
For a measure, you continue by adding the descriptive data for the measure.

Adding an expression through the properties panel

You can add an expression to a visualization through the properties panel.

1. Open the sheet with the visualization that you want to edit.
2. Click **Edit sheet** to open the properties panel. (If it is hidden, click **Show properties** in the lower right-hand corner to open it.)
3. Click the visualization that you want to edit.
The properties panel for that visualization is displayed on the right-hand side.
4. Under **Data**, click **Add data** and select **Dimension**.
A dimension text box is displayed.
5. Type your expression. The expression must begin with an equals sign (=), otherwise the string will be interpreted as text.

Detailed syntax help

You can open the online help with the full description of how to use the current function by double-clicking the function name in the expression editor and pressing Ctrl+H on the keyboard. This feature becomes available after having entered the first parenthesis of the expression after the function name.



This feature is supported only when using a computer with a keyboard.

Adding the descriptive data for the dimension

After having entered the expression, you need to add some descriptive information.

1. Type a name for the dimension.
2. Type a description for the dimension (optional).
3. Optionally, click ▼ in the color drop down and select a color through one of the following methods:

- Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field: #.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
4. Add tags (optional).
 5. Click **Create**.
 6. Click **Done** to close the dialog.

The calculated dimension is now saved in the **Dimensions** category of the **Master items**, and you can use it in visualizations.

Editing a master dimension

When you update a master dimension, the changes will be reflected in all its instances, including all visualizations that use it.

Edit a master dimension from the assets panel or while editing visualization properties.

Editing a master dimension from the assets panel

Use the assets panel to edit a master dimension when you want to preview the dimension or edit several master dimensions.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to display the master items.
3. Click the dimension that you want to edit.
The preview opens.
4. Click  at the bottom of the preview.
If the dimension is used on a sheet, a dialog is displayed to inform you that any changes to the master dimension will be applied to all its instances on the sheets.
5. Click **OK**.
The **Edit dimension** dialog opens, where you update the field or expression, name, description and tags.
6. Click **Save**.

The master dimension is now updated.



You can also right-click the master item, and then click **Edit**.

Editing a master dimension from visualization properties

When a dimension in a visualization is linked to a master dimension, you can edit the master dimension from the properties panel. All visualizations in the app that use the master dimension will be updated. To edit a linked dimension only in this visualization, without affecting the master item, you must first unlink it from the master. See *Unlinking from a master item (page 119)*.

Do the following:

1. In edit mode, select the visualization.
2. In the properties panel, click the **Data** tab.
If it is hidden, click **Show properties**  in the lower right-hand corner.
3. Under **Dimensions**, click the linked dimension to edit.
4. Under **Master item**, click **Edit**.
A dialog is displayed to inform you that any changes to the master dimension will be applied to all visualizations that use the dimension.
5. Make your changes to the master dimension, and then click **Save**.

Deleting a master dimension or master measure

You can delete dimensions and measures from the master items as long as the app is not published.



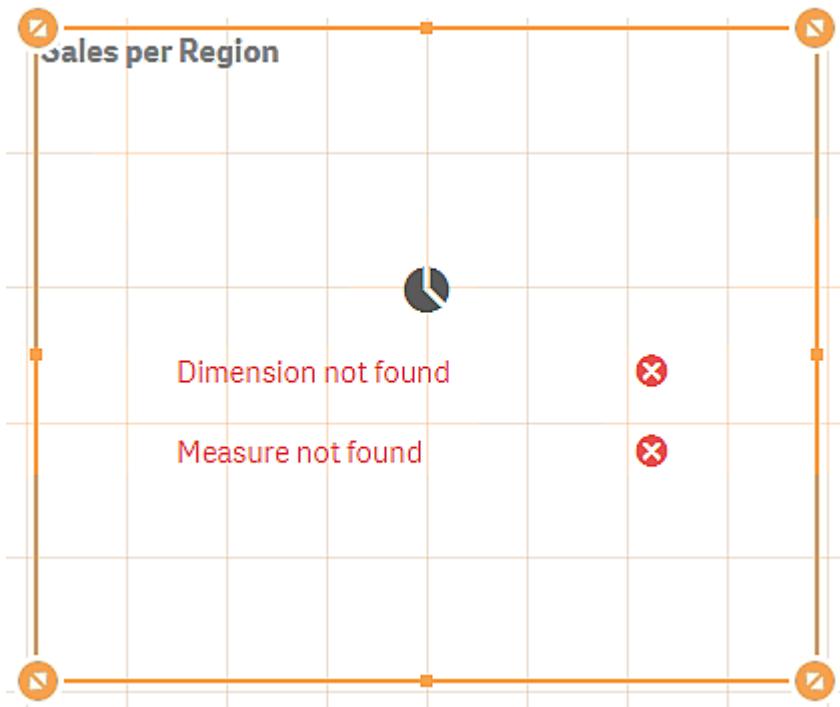
If you delete a master dimension or master measure, the visualizations that use the deleted master item will not work unless you replace it with a new dimension or measure.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to display the master items.
3. Click the dimension or measure that you want to delete.
The preview opens.
4. Click  at the bottom of the preview.
A dialog is displayed stating that all visualizations that use the dimension or measure will stop working.
5. Click **OK**.

The dimension or measure is deleted from the master items, and all visualizations on the sheets that used the deleted item do not work anymore. You see the text **Dimension not found** or **Measure not found** on those visualizations.

A visualization that is lacking both a dimension and a measure, both which have been deleted from the master items.



Replacing an invalid dimension or measure

When a dimension or measure has been deleted from the master items, all visualizations that reference to the deleted master item will not work anymore, until the missing dimension or measure is replaced.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side. Click to display the master items.
2. Drag a dimension or measure from the **Dimensions** or **Measures** sections to the visualization on the sheet.
The shortcut menu opens.
3. Select **Replace invalid dimension** or **Replace invalid measure**.

The visualization is complete and works again.

Reusing measures with master measures

When you are working with an unpublished app, you can create master measures so that they can be reused. Users of a published app will have access to the master measures, but will not be able to modify them.

A master measure is a combination of an expression and descriptive data, such as name, description and tags. You can create a master measure in different ways.

You can also create a master measure based on a measure created in a visualization. Under the measure in the properties panel of a visualization, click **Add new** under **Master items**.

Creating a master measure from a field

When you are working with an unpublished app, you can create master measures so that they can be reused. You can create a master measure from the **Fields** section of the assets panel.

When naming an entity, avoid assigning the same name to more than one field, variable, or measure. There is a strict order of precedence for resolving conflicts between entities with identical names. This order is reflected in any objects or contexts in which these entities are used. This order of precedence is as follows:

- Inside an aggregation, a field has precedence over a variable. Measure labels are not relevant in aggregations and are not prioritized.
- Outside an aggregation, a measure label has precedence over a variable, which in turn has precedence over a field name.
- Additionally, outside an aggregation, a measure can be re-used by referencing its label, unless the label is in fact a calculated one. In that situation, the measure drops in significance in order to reduce risk of self-reference, and in this case the name will always be interpreted first as a measure label, second as a field name, and third as a variable name.

1. Click  to select the fields tab.
2. Click the field you want to use to create a measure.
The preview opens.
3. Click  at the bottom of the preview.
The **Create new measure** dialog opens with the field you selected as the name of the measure and as a part of the expression.
4. Click **Fields**.
5. Right-click the field you want to use as a measure and click **Create measure**.
The **Create new measure** dialog opens with the field you selected. The field name is also used as the name of the measure.
6. Click  in the **Expression** field to open the **Edit expression** dialog.
7. Type the expression directly in the expression editor (the main window).
The field names you use in the expression are checked, and the syntax of the expression is validated.



As you type in the expression editor, the expression is validated continuously. If there is an error, you see a hint about what is incorrect in the lower left-hand corner. Additional error information may be available by clicking the icon next to the hint.

Each line in the expression editor is numbered and syntax highlighting is used.



You can open the online help with the full description of how to use the current function by double-clicking the function name in the expression editor and pressing Ctrl+H on the keyboard. This feature becomes available after having entered the first parenthesis of the expression after the function name, and only when using a computer with a keyboard.

8. Click **Apply** to close the **Add expression** dialog.
Now you need to add some descriptive data for the measure.

9. Edit the name if you want to.
10. Type a description for the measure (optional).
11. If you want to specify a color, click ▾ in the color drop down and select a color through one of the following methods:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
12. Add tags (optional).
13. Under **Number formatting**, you can choose:
 - Auto
 - Number
 - Money
 - Date
 - Duration
 - Custom
 - Measure expression
14. Click **Create**.

The measure is now saved in the **Measures** category in the master items, and you can use it in visualizations.



*Direct Discovery fields are indicated by  in the **Fields** section of the assets panel.*

Creating a master measure with a common aggregation function

When you are working with an unpublished app, you can create master measures so that they can be reused. You can easily create a measure using one of the most common aggregation functions by selecting the function and the field from drop-down lists.

When naming an entity, avoid assigning the same name to more than one field, variable, or measure. There is a strict order of precedence for resolving conflicts between entities with identical names. This order is reflected in any objects or contexts in which these entities are used. This order of precedence is as follows:

- Inside an aggregation, a field has precedence over a variable. Measure labels are not relevant in aggregations and are not prioritized.
- Outside an aggregation, a measure label has precedence over a variable, which in turn has precedence over a field name.
- Additionally, outside an aggregation, a measure can be re-used by referencing its label, unless the label is in fact a calculated one. In that situation, the measure drops in significance in order to reduce risk of self-reference, and in this case the name will always be interpreted first as a measure label, second as a field name, and third as a variable name.

Do the following:

1. Click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
 2. Click  to select the master items tab.
 3. Click the **Measures** heading to expand that category.
 4. Click **Create new**.
The **Create new measure** dialog opens.
 5. Click  in the **Expression** field to open the **Add expression** dialog.
You find drop-down lists for selecting a field and a common function on the right-hand side.
 6. If you want to show fields from a particular table, select this table in the top drop-down list (optional).
 7. Select a field from the **Field** drop-down list.
 8. Select a function from the bottom drop-down list.
-  *You can insert just a field by not selecting a function.*
9. Click **Insert** to insert the field and the function into the expression editor.
-  *You can open the online help with the full description of how to use the current function by double-clicking the function name in the expression editor and pressing Ctrl+H on the keyboard. This feature becomes available after having entered the first parenthesis of the expression after the function name, and only when using a computer with a keyboard.*
10. Click **Apply** to close the **Add expression** dialog.
Now you need to add some descriptive data for the measure.
 11. Type a name for the measure.
 12. Type a description for the measure (optional).
 13. If you want to specify a color, click ▾ in the color drop down and select a color through one of the following methods:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
 14. Add tags (optional).
 15. Under **Number formatting**, you can choose:
 - Auto
 - Number
 - Money
 - Date

- Duration
- Custom
- Measure expression

16. Click **Create**.

The measure is now saved in the **Measures** category in the master items, and you can use it in visualizations.

Creating a master measure by typing the expression

When you are working with an unpublished app, you can create master measures so that they can be reused. You can add complex expressions by typing the expression into the expression editor.

1. Click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to select the master items tab.
3. Click the **Measures** heading to expand that category.
4. Click **Create new**.
The **Create new measure** dialog opens.
5. Click  in the **Expression** field to open the **Add expression** dialog.
6. Type the expression directly in the expression editor (the main window).
The field names you use in the expression are checked, and the syntax of the expression is validated.



As you type in the expression editor, the expression is validated continuously. If there is an error, you see a hint about what is incorrect in the lower left-hand corner. Additional error information may be available by clicking the icon next to the hint.

Each line in the expression editor is numbered and syntax highlighting is used.



You can open the online help with the full description of how to use the current function by double-clicking the function name in the expression editor and pressing Ctrl+H on the keyboard. This feature becomes available after having entered the first parenthesis of the expression after the function name, and only when using a computer with a keyboard.

7. Click **Apply** to close the **Add expression** dialog.
Now you need to add some descriptive data for the measure.
8. Type a name for the measure.
9. Type a description for the measure (optional).
10. If you want to specify a color, click ▾ in the color drop down and select a color through one of the following methods:

- Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field.
 - Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
11. Add tags (optional).
 12. Under **Number formatting**, you can choose:
 - Auto
 - Number
 - Money
 - Date
 - Duration
 - Custom
 - Measure expression
 13. Click **Create**.

The measure is now saved in the **Measures** category in the master items, and you can use it in visualizations.

Editing a master measure

When you update a master measure, the changes will be reflected in all its instances, including all visualizations that use it.

Edit a master measure from the assets panel or while editing visualization properties.

Editing a master measure from the assets panel

Use the assets panel to edit a master measure when you want to preview the measure or edit several master measures.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to select the master items tab.
3. Click **Master items**.
4. Click the measure that you want to edit.
The preview opens.
5. Click  at the bottom of the preview.
A dialog is displayed to inform you that any changes to the master measure will be applied to all visualizations that use the measure.
6. Click **OK**.
The **Edit measure** dialog opens, where you update the expression, name, description, color, and tags.
7. Click **Save**.

The measure is now updated.



You can also right-click the master item, and then click **Edit**.

Editing a master measure from visualization properties

When a measure in a visualization is linked to a master measure, you can edit the master measure from the properties panel. All visualizations in the app that use the master measure will be updated. To edit a linked measure only in this visualization, without affecting the master item, you must first unlink it from the master. See *Unlinking from a master item (page 119)*.

Do the following:

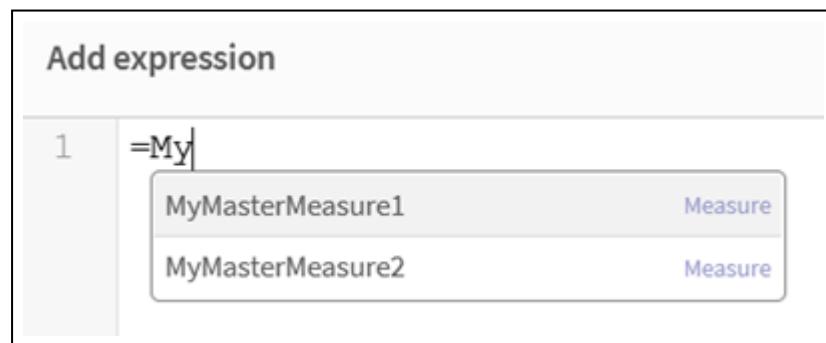
1. In edit mode, select the visualization.
2. In the properties panel, click the **Data** tab.
If it is hidden, click **Show properties**  in the lower right-hand corner.
3. Under **Measures**, click the linked measure to edit.
4. Under **Master item**, click **Edit**.
A dialog is displayed to inform you that any changes to the master measure will be applied to all visualizations that use the measure.
5. Make your changes to the master measure, and then click **Save**.

Using master measures in expressions

You can use master measures in expressions. You can use the master measure by itself, or use it to build a more complex expression.

When you type in the **Expression editor**, an auto-complete list of matching master measures opens.

Master measure in Expression editor



The screenshot shows the 'Add expression' dialog in the Expression editor. A user has typed '=My' into the input field. An auto-complete dropdown menu is open, listing two items: 'MyMasterMeasure1' and 'MyMasterMeasure2', each followed by a 'Measure' button. The background of the dialog is light gray, and the dropdown menu has a white background with a thin gray border.



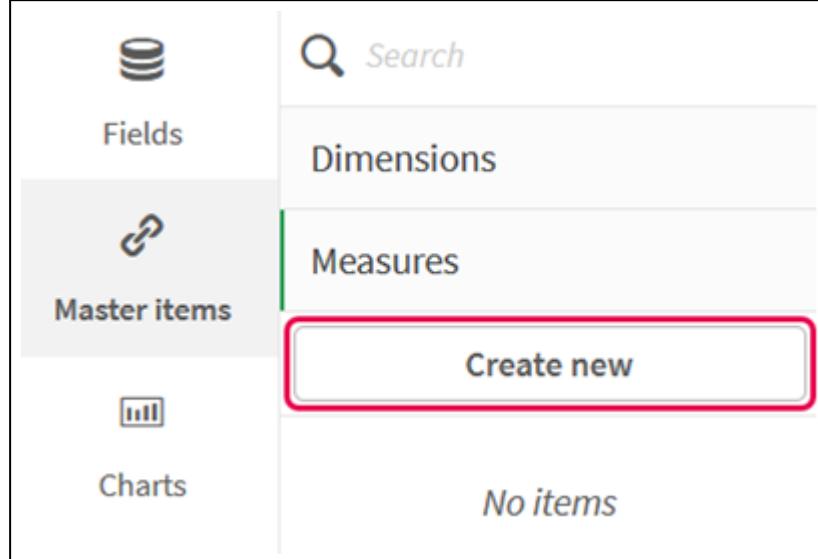
If the master measure is renamed or deleted, references in expressions are not automatically updated. The old reference will return NULL in the expression as there is no measure by that name.

Example: Using a master measure in an expression

This example creates a master measure, and then uses the master measure in a chart expression.

1. Create the master measure.
 - i. Click **Master items** in the assets panel.
 - ii. Under **Measures**, click **Create new**.

Create new measure button



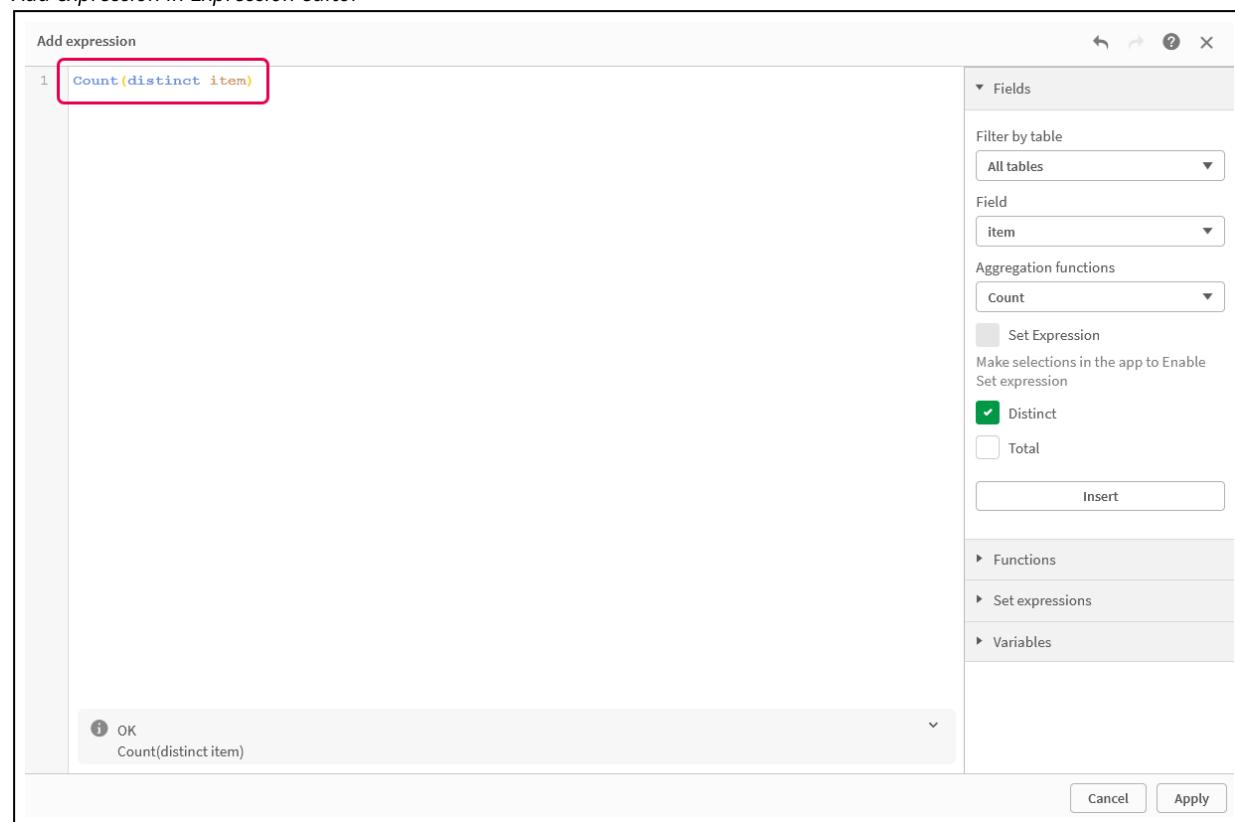
- iii. In the **Create new measure** dialog box, enter a **Name** for the master measure, and then click **fx** to open the Expression editor.

Create new measure dialog box

The screenshot shows the 'Create new measure' dialog box. The 'Create new measure' tab is active. The 'Name' field contains 'MyMasterMeasure1'. The 'Expression' field has an 'fx' button highlighted with a red box. The 'Label expression' and 'Number formatting' fields are also visible. At the bottom are 'Cancel' and 'Create' buttons.

- iv. In the Expression editor, create the expression.
You can type the expression, or use the options available in the expression builder panel.
Insert the expression using the expression builder *Count(distinct item)*, and then click **Apply**.

Add expression in Expression editor



- v. In the **Create new measure** dialog box, click **Create**.

Create measure

Create new measure

Segment colors

Name
MyMasterMeasure1

Description

Measure color

Tags

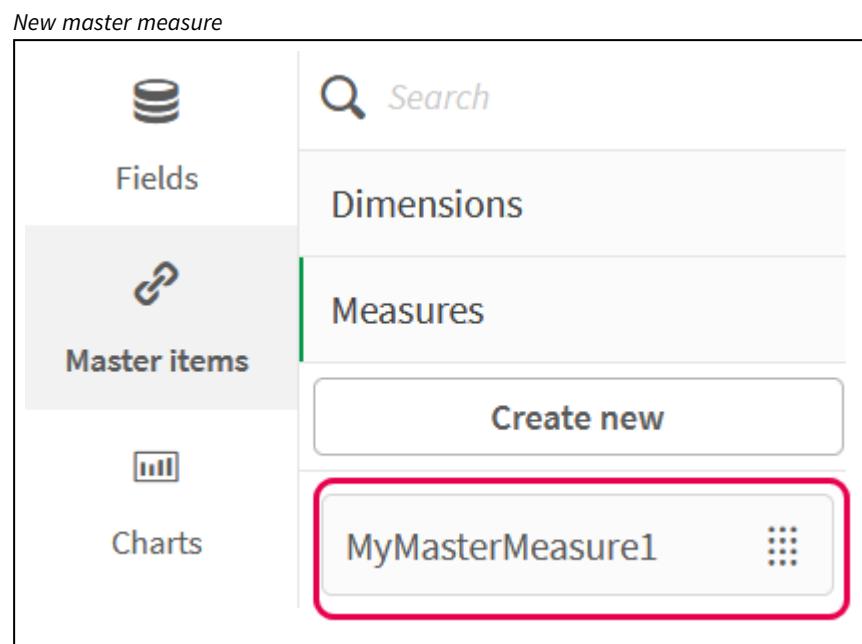
Expression
Count(distinct item)

Label expression

Number formatting
Auto

Cancel Create

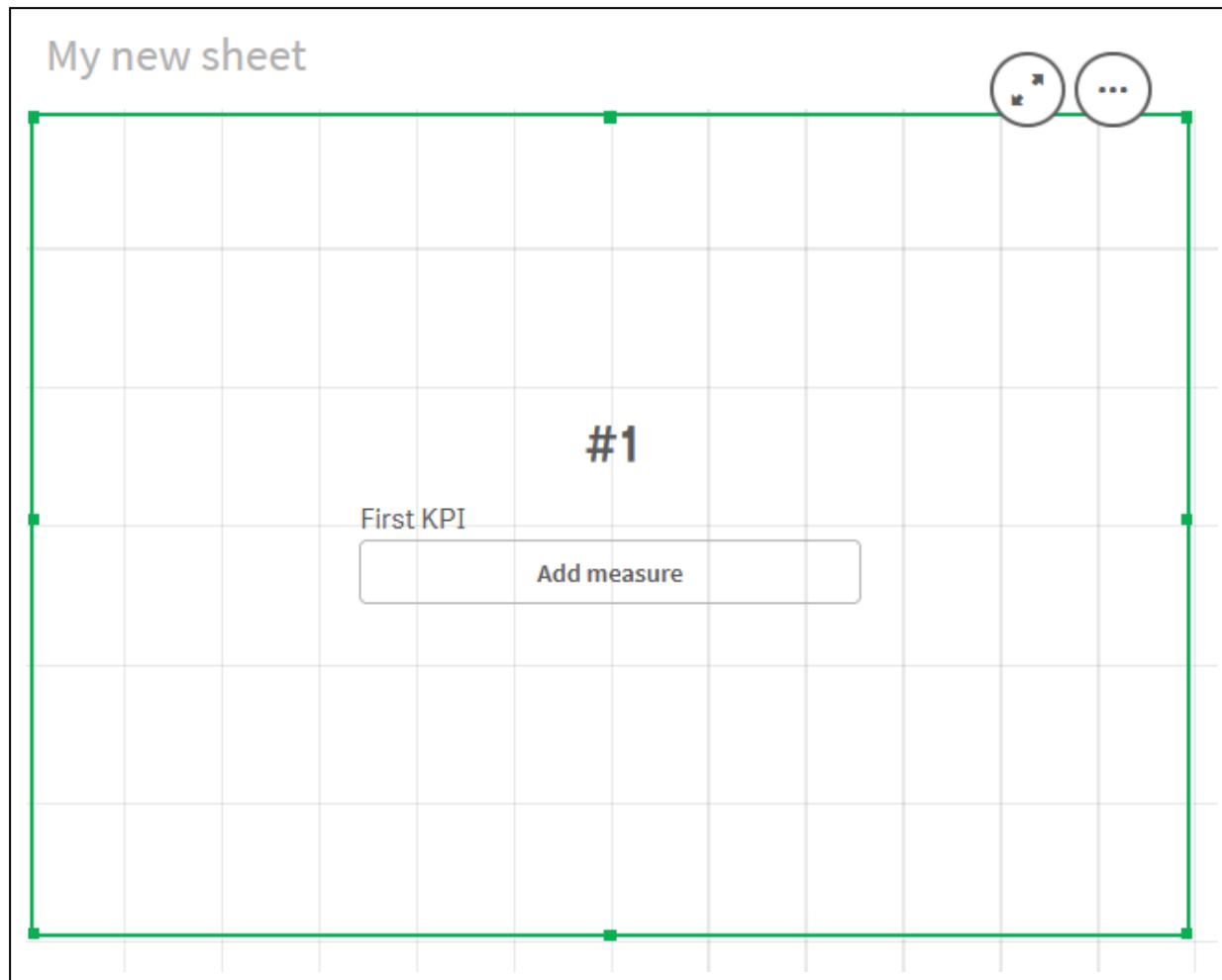
The new master measure is created.



2. Use the master measure in a chart expression.

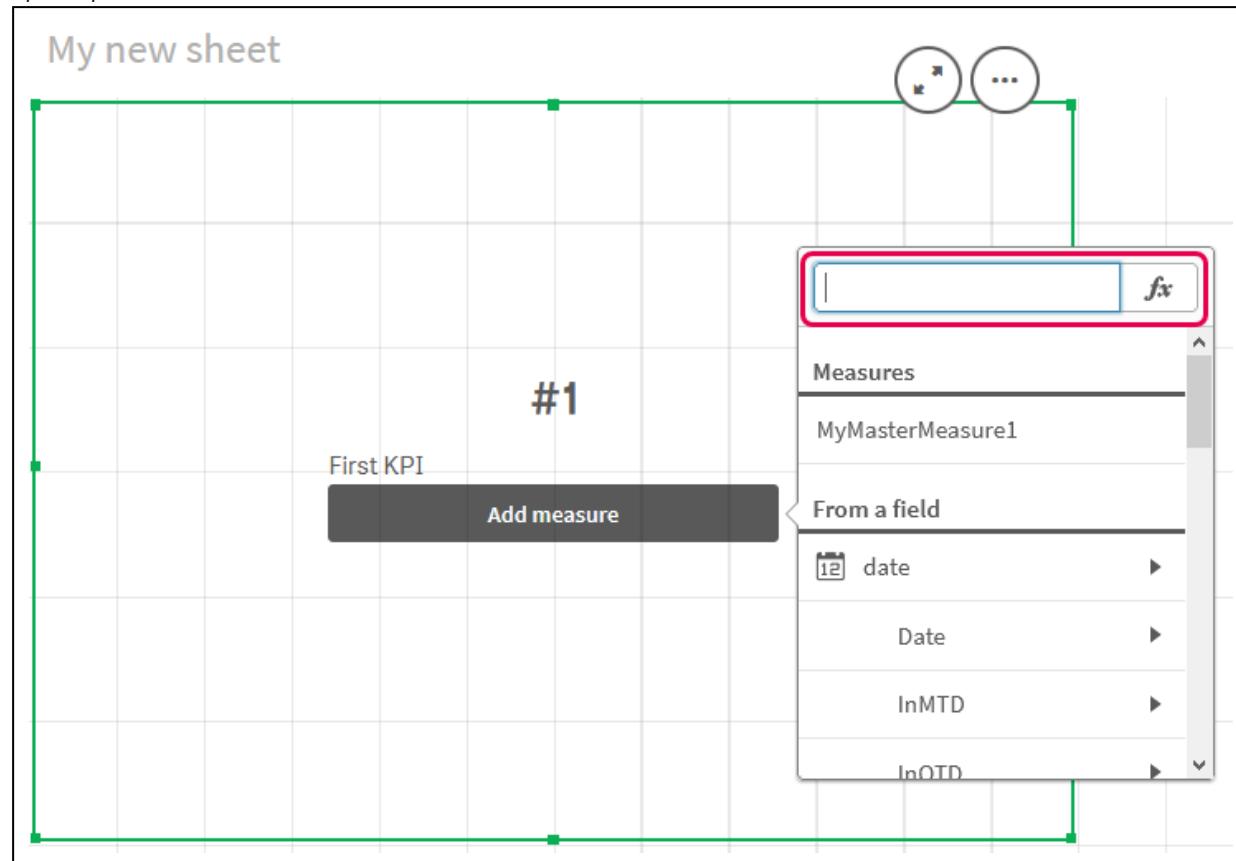
- i. Add a **KPI** chart to a sheet in an app.

New KPI



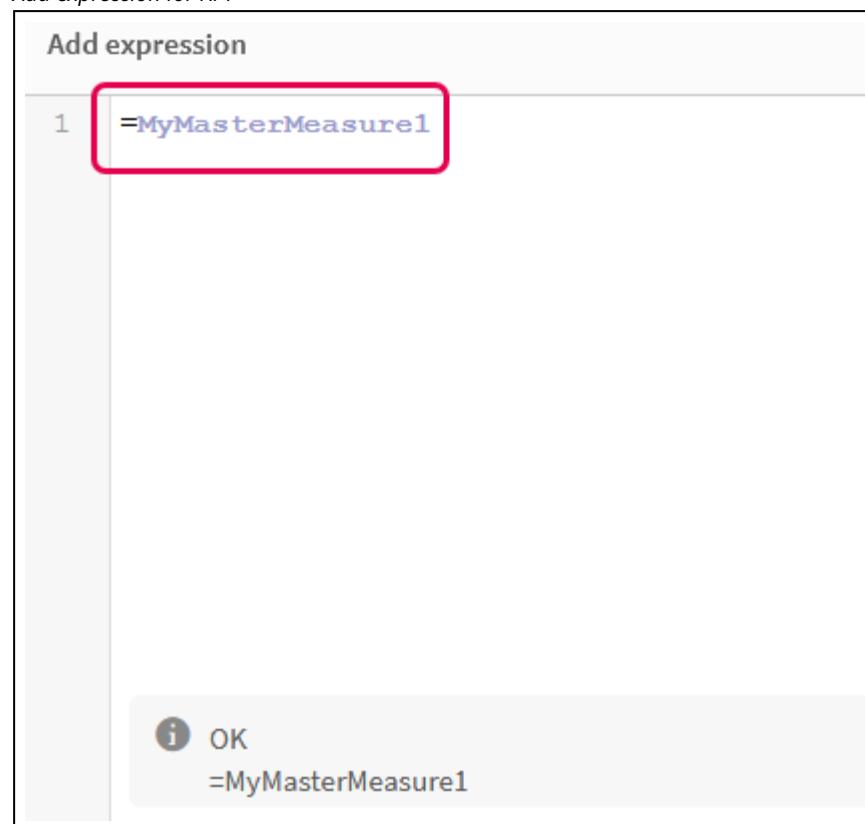
- ii. Click **Add measure**, and then click **fx** to open the Expression editor for the **KPI**.

Open Expression editor



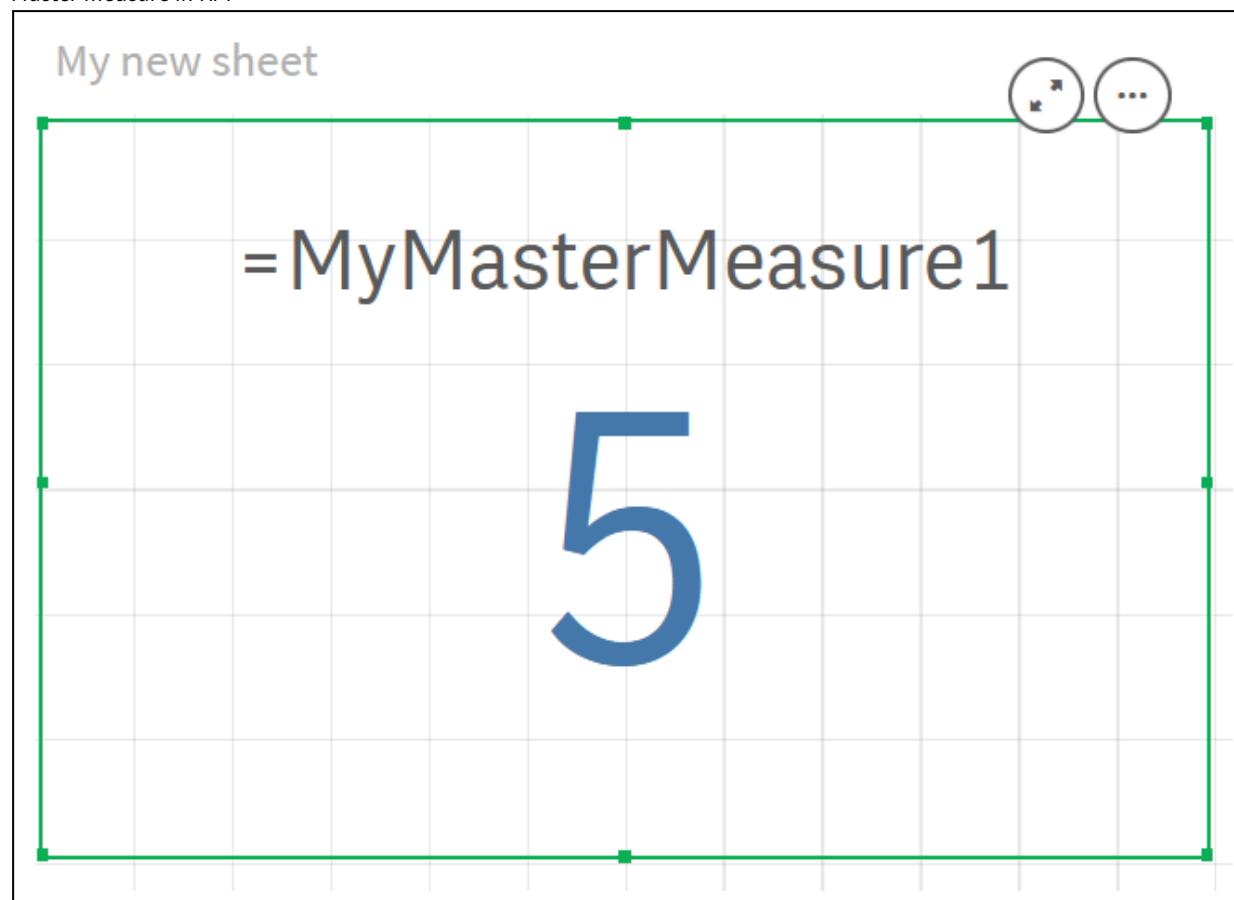
- iii. Type the master measure into the expression. As you type, you will get an auto-complete list of any available master measures. If required, the master measure can be used to build more complex expressions in the Expression editor.

Add expression for KPI



- iv. Close the Expression editor to view the **KPI**. You can change properties for the KPI, including the label, in the properties panel.

Master measure in KPI



Assigning colors to master items

You can assign colors to your master items. Colors assigned to master dimensions and master measures persist across all instances of those master items in all visualizations.

If you change the color used for the master item, the color will be updated across all instances of that master item. Optionally, master item colors can be disabled for individual visualizations.

Visualizations use master dimension colors when **Single color** is selected in the **Colors and legend** section of the visualization properties panel. Master measure colors are used when **Single color** or **Multicolored** are selected in the **Colors and legend** section of the visualization properties panel.

By default, if a visualization's default **Auto** settings use the **Single** or **Multicolored**, master item colors will be applied automatically. If they do not, you must switch to **Custom** and select a supported setting. Master item colors can be disabled in individual visualizations.

In a visualization with colors specified for both master dimension and master measure, Qlik Sense defaults to the master dimension color. Master measure colors can be applied using the **Use library colors** drop-down and selecting **Measure** in the **Colors and legend** section of the visualization properties panel. In visualization with a mix of master measures with assigned colors and measures without assigned colors, the other measures will use the default palette colors.

Master dimensions can have colors assigned to their distinct values.

Master measures can have custom color gradients applied to their values in visualizations. .

When assigning colors to your master items, you have the following options:

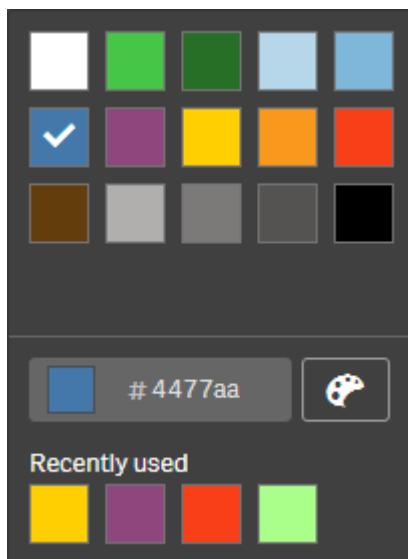
- Choose a color from the default palette
- Type a hexadecimal color code
- Choose a color using advanced color options

Assigning a color from the default color palette

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click  to display the master items.
3. Select a master item from your library.
4. Click .
5. Click ▼ in the color drop-down.
6. Select one of the colors in the palette.
7. Click outside the dialog.
8. Click **Save**.

The color dialog with the default color palette and a blue color selected.



Assigning a color using a hexadecimal color code

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click  to display the master items.
3. Select a master item from your library.
4. Click .

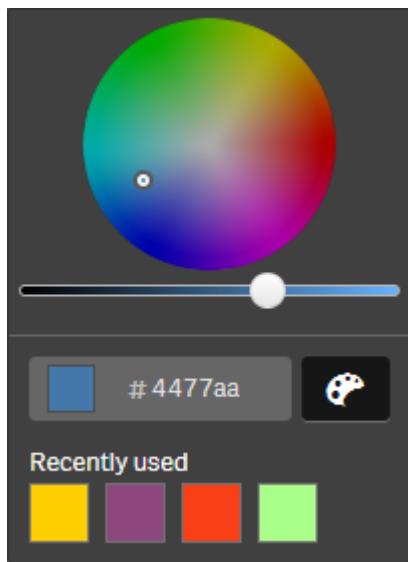
5. Click ▼ in the color drop-down.
6. Type a 6 character color code in the input field: #.
7. Click outside the dialog or press Enter.
8. Click **Save**.

Assigning a color using advanced color options

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click  to display the master items.
3. Select a master item from your library.
4. Click .
5. Click ▼ in the color drop-down.
6. Click  at the bottom of the dialog.
The dialog displays the advanced options.
7. Select a color in the color wheel.
The selected color changes and the Hex color code updates accordingly.
8. Optionally, adjust the color saturation using the slider.
The saturation changes and the Hex color code updates accordingly.
9. Click outside the dialog.
10. Click **Save**.

The color dialog with the color wheel in the advanced options and a blue color selected.



Assigning colors to master dimension values

You can assign colors to the distinct values contained in a master dimension. This ensures that the distinct values of your dimensions use the same colors in all visualizations.

Colors assigned to values are used when you choose to color **By dimension** and have **Library colors** enabled. If you choose to color by Single color, the master dimension's color is used instead.

Assigning colors to master dimension values has the following limitations:

- A maximum of 100 colors can be assigned to a dimension's values.
- Dimension values cannot be longer than 1024 characters. Longer values are disabled in the list of values.
- Drill-down dimensions do not support assigning colors to dimension values



Do not assign colors to master dimension values if you use section access or work with sensitive data, because the values might be exposed by the color configuration.

The **Value colors** section of **Edit dimensions** contains the options for assigning colors to a dimension's distinct values. You can search the list of values with

Qlik Sense automatically applies an auto fill to your values to provide a default color. The auto fill is either a single color or a color scheme. Changing your auto-fill settings will not change colors you have assigned to values.

In addition to distinct values, you can set colors for the values classified as others in a visualization, that is, the collection of distinct values that fall outside the displayable values in a dimension. You can also set colors for null values.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
2. Click to display the master items.
3. Select a master dimension from your library.
4. Click
5. Click **Value colors**.
6. If you want to change the auto fill settings, set **Auto fill** to **Custom**, select the auto fill method, and either select a single color or color scheme.
7. Select a value and do one of the following:
 - Select a color in the color wheel, and optionally adjust the saturation slider.
 - Type a 6 character color code in the Hex input field: #.

Values assigned colors display the

8. If you want to remove an assigned color, select the value and click
9. Optionally, to assign a color to **Others** or **Null values**, click and do one of the following:
 - Click one of the colors in the palette.
 - Type a 6 character color code in the Hex input field: #.

- Click  at the bottom of the dialog, select a color in the color wheel, and optionally adjust the saturation slider.
10. Click **Save**.

Assigning colors to master measure values

You can assign a color gradient or segment scheme to a master measure, enabling you to color visualizations by measure, using colors other than the default color schemes.

If you choose to color by single color, the master measure's color is used instead of the color scheme.

To assign custom segment or gradient colors to the values in a master measure, do the following:

1. Access and enable segment colors.
2. Select the template scheme.
3. Select the segment format.
4. Optionally, add or remove limits.
5. Optionally, edit your segment limits.
6. Assign colors to your segments.
7. Save.

Accessing and enabling segment colors

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click  to display the master items.
3. Select a master measure from your library.
4. Click .
5. Click on the **Segment colors** tab.
6. Set the **Segment colors** button to **Custom** coloring.

Selecting the template scheme

The template scheme provides a default template to edit. You can choose from the default classes and gradients.

Do the following:

- Under **Select a template scheme**, select a template scheme.

Selecting the segment format

The segment format determines how colors will be applied. **Percentage** adds value classes based on where values fit into the percentile of their value. **Fixed value** uses defined values to set the limits of each segment.

Do the following:

- Set **Segment format** to **Fixed value** or **Percentage**.

Adding, editing and removing limits

By default, the number of segments depends on the scheme selected. You can create additional segments by adding limits. When a new segment is created a pointer marks the value of its limit.

Do the following:

- Click on the **Add limit** button to add a limit to the gauge.
- Use the slider to change the value of a limit. You can also type a value for the selected limit.
- Click on the **Remove limit** button to remove the selected limit.

Editing segments

By default, segment colors depend on the scheme selected. You can change the color of a segment and its ending.

Do the following:

- Click the segment to change colors.
- Select **Gradient** to use different shades of colors in the transition between segments for the selected limit.

Create calendar measures

To analyze data over relative time ranges, use calendar measures. For example, you might want to compare current year-to-date sales figures with figures from the same period the previous year.

Calendar measures aggregate data from a field over a time range, and are saved in the **Measures** category in the master items. Calendar measures comprise a field to be aggregated, an aggregation, a date field, and a time range for that date field that sets which data is included in the aggregation. You create calendar measures under **Fields** in the **Assets** panel, using the **Create calendar measures** dialog.

You can aggregate fields from tables loaded in **Data manager** or from a script in **Data load editor**, as long as the field is in the same table as the date field, or is in an associated table.

However, the date field must be from a table that has been loaded using **Data manager**, because calendar measures use expressions tagged as date & time fields that are declared in autoCalendar, and date fields are only mapped to autoCalendar when loaded in **Data manager**. Calendar measures support the following aggregations with the aggregated field: Sum, Count, Avg, Min, and Max.



*Calendar measures do not support calendars created using the **Data load editor**. If you use calendars created using **Data load editor** and want to create calendar measures, you must load a table containing a date field using **Data manager** for use with your tables loaded using **Data load editor**.*



If your date field is subject to more than one calendar and both calendars are qualified for use with calendar measures, then the first calendar loaded in the data load script is used in the calendar measures.

A calendar measure can use one of the following time ranges are available for use with calendar measures: weekly, monthly, quarterly, and yearly. Within each time range, different measures exist for periods such as current month, year-to-date, and current week last year. The following time ranges and measures are available for creating calendar measures:

Calendar measures for time range Yearly

Measure	Description
YTD	The year to date for all years.
YTD Current Year	The year to date for the current year.
YTD Last Year	The year to date for last year.

Calendar measures for time range Monthly

Measure	Description
MTD	The month to date for all months and years.
MTD Current Month	The month to date for the current month.
MTD Last Month	The month to date for last month.
Current Month	All dates this month.
Current Month Last Year	All dates this month last year.
Last Month	All dates last month.

Calendar measures for time range Quarterly

Measure	Description
QTD	The quarter to date for all years.
QTD Current Quarter	The quarter to date for the current quarter.
QTD Last Quarter	The quarter to date for the last quarter.
Current Quarter	All dates in the current quarter.
Current Quarter Last Year	All dates in the current quarter last year.
Last Quarter	All days for the last quarter.

Calendar measures for time range Weekly

Measure	Description
WTD	The week to date for all weeks across all years.
WTD Current Week	This week to date for the current week.
WTD Last Week	The week to date for the last week.
Current Week	All dates this week.
Current Week Last Year	All dates this week last year.
Last Week	All dates last week.

Once created, calendar measures are treated identically to master measures. That is, calendar measures are reusable and editable while an app is unpublished. Users of a published app will have access to the calendar measures, but will not be able to modify them.

Creating a calendar measure from a field

On a sheet in edit mode with the **Fields** tab open, do the following:

1. Right-click a field and select **Create calendar measures**.
By default, the field you right-clicked will be included as the **Aggregated field**.
2. Select a date field from the **Date field** drop-down list.
3. Select a field from the **Aggregated field** drop-down list.
4. Select an aggregation from the **Aggregation** drop-down list.
5. Select a time range from the **Time range** drop-down list.
A list of the available measures displays under **Preview of measures**.
You can toggle the display of the measures' expressions by selecting the **Preview of measures** switch.
6. Select the calendar measures to add to your master items.
By default, all measures are selected.
7. Click **Save to master items**.
8. Click **Close**.

Tagging master items

You can use tags to organize master items. You will find matches in tags when searching in the assets panel. You can also tag master items with synonyms for Insight Advisor. Use the format *alt:<term>* in synonym tags. For example, *alt:cities*.

Each tag can contain a maximum of 31 characters, and each master item can have up to 30 tags.

Adding tags to a master item

You can add tags when creating or editing a master item.

Adding tags while creating a new measure.

Edit measure

Expression:

Sum([City Code])fx

Name:

Sales

Description:

Measure color

█ ▾

Tags:

MyTag +

Invoicing × Sales ×

Cancel Save

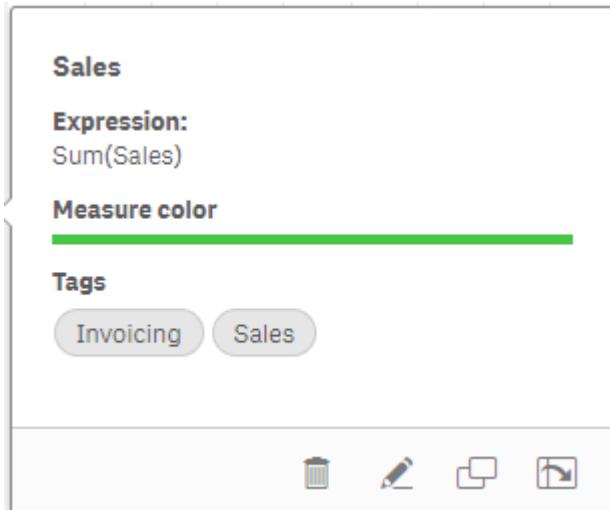


You can add tags by clicking + or by pressing Enter.

Previewing tags

In the assets panel, tags are displayed when previewing dimensions, measures and visualizations.

The preview displays the dimension type, which fields are included in the drill-down dimension and tags.



Unlinking from a master item

Unlink a dimension or measure in a visualization from its master item to edit only the current instance of the dimension or measure. Unlinking removes the connection to the master item, and the dimension or measure becomes independent. If you want to update all instances of a linked dimension or linked measure, you must edit the master item. For more information, see *Editing a master dimension (page 93)* or *Editing a master measure (page 100)*.

Unlinking from the properties panel

Unlink from a master dimension or master measure in a visualization using the properties panel.

Do the following:

1. Click **Edit sheet**.
The properties panel opens on the right-hand side.
2. Select the visualization on the sheet.
The properties for the selected visualization are displayed in the properties panel.
3. Under **Data**, locate and click the dimension or measure.
A linked dimension or measure is indicated by the symbol .
4. Click .
A dialog is displayed that you are about to unlink from a master item.
5. Click **OK**.

You can now edit the dimension or measure in the visualization without affecting any visualizations that use the master items.



You can only unlink from a single dimension, not a drill-down dimension.

Unlinking from a master visualization

Unlink a visualization from its master visualization to edit only the current instance. Unlinking removes the connection between the visualization on your sheet and the master item. The current visualization becomes independent. To edit all visualizations that are linked to a master visualization, see *Editing a master visualization (page 87)*.

Do the following:

1. Click  **Edit sheet**.
2. Click  in the top right corner of a linked visualization, and then click **Unlink visualization**.
A dialog is displayed that you are about to unlink from a master visualization.
3. Click **OK**.

The visualization on the sheet can now be edited and the changes will not affect any other visualizations.



*You can also right-click the linked visualization, and then click **Unlink visualization** from the shortcut menu.*

Deleting a master dimension or master measure

You can delete dimensions and measures from the master items as long as the app is not published.



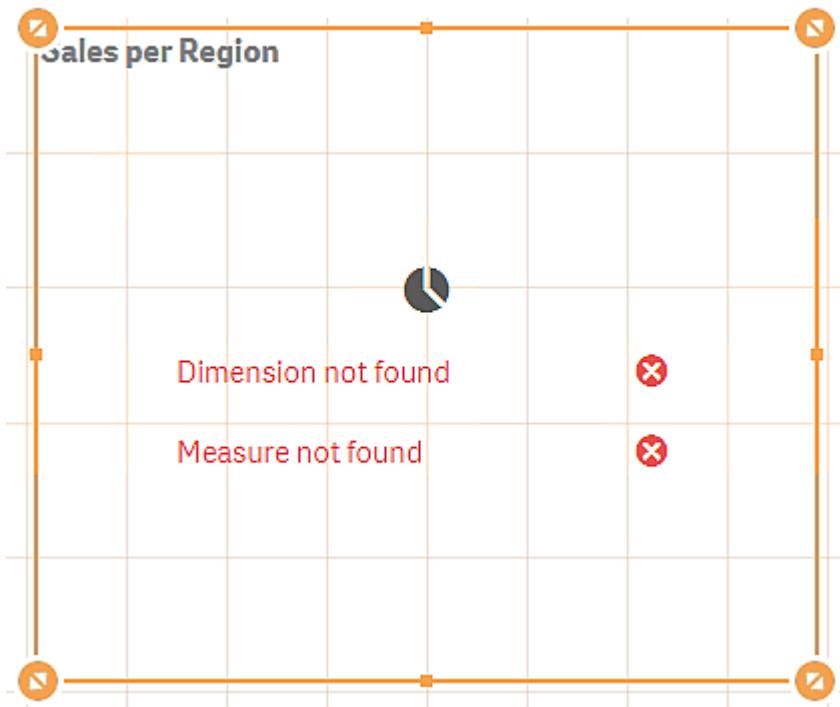
If you delete a master dimension or master measure, the visualizations that use the deleted master item will not work unless you replace it with a new dimension or measure.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click  to display the master items.
3. Click the dimension or measure that you want to delete.
The preview opens.
4. Click  at the bottom of the preview.
A dialog is displayed stating that all visualizations that use the dimension or measure will stop working.
5. Click **OK**.

The dimension or measure is deleted from the master items, and all visualizations on the sheets that used the deleted item do not work anymore. You see the text **Dimension not found** or **Measure not found** on those visualizations.

A visualization that is lacking both a dimension and a measure, both which have been deleted from the master items.



Replacing an invalid dimension or measure

When a dimension or measure has been deleted from the master items, all visualizations that reference to the deleted master item will not work anymore, until the missing dimension or measure is replaced.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side. Click to display the master items.
2. Drag a dimension or measure from the **Dimensions or Measures** sections to the visualization on the sheet.
The shortcut menu opens.
3. Select **Replace invalid dimension** or **Replace invalid measure**.

The visualization is complete and works again.

Deleting a master visualization

You can delete visualizations from the master items as long as the app is not published.

Do the following:

1. In sheet view, click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
2. Click **Advanced options**.
3. Click to display the master items.

4. Click the visualization that you want to delete.
The preview opens.
5. Click  at the bottom of the preview.
A dialog is displayed stating that wherever this visualization is used on sheets, there will be invalid instances of it.
6. Click **OK**.

The visualization is deleted from the master items, and on all sheets where this visualization was used, you see invalid visualizations. You now need to replace the invalid visualization with other ones, or delete the instances.



You can also delete visualizations from the preview that is displayed when clicking the visualization in the master items.

Replacing an invalid visualization on a sheet

The representation of the invalid visualization is there to tell you that there used to be a visualization at a certain location on the sheet, but the invalid visualization serves no purpose.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side. Click  to display the master items.
2. Drag a visualization from the master items to the location of the invalid visualization on the sheet.

The invalid visualization is replaced.

Deleting an invalid visualization

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Right-click on the invalid visualization and select **Delete** in the shortcut menu.

The invalid visualization is deleted.

Using expressions in visualizations

Visualizations in Qlik Sense are built from charts, which in turn, are built from dimensions and measures, depending on the type of chart. Visualizations can have titles, subtitles, footnotes, and other elements to help convey information. All of the elements that make up a visualization can be simple: a dimension consisting of a field representing data, a title consisting of text, for example.

For visualizations that contain measures, the measures are aggregations based on fields. Specifically, the measures are calculations that span multiple records. For example **Sum(Cost)** means all the values of the field **Cost** are aggregated using the function **Sum**. In other words, **Sum(Cost)** is an expression.

What is an expression?

An expression is a combination of functions, fields, and mathematical operators (+ * / =), and other measures. Expressions are used to process data in the app in order to produce a result that can be seen in a visualization. They are not limited to use in measures. You can build visualizations that are more dynamic and powerful by using expressions for titles, subtitles, footnotes, and even dimensions.

This means, for example, that instead of the title of a visualization being static text, it can be made from an expression whose result changes depending on the selections made.

Where can I use expressions?

Expressions can be used in a visualization wherever the symbol **fx** is seen in the properties panel while editing a visualization. The **fx** symbol indicates an expression field. By clicking **fx**, you enter the expression editor, which is designed to help you build and edit expressions. Expressions can also be entered directly into the expression field, without using the expression editor.

An expression cannot be saved directly as a master item, but if an expression is used in a measure or dimension, which is then saved as a master item, with its descriptive data, such as name, description, and tags, the expression in the measure or dimension is preserved.

Expressions are used both in scripts and in chart visualizations. They can be simple, involving only basic calculations, or complex, involving functions fields and operators. Expressions can be used in several different situations. The difference between measures and expressions is that expressions have no name or descriptive data.



In a script, an expression is evaluated as the script execution passes it by. In visualizations (including charts and tables), expressions are evaluated automatically whenever any of the fields, variables or functions that the expression contains change value or logical status. A few differences exist between script expressions and chart expressions in terms of syntax and available functions.



For detailed reference regarding script functions and chart functions, see the Script syntax and chart functions.

Working with the expression editor

You can enter the expression editor to add or edit an expression wherever you see the symbol **fx** in the properties panel while editing a visualization. The **fx** symbol indicates an expression field. Click **fx** to enter the expression editor.

You can add expressions in two ways. Expressions can be created using the **Fields**, **Functions**, **Variables**, and **Set expressions** sections by making selections and inserting them into the expression field. You can edit the inserted expressions and add more complex expressions by typing directly into the expression field. You can undo an action by clicking the **Undo button** ↺ .

Inserting an expression using Fields

You can insert an expression by making selections from the **Fields** section at the right-hand side of the expression editor dialog.

Do the following:

1. If you want to limit the available fields to those from a particular table in your data model, select a table from the **Filter by table** drop-down list.
2. Select a field from the **Field** drop-down list.
3. Select the aggregation function you want to use. The functions available are from the group of basic aggregation functions.
4. If you want to insert the current selection as a set expression in your aggregation, use the **Set expression** check box. The current selection is always based on the default state.
5. If you want to insert a **Distinct** or a **Total** clause together with the aggregation function, use the check boxes for each clause. Each clause can be used separately.
6. Click **Insert** to insert the field and the function into the expression field.



If you do not select an aggregation function, only the field will be inserted into the expression. Clauses can only be added if you select an aggregation function.

7. Click **Apply** to close the **Add expression** dialog.

For a dimension or measure, you continue by adding descriptive data for the dimension or measure.

Inserting a function using Functions

You can insert more than the basic aggregation functions by making selections from the **Functions** section at the right-hand side of the expression editor dialog.

1. If you want to filter the list of available functions, select a function category from the **Function category** drop-down list.
2. Select a function from the **Function name** drop-down list.
You see a preview of the function.
3. Click **Insert** to insert the function and the first parenthesis into the expression field.
4. Type the rest of the expression in the expression editor according to the syntax.
5. Click **Apply** to close the **Add expression** dialog.

For a dimension or measure, you continue by adding descriptive data for the dimension or measure.

Inserting a variable using Variables

You can insert a variable by making selections from the **Variables** section at the right-hand side of the expression editor dialog.

1. If you want system variables to appear in the list of available variables use the **Show system variables** check-box.
2. Select a variable from the **Variable** drop-down list.
You see a **Definition** and **Value** of the variable, if available.

3. Click **Insert** to insert the variable into the expression field.

4. Click **Apply** to close the **Add expression** dialog.

For a dimension or measure, you continue by adding descriptive data for the dimension or measure.

Inserting a set expression using Set expressions

You can insert a set expression by making selections from the **Set expressions** section at the right-hand side of the expression editor dialog.

1. Select whether to **Use current selections** or **Use bookmark**, as a set expression.

You can only **Use current selections** if a selection has been made in the app. You can only **Use bookmark** if a bookmark is available. Use the **Bookmark** drop-down menu to choose between bookmarks.

2. If you have defined alternate states, you can change the alternate state which the set expression is based on. Select the alternate state from **Alternate state** drop-down menu.

3. Click **Insert** to insert the set expression into the expression field.

4. Click **Apply** to close the **Add expression** dialog.

For a dimension or measure, you continue by adding descriptive data for the dimension or measure.

Adding or editing an expression by typing

Do the following:

1. Type the expression directly in the expression field.

The field names you use in the expression are checked, and the syntax of the expression is validated.



As you type in the expression editor, the expression is validated. If there is an error, you see a hint about what is incorrect. Additional error information may be available by clicking the icon next to the hint.

When you type the name of a function in the expression, a tooltip appears that provides information to help you enter the function correctly, including argument names and qualifiers.



The tooltip for some chart functions shows the ALL qualifier. It is recommended that you do not use the ALL qualifier. Instead, use the set expression {1}.

Each line in the expression editor is numbered and syntax highlighting is used.

2. Click **Apply** to close the **Add expression** dialog.

For a dimension or measure, you continue by adding descriptive data for the dimension or measure.

Auto-complete, evaluation and color coding

When typing in the expression editor, you get an auto-complete list of matching fields, variables and functions to select from. The list is narrowed down as you continue to type. The color coding helps you to see where fields, variables and functions are used in the expression.

The color coding applies both in the auto-complete list and in the expression itself.

You can verify that any dollar-sign expansions added in your expression give the correct result, by checking the expression evaluation, in the dollar-sign expansion preview.

Detailed syntax help

You can activate help mode by clicking the **Help button** . When active, all functions in the expression editor act like hyperlinks. Clicking on a function opens a browser tab to the online help section with the full description of how to use the specific function. When a function is clicked, help mode is exited.

References to fields, measures, and variables

An expression can contain references to fields, variables, and measures.

In most cases, expressions are aggregations, that is calculations that potentially can span multiple records. This means that all field references in an expression must be wrapped in an aggregation function. If no aggregation function is used, the Only() function is used.

When the name of a measure is used inside an expression, it is interpreted as an alias for the measure. This allows you to re-use an already defined entity. You can use the labels of master measures and chart measures in this way. However, the chart measure must be in the same chart. You cannot use labels of measures in other charts.

You can define a measure recursively. In other words, the measure can contain a reference to itself. However, you should only do this in charts with few rows. Performance degrades if the chart has many rows. If there are hundreds of rows, the recursive definition will stop working altogether.

A variable can be used in two different ways in a direct reference or in a dollar expansion. If you use a direct reference, the variable value will be used in the calculation. If you use a dollar expansion, the entire dollar expansion will be replaced by the variable value before the expression is parsed. Therefore, the two different methods may return different results.

How names are interpreted

A name inside an expression can be a reference to a field, a variable, a function, or a measure. Depending on situation, the name is interpreted differently.

Example:

The string XXX represents a field, a variable, a function, or a measure. XXX will be interpreted as one of these depending on how you create the expression.

Examples of how names are interpreted

Expression	XXX interpreted as
xxx	measure, variable, or field
\$ (xxx)	variable
count (xxx)	field or variable
xxx ()	function

You should not use the same name for a field and a variable (or a measure). But if you do and there is ambiguity, the following order of precedence is used:

- If the name is found inside an aggregation function, a field has precedence over a variable.
- If the name is found outside an aggregation function, a measure label has precedence over a variable name, which in turn has precedence over a field name.

Rules for expressions

The following rules apply for chart expressions:

- If a field reference isn't wrapped in an aggregation function, the Only() function is used by the engine.
- All expressions return a number and/or a string, whichever is appropriate.
- Logical functions and operators return 0 for False, -1 for True. Number-to-string conversions and string-to-number conversions are implicit.
- Logical operators and functions interpret 0 as False and all else as True.
- Expressions that cannot be correctly evaluated, for example as a result of incorrect parameters or functions, return NULL.

Using functions in charts

A function is a type of procedure or routine that performs a specific task on data in apps. Qlik Sense contains several hundred ready-made functions that can be used in charts when creating visualizations. Functions can be, for example, mathematical, logical, can operate on financial or date and time information, can be used to manipulate strings, and other situations.

Functions can be grouped into the types:

- Aggregation functions, which use several records as input and produce a single value result.
- Scalar functions, which take a single input and produce a single output.
- Range functions, which produce a single value based on a range of input values.
- Range-producing functions, which are like range functions, but produce a range of values as output.

Many of the functions can be used in both chart expressions and scripts, but some are specific for chart expressions.

The following list shows some examples of functions:

- **Max**: an aggregation function that can be used in scripts and charts.
For example: **Max(Sales)** calculates the highest value in the field Sales.
- **IF**: a conditional function that can be used in scripts and charts.
For example: **IF(Amount>0, 'OK', 'Alarm')** determines if the condition 'is the value of Amount greater than zero?' is met. If it is, OK is written, otherwise Alarm is written.
- **Date#**: an interpretation function that can be used in scripts and charts.
For example: **Date#(A)** takes the input value A and evaluates it as a date.



A few differences exist between script expressions and chart expressions in terms of syntax and available functions. The most important difference is the role of the aggregation functions and the use of field references. The basic rule is that any field name in a chart expression must be enclosed by exactly one aggregation function. An aggregation function can never have another expression containing an aggregation function as argument.



For detailed reference regarding script functions and chart functions, see the [Script syntax and chart functions](#).

Using variables in expressions

A variable in Qlik Sense is a container storing a static value or a calculation, for example a numeric or alphanumeric value. When you use the variable in the app, any change made to the variable is applied everywhere the variable is used. You can define variables in the variables overview, or in the script using the data load editor. You set the value of a variable using **Let** or **Set** statements in the data load script.



When using variables in expressions, you can change the expression used in a range of charts simultaneously simply by editing the variable.

You open the **Variables** overview by clicking in the edit bar when editing a sheet.



The **Variables** overview is not available in published apps. If you need to add or change variables in an published app, use the variable input control available with the Dashboard bundle.

The following actions are available in the variables overview:

- Create a new variable.
- Edit the selected variable.
- Delete the selected variable.



To edit or delete a variable that is defined in the script, you must edit the script.

Getting an overview of all variables in an app

You can get an overview of all variables in an unpublished app.

Do the following:

- When editing a sheet, click in the edit bar on the sheet to open the variables dialog. The variables dialog opens and displays a list of all variables in the app and their definitions (if any).

See also:

[Working with the expression editor \(page 123\)](#)

Creating a variable

A variable in Qlik Sense is a named entity, containing a data value. When a variable is used in an expression, it is substituted by its value or the variable's definition. Variables are defined using the variables dialog or in the script using the data load editor.

You can create a new variable from the variables dialog, when editing a sheet in an unpublished app.

You can duplicate an existing variable by clicking  and selecting **Duplicate**.

For a visual demo about creating variables, see [Creating a variable](#).

Do the following:

1. When editing a sheet, click  in the edit bar on the sheet to open the variables dialog.

The variables overview opens.

2. Click **Create new**.

The following input fields for the variable are displayed:

- **Name** (mandatory)
- **Definition**
- **Description**
- **Tags**

Press Esc or click **Cancel** if you want to cancel creating the new variable.

3. Type a name for the variable (mandatory). Use the following guidelines when choosing a name:

- You cannot change the name once you have created the variable.
- Use a letter as the first character, do not use a number or a symbol.
- It is not recommended to name a variable identically to a field or a function in Qlik Sense.
For more information, see [How names are interpreted \(page 131\)](#).
- Do not use the following characters when naming a variable: \$ () [] "
- The name must be unique. You cannot name a variable using a name used for a reserved variable or a system variable. These variables are not listed in the variables dialog, but if you are not allowed to use a certain name, even though you cannot find a duplicate in the variables dialog, a reserved variable or a system variable already has this name.
- A long name is not recommended. If a variable's name is too long, the name cannot be fully displayed in the variables overview.

4. Optionally, type a description for the variable.

5. Create a definition for the variable. You can enter the expression editor by clicking .

For more information, see [Working with the expression editor \(page 123\)](#).

Example:

Set the variable's value to today's date, presented as a number:

`Num(Today())`

6. Optionally, enter tags for the variable.
7. Click **Create** to save the variable.

Editing a variable

You can edit variables from the variables dialog.



You can edit variables defined in the script in the variables dialog. If you reload the script, however, your changes will be undone.

Do the following:

1. When editing a sheet, click in the edit bar on the sheet to open the variables dialog.
The variables dialog opens.
2. In the row of the variable you want to delete, click and select **Edit**.
3. Edit the variable as desired.
4. Click **Save**.

Deleting a variable

You can delete variables, from an unpublished app, by deleting them from the variables dialog.

Do the following:

1. When editing a sheet, click in the edit bar on the sheet to open the variables dialog.
The variables dialog opens.
2. Select the variables you want to delete and click **Delete**. You can select up to 20 variables.
You can also click in the row of the variable you want to delete and select **Delete**.



If you remove a variable from the script and reload the data, the variable stays in the app. If you want to fully remove the variable from the app, you must also delete the variable from the variables dialog.



Deleting a variable cannot be undone.

3. Click **Delete**.

Examples of using a variable in an expression

A variable in Qlik Sense is a named entity, containing a data value. When a variable is used in an expression, it is substituted by its value or the variable's definition.

Example:

The variable x contains the text string $\text{Sum}(\text{Sales})$.

In a chart, you define the expression $\$(x)/12$. The effect is exactly the same as having the chart expression $\text{Sum}(\text{Sales})/12$.

However, if you change the value of the variable x to for example $\text{Sum}(\text{Budget})$, the data in the chart are immediately recalculated with the expression interpreted as $\text{Sum}(\text{Budget})/12$.



When using variables in expressions, you can change the expression used in a range of charts simultaneously simply by editing the variable.

How names are interpreted

It is not recommended to name a variable identically to a field or a function in Qlik Sense. But if you do, you must know how to use them in an expression.

Example:

The string XXX represents a field, a variable, a function, or a measure. XXX will be interpreted as one of these depending on how you create the expression.

Examples of how names are interpreted

Expression	XXX interpreted as
xxx	measure, variable, or field
$\$(xxx)$	variable
$\text{Count}(xxx)$	field or variable
$xxx()$	function

When naming an entity, avoid assigning the same name to more than one field, variable, or measure. There is a strict order of precedence for resolving conflicts between entities with identical names. This order is reflected in any objects or contexts in which these entities are used. This order of precedence is as follows:

- Inside an aggregation, a field has precedence over a variable. Measure labels are not relevant in aggregations and are not prioritized.
- Outside an aggregation, a measure label has precedence over a variable, which in turn has precedence over a field name.
- Additionally, outside an aggregation, a measure can be re-used by referencing its label, unless the label is in fact a calculated one. In that situation, the measure drops in significance in order to reduce risk of self-reference, and in this case the name will always be interpreted first as a measure label, second as a field name, and third as a variable name.

Variable calculation

There are several ways to use variables with calculated values in Qlik Sense, and the result depends on how you define it and how you call it in an expression.

This example requires the following data is loaded in the data load editor:

```
LOAD * INLINE [
    Dim, Sales
    A, 150
    A, 200
    B, 240
    B, 230
    C, 410
    C, 330
];
```

Let's define two variables, from the variables dialog:

- **Name vSales Definition**'Sum(Sales)'
- **Name vSales2 Definition**'=Sum(Sales)'

In the second variable, we add an equal sign before the expression. This will cause the variable to be calculated before it is expanded and the expression is evaluated.

If you use the vSales variable as it is, for example in a measure, the result will be the string Sum(Sales), that is, no calculation is performed.

If you add a dollar-sign expansion and call \$(vSales) in the expression, the variable is expanded, and the sum of Sales is displayed.

Finally, if you call \$(vSales2), the variable will be calculated before it is expanded. This means that the result displayed is the total sum of Sales. The difference between using=\$(vSales) and=\$(vSales2) as measure expressions is seen in this chart showing the results:

Results		
Dim	\$(vSales)	\$(vSales2)
A	350	1560
B	470	1560
C	740	1560

As you can see, \$(vSales) results in the partial sum for a dimension value, while \$(vSales2) results in the total sum.

See also:

Working with the expression editor (page 123)

Using chart level scripting in visualizations

With chart level scripting, you can add script-like functionality to the results of a visualization. It complements chart expressions, as chart level scripting include control statements, such as loops and recursive actions, for more complex mathematical calculations. You can also modify multiple columns, as long as a placeholder exists in the chart, and add new rows to a result.

You can use chart level scripting on most visualization types and you access it from the properties panel. Some visualizations, or specific configurations, do not support chart level scripting. In those cases, you cannot access chart level scripting from the properties panel, or you will receive an error message.

When modifying chart data, you use a sub-set of the Qlik Sense script which consists of a number of statements. See [Chart level scripting](#).

You enable chart level scripting in the app settings. When enabled, a **Scripts** control is available on the **Data** section of the properties panel for a visualization. See [Turning on chart level scripting](#).

Even when disabled, existing chart scripts will be processed. Disabling once authored content has been developed is a useful way of preventing new chart scripts being created by other users.

Adding a chart script to a visualization

You can add chart scripts to existing visualizations, or when you create new visualizations.

Chart level scripting must be enabled in the app before you can add a chart script to a visualization, see [Turning on chart level scripting](#).

You must be in  **Edit sheet** mode with **Advanced options** turned on to be able to add a chart script to a visualization.

Adding chart scripts to a new visualization

Do the following:

1. Drag the visualization from the assets panel onto the sheet, or double-click the visualization.
2. On the **Data** tab, add dimensions and measures to the visualization.
3. Also on the **Data** tab, add a chart script in the **Scripts** section.
You can add maximum five chart scripts per visualization.

Adding chart scripts to an existing visualization

Do the following:

1. When in  **Edit sheet** mode, select the visualization.
2. On the **Data** tab, add a chart script in the **Scripts** section.
You can add maximum five chart scripts per visualization.

Example

This example assumes that you have created a visualization, for example a bar chart, and that you have added a dimension and a measure.

This example takes the values of the first dimension and puts them in the first measure in reverse order.

```
Let P = HCNoRows();
For J = 1 to P
Put #hc1.measure.1(J) = HCValue(#hc1.dimension.1, P - J + 1);
Next
```

Limitations of chart level scripting

Chart level scripting cannot be used in the standard edit mode.

Chart level scripting does not work for visualizations with **Chart suggestions** enabled.

You can add maximum five chart scripts per visualization.

Chart level scripting is not available for the following visualizations:

- Bar charts in stacked mode
If you have created a chart script in grouped mode and switch to stacked mode, an invalid hypercube error will be shown.
- Box plot
- Button
- Distribution plot
- Filter pane
- Line charts with two dimensions
- Pivot table
- Text & image
- Treemap
- Grid Chart (Visualization bundle)
- Trellis Container (Visualization bundle)

Searching in the assets panel

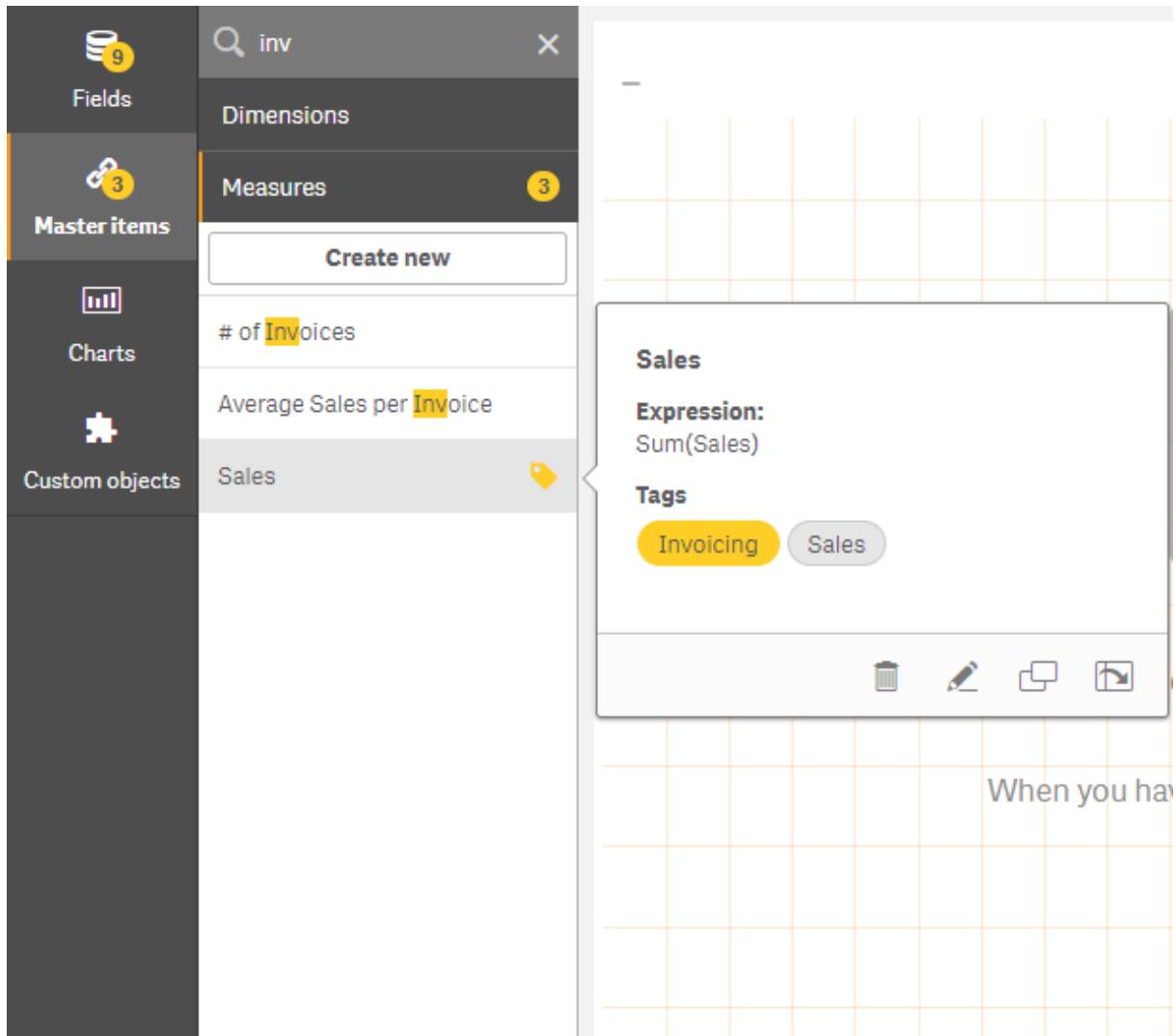
Find assets using the search field. As you start to type in the search field, all items that have a name or a tag that contains the search string, are presented.



The search function is not case sensitive.

When a matching string is found in a tag, it is indicated by a tag icon in the search result. Clicking the item in the list of results, brings up a preview of it.

Searching for "inv" among the measures presents any measure that contains that string in its name or a tag.



Designing visualizations with Direct Discovery

Designing visualizations from Direct Discovery fields requires some background knowledge about the nature of the field types that can be used.

Direct Discovery fields in visualizations

DIMENSION

- The Direct Discovery DIMENSION field type can be used in the same way as any other dimension.

MEASURE

- A Direct Discovery MEASURE must be aggregated with one of the common functions before it can be used in a visualization.
- A regular field that is dropped onto a sheet becomes a filter pane, but not a Direct Discovery MEASURE as it must be aggregated first.

DETAIL

The Direct Discovery DETAIL field type can only be used in tables, and not in combination with a measure.



When creating a table using a Direct Discovery DETAIL field, and the number of rows being retrieved exceeds a value defined in the load script (by default, 1000 rows), you may see a warning message and the table may appear to be invalid. To make the table work normally, try making selections in the app to reduce the data set.



This functionality is not available in all editions of Qlik Sense.

3.5 Best practices for choosing visualization types

A good visualization clearly presents relationships among many values, and lets you analyze data at a glance. Qlik Cloud Analytics offers a range of Qlik Sense visualizations and charts. Each chart excels at visualizing data in different ways for different purposes. You should select your charts by deciding what you want to see from the data in your charts.

If you are unsure of what visualizations to use, Qlik Sense can offer recommendations. For more information, see *Creating visualizations with Insight Advisor (page 412)* and *Creating visualizations using Insight Advisor chart suggestions (page 434)*.

The following lists the purposes for viewing data and the chart type recommended to achieve that purpose:

Viewing comparisons

Comparison charts are used to compare values against each other. They show the differences in values, such as the difference between categories, or how values are changing over time.

Questions that might be answered by comparison charts include:

- What product has the highest total sales this year?
- How have product sales risen or fallen over the last 24 months?

Chart types for viewing comparisons

Chart type	Common purpose
<i>Bar chart (page 142)</i>	Comparing categories against the same measure or measures.
<i>Line chart (page 184)</i>	Comparing trends over time.
<i>Combo chart (page 162)</i>	Comparing measures that are different in scale.

If you want to compare different values of the same dimension, you can use alternate states.

Viewing relationships

Relationship charts are used to explore how values relate to each other. A relationship chart allows you to find correlations, outliers, and clusters of data.

Questions that might be answered by relationship charts include:

- Is there a correlation between advertising spending and sales for our products?
- How do expenses and income vary per region?

Chart types for viewing relationships

Chart type	Common purpose
<i>Scatter plot</i> (page 262)	Viewing the relationship between two or three measures for a dimension.

Viewing compositions

Composition charts take a total value and discover what component values make up that total. Composition charts can be static, showing the current composition of a total value, or they can show how the composition of a total value changes over time. Composition charts can display compositions either by percentage of the total value or the fixed values in the total value.

Questions that might be answered by composition charts include:

- What percentages of our total sales come from which regions?
- What is each department's allotment of our total quarterly budget over the past year?

Chart types for viewing compositions

Chart type	Common purpose
<i>Bar chart</i> (page 142)	Viewing the changing composition of a value over a short period of time.
<i>Line chart</i> (page 184)	Viewing the changing composition of a value over a long period of time.
<i>Pie chart</i> (page 248)	Viewing the static composition of a value.
<i>Waterfall chart</i> (page 292)	Viewing the static composition of a value with accumulation or subtraction to the total.
<i>Treemap</i> (page 287)	Viewing the static composition of a value's accumulation to the total.

Viewing distributions

Distribution charts are used to explore how the values within data are grouped. Distribution charts show you the shape of your data, the range of its values, and possible outliers.

Questions that might be answered by distribution charts include:

- What is the number of customers per age group?
- What cities have the highest use of our services?

Chart types for viewing distributions

Chart type	Common purpose
<i>Histogram</i> (page 178)	Viewing the how data is distributed over intervals.
<i>Scatter plot</i> (page 262)	Viewing the distribution of two measures.
<i>Distribution plot</i> (page 168)	Viewing the distribution of measure values in a dimension.
<i>Box plot</i> (page 155)	Viewing the range and distribution of numerical data.

Viewing performances

Performance charts provide a quick view of a performance measure. Looking at a performance chart, a user can quickly identify the measure value and whether the results are as expected or not.

Questions that might be answered by performance charts include:

- What are the current total sales for this quarter?
- Are current total sales for this quarter meeting the projected sales for the quarter?
- How is performance for this product line compared with other product lines?

Chart types for viewing performances

Chart type	Common purpose
<i>Bullet chart</i> (page 159)	Comparing performance of a measure for several dimensions.
<i>Gauge</i> (page 176)	Viewing a performance value to understand performance immediately.
<i>KPI</i> (page 181)	Viewing one or two performance measures.
<i>Text & image</i> (page 284)	Viewing text or several measures with an image.

Viewing data

Data charts present detailed data rather than a visualization of the data. Data charts are useful when you need to view precise values, and when you want to compare individual values.

Questions that might be answered by data charts include:

- What are the records for each transaction for this month?
- What are the quantity and sales for each item in each product group for each of our customers?

Chart types for viewing data

Chart type	Common purpose
<i>Table</i> (page 268)	Viewing precise values from your data without trends or patterns.
<i>Pivot table</i> (page 252)	Viewing precise value for several dimensions and measures.

Viewing geography

Geographical charts let you visualize your data by geography, displaying your data on a map either as points or areas.

Common questions that might be answered by geographical charts include:

- What cities have the highest use of our services?
- Which countries have the most customers?

Chart types for viewing geography

Chart type	Common purpose
<i>Map chart</i> (page 187)	Viewing data represented geographically by point or area.

What if no standard chart suits my purpose?

You can use controls or objects from a bundle supplied by Qlik:

- *Dashboard bundle* (page 313)
- *Visualization bundle* (page 331)

You can also create custom visualization objects if none of the standard charts provided fit your requirements for visualizing your data.

For more information, see *Creating a visualization using a custom object* (page 446).

3.6 Visualizations

You can use visualizations to present the data that is loaded into the app. For example, you can use a bar chart to compare sales numbers for different regions, or use a table to show precise values for the same data.

The selections you make in a visualization are reflected in all associated visualizations on all sheets.

Creating a visualization

You create visualizations from predefined charts or custom objects. You must be in  **Edit** mode to be able to add a visualization to the sheet.

1. Drag the visualization from the assets panel onto the sheet, or double-click the visualization.
2. Add dimensions and measures to the visualization. The number of dimensions and measures that are required depends on which visualization you selected.
Dimensions determine how the data in a visualization is grouped. For example: total sales per country or number of products per supplier. For more information, see *Dimensions* (page 74).
Measures are calculations used in visualizations, typically represented on the y-axis of a bar chart or a column in a table. Measures are created from an expression composed of aggregation functions, such

as **Sum** or **Max**, combined with one or several fields. For more information, see *Measures (page 77)*.

3. Adjust the presentation, for example sorting, coloring, or labeling.

You can convert from one visualization type to another by dragging a new chart to a visualization on a sheet.

For other methods of creating a visualization, see *Creating and editing visualizations (page 406)*.

Reusing a visualization

If you have created a visualization that you want to reuse in other sheets of the app, you can save it as a master visualization. You can only create master visualizations in an unpublished app. When the app is published, all users can add the visualization to their own sheets, but not modify it.



Right-click on a visualization and select **Add to master items** to save it as a master visualization.

You can find master visualizations under in the assets panel.

Which visualizations are available?

There are two basic types of visualizations available in the assets panel.

- Charts illustrate the data with visual elements like bars, lines, or points.
- Text-based visualizations presents data in text form, for example, tables or filters.

There are dashboard objects available in the assets panel.

The best choice of chart type depends on the purpose of the visualization.

If the predefined visualizations does not fill your purpose, you can use a visualization extension. You find them in the assets panel under .

Charts

Chart	Icon	Description
Bar chart		The bar chart displays a bar for each dimension value. The bar length corresponds to its numerical measure value.
Box plot		The box plot is suitable for comparing range and distribution for groups of numerical data, illustrated by a box with whiskers, and a center line in the middle.
Bullet chart		Bullet charts can be used to visualize and compare performance of a measure to a target value and to a qualitative scale, such as poor, average, and good.

Chart	Icon	Description
Combo chart		The combo chart combines bars and lines in the same chart. The bars and lines have different axes to enable comparing percentages and sums.
Distribution plot		The distribution plot is suitable for comparing range and distribution for groups of numerical data. Data is plotted as value points along an axis.
Gauge		The gauge is used to display the value of a single measure, lacking dimensions.
Histogram		The histogram is suitable for visualizing distribution of numerical data over a continuous interval, or a certain time period. The data is divided into bins.
Line chart		The line chart displays data lines between values. Line charts are often used to visualize a trend in data over intervals of time.
Map		The map is used to combine geospatial data and measure values, such as the sales for a region or a store.
Pie chart		The pie chart shows the relation between a single dimension and a single measure.
Scatter plot		The scatter plot presents values from two measures. This is useful when you want to show data where each instance has two numbers, for example, country (population and population growth). An optional third measure can be used and is then reflected in the size of the bubbles. When showing large data sets colors will be used instead of bubble size to represent the measure size.
Treemap		The treemap shows hierarchical data. A treemap can show a large number of values simultaneously within a limited space.
Waterfall chart		The waterfall chart illustrates how an initial value is affected by intermediate positive and negative values.
<i>Visualization bundle (page 331)</i>		The Visualization bundle is a set of charts that can be used to enhance and increase your Qlik Sense app's charting capacity.

Text-based visualizations

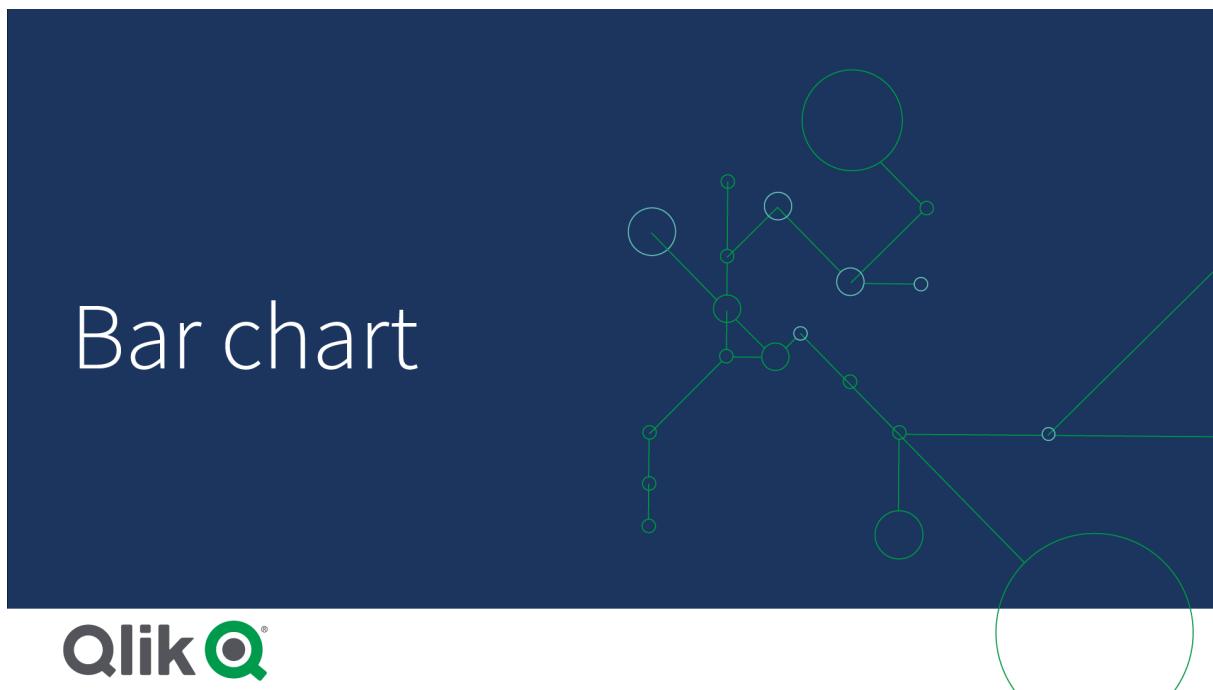
Chart	Icon	Description
Filter pane		The filter pane allows you to control what data that is shown in the visualizations on a sheet. A filter pane can filter the data of several dimensions at once.
KPI		The KPI is used to present central performance figures. You can add a link to a sheet.
Pivot table		The pivot table presents dimensions and measures as rows and columns of a table. The pivot table allows you to analyze data in multiple dimensions at a time. The data in a pivot table may be grouped based on a combination of the dimensions, and partial sums can be shown.
Table		The table displays values in record form, so that each row of the table contains fields calculated using measures. Typically, a table includes one dimension and multiple measures.
Text & image		You can use the text & image visualization to add text, images, measures and links to a webpage.

Dashboard objects

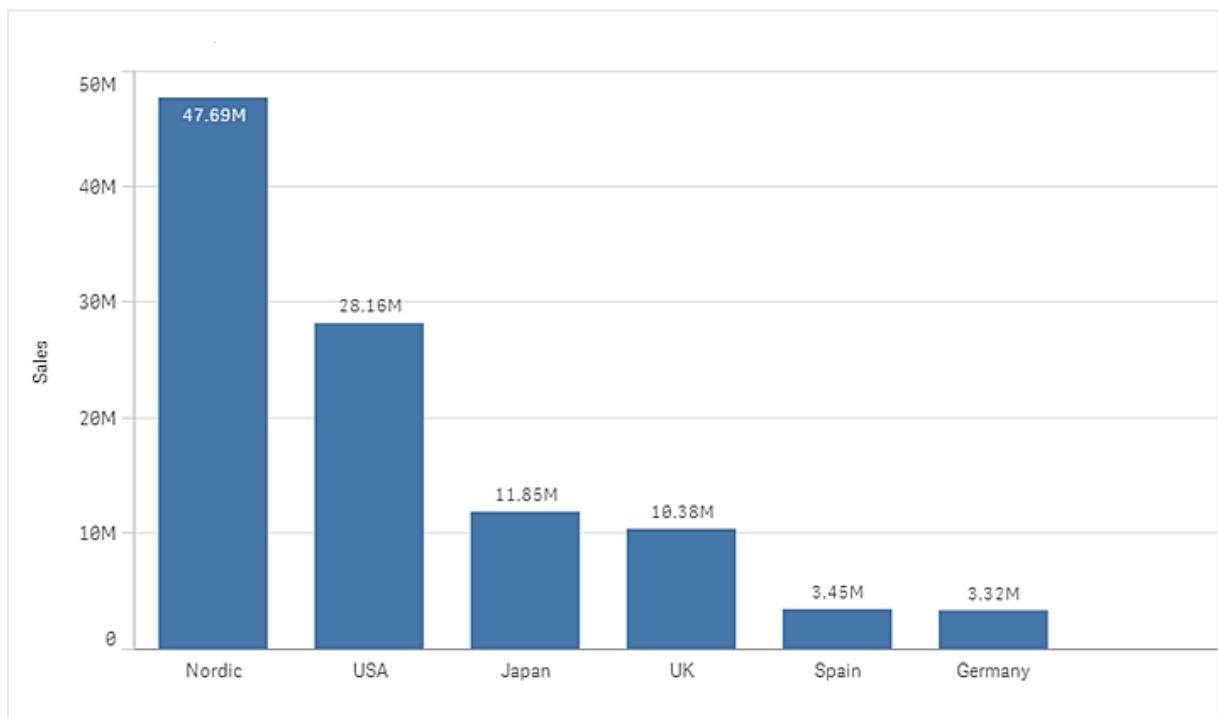
Object	Icon	Description
Button		You can use buttons to add quick links for easy selection and navigation in your app.
Container		You can add visualizations in a limited space and show or hide the visualizations inside the container based on conditions.
<i>Dashboard bundle (page 313)</i>		A set of controls that you can use to enhance navigation and selection in your Qlik Sense app.

Bar chart

The bar chart is suitable for comparing multiple values. The dimension axis shows the category items that are compared, and the measure axis shows the value for each category item.

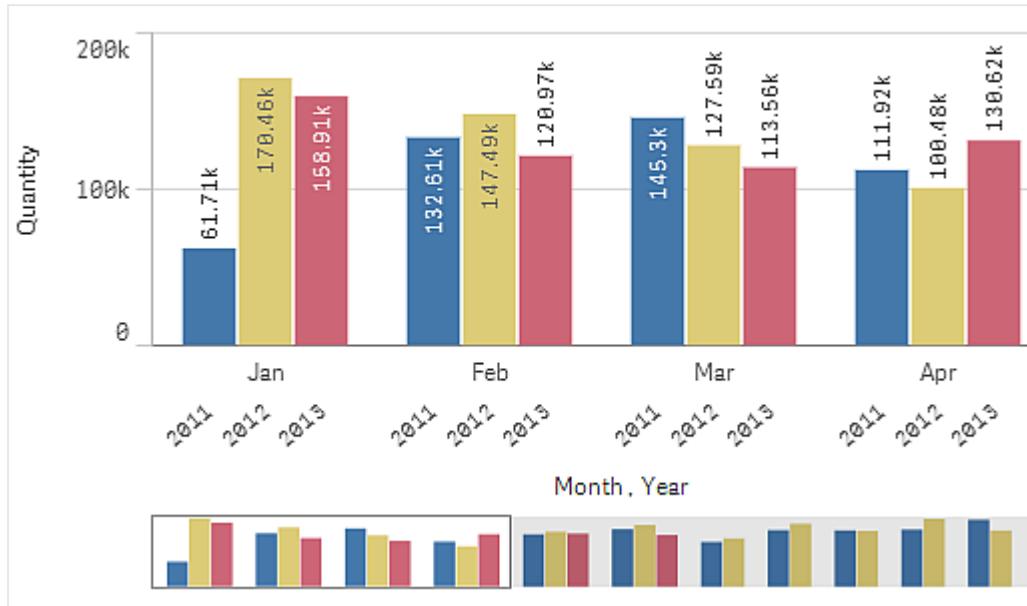


In the image, the dimension values are different regions: Nordic, USA, Japan, UK, Spain, and Germany. Each region represents a dimension value, and has a corresponding bar. The bar height corresponds to the measure value (sales) for the different regions.

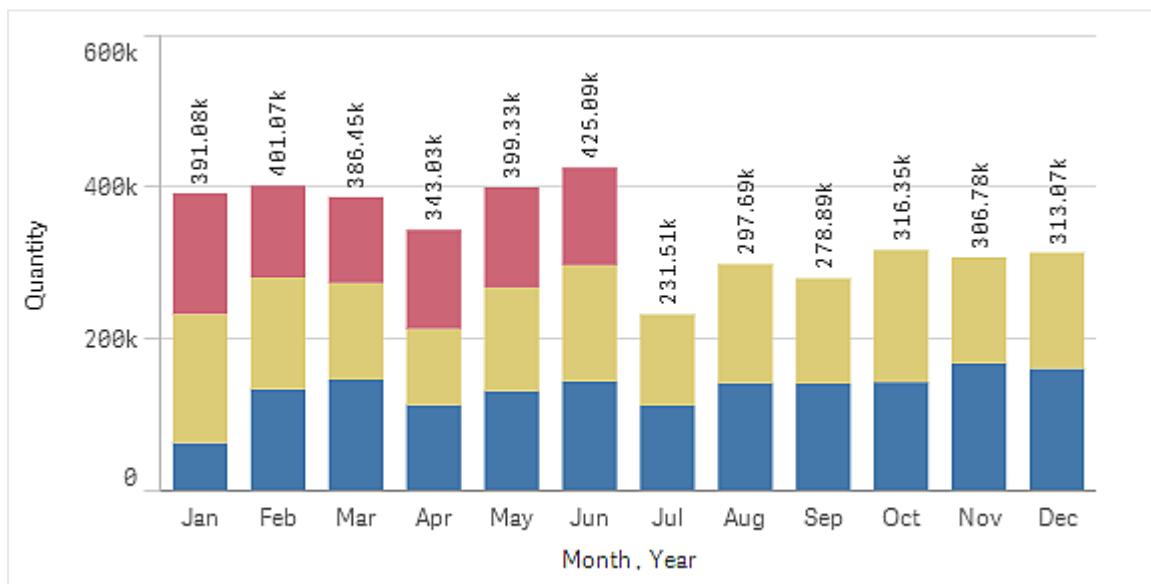


You can make more complex comparisons of data by using grouped or stacked bars. This requires using two dimensions and one measure. The two example charts use the same two dimensions and the same measure:

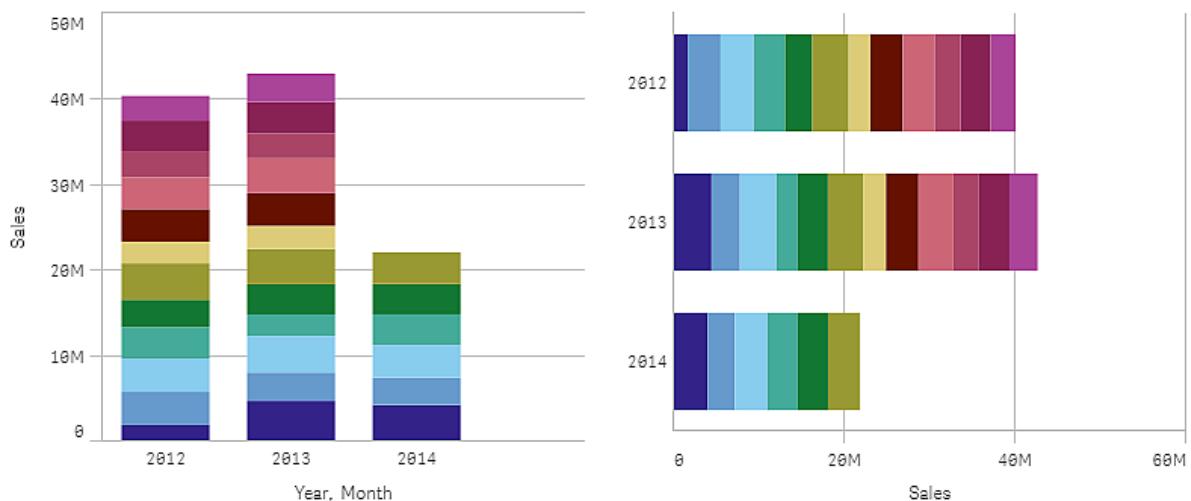
Grouped bars: With grouped bars, you can easily compare two or more items in the same categorical group.



Stacked bars: With stacked bars it is easier to compare the total quantity between different months. Stacked bars combine bars of different groups on top of each other and the total height of the resulting bar represents the combined result.



The bar chart can be displayed horizontally or vertically, as in the example below:



When to use it

Grouping and stacking bars makes it easy to visualize grouped data. The bar chart is also useful when you want to compare values side by side, for example sales compared to forecast for different years, and when the measures (in this case sales and forecast) are calculated using the same unit.

Advantages: The bar chart is easy to read and understand. You get a good overview of values when using bar charts.

Disadvantages: The bar chart does not work so well with many dimension values due to the limitation of the axis length. If the dimensions do not fit, you can scroll using the scroll bar, but then you might not get the full picture.

Creating a bar chart

You can create a simple bar chart on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty bar chart to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.

In a bar chart you need at least one measure.

You can include up to two dimensions and one measure, or one dimension and up to 15 measures in a bar chart. Each bar corresponds to a dimension, and the values of the measures determine the height or length of the bars.

You can also create a bar chart with no dimension and up to 15 measures. In this case, one bar is displayed for every measure. The value of the measure determines the height or length of a bar.

Creating a bar chart

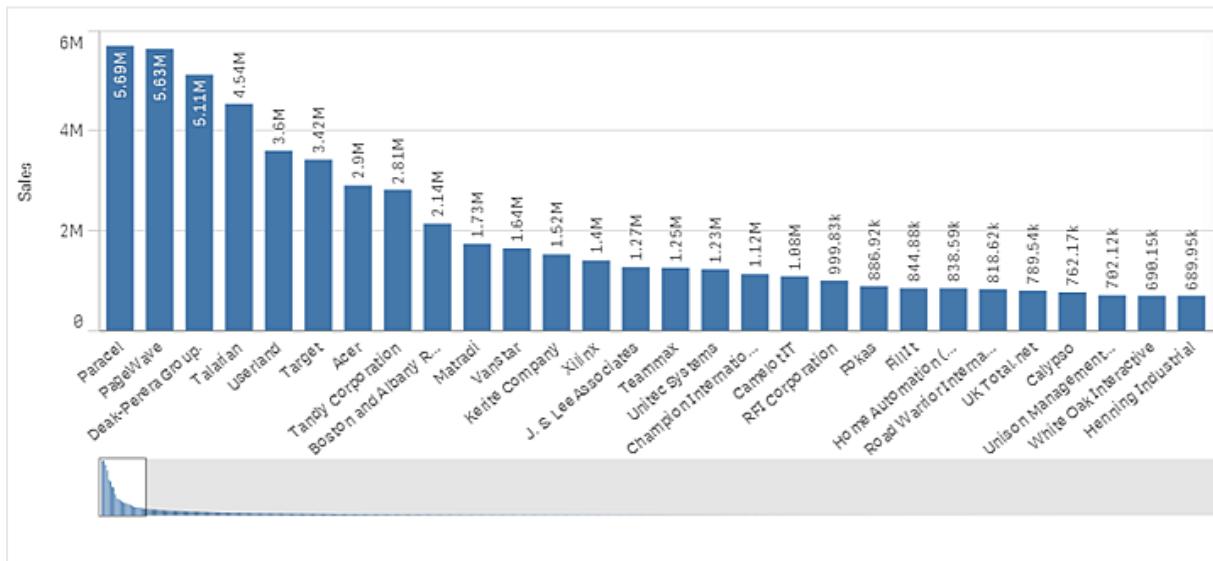
Dimensions	Measures	Result
1 dimension	1 measure	A simple bar chart with one bar for each dimension value.
2 dimensions	1 measure	A grouped or a stacked bar chart with one bar for each value of the two dimensions.
No dimension	up to 15 measures	A simple bar chart with one bar for each measure.
1 dimension	up to 15 measures	A grouped or a stacked bar chart with one bar for each value of each measure..

When you have created the bar chart, you may want to adjust its appearance and other settings in the properties panel.

Display limitations

Displaying large numbers of dimension values

When the number of dimension values exceeds the width of the visualization, a mini chart with a scroll bar is displayed. You can scroll by using the scroll bar in the mini chart, or, depending on your device, by using the scroll wheel or by swiping with two fingers. When a large number of values are used, the mini chart no longer displays all the values. Instead, a condensed version of the mini chart (with the items in gray) displays an overview of the values, but the very low and the very high values are still visible.



You can exchange the mini chart with a regular scrollbar, or hide it, with the **Scrollbar** property.

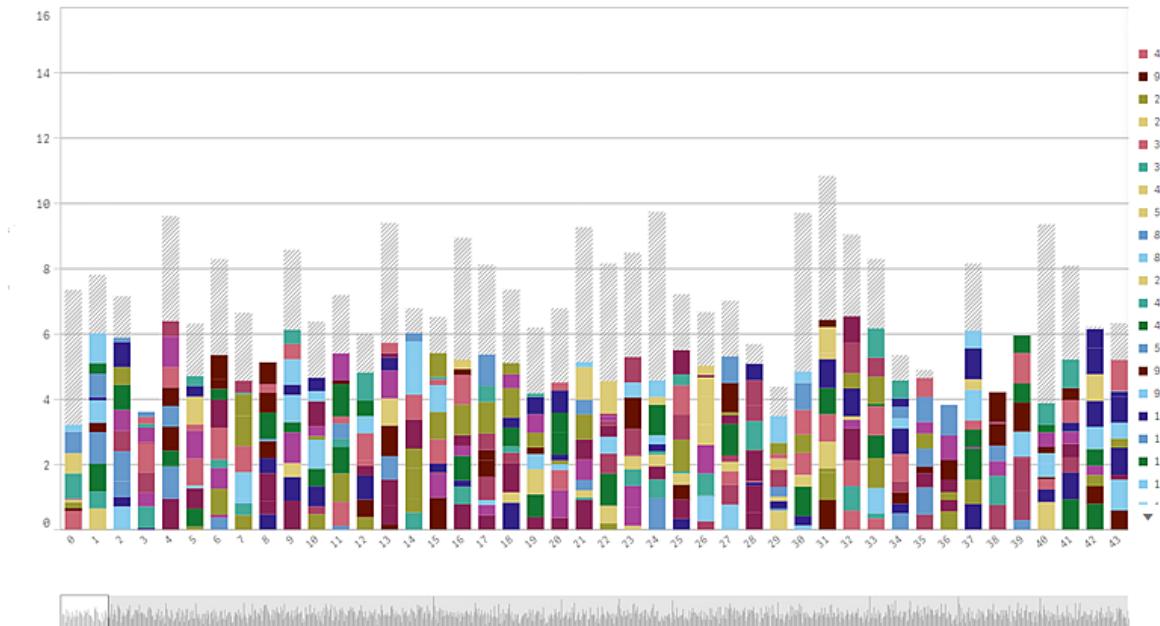
Displaying out of range values

In the properties panel, under **Appearance**, you can set a limit for the measure axis range. Without a limit, the range is automatically set to include the highest positive and lowest negative value, but if you set a limit you may have values that exceed that limit. A bar that exceeds the limit will be cut diagonally to show that it is out of range.

When a reference line is out of range, an arrow is displayed together with the number of reference lines that are out of range.

Displaying large amounts of data in a stacked bar chart

When displaying large amounts of data in a stacked bar chart, there may be cases when not each dimension value within a bar is displayed with correct color and size. These remaining values will instead be displayed as a gray, striped area. The size and total value of the bar will still be correct, but not all dimension values in the bar will be explicit.



To remove the gray areas, you can either make a selection or use dimension limits in the properties panel.

The approximate limit for how many stacked bars that can be displayed without gray areas is 5000 bars, assuming that each bar consists of 10 inner dimension values and one dimension value and one measure value for the whole bar.

The initial data load is 500 dimension values or dimension stacks. (The value 500 refers to the outer dimension values, not each dimension value in a stack.) When you have scrolled past those 500 values, an incremental load is performed, where values are instead loaded based on the current view or scroll position.

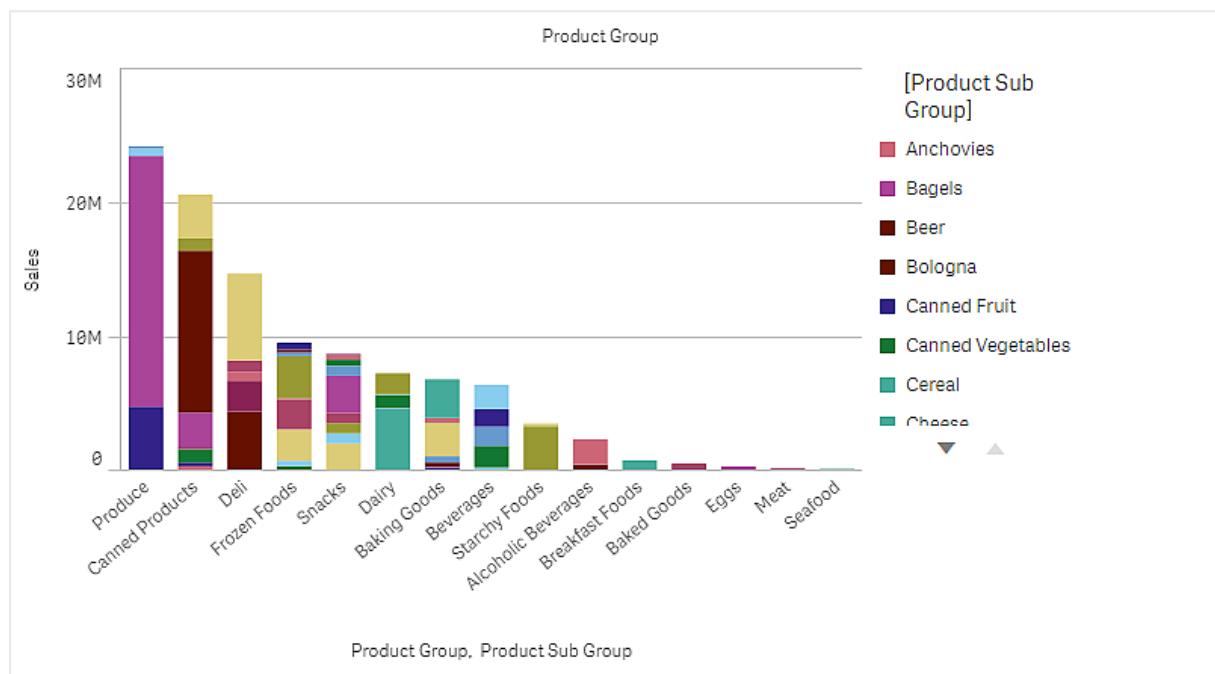
Displaying large amounts of data in a bar chart with continuous scale

If the chart uses a continuous scale, a maximum of 2000 data points are displayed. Above that number, data points are neither displayed, nor included in selections made in the chart. Additionally, only twelve dimension values are displayed for the second dimension in a chart with two dimensions and continuous scale.

To avoid displaying limited data sets, you can either make a selection or use dimension limits in the properties panel.

Comparing categories against a measure with a bar chart

This example shows how to make a bar chart to visualize sales data and how to compare different product groups against the same measure.



Dataset

In this example, we'll use two data files available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial, and the files are available in the *Tutorials source* folder:

- *Sales.xls*
- *Item master.xls*

To download the files, go to [Tutorial - Building an App](#).

Add the two data files to an empty app, and make sure that they are associated by *Item Number*.

The dataset that is loaded contains sales data. The *Item master* table holds the information about the items ordered, such as product groups.

Measure

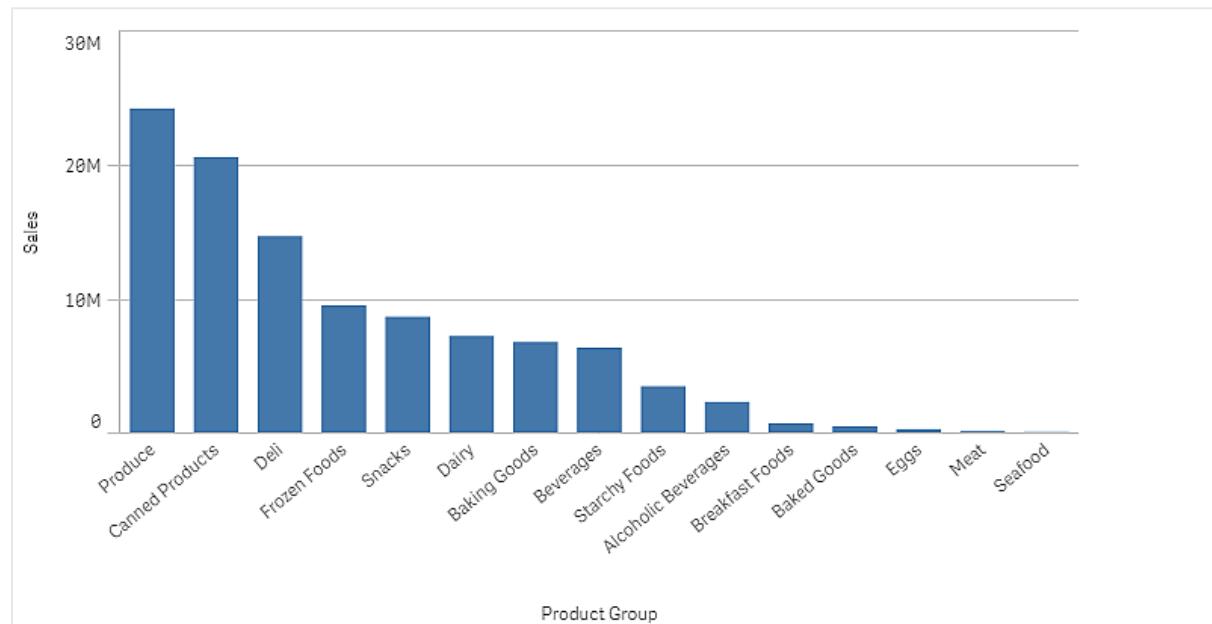
We use the sales volume as the measure, by creating a measure in Master items with the name *Sales*, and the expression `Sum(Sales)`.

Visualization

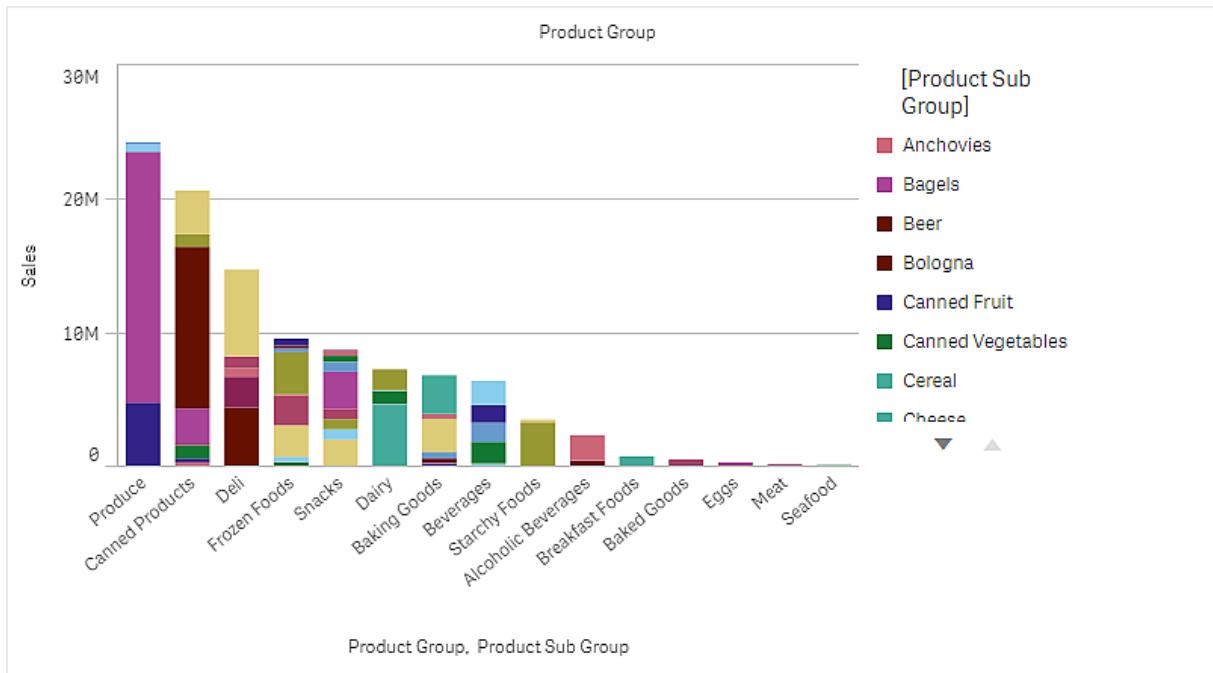
We add a bar chart to the sheet and set the following data properties:

- **Dimension:** Product Group (product group).
- **Measure:** *Sales*; the measure that was created as a master item.

The following bar chart is created, with a bar showing the sales for each product group:



But we want to have some more detailed information about the product sales, by adding the Product Sub Group as a dimension. The Product Sub Group field divides the product groups into sub groups. By default, a grouped chart is selected when adding the second dimension. We want to display a stacked chart instead, that is changed under **Appearances** in the properties panel.



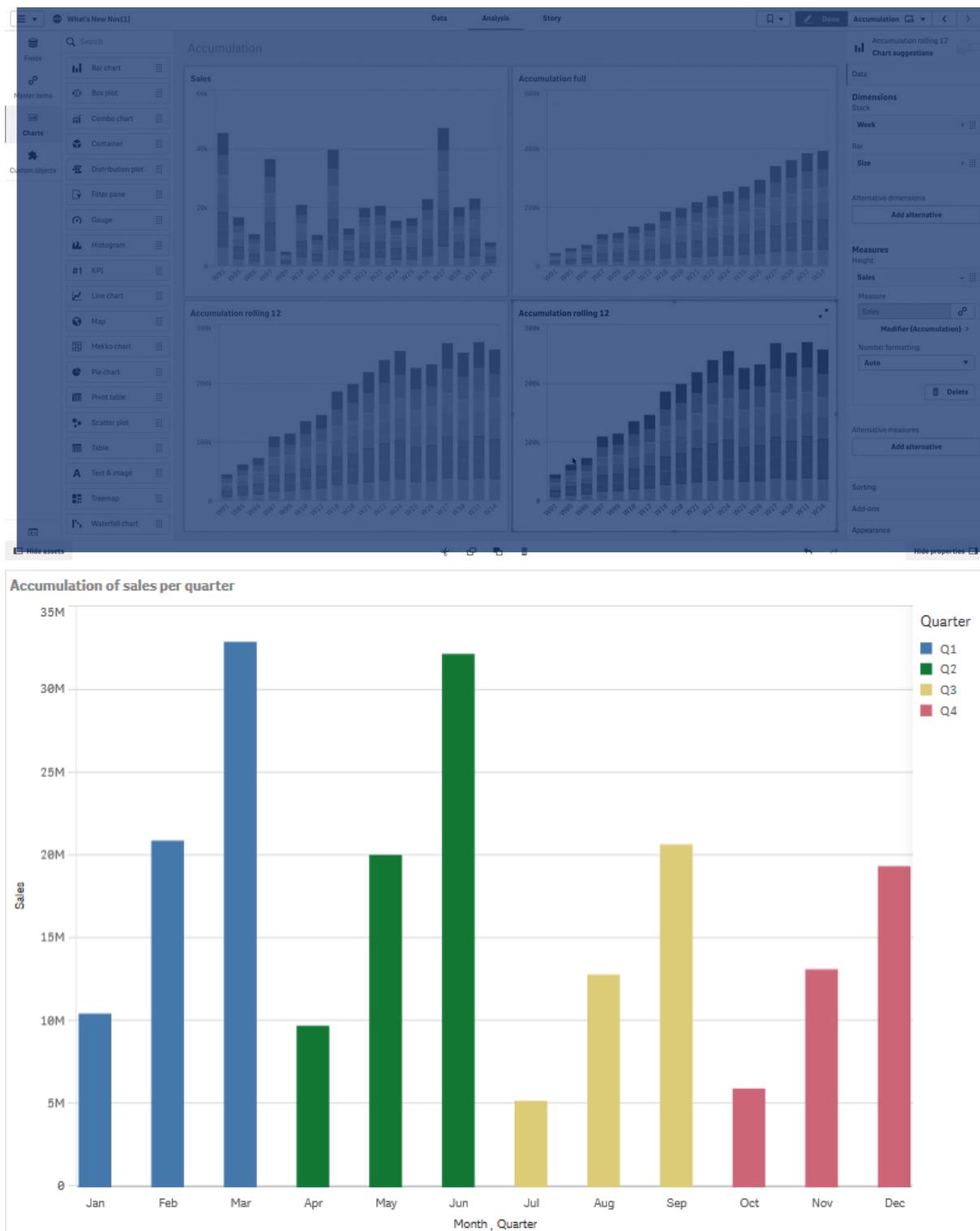
Discovery

The bar chart visualizes the sales volume of different product groups, divided into product sub groups. The visualization is sorted in order of sales volume per product. You can hover the mouse pointer over a product sub group and view the details.

In the bar chart we can see that Produce has the highest sales volume. One of the sub products contribute to most of the Produce sales, if we hover over that part of the bar we can see it is Fresh Vegetables.

Accumulating values over a dimension in a bar chart

This example shows how to use a bar chart to visualize sales data which accumulate over a dimension.



Dataset

In this example, we will use a data file available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial. The file is available in the *Tutorials source* folder: *Sales.xls*

To download the file, go to [Tutorial - Building an App](#).

Add the data file to an empty app. The dataset that is loaded contains sales data.

Measure

We use the sales volume as the measure that we create in Master items:

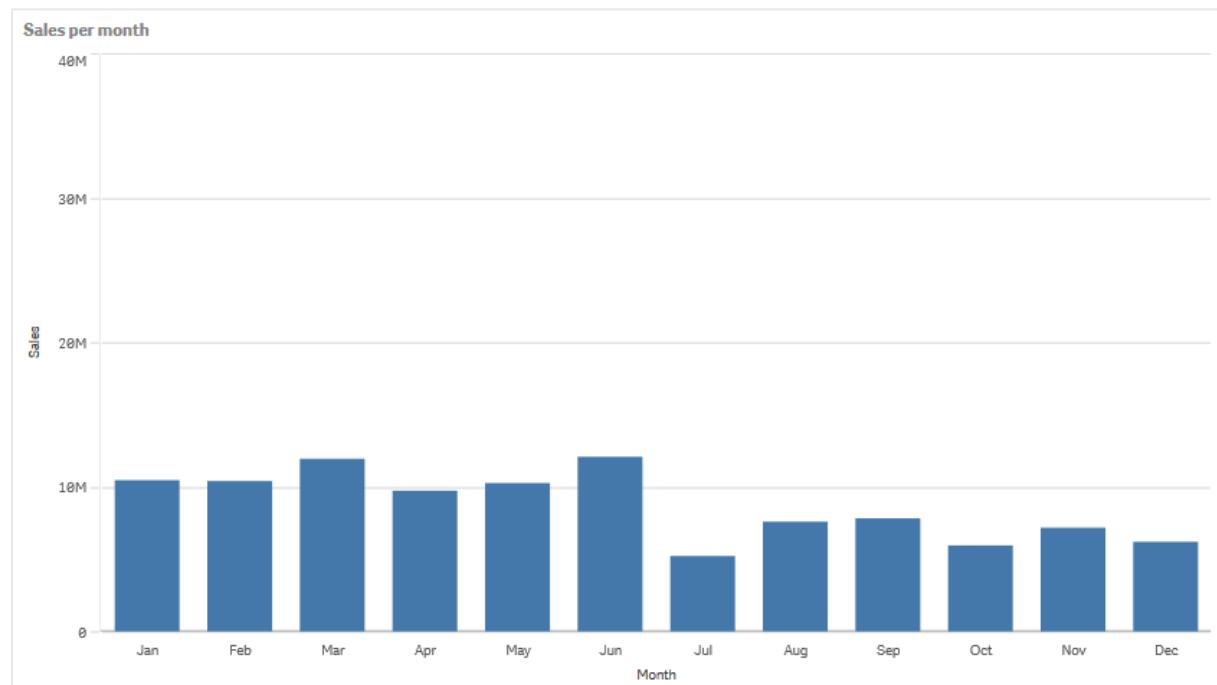
- *Sales* with the expression `sum(sales)`. This is the sum of the sales volume.

Visualization

We add a bar chart to the sheet and set the following data properties:

- **Dimension:** Month (`Date.Month`).
- **Measure:** *Sales*; the measure that was previously created.

The following bar chart is created, with a bar showing the sum of sales for each month.



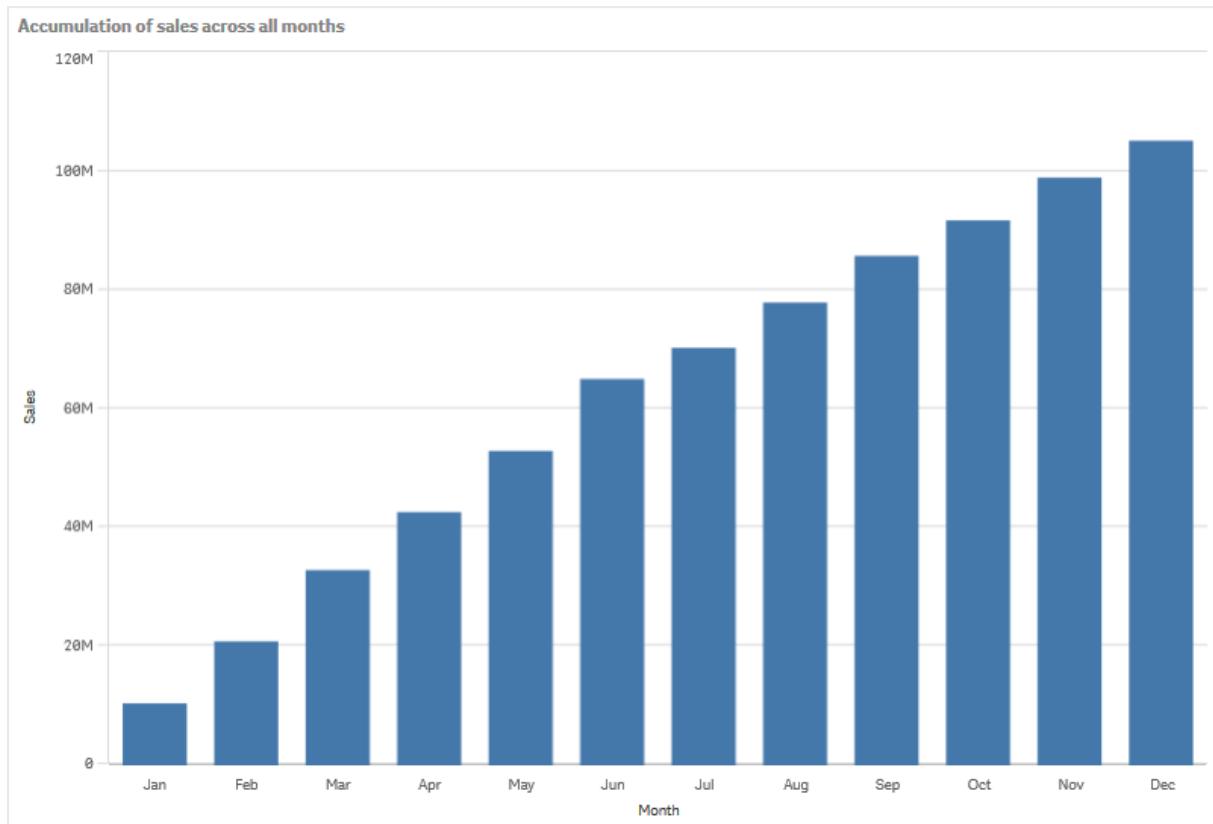
Accumulation

To have our sales data accumulate over one dimension, we need to apply a modifier to our *Sales* measure.

Do the following:

- Under **Measure: Sales** set the **Modifier** to Accumulation. This will set the measure to accumulate over one dimension.

Our bar chart becomes as follows, with the sales accumulating from one month to the next.

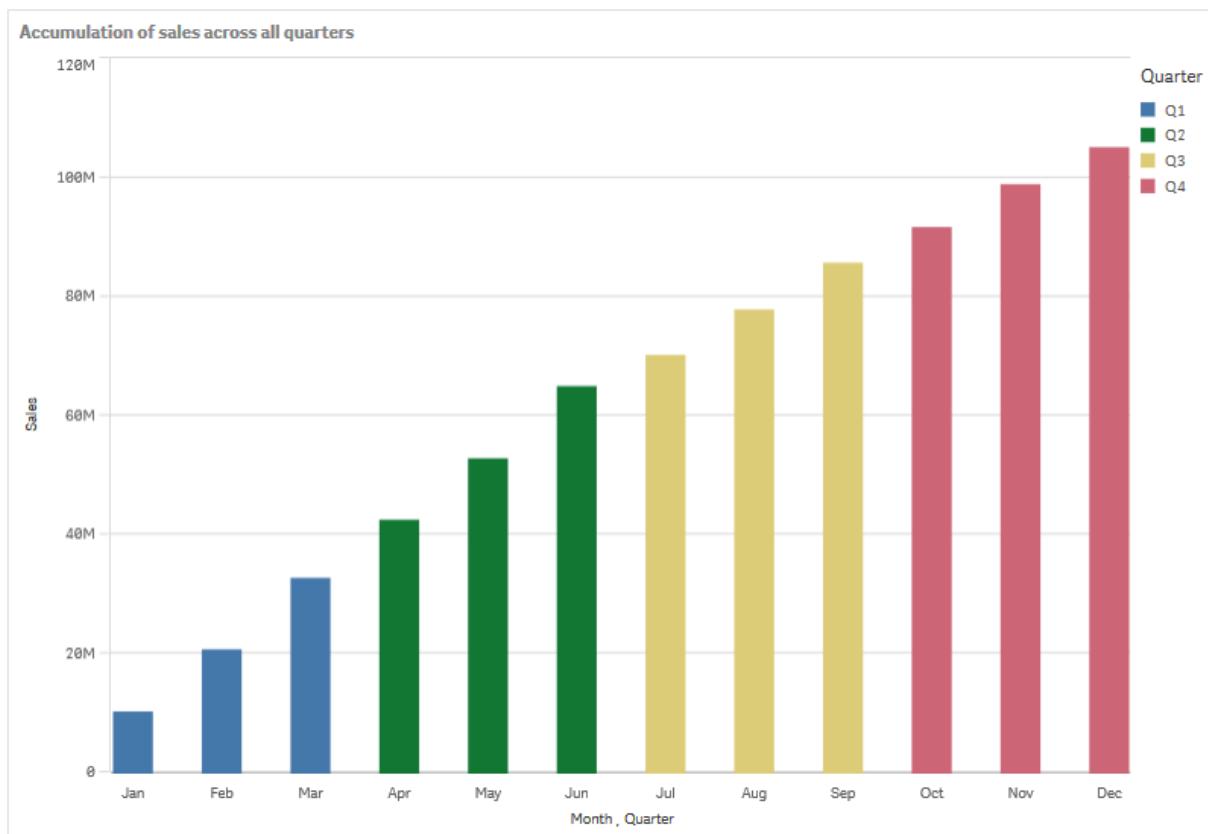


We want to achieve a visual separation of the periods of time, by adding the Quarter as a second dimension. The Quarter field groups the month bars into larger groups. By default, a grouped chart is selected when adding a second dimension.

Do the following:

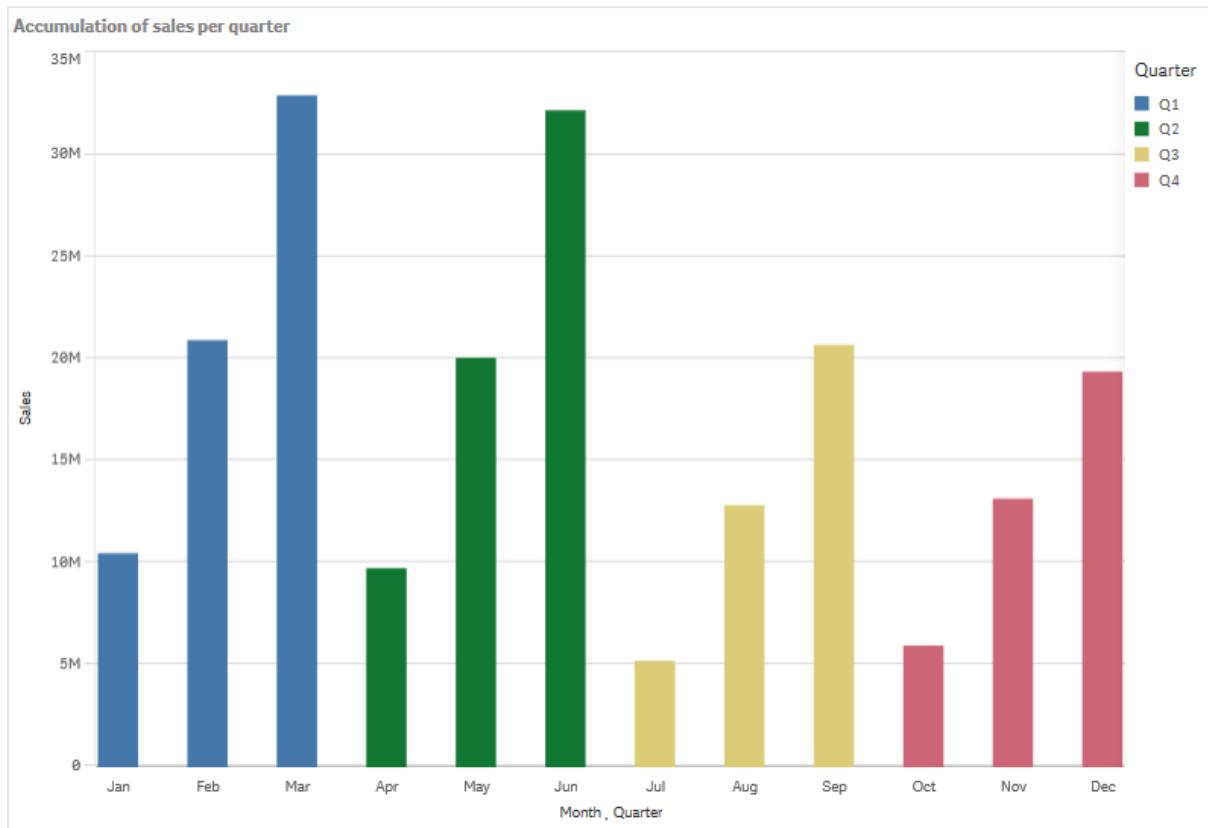
1. Add **Dimension**: Quarter (Date.Quarter).
2. Under **Measure: Sales** set the **Modifier>Dimension** to *Month*. This sets the dimension over which the accumulation will take place.

If **Across all dimensions** is turned on, the accumulation continues across all quarters.



Notice how the bars of the previous two bar charts are the same.

If **Across all dimensions** is turned off, the accumulation starts again at the beginning of each quarter. Our bar chart becomes as follows, with the sales accumulating from one month to the next, inside each quarter.



It is good practice to have the title of your charts represent their content. So consider changing the title of your new bar chart to reflect that the bars are now an accumulation of sales.

Discovery

The bar chart visualizes the sales volume of each month, grouped into quarters. By having the sales accumulate inside each quarter, we get a better understanding of the volume of sales for each quarter. You can hover the mouse pointer over a month and view the details.

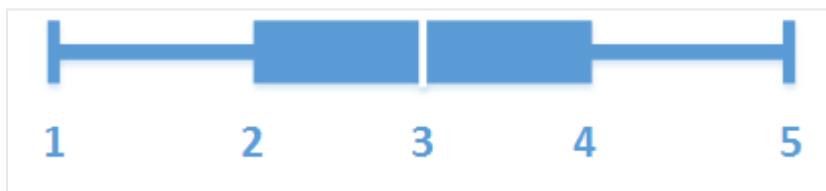
In the bar chart we can see that Q1 has the highest sales volume. We can also see that even though Q4 had the lowest sales volume, the first two months of the quarter were better than the ones from Q3.

Box plot

The box plot is suitable for comparing range and distribution for groups of numerical data, illustrated by a box with whiskers, and a center line in the middle. The whiskers represent high and low reference values for excluding outlier values.



You can define the box start and end points, and whiskers ranges with a few different presets, or define your own settings using expressions.



1. First whisker
2. Box start
3. Center line
4. Box end
5. Last whisker



You can hover over a box to display a popup showing the respective values of the box plot elements.

When to use it

The box plot is suitable for comparing range and distribution for groups of numerical data.

Advantages: The box plot organizes large amounts of data, and visualizes outlier values.

Disadvantages: The box plot is not relevant for detailed analysis of the data as it deals with a summary of the data distribution.

Creating a box plot

You can create a box plot on the sheet you are editing.

In a box plot you need to use one or two dimensions, and one measure. If you use a single dimension you will receive a single box visualization. If you use two dimensions, you will get one box for each value of the second, or outer, dimension.



You cannot use calculated dimensions in a box plot.

Do the following:

1. From the assets panel, drag an empty box plot to the sheet.
2. Add the first dimension.
This is the inner dimension, which defines a box.
3. Add a second dimension.
This is the outer dimension, which defines the boxes shown on the dimension axis.
4. Click **Add measure** and create a measure from a field. The measure does not have to contain an aggregation.

When you have created the box plot, you may want to adjust its appearance and other settings in the properties panel. By default, the **Standard (Tukey)** preset is used.

Changing the definition of the box plot

You can use one of the three presets, found under **Box plot elements** in the properties panel, to define your box plot.

- **Standard (Tukey)**

This preset is based on the original box plot definition by J. Tukey. The center line represents the median (second quartile), and the box start and end points represent the first and third quartiles. Whisker length can be set to 1, 1.5 or 2 inter-quartile ranges. An inter-quartile range represents the difference between the first and third quartiles.

- **Percentile-based**

This preset is also defined with the box start and end points representing the first and third quartiles, and the center line representing the median, but the whisker length is adjusted by setting a percentile based whisker position.

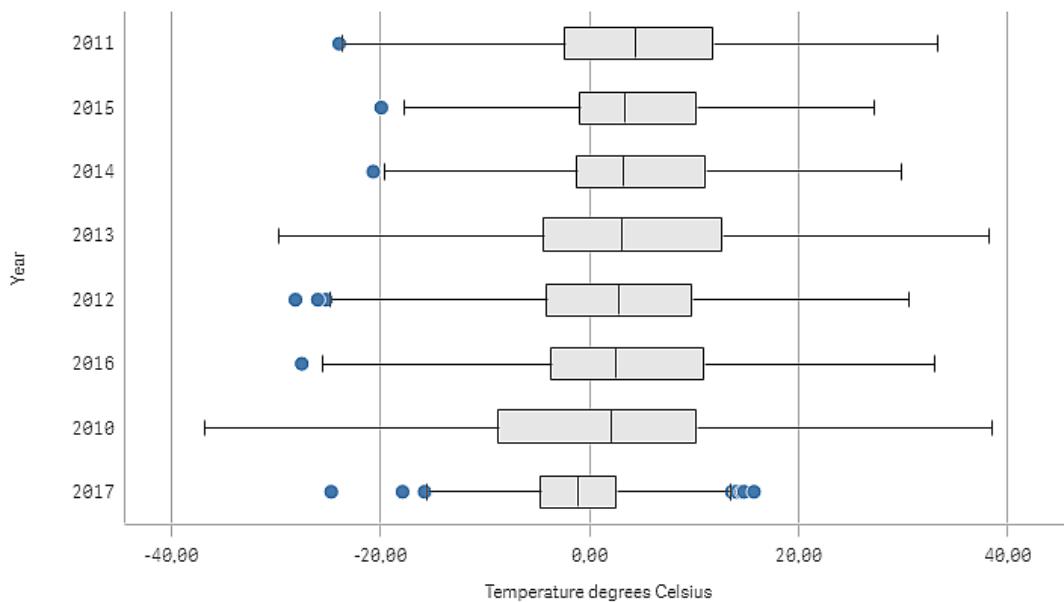
- **Standard deviation**

This preset is based on standard deviations, with the center line representing the average value, and the box start and end points representing one standard deviation variance. You can set the whisker length to a multiple of standard deviations.

You can also define a custom box plot where you set the value of each box plot element using an expression.

Visualizing range and distribution of numerical data with a box plot

This example shows how to make a box plot to visualize range and distribution of numerical data using daily temperature measurements.



Dataset

In this example, we'll use the following weather data.

- Location: Sweden > Gällivare Airport
- Date range: all data from 2010 to 2017
- Measurement: Average of the 24 hourly temperature observations in degrees Celsius

The dataset that is loaded contains a daily average temperature measurement from a weather station in the north of Sweden during the time period of 2010 to 2017.

Measure

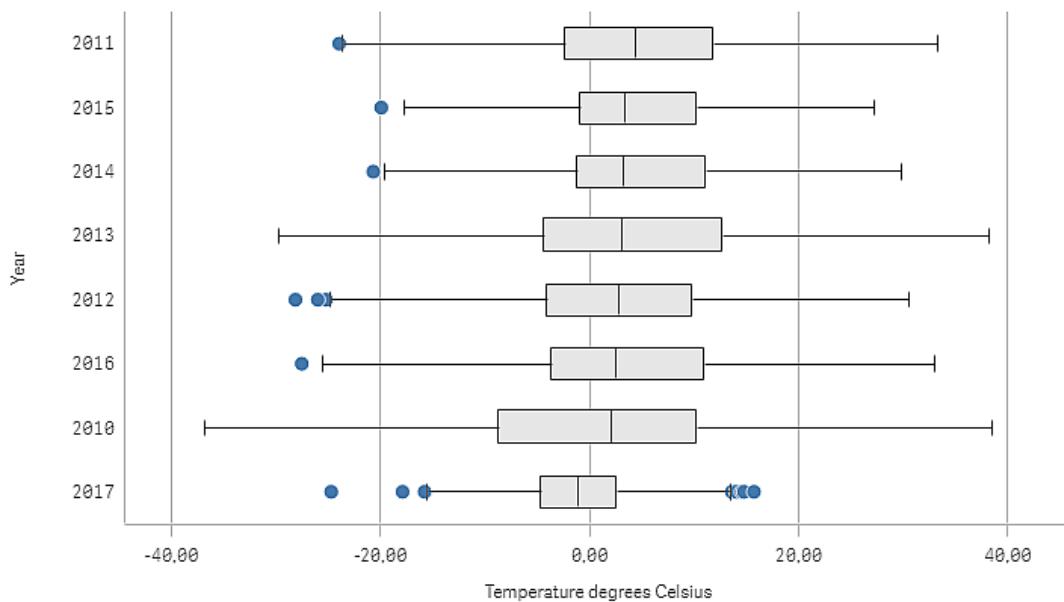
We use the average temperature measurement in the dataset as the measure, by creating a .measure in Master items with the name *Temperature degrees Celsius*, and the expression `Avg([Average of the 24 hourly temperature observations in degrees Celsius])`.

Visualization

We add a box plot to the sheet and set the following data properties:

- **Dimension:** Date (date) and Year (year). The order is important; Date needs to be the first dimension.
- **Measure:** *Temperature degrees Celsius*; the measure that was created as a master item.

In this example we use the default box plot preset, **Standard (Tukey)** with the whisker length **1.5 interquartile range**.



Discovery

The box plot visualizes the distribution of the daily temperature measurements. The visualization is sorted in mean temperature order. The mean temperature for each year is illustrated by the middle line in each box. The box stretches from the first quartile to the third quartile, and the whiskers stretch 1.5 inter-quartile ranges. There are also a number of outlier values, the points that are placed outside the whiskers. You can hover the mouse pointer over an outlier point and view the details.

In the box plot we can see that the year 2010 has the longest box and whiskers. That shows that the year 2010 has the largest distribution of temperatures measured. It also seems to be the coldest year in average.

The range of 2017 is small, as the dataset only contains measurements from the first months of the year.

Bullet chart

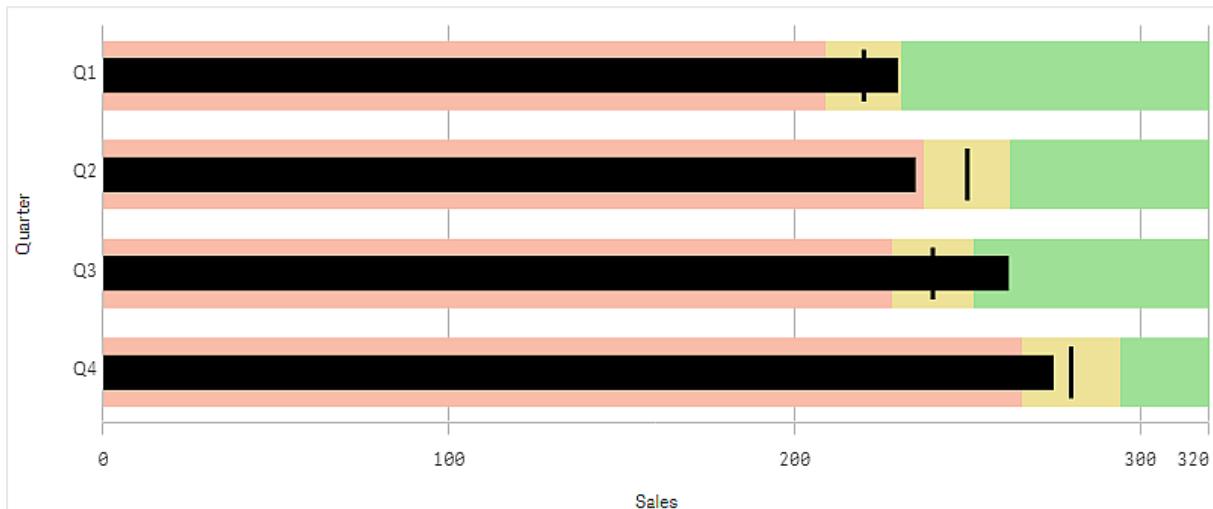
The bullet chart displays a gauge with extended options. Bullet charts can be used to visualize and compare performance of a measure to a target value and to a qualitative scale, such as poor, average, and good.

In a bullet chart you need one measure, which determines the length of the bar.

You can also add a dimension. This will show one gauge for every dimension value. If you do not define a dimension, the chart will show a single gauge.

Example:

A bullet chart showing sales performance for each value of the dimension (quarter)



The example shows a bullet chart with sales performance for each quarter. It also shows the performance relative to target and performance range, which are different for each quarter.

When to use it

Bullet charts let you compare and measure performance with more enriched information than a common gauge. This is helpful when comparing performance according to a target and a simple performance rating. For example: you can show how sales relate to a target value, and in context of poor, good, and stretched performance.

Creating a bullet chart

You can create a bullet chart on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty bullet chart to the sheet.
2. Click **Add dimension** to select the dimension that define how many gauges are shown.
3. Click the **Add measure** button to select the value measure of the chart that defines the length of the bar.

Once the measure is selected the bullet chart is displayed with default settings. Each gauge is displayed with an individual range. If you want to use a common range, you can set it with **Appearance>Y-axis>Common range**.

4. To add a target value, click **Target** under the measure. You can define a fixed value or use a measure with target values.
5. To add performance ranges, set **Use segments** under the measure to **On**. Click **Add limit** to set a range limit for the segments. You can adjust the color of each segment by clicking on it. You can define a fixed limit value or an expression.

The bullet chart is now displayed with the dimension and measure you selected.

Setting a target value

You can add a target value which is displayed as a marker line. If the measure contains sales numbers, this could be budgeted sales for example.

You can define a fixed value or use a measure with target values.

Setting performance ranges

To add performance ranges, set **Use segments** under the measure to **On**.

You need to add the limits for the ranges you want to use for showing indicators with **Add limit**. You can set a limit value in three ways.

- Use the slider.
- Type a value in the text box.
- Set an expression that returns the limit value.

Changing the color scheme

You can change color scheme of the value bar and the target by setting **Appearance > Colors > Colors** to **Custom**. You can set single colors or use an expression.



It's a good idea to use a bar color that is visually dominant to the range colors.

When you have added the limits, you can select the color and the symbol of the indicator for each defined range.

Adding a custom tooltip

You can add measures, charts, and images as tooltips. To add custom tooltips, select **Appearance>Tooltip>Custom**.

- If you want to add a measure as a tooltip, you can add it from a field using **From a field**. You can also use an expression to add a measure. Click **fx** to open the expression editor. The existing expression is displayed by default. You can add a title and description to the tooltip. You can change the label of the tooltip using **Label** as well as its formatting using **Number formatting**.
- If you want to add a chart as a tooltip, select a master visualization from the list of **Master items**.
 - Set the size of the tooltip container. Qlik Sense fits the chart into the container.
 - **Small**. Sets the width and height of the container to 140 pixels.
 - **Medium**. Sets the width and height of the container to 200 pixels.
 - **Large**. Sets the width and height of the container to 340 pixels.



*After you add the chart, you can edit it by clicking **Edit master item**. See *Editing a master visualization* (page 87).*

- If you want to add an image as a tooltip, choose whether to add an image from your **Media library** or from a **URL**.
 - Set the size of the tooltip container. Qlik Sense fits the image into the container.
 - **Small**. Sets the width and height of the container to 50 pixels.
 - **Medium**. Sets the width and height of the container to 200 pixels.
 - **Large**. Sets the width and height of the container to 340 pixels.
 - **Original**. Qlik Sense fits the image into the container. If the image is larger than 340 pixels, it is scaled down.
 - **Media library**: Appears when you choose **Media library** for **Type**. Click to select an image from your media library.
 - **URL**: Appears when you choose **URL** for **Type**. Enter a **URL**.

Setting the scale of the axis

If you use a dimension to show several gauges, you can select how to show the scale of the axis with **Appearance>Y-axis>Common range**.

- If you want each dimension gauge to use the same scale, enable **Common range**. If the range measure depends on the dimension value, the range bars will have different lengths.
This is useful when you want to be able to compare the actual values.
You can also set a common axis for all gauges with **Appearance>Y-axis>Common range**.
- If you want each range bar to be equally long, disable **Common range**.
This is useful when you want to be able to compare the relative performance of each dimension value.

Changing the orientation of the labels

To change the orientation of the chart's labels, the chart's presentation must be vertical. This can be set with **Appearance > Presentation > Vertical**. Label orientation can then be selected with **Appearance > X-axis > Label orientation**. The following orientations are available:

- **Auto**: Automatically selects one of the other options depending on the space available on the chart.
- **Horizontal**: Labels are arranged in a single horizontal line.
- **Tilted**: Labels are stacked horizontally at an angle.
- **Layered**: Labels are staggered across two horizontal lines.

To view examples of label orientation, see *X-axis and Y-axis (page 458)*.

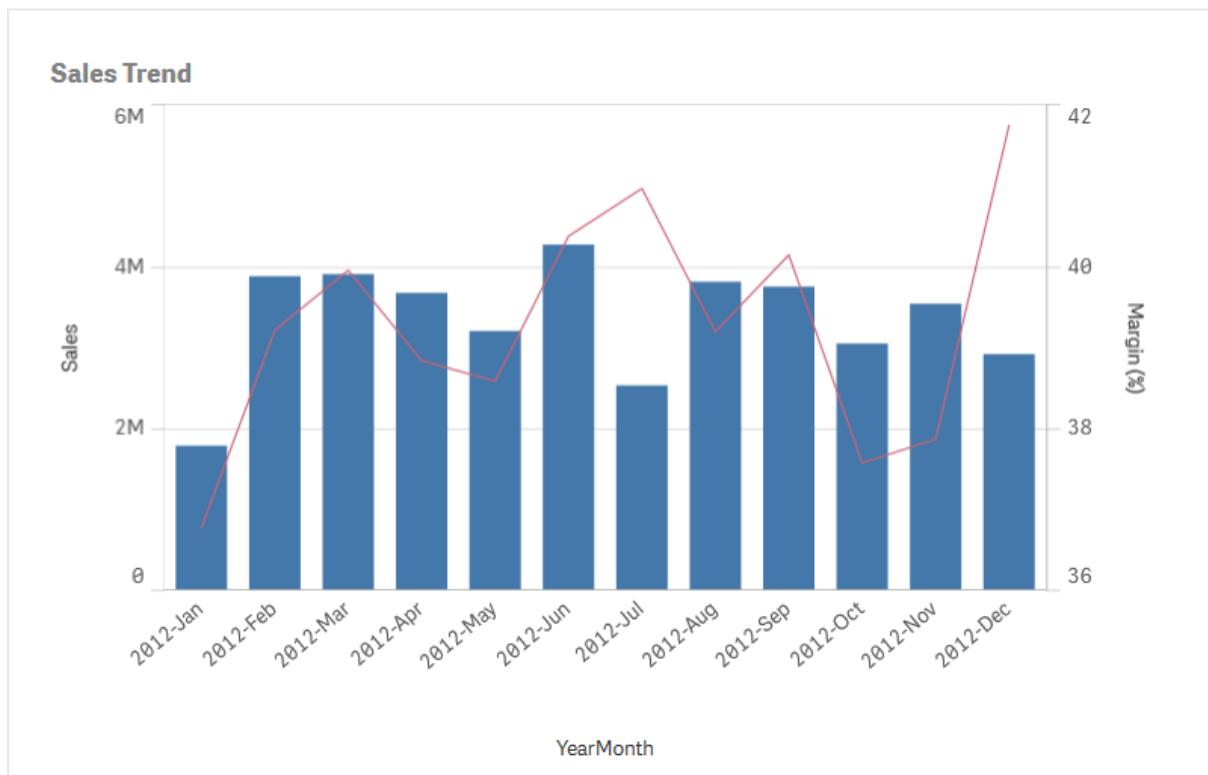
Combo chart

The combo chart is suitable for comparing two sets of measure values that are usually hard to compare because of the differences in scale. It is basically a bar chart combined with a line chart.



A typical example is when you have a bar chart with sales figures and want to combine these figures with the margin values (in percent). In a regular bar chart, the bars for sales would be displayed as usual, but the margin values would be almost invisible because of the very large difference between the numeric values for sales and margin.

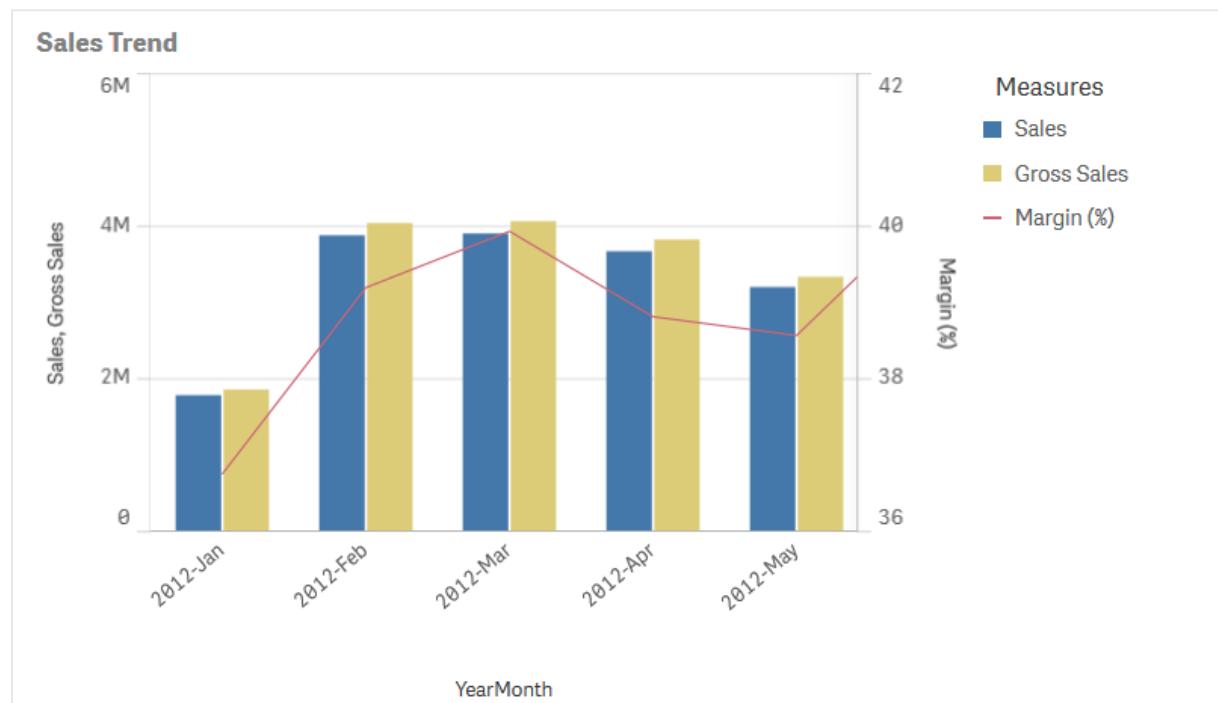
A combo chart with the margin values (in percent) and bars with sales figures.



With a combo chart you can combine these values by, for example, using bars for the sales values and a line for the margin values. By default, the bars have the measure axis on the left and the margin values have a separate axis to the right. The two measures use the same dimension (YearMonth).

If you have yet another measure, for example, gross sales, with values that are roughly in the same range as the sales values, you can add the third measure as bars and either stack or group the new measure values with the sales values. With grouped bars, you can easily compare two or more items in the same categorical group. Stacked bars combine bars of different groups on top of each other and the total height of the resulting bar represents the combined result.

A combo chart with three measures; the margin values (in percent), bars with sales figures and the measure Gross sales grouped with the sales values.



The combo chart can only be displayed vertically.

When to use it

With the possibility to have different measure scales, one to the left and one to the right, the combo chart is ideal when you want to present measure values that are normally hard to combine because of the significant difference in value ranges.

But a combo chart can also be quite useful when comparing values of the same value range. In the image above, the combo chart only has one measure axis, but the relationship between the two categories sales and cost is clear.

Advantages

The combo chart is the best choice when combining several measures of different value ranges.

Disadvantages

The combo chart only supports one dimension, and can therefore not be used when you need to include two or more dimensions in the visualization.

Creating a combo chart

You can create a combo chart on the sheet you are editing. In a combo chart, you need at least one dimension and one measure.

Do the following:

1. From the assets panel, drag an empty combo chart to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field. Select to show the measure as a bar.
4. Add another measure by selecting **Add** under **Height of line**. Enter an expression, or master measure item, or a field with an aggregation function applied. By default a line will appear as the measure. You can select **More Properties** to choose **Presentation** of the measure as either bars, line, or marker. You can select drop-down options to switch between the **Primary axis** to the left or the **Secondary axis** to the right (right and left axes are reversed if **Right-to-left** is turned on in **App Settings**). For markers you can choose between several different shapes.

You can only have one dimension, but you can continue adding up to 15 measures. You can only have two measure axes though. This means, if you add three or more measures with a large difference in value range it can be hard to display all measures with a good distribution of values.

When you have created the combo chart, you may want to adjust its appearance and other settings in the properties panel.

Display limitations

Displaying out of range values

In the properties panel, under **Appearance**, you can set a limit for the measure axis range. Without a limit, the range is automatically set to include the highest positive and lowest negative value, but if you set a limit you may have values that exceed that limit. A bar that exceeds the limit will be cut diagonally to show that it is out of range. For a line data point value that is out of range, an arrow indicates the direction of the value.

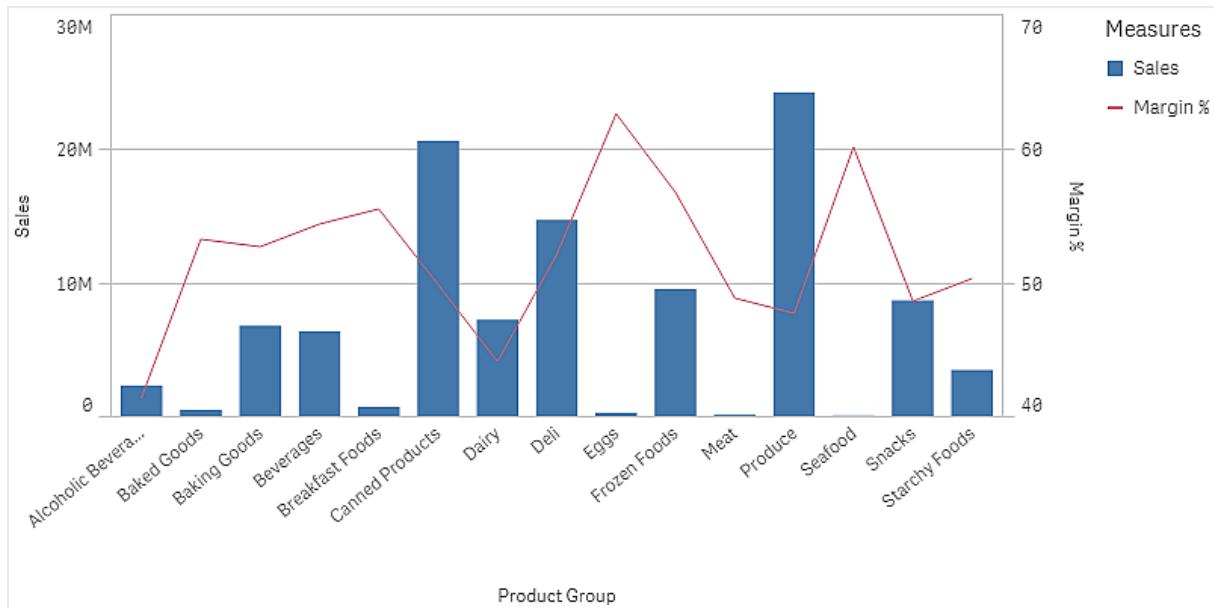
Displaying large amounts of data in a combo chart

If the chart uses a continuous scale, a maximum of 2000 data points are displayed. Above that number, data points are neither displayed, nor included in selections made in the chart.

To avoid displaying limited data sets, you can either make a selection or use dimension limits in the properties panel.

Comparing measures with a different scale using a combo chart

This example shows how to make a combo chart that visualizes sales data. You will also compare different product groups against two measures with a different scale.



Dataset

In this example, we will use two data files from the Qlik Sense Tutorial - Building an App. To download the files, go to [Tutorial - Building an App](#). Download and unzip the tutorial, and find the files in the *Tutorials source* folder:

- *Sales.xls*
- *Item master.xls*

Create a new app, and add the two data files. Make sure that they are associated by *Item Number*.

The dataset that is loaded contains sales data. The *Item master* table holds the information about the items ordered, such as product groups.

Measures

We need to create two measures in Master items:

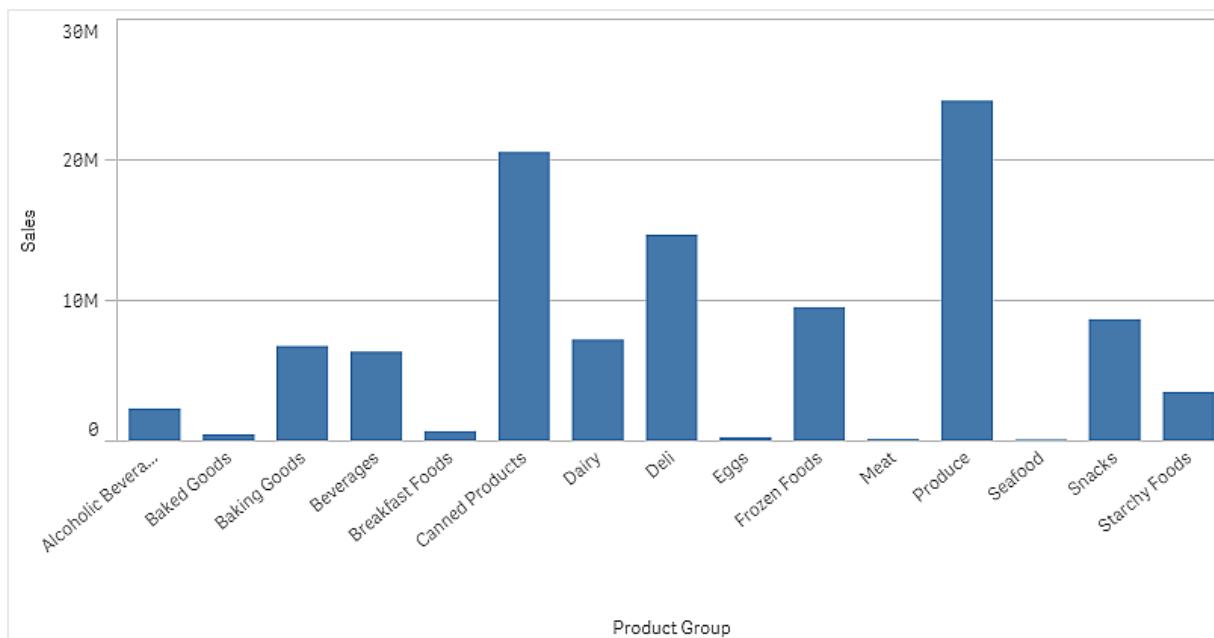
- Sales volume: with the name *Sales*, and the expression `sum(Sales)`.
- Sales margin in percent: with the name *Margin %*, and the expression `Avg(Margin/sales)*100`.

Visualization

We add a combo chart to the sheet and set the following data properties:

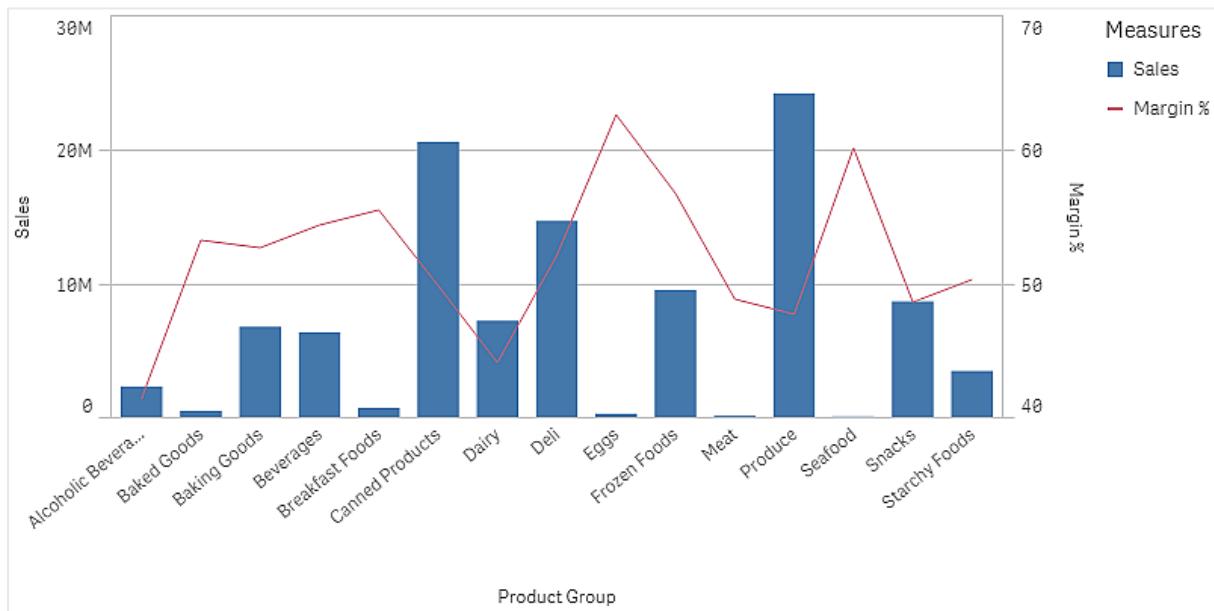
- **Dimension:** Product Group (product group).
- **Measure:** *Sales* (the master measure you created).

The following chart is created, with a bar showing the sales for each product group. It is a bar chart at this stage.



But we also want to show the sales margin, which has a different scale than the sales volume. Sales volume is in the scale of millions, while the margin is a percentage between 0 and 100. If we add margin as a bar next to sales volume, it would be too small to distinguish.

In the properties pane, go to **Measures > Height of line**. Use the drop down to add *Margin %* as a measure.



Discovery

The combo chart visualizes the sales volume and margin of different product groups. You can hover the mouse pointer over a product group and view the details. The bars show the sales volume with the scale to the left, and the line shows the margin with the scale to the right.

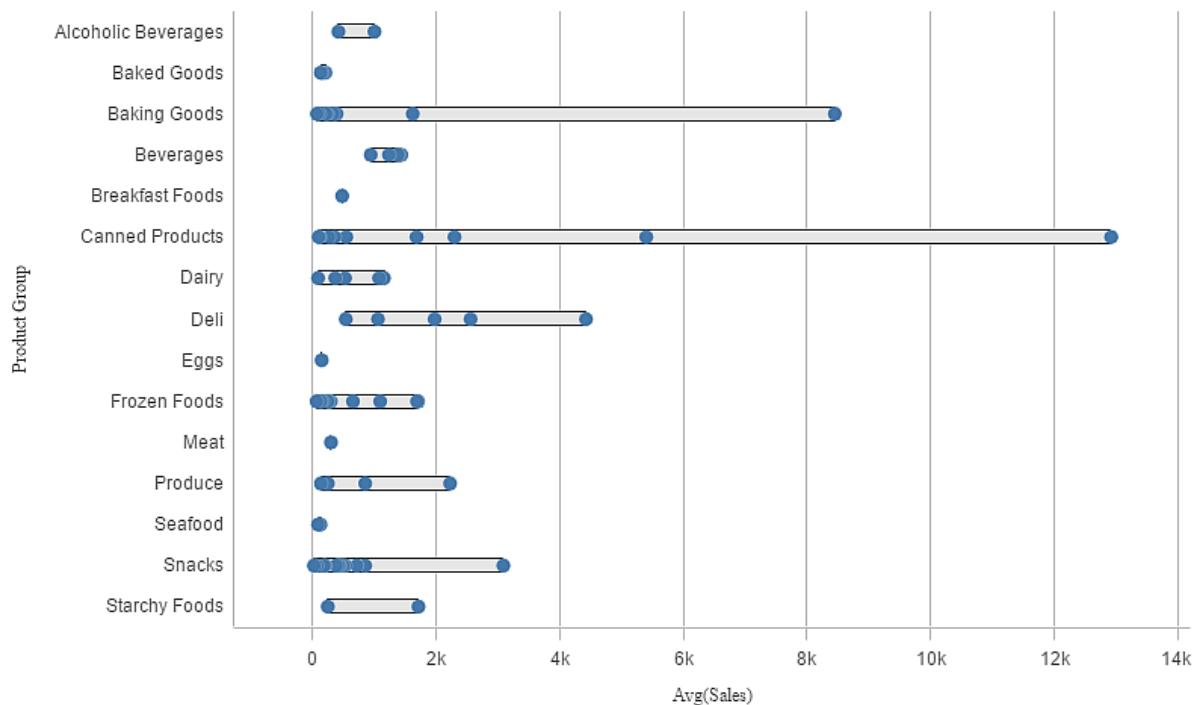
In the chart we can see that Produce and Canned Products have the highest sales volumes. Both groups have a lower margin than most other product groups.

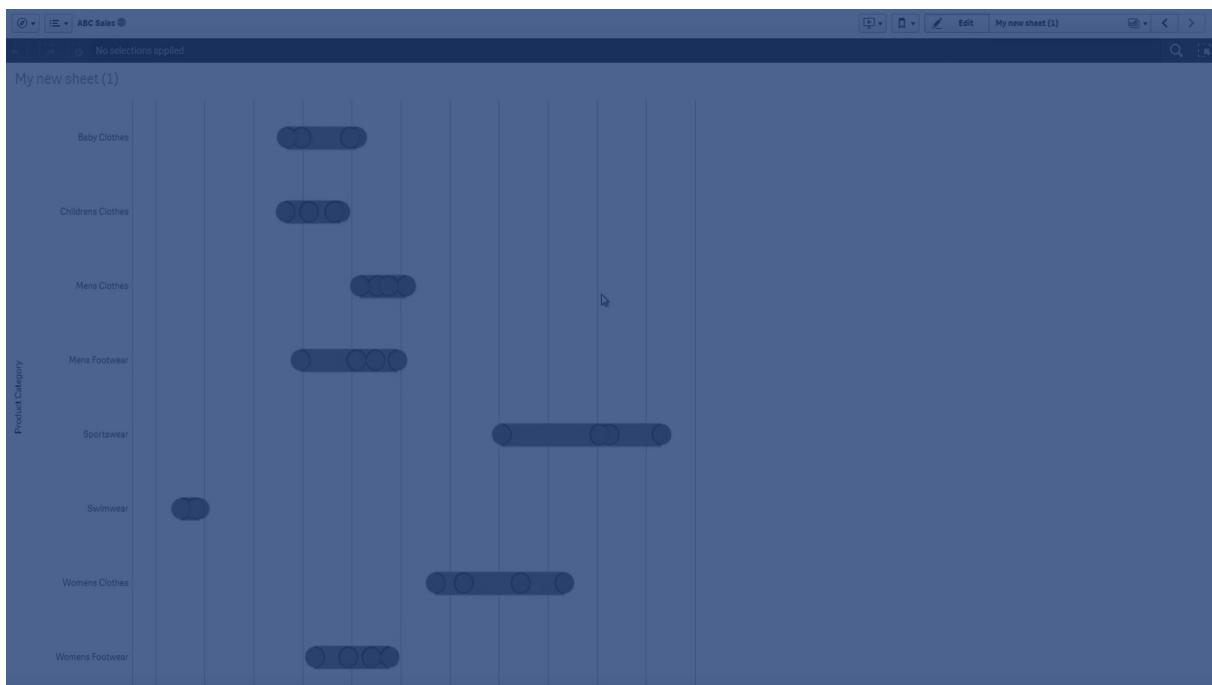
Some product groups with low sales volume, like Eggs and Seafood, have a significantly higher margin.

Distribution plot

The distribution plot is suitable for comparing range and distribution for groups of numerical data. Data is plotted as value points along an axis.

You can choose to display only the value points to see the distribution of values, a bounding box to see the range of values, or a combination of both as shown here:





When to use it

The distribution plot is suitable for comparing range and distribution for groups of numerical data.

Advantages

The distribution plot visualizes the distribution of data.

Disadvantages

The distribution plot is not relevant for detailed analysis of the data as it deals with a summary of the data distribution.

Creating a distribution plot

You can create a distribution plot on the sheet you are editing.

In a distribution plot you need to use one or two dimensions, and one measure. If you use a single dimension you will receive a single line visualization. If you use two dimensions, you will get one line for each value of the second, or outer, dimension.

Do the following:

1. From the assets panel, drag an empty distribution plot to the sheet.
2. Add the first dimension.
This is the inner dimension, which defines the value points.
3. Add a second dimension.
This is the outer dimension, which defines the groups of value points shown on the dimension axis.
4. Click **Add measure** and create a measure from a field.

Viewing the distribution of measure values in a dimension with a distribution plot (page 170)

When you have created the distribution plot, you may want to adjust its appearance and other settings in the properties panel.

Display limitations

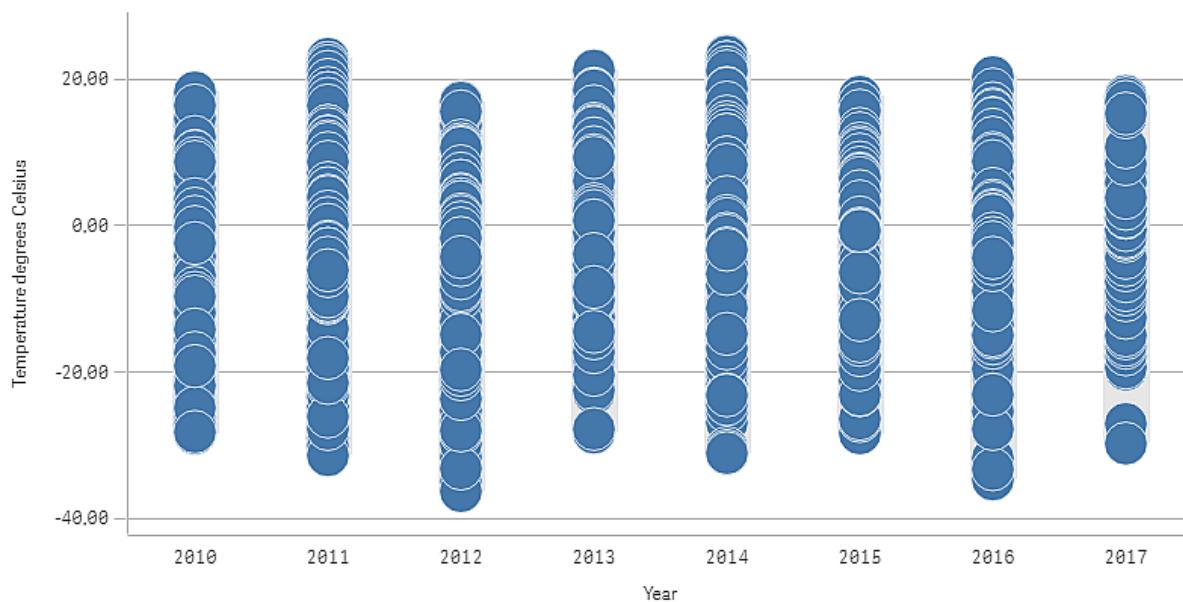
Displaying large amounts of data in a distribution plot

When displaying large amounts of data in a distribution plot, the message "**Currently showing a limited data set.**" is shown to indicate that not all data is displayed.

- If the chart uses more than one dimension, 3000 data points are displayed.

Viewing the distribution of measure values in a dimension with a distribution plot

This example shows how to make a distribution plot to view the distribution of measure values in a dimension, using weather data as an example.



Dataset

In this example, we'll use the following weather data.

- Location: Sweden > Gällivare Airport
- Date range: all data from 2010 to 2017
- Measurement: Average of the 24 hourly temperature observations in degrees Celsius

The dataset that is loaded contains a daily average temperature measurement from a weather station in the north of Sweden during the time period of 2010 to 2017.

Measure

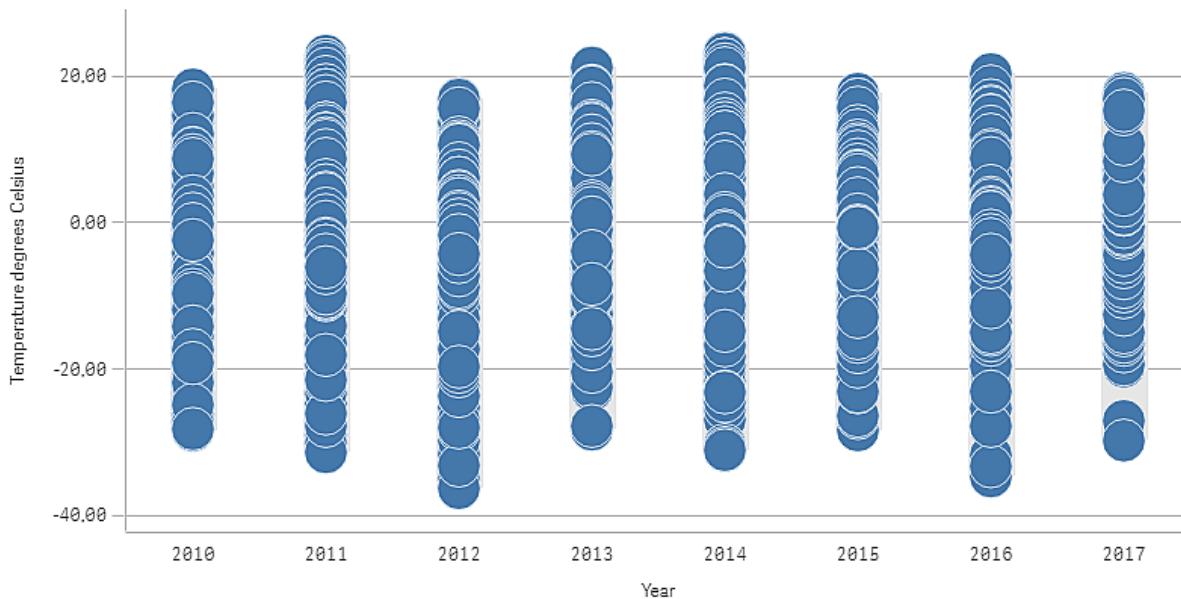
We use the average temperature measurement in the dataset as the measure, by creating a .measure in Master items with the name *Temperature degrees Celsius*, and the expression `Avg([Average of the 24 hourly temperature observations in degrees celsius])`.

Visualization

We add a distribution plot to the sheet and set the following data properties:

- **Dimension:** Date (date) and Year (year). The order is important, Date needs to be the first dimension.
- **Measure:** *Temperature degrees Celsius*, the measure that was created as a master item.

Distribution plot with the dimensions Date (date) and Year (year) and the measure Temperature degrees Celsius.



Discovery

The distribution plot visualizes the distribution of the daily temperature measurements. The visualization is sorted by year, and each point represents a temperature measurement.

In the visualization we can see that the year 2012 has the lowest extreme temperature measurement, close -40 degrees Celsius. We can also see that the year 2016 seems to have the largest distribution of temperature measurements. With this many points in the distribution plot, it can be hard to spot clusters and outliers, but the year 2017 has two low temperature measurements that stand out. You can hover the mouse pointer over a point and view the details.

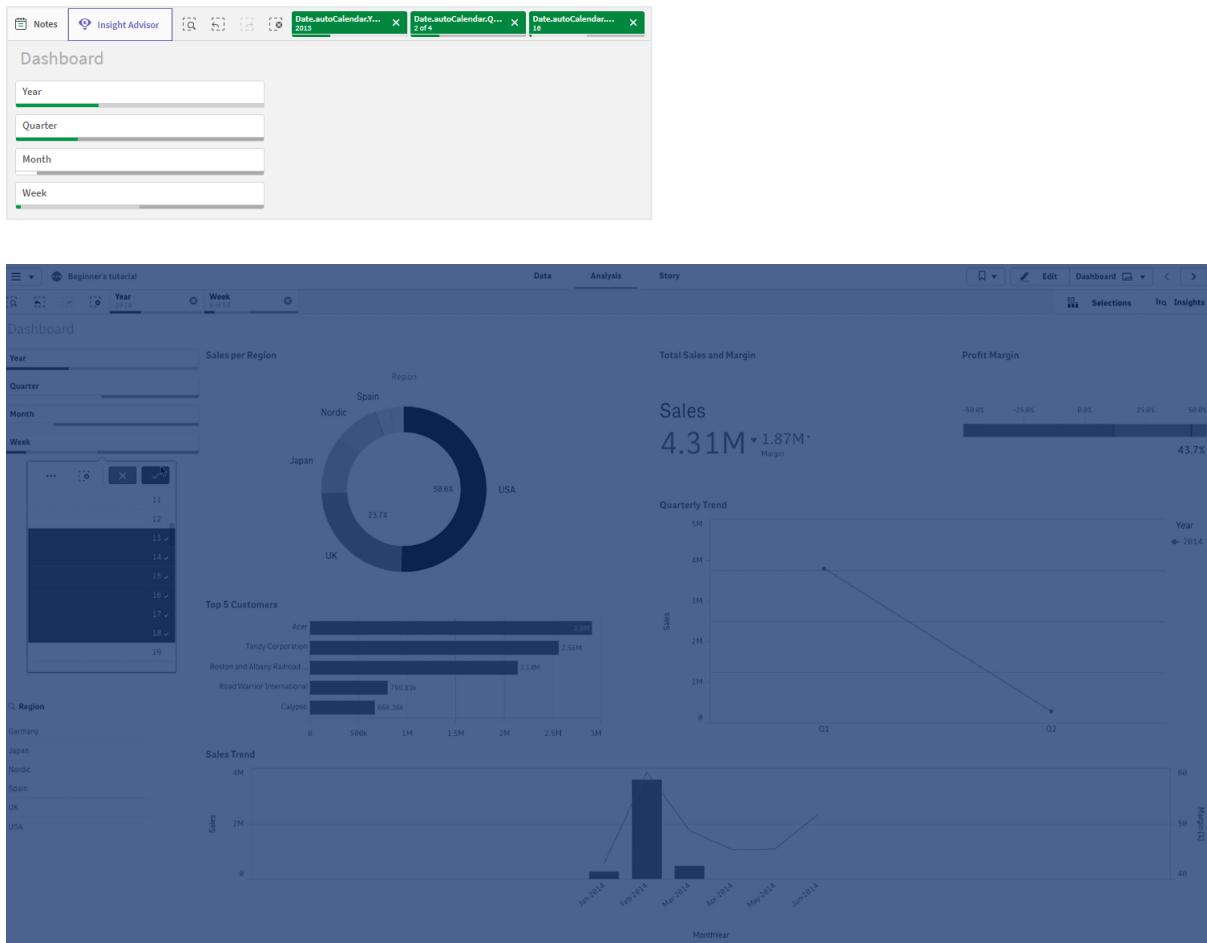
Filter pane

You can add a filter pane to control what data that is shown in the visualizations on a sheet. A filter pane can filter the data of several dimensions at once.

For example, if you have a chart of sales over time, you can use a filter pane to limit the data in the chart to only show sales from a selected time period, from certain product categories, and from a certain region.

When a dimension is added, it is placed to the right of the previous dimensions, or below, depending on the available space. As long as there is enough space, the dimensions are displayed as expanded lists. If there is not enough space, the dimensions that were added first are turned into filter panes.

Selections have been made in the dimensions Year, Quarter, and Week.



When to use it

With filter panes, you can easily make several selections to define your data set exactly like you want it. With your well-defined data set, you can explore data of particular interest.

By using the selection menu options in the filter panes (select possible, select alternative, and select excluded), you can make adjustments to the data set and compare the results with the previous selection.

Advantages

Filter panes are good for making selections and defining data sets. But they also show the relationship between different values, the associations. The green, white, and gray colors reflect the data associations that exist - and that do not exist. And by analyzing those associations, you can make new discoveries, for example, that a sales representative has too many customers, or that a region lacks a sales representative.

Disadvantages

When the dimensions contain a very large amount of values, it may be hard to manage the data.

Creating a filter pane

You can create a filter pane on the sheet you are editing.

In a filter pane you can use up to 1000 dimensions.

Do the following:

1. From the assets panel, drag an empty filter pane to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. If you want to add more dimensions, click **Add dimension** again.

When you have created the filter pane, you may want to adjust its appearance and other settings in the properties panel.



If you double-click or drag a field or a dimension from the assets panel, a filter pane is added to the sheet using the dimension. If you then double-click more dimensions, they are automatically added to the new filter pane.

Configuring filter panes

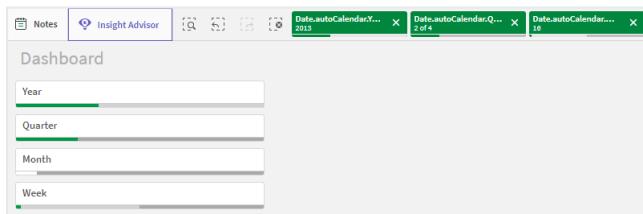
In the properties panel, you can customize the look and feel of the filter pane. Listboxes for each dimension can be customized individually.

Selections in filter panes

During analysis you click a compressed filter pane dimension to open a selection list.

When you make a selection, it is reflected in the small bars at the bottom of each filter pane dimension. Four states can be displayed in the bars: selected (green), possible (white), alternative (light gray), and excluded (dark gray). Locked values are indicated by a lock icon. The details of the selections are displayed in the selections bar, above the sheet. You can click an item to see the details and change your selection.

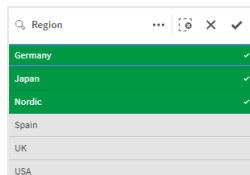
Fields are filtered out from each dimension to be shown in the visualizations on the sheet.



Making selections in filter pane lists

When there is space enough in a filter pane, the dimension values are displayed in a list. In lists, you can click to select a single value or draw to select several values. On a touch device, you can two-finger-tap in the list to select a range of values.

Germany, Japan, and Nordic selected in the Region filter pane.



The selections tool

The selections tool offers an option to get an overview of the fields and dimensions in an app. In the selections tool you can make selections in all the fields and dimensions in the app, regardless of whether they are used in the app or not.

During analysis, click **Selections** to open selections view.

Display limitations

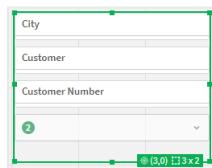
Responsive design

The filter pane has a responsive design and renders as many dimensions as possible. When space is limited, this could involve reducing the size of each dimension so that all dimensions are displayed.

Example:

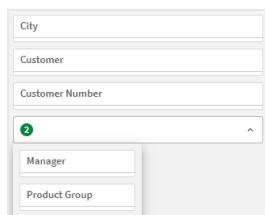
The following image shows a filter pane while it is being edited. Only three out of five dimensions are displayed. The other dimensions are represented by a dropdown list with a numeric value indicating how many dimensions are not displayed.

Filter pane in edit mode.



When you have finished editing the filter pane and enter analysis mode, you will see the filter pane with all the dimensions displayed. If all items cannot be shown due to lack of space, the dropdown list below the displayed dimensions can be expanded to show additional dimensions.

Filter pane in analysis mode, with expanded dropdown list showing additional dimensions.



Full screen view

In full screen view, the filter pane is maximized and displays as many dimensions as possible expanded. When not all dimensions can be displayed expanded, the priority order is that the most recently added dimensions are expanded to the right. You can change the priority order in the properties panel, under **Dimensions**. Drag the dimensions to change the order.

Keyboard navigation

You can use your keyboard to navigate a filter pane. Use the arrow keys or the Tab key to alternate between the different objects on the sheet. Once you have moved the focus to the filter pane, use the Enter, Space and Esc keys to move the focus between the following levels:

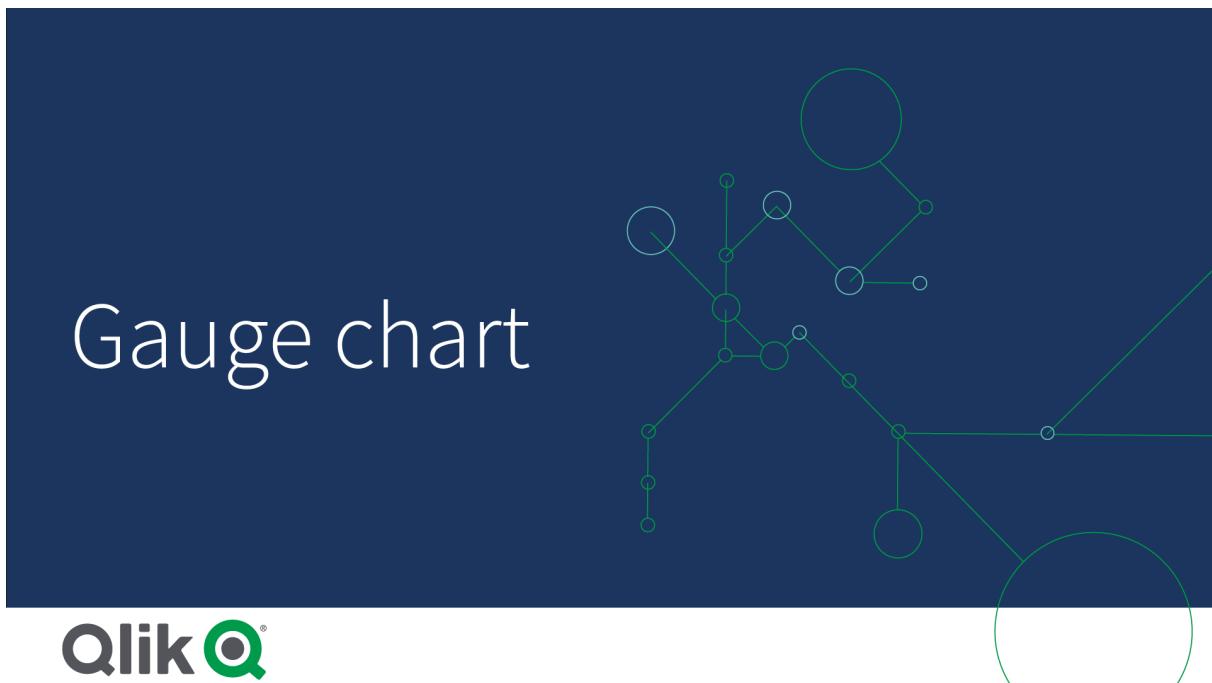
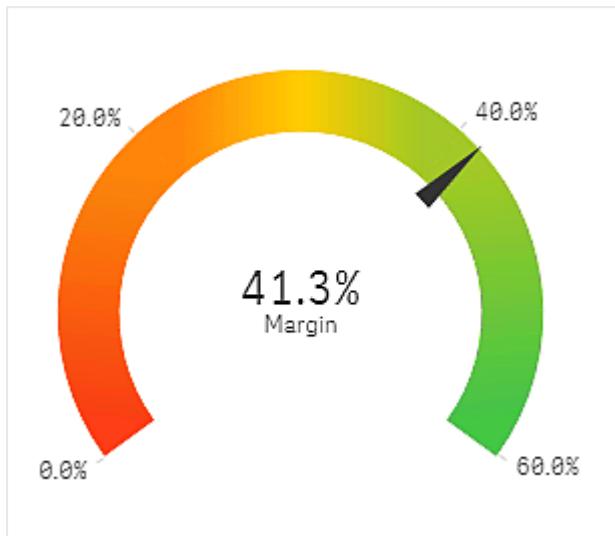
- Entire filter pane
- Listboxes inside the filter pane
- Values inside each listbox

Keyboard controls in filter panes

Keyboard navigation	Description
Space or Enter	Moves focus to the first listbox in the filter pane. Another key press brings the focus to the first value inside that listbox. If the focus is on a collapsed listbox, press Enter to open the listbox in a pop-over.
Esc	Moves focus back one level (from the values to the listbox to the filter pane object).
Right arrow	If the focus is on an individual listbox within the filter pane, the right arrow key moves the focus to the next listbox on the right. If the focus is on a value inside a listbox, the right arrow key moves to the next value in the listbox.
Left arrow	If the focus is on an individual listbox within the filter pane, the left arrow key moves the focus to the next listbox on the left. If the focus is on a value inside a listbox, the left arrow key moves to the previous value in the listbox.
Up arrow	If the focus is on an individual listbox within the filter pane, the up arrow key moves the focus to the previous listbox (to the left, or above, the current listbox). If the focus is on a value inside a listbox, the up arrow key moves to the previous value in the listbox.
Down arrow	If the focus is on an individual listbox within the filter pane, the down arrow key moves the focus to the next listbox (to the right, or below, the current listbox). If the focus is on a value inside a listbox, the down arrow key moves to the next value in the listbox.

Gauge

The gauge shows a single measure value and visualizes how to interpret that value.



When to use it

The gauge is often used to present KPIs, for example, on an executive dashboard, and together with segmenting and color coding, it is an effective way of illustrating a performance result.

It is important to set relevant max and min values to support the interpretation of the value. You can use a reference line to provide additional context.

Advantages

A gauge is easy to read and understand and gives an instant indication of the performance within an area.

Disadvantages

The gauge is quite space-demanding in relation to the single value it visualizes.

Although visually compelling, the gauge is not always the best choice for presenting a single measure value. Problems when deciding the max and min values can indicate that some other visualization should be used.

If you only want to show a performance value, without a gauge, consider using a KPI instead.

Creating a gauge

You can create a gauge on the sheet you are editing. In a gauge you can only have one measure and no dimensions.

Do the following:

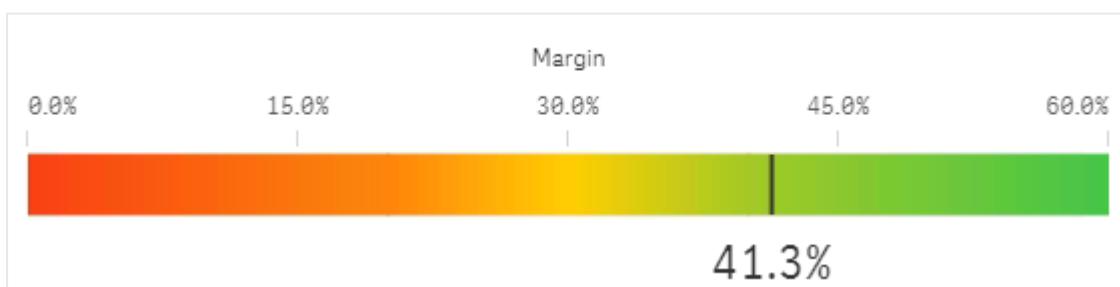
1. From the assets panel, drag an empty gauge to the sheet.
2. Click **Add measure** and select a measure or create a measure from a field.

When you have created the gauge, you may want to adjust its appearance and other settings in the properties panel.

The following settings are used by default in a gauge:

- A radial gauge.
- A single (blue) color.
- Range limits: min (0), max (100).
- No segments.
- Label and title are displayed in medium scale.

For example, you can change the radial gauge to a bar, and use a color gradient.

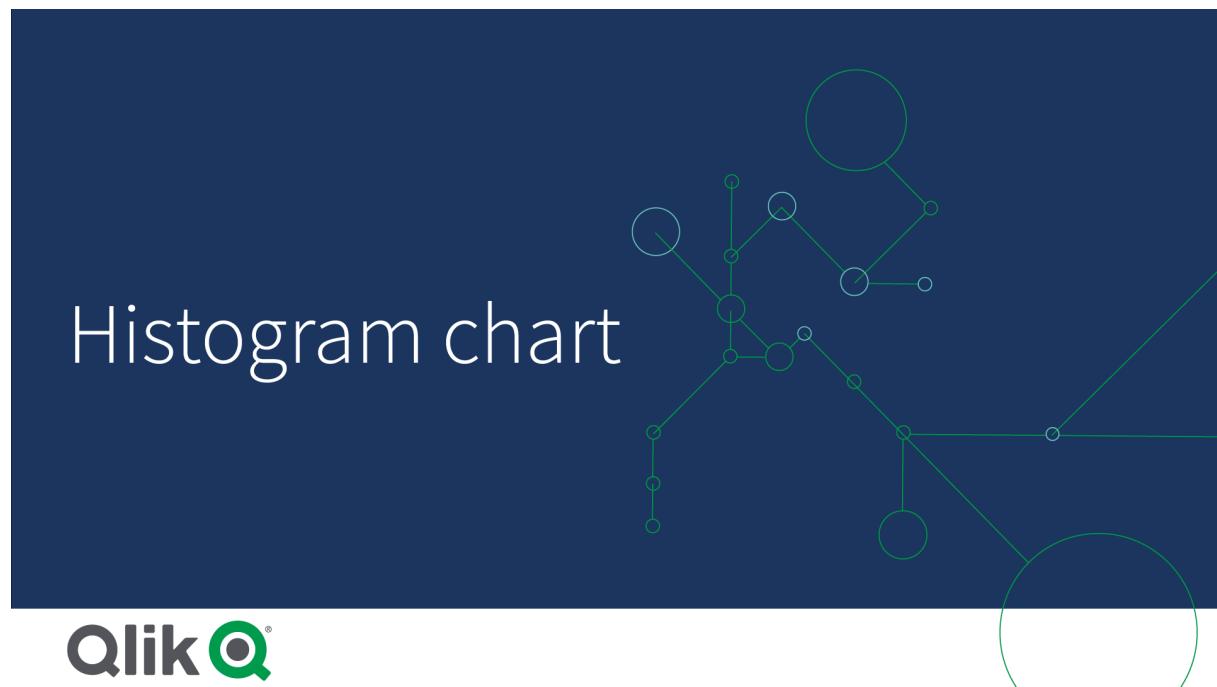
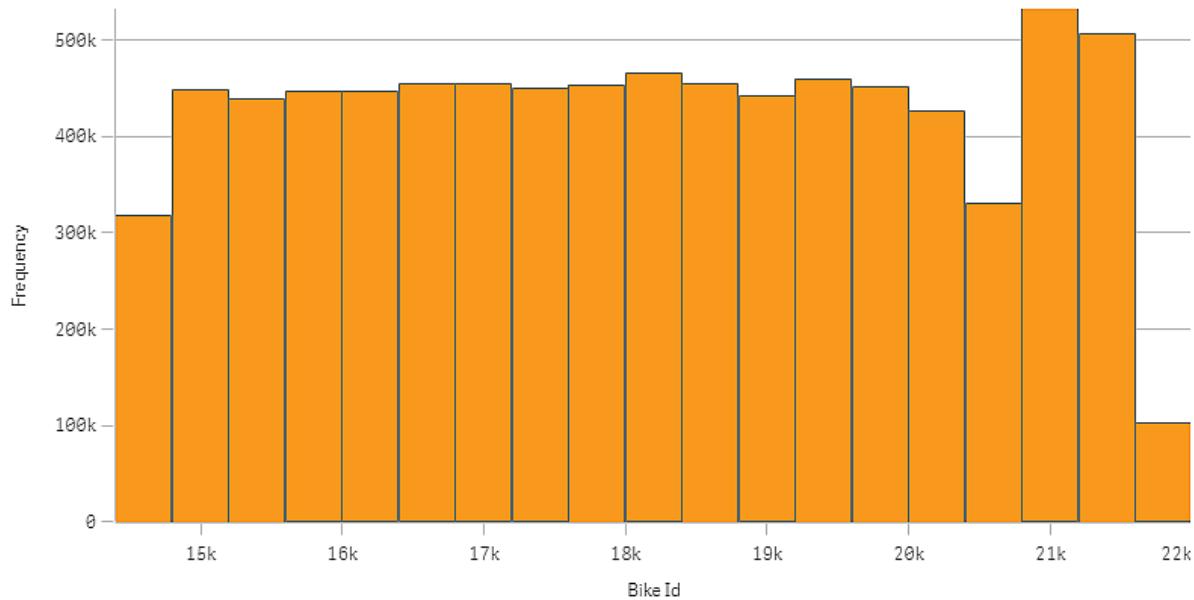


Display limitations

When a measure value is outside the range limits, an arrow indicates whether the measure value is higher or lower than the range values.

Histogram

The histogram is suitable for visualizing distribution of numerical data over a continuous interval, or a certain time period. The data is divided into bins, and each bar in a histogram represents the tabulated frequency at each bin.



When to use it

The histogram is suitable for visualizing distribution of numerical data over a continuous interval, or a certain time period.

Advantages

The histogram organizes large amounts of data, and produces a visualization quickly, using a single dimension.

Disadvantages

The histogram is not relevant for detailed analysis of the data as it deals with a summary of the data distribution.

Creating a histogram

You can create a histogram on the sheet you are editing. You can only apply a single dimension to a histogram. Histograms do not need a measure, as the frequency of the binned data is automatically calculated.

Do the following:

1. From the assets panel, drag an empty histogram to the sheet.
2. Add the dimension to calculate the frequency on.

When you have created the histogram, you may want to adjust its appearance and other settings in the properties panel.

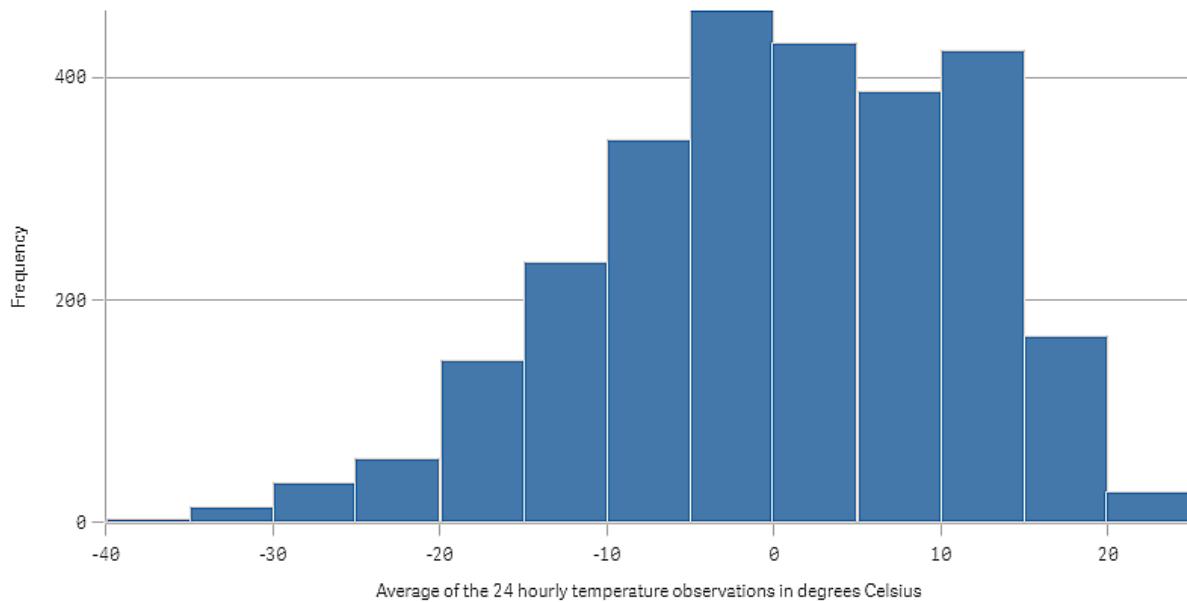
Dimension limitations

There are some limitations to the dimension used in a histogram.

- The dimension must be a numerical field.
- You cannot use a master dimension that was created using the expression editor, even if the resulting field is numeric.
- The dimension cannot be based on an aggregation function.

Viewing the distribution of data over intervals with a histogram

This example shows how to make a histogram to the distribution of data over intervals, using weather data.



Dataset

In this example, we'll use the following weather data.

- Location: Sweden > Gällivare Airport
- Date range: all data from 2010 to 2017
- Measurement: Average of the 24 hourly temperature observations in degrees Celsius

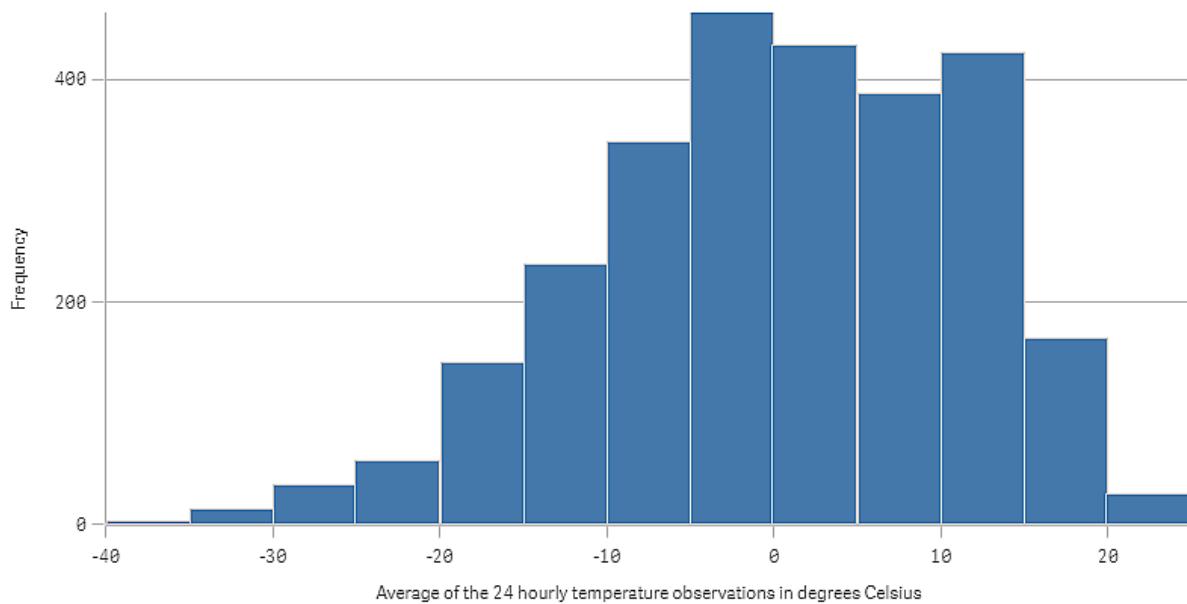
The dataset that is loaded contains a daily average temperature measurement from a weather station in the north of Sweden during the time period of 2010 to 2017.

Visualization

We add a histogram to the sheet and add the field *Average of the 24 hourly temperature observations in degrees Celsius* as dimension.

The visualization creates a frequency measure automatically, and sorts the temperature measurements into a number of bars according to frequency distribution.

We can adjust the size of the bars to get even intervals, by setting **Bars** to **Custom** and **Bar width (x-axis)** with a width of 5. This adjusts the bars to be intervals of 5 degrees Celsius as shown below:



Discovery

The histogram visualizes the frequency distribution of the temperature measurements. You can hover over the mouse over a bar to see more details of the frequency.

We can see that most days, the temperature is between -5 and 15 degrees Celsius. There are days below -30, but they are not many.

KPI

The KPI visualization can show one or two measure values, and is used to track performance.

A KPI visualization with two measure values, using conditional colors and symbols.





When to use it

Use KPIs to get an overview of performance values that are central to an organization. Use color coding and symbols to indicate how the figures relate to the expected results.

Advantages

KPIs give a quick understanding of the performance within an area.

Disadvantages

The KPI is somewhat limited when it comes to graphical components. You can use symbols to help illustrate the performance, but if you want a more conspicuous component, consider using a gauge.

Creating a KPI

You can create a KPI visualization on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty KPI chart to the sheet.
2. Click **Add measure** and select a measure or create a measure from a field.

In a KPI visualization, you can have one or two measures and no dimensions. With two measures, the second value automatically becomes a complementary value and is shown with a smaller font size. You can easily switch their order by dragging the measures in the properties panel under **Measures**.

When you have created the KPI visualization, you may want to adjust its appearance and other settings in the properties panel.

The following settings are used by default in a KPI visualization:

- Centered alignment.
- Black text color.
- No background color.
- Responsive layout behavior.
- Medium font size.
- No titles.
- Measure label displayed.
- Conditional colors and symbols are turned off.
- No link to sheet.



If you double-click or drag a measure from the assets panel, a KPI visualization is added on the sheet using the measure.

Using conditional colors and symbols

You can set your KPI visualization to display in different colors, and with different symbols, depending on the value of the selected measure. Conditional colors and symbols can be configured in the properties panel.

You can do this by adding multiple range limits to the KPI, creating subsections that indicate performance. For example, you can set your KPI so that it displays in:

- Green with a check mark symbol when performance is strong.
- Yellow with a caution symbol when performance falls below expectations.
- Red with an X symbol when performance is low.

You can also set range limits with expressions, rather than defining a single value.

Do the following:

1. In the properties panel for a KPI visualization, select **Appearance** and expand **Color**.
2. If necessary, turn off **Library colors** and turn on **Conditional colors**.
3. Click **Add limit** to create a new limit. Multiple limits can be added to a single KPI chart.
4. Specify a value for the limit, or enter an expression using the expression editor.
5. On the **Value** color bar, click the range area of the KPI you would like to modify.
6. Under **Colors**, select a preset color or use a custom color. Toggle, if needed, to **Symbols** to choose the symbol displayed when your KPI falls within the specified limit.

Linking to another sheet

You can link from the KPI visualization to a sheet in the app. When making data analysis and clicking the visualization, you can click a second time to go to a predefined sheet. The sheet is opened in a new tab. When hovering over , the name of the sheet is displayed. The icon is only displayed when **Show title** is selected, under **Presentation**.

Line chart

The line chart is used to show trends over time. The dimension is always on the x-axis, and the measures are always on the y-axis.



Your data set must consist of at least two data points to draw a line. A data set with a single value is displayed as a point.

If you have a data set where data is missing for a certain month, you have the following options for showing the missing values:

- As gaps
- As connections
- As zeros

When a month is not present at all in the data source, it is also excluded from the presentation.



When to use it

The line chart is primarily suitable when you want to visualize trends and movements over time, where the dimension values are evenly spaced, such as months, quarters, or fiscal years.

Advantages

The line chart is easy to understand and gives an instant perception of trends.

Disadvantages

Using more than a few lines in a line chart makes the line chart cluttered and hard to interpret. For this reason, avoid using more than two or three measures.

Creating a line chart

You can create a line chart on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty line chart to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.

In a line chart you need at least one dimension and one measure.

You can include up to two dimensions and one measure, or one dimension and up to 15 measures in a line chart.

Creating a line chart

Dimensions	Measures	Result
1 dimension	1 measure	A simple line chart with a single line.
2 dimensions	1 measure	A line chart with the first dimension on the X-axis, and a line for each value of the second dimension.
1 dimension	up to 15 measures	A line chart with one line for each measure.

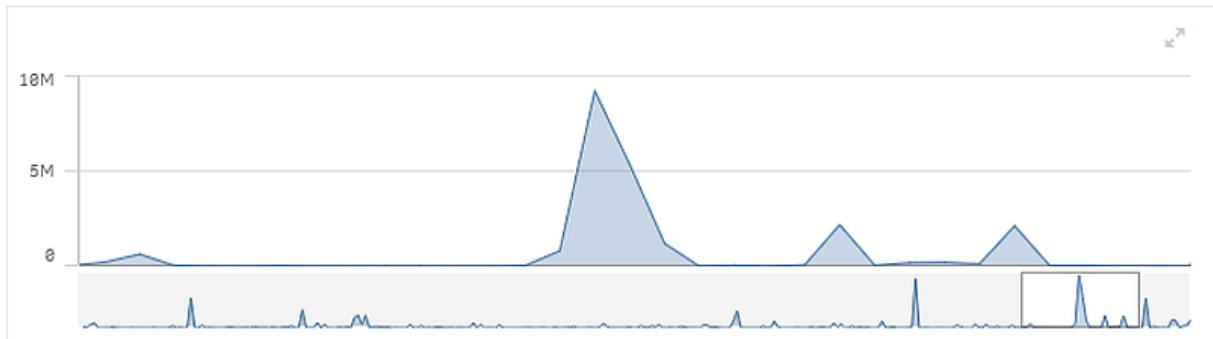
When you have created the line chart, you may want to adjust its appearance and other settings in the properties panel.

Display limitations

Displaying large numbers of dimension values

When the number of dimension values exceeds the width of the visualization, a mini chart with a scroll bar is displayed. You can scroll by using the scroll bar in the mini chart, or, depending on your device, by using the scroll wheel or by swiping with two fingers. When a large number of values are used, the mini chart no longer displays all the values. Instead, a condensed version of the mini chart (with the items in gray) displays an overview of the values, but the very low and the very high values are still visible. Note that for line charts with two dimensions, the mini chart is only available in stacked area mode.

Line chart with a mini chart, since the dimension values exceeds the width of the visualization.



Displaying out of range values

In the properties panel, under **Appearance**, you can set a limit for the measure axis range. Without a limit, the range is automatically set to include the highest positive and lowest negative value, but if you set a limit you may have values that exceed that limit. When a data point value cannot be displayed, due to the range limits, an arrow indicates the direction of the value.

When a reference line is out of range, an arrow is displayed together with the number of reference lines that are out of range.

Displaying large amounts of data in a line chart

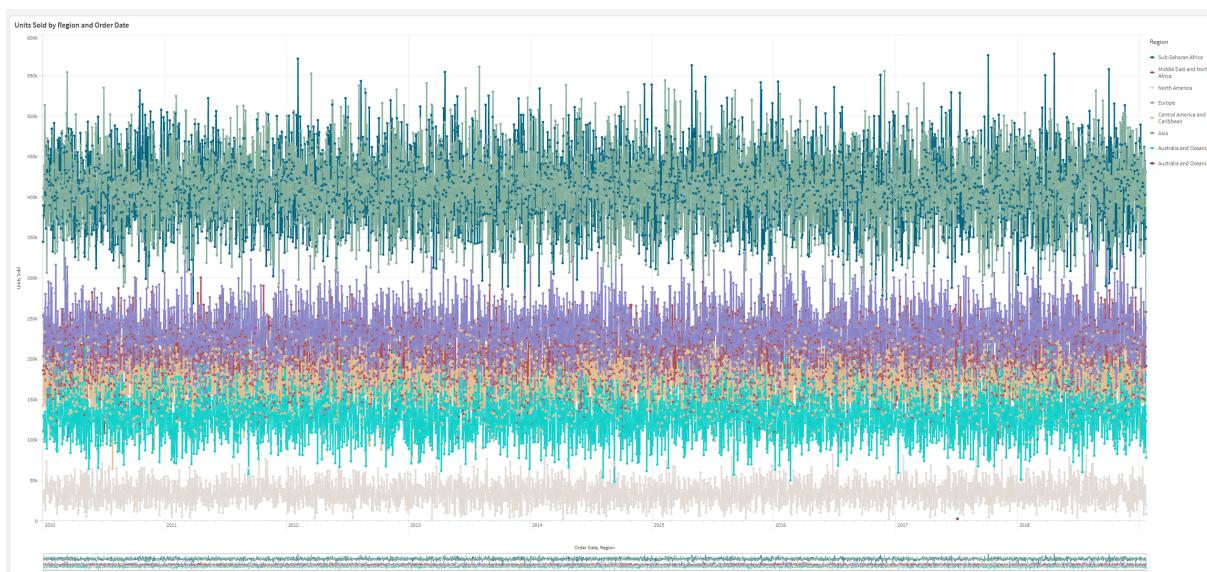
You can set the maximum number of visible points and lines in advanced edit mode. In the properties panel, go to **Presentation**. Then adjust the following:

- **Max visible points:** Set the maximum number of points that will be displayed. The default is 2,000. The maximum is 50,000. If you set a number less than 1,000, the line chart will behave as if the maximum is 1,000 visible points.
- **Max visible lines:** Set the maximum number of lines that will be displayed. The default is 12. The maximum is 1,000.

If there are more data points than the number set in **Max visible points**, you will not see any points, only lines. If there are more than 5,000 visible points, then labels will not be shown. If you have a large number of lines, not all lines will be displayed, or lines may overlap.

If you have a large number of points or lines, it may take your chart longer to render when you zoom or pan. You cannot make selections when the line chart is rendering.

A line chart with a large data set. This chart is set to display 50 lines, and 20,000 data points.



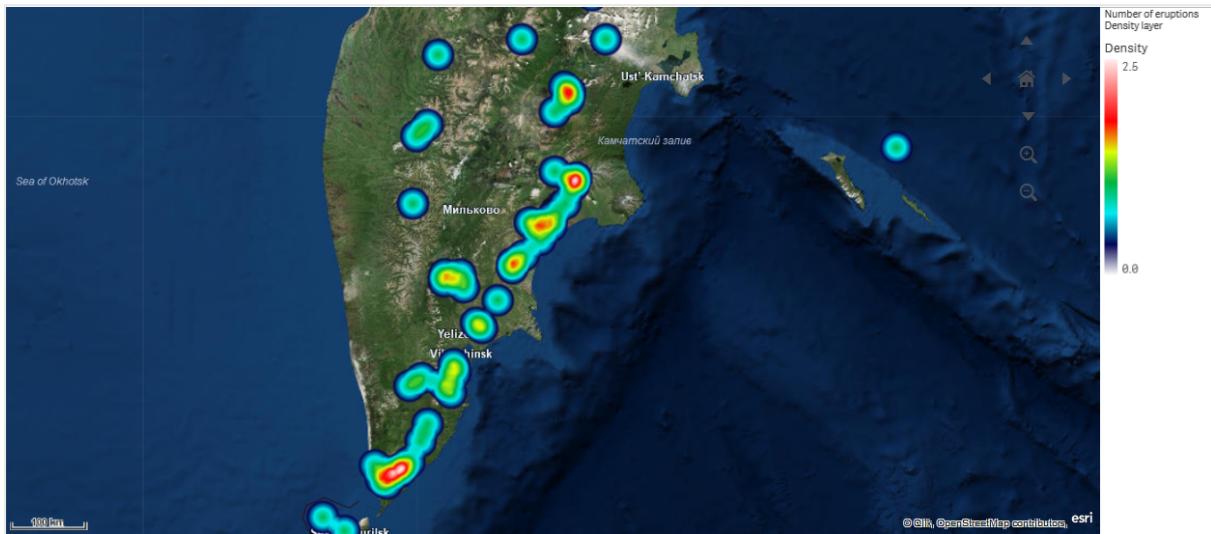
To avoid displaying limited data sets, you can either make a selection or use dimension limits in the properties panel.

Map chart

Maps enable you to view your data geographically.

Maps have many ways to present your data. You can add multiple layers to your map to display different types of information on the same map. You can set a custom scope for locations so that if two locations have the same name, you display the locations and their data correctly. You can use drill-down dimensions to create a hierarchy of geographic areas for selection. You can limit the pan of a map to a specific view and scope of the map, such as a region of interest, out of which users cannot pan or zoom out. You can add custom base maps to your map and use non-WGS-84 coordinates.

Map with density layer displaying number of global volcanic eruptions.



When to use maps

You can use a map to show the geographical distribution of offices, stores, and other sites of business interest. You can visualize not only locations but also sales values and other measures and display the value differences by bubble size or color.

Advantages

The map is a versatile visualization that efficiently presents the geographical distribution of key values related to location or area.

Disadvantages

With a large number of values, it may be hard to get a good overview. Values may be placed on top of each other and not visible until zoomed in.

Server connection requirements

In order for a Qlik Sense map chart to be able to perform location lookup and show a background map (base map and layers), your web browser needs to be able to establish a connection with both of the following servers on port 443 (HTTPS):

- mapsqlikcloud.com (required for location lookups and background map)
- ibasemaps-api.arcgis.com (required for Satellite base map)

For more information, refer to the Map service connection requirements on [Ports used by user web browser](#).

If you are instead using an on-premise GeoAnalytics server, your map chart will not need access to mapsqlikcloud.com and can function without an internet connection. However, in this setup, the Satellite base map will not be displayed as it requires a connection with ibasemaps-api.arcgis.com. For more information, see [Using GeoAnalytics with Qlik Sense map](#).

Base map

The base map provides the background for the data contained in your layers. You can select your base map in **Map settings**. Qlik Sense has four default base maps:

- **Default:** An OpenStreetMap-based map.
- **Pale:** A paler version of **Default**.
- **Dark:** A darker version of **Default**.
- **Satellite:** A satellite image map.
- **None:** No base map.



As of December 7, 2021 the map tile service used by Qlik Sense for satellite base maps changed from services.arcgisonline.com to ibasemaps-api.arcgis.com. If your maps are not functioning as expected, contact your Qlik administrator. They might need to allow this new service.

Additionally, custom base maps can be added using background layers to add your own custom base maps. For example, you could add the floor plan of an airport or office as a custom base map.

Layers

Layers contain visualized dimension and measure data that is displayed over your map. You can stack layers on top of each other. You can also control at what zoom levels different layers appear in or have layers that appear only if other values in a drill-down dimension are selected. This enables you to create different levels of detail as you make selections and zoom in and out of areas of interest on your map. The following layers are available:

- **Point layer:** A point layer overlays individual locations on a map, representing them with shapes.
Point layers (page 192)
- **Area layer:** An area layer presents areas on your map, such as countries or states. With polygon geometry loaded into a field, it can present any custom area.
Area layers (page 193)
- **Line layer:** A line layer enables you to display lines between points on your map.
Line layers (page 194)
- **Density layer:** A density layer enables you to visualize the density of points in an area using a color ramp.
Density layers (page 196)
- **Chart layer:** A chart layer enables you to display small pie charts or bar charts over locations in your map.
Chart layers (page 197)
- **Background layer:** Background layers enable you to display a custom base map for your map visualization.
Background layers (page 199)

Layers that use drill-down dimensions can be used to create drill-down layers. Drill-down layers enable you to drill-down different hierarchical dimensions in a single layer or in multiple layers. You could, for example, switch between area and point layers as selections are made. For more information, see *Drill-down layers (page 202)*. For an example map that uses drill-down dimensions and layers, see *Controlling visible map data with drill-down layers (page 202)*.

Location data for map layers

Maps support several ways for determining locations in a layer. You can use the dimension added to the layer. You can alternatively specify fields containing location data for the layer, if the layer dimension does not contain geographic data. In the **Location** properties, you can specify additional parameters for the location field, such as adding additional fields that include country or administrative area information. For example, if you have a field containing custom area geometries and a field containing the names of the custom areas, you can set the name field as the dimension and then set the area geometry field as the location field in **Location** in the map properties.

Locations can be either geometries or names of locations such as names of countries, regions, cities, postal codes etc. Layer locations can be defined using fields that contain names and codes. Qlik Sense can identify the following types of locations:

- Continent names
- Country names
- ISO alpha 2 country codes
- ISO alpha 3 country codes
- First-order administrative area names, such as a state or province names
- Second-order administrative area names
- Third-order administrative area names
- Fourth-order administrative area names
- Postal codes or ZIP Codes
- City, village, or other populated place names
- IATA airport codes
- ICAO airport codes



Availability of locations may vary by country. If the named location is not available, use coordinate or area data for the location.

Qlik Sense uses map and location data obtained from recognized field leaders who use accepted methodologies and best practices in marking borders and naming countries within their mappings. Qlik Sense provides flexibility to enable users to integrate their own, separate background maps. If the standard maps do not fit, Qlik Sense offers the option to load customer provided background maps, borders, and areas.

Geometries can either be added at load time by the data preparation service or loaded from geographic sources such as KML. Point layers also support latitudes and longitudes in separate fields. For area layers, areas can be defined using geometries from a geographic data source such as KML files. Line layers support the same point data as point layers. Line layers also support strings with line geometries in GeoJSONLineString or MultiLineString formats.

If you are using a custom map in a background layer that uses non-WGS-84 coordinates, you can use a field with locations defined in the coordinate system the map uses (either degrees or meters). For more information, see *Using non-WGS-84 coordinate systems (page 192)*.

Creating maps

You can add a map to the sheet you are editing.



You can create several map visualizations based on different point data or area data, which use the same dimension data.

Do the following:

1. From the assets panel, drag an empty map to the sheet.
2. From **Map settings**, select the **Base map**.
3. From **Layers** in the properties panel, add layers to your map.

Layers are placed above layers of the same type. You can set the order of layers by dragging layers up and down in the list. The layer at the top of your list is also overlaid on top of the other layers on your map.

For information on configuring layers, see:

- *Point layers (page 192)*
- *Area layers (page 193)*
- *Line layers (page 194)*
- *Density layers (page 196)*
- *Chart layers (page 197)*
- *Background layers (page 199)*

Once the map is created, you can adjust its appearance and other settings for the map and its layers in the properties panel.



ⓘ will appear in the top-left corner of your map if there is an issue with the added layer. Click the icon to see the error message.

Limiting location scope in map layers

By default, Qlik Sense searches a broad scope of locations for the location field when names are used. In cases where there are multiple possible matches, you may need to limit the scope of locations searched to ensure Qlik Sense displays the desired location.

Do the following:

1. From **Layers** in the properties panel, click **Location**.
2. Set **Scope for location** to **Custom**.
3. After **Location type**, select the type of data contained in the field or expression selected as **Location field**.
4. Optionally, after **Country**, enter a field, expression, or text indicating the country the locations in the **Location field** are located.

5. Optionally, after **Administrative area (Level 1)**, enter a field, expression, or text indicating the first-level administrative area the locations in the **Location field** are located.
6. Optionally, after **Administrative area (Level 2)**, enter a field, expression, or text indicating the second-level administrative area the locations in the **Location field** are located.

Limiting panning in your map

You can limit the panning settings in your map to keep the view in your map limited to a set area, such as a specific region of interest. When you limit the panning in a map, you lock the map to the current view. The map will not zoom out further than the current view. You cannot pan the map beyond the limits of the current view. If you zoom in, you can pan, but you are limited by the boundaries of the pan limit.

Do the following:

1. From the properties panel, click **Appearance**.
2. In **Presentation**, set **Limit pan navigation** to **Custom**.
3. Set your map to the view you want to which you want to limit the pan.
4. Click **Set pan limit**.

You can toggle the set pan limit on and off with **Limit pan navigation**. You can clear the saved pan limit by clicking **Clear pan limit**.

Using non-WGS-84 coordinate systems

If you have data and a background map in a projection other than WGS-84, you should set the base map to **None**, add your background map as a background layer, and then set your projection to use undefined degrees or meters depending on the unit of length used in your base map.

When using non-WGS-84 coordinates, you must use geometries loaded into a field. Location names cannot be used.

Do the following:

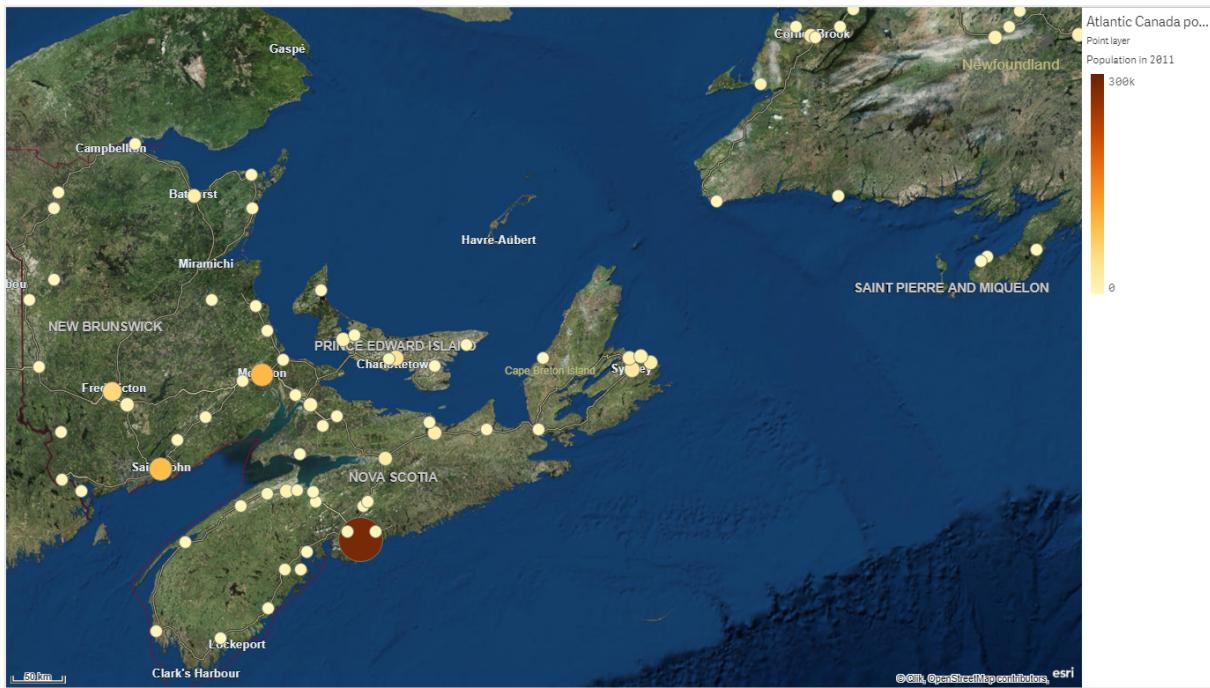
1. From the properties panel, click **Map settings**.
2. In **Base map**, select **Empty (undefined degrees)** or **Empty (undefined meters)**.
3. Select **Background layer**.
4. After **URL**, enter the URL to a slippy map server.
For example, `http://a.tile.opencyclemap.org/cycle/${z}/${x}/${y}.png`.
5. After **Attribution**, enter the attribution string for the map.
For example, © `OpenCycleMap. Map data © OpenStreetMap contributors.`
6. From **Layers**, click **Add layer**.
7. Select your layer type.
8. In **Dimensions**, click **Add** and select a field containing data in your map's coordinate system.

Point layers

A point layer overlays individual locations on a map, representing them with shapes.

By default, point layers use circular bubbles, but you can also use several other shapes. You can also use custom images as the points in a point layer. The size of the points in your layer can be fixed or you can specify a measure or expression to set the sizes of the different points. You can highlight these differences in values further by coloring by measure.

Map with point layer displaying cities in Atlantic Canada colored and sized by population.



Adding a point layer

Do the following:

1. Drag and drop a field onto the map, select **Add as new layer**, and select **Add as point layer**.
2. From **Layers** in the properties panel, click **Add layer** and select **Point layer**. In **Dimensions**, click **Add** and select a field containing point data to use as the dimension.
3. If there are issues with the point locations, adjust the location settings in **Locations** in the properties panel.
For more information, see *Limiting location scope in map layers (page 191)*.

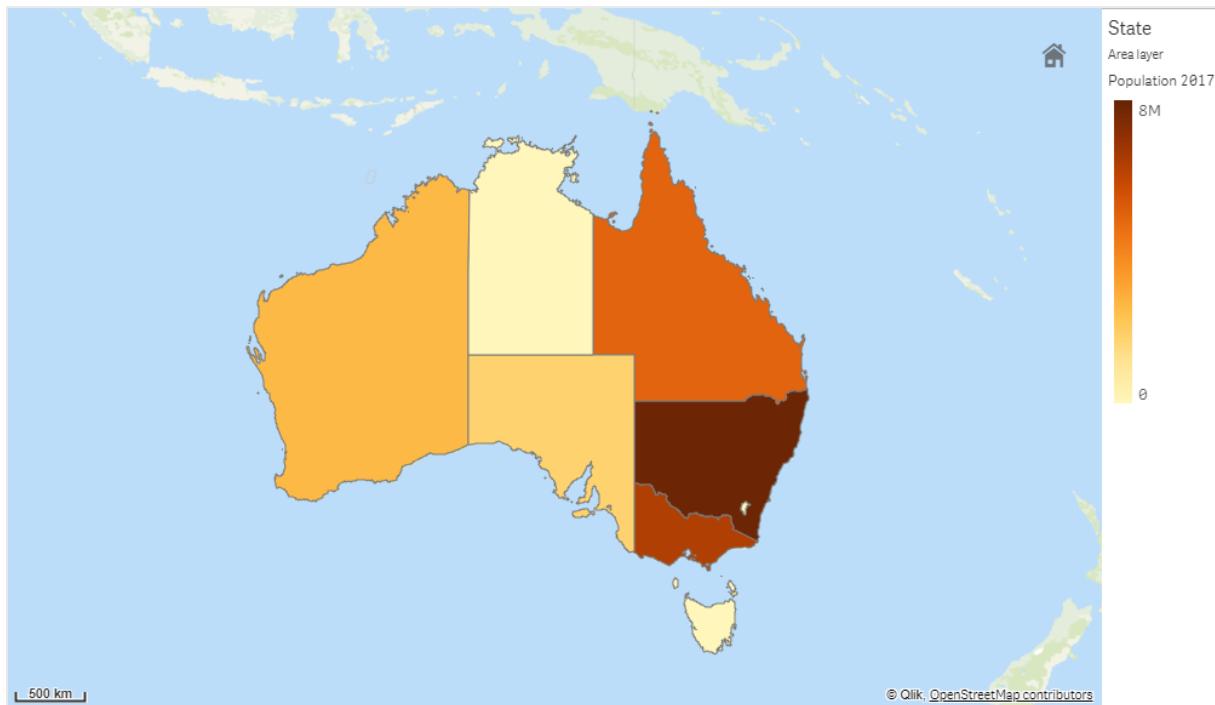
Once the layer is added, you can adjust settings for the layer in the properties panel.

Area layers

An area layer presents areas on your map, such as countries or states. With polygon geometry loaded into a field, it can present any custom area.

With an area layer, each dimension value corresponds to a presented area. By using colors with your area layer, you can present different measure values for the areas. In the properties panel, under **Appearance > Colors and legend**, switch **Colors** to **Custom** where the options **By measure** and **By expression** are available.

Map with area layer displaying Australian states and territories colored by population.



Adding an area layer

Do the following:

1. Do one of the following:
 - Drag and drop a field onto the map, select **Add as new layer**, and select **Add as area layer**.
 - From **Layers** in the properties panel, click **Add layer** and select **Area layer**. In **Dimensions**, click **Add** and select a field containing area data to use as the dimension.
2. If there are issues with the point locations, adjust the location settings in **Locations** in the properties panel.
Limiting location scope in map layers (page 191)

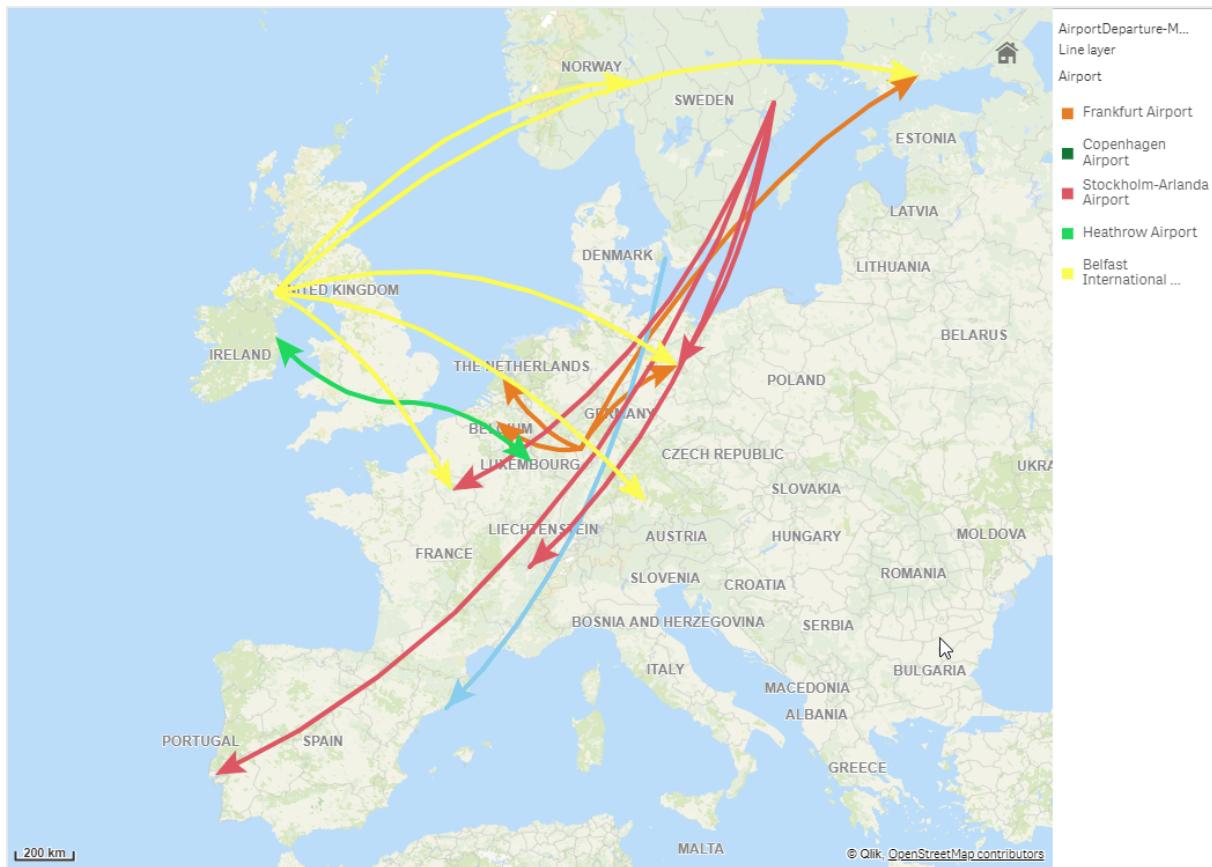
Once the layer is added, you can adjust settings for the layer in the properties panel.

Line layers

A line layer enables you to display lines between points on your map.

For example, you can use a line layer to show aircraft flights between cities. With a line layer, you can use two fields containing point data to define the start points and end points for lines in the layer. Alternatively, you can use a field containing line geometry in either GeoJSON LineString or MultiLineString format. You can customize the width and curvature of lines in the line layer and add directional arrows to your lines.

Map with line layer displaying departing flights between airports in Europe.



Adding a line layer

Line layers display lines using either start and end points, where two fields are used to determine where lines start and end, or using line geometries in GeoJSONLineString or MultiLineString format.

If you are using a start and end point line layer, the dimension you select for your line layer should represent the fields selected as the start and end points in your **Location** settings. For example, if you wanted to visualize where your shipments are being sent, you could set *Shipments* as the dimension and then use *Distribution Center Location* and *Shipping Destination* as the start point and end point in **Location**.

Alternatively, you can add two dimensions to the line layer and use these as the start and end points. This is useful if you want to display lines between all locations in the first dimensions to all locations in the second dimension, which has an association to the first dimension.

Adding a line layer with start and end points

Do the following:

1. Do one of the following:
 - Drag and drop a field containing start point data onto the map, select **Add as new layer**, and select **Add as line layer**.

- From **Layers** in the properties panel, click **Add layer** and select **Line layer**. In **Dimensions**, click **Add** and select a field to use as the dimension.
2. If your dimension contains start point data, add a field containing end point data. Do one of the following:
 - Drag and drop a field containing end point data onto the map, select **Use in <layer name>**, and select **Add <field name> as a second dimension**.
 - In **Data**, click **Add** and select a field containing end point data.
 3. If your dimension does not contain start or end point data, in **Location**, add the fields containing the start point data and end point data as location fields.
 4. If there are issues with the start and end point locations, adjust the location settings in **Location** in the properties panel.
Line layers using start point field and end point fields each have separate location settings in **Location**. For more information, see *Limiting location scope in map layers (page 191)*.

Once the layer is added, you can adjust settings for the layer in the properties panel.

Adding a line layer with line geometries

Do the following:

1. Do one of the following:
 - Drag and drop a field containing line geometries onto the map, select **Add as new layer**, and select **Add as line layer**.
 - From **Layers** in the properties panel, click **Add layer** and select **Line layer**. In **Dimensions**, click **Add** and select a field to use as the dimension.
2. In **Location**, select **Line geometry** and then select a field in **Line geometry field**.
By default, your dimension is selected as the **Line geometry field**.

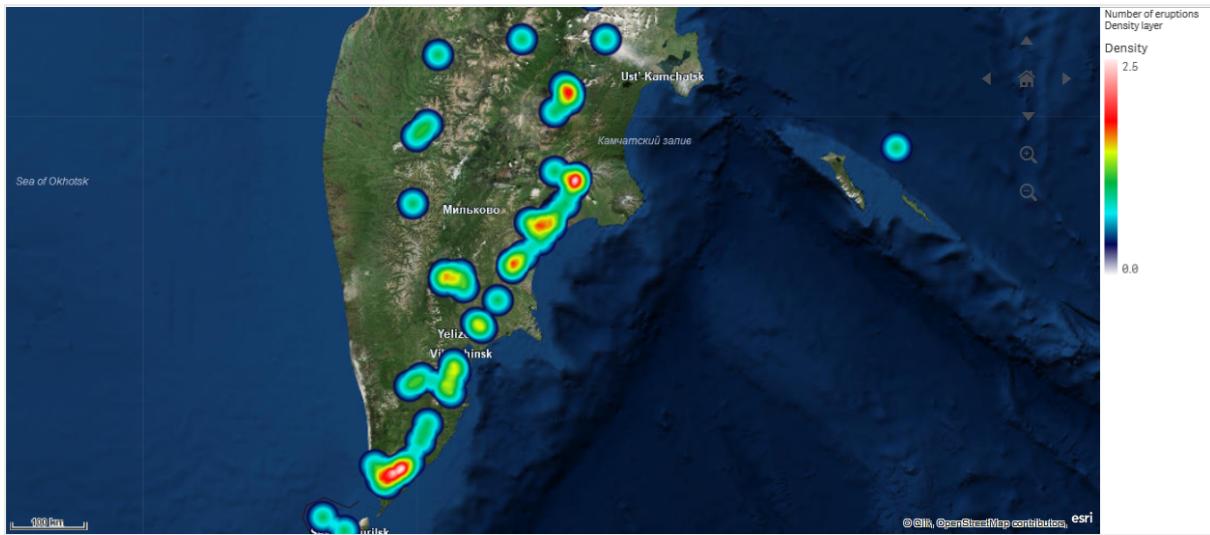
Once the layer is added, you can adjust settings for the layer in the properties panel.

Density layers

A density layer enables you to visualize the density of points in an area using a color ramp.

Each point influences a circular area, with the highest influence at the center and declining towards its outer perimeter. You can specify the influence radius of the points, change its color and scale it to suit your visualization. Density layers can be used to show hotspots of activity, population densities and more.

Map with density layer displaying number of global volcanic eruptions.



Adding a density layer

Do the following:

1. Do one of the following:
 - Drag and drop a field containing point data onto the map, select **Add as new layer**, and select **Add as density layer**.
 - From **Layers** in the properties panel, click **Add layer** and select **Density layer**. In **Dimensions**, click **Add** and select a field containing point data to use as the dimension.
2. If there are issues with the point locations, adjust the location settings in **Locations** in the properties panel.
For more information, see *Limiting location scope in map layers (page 191)*.

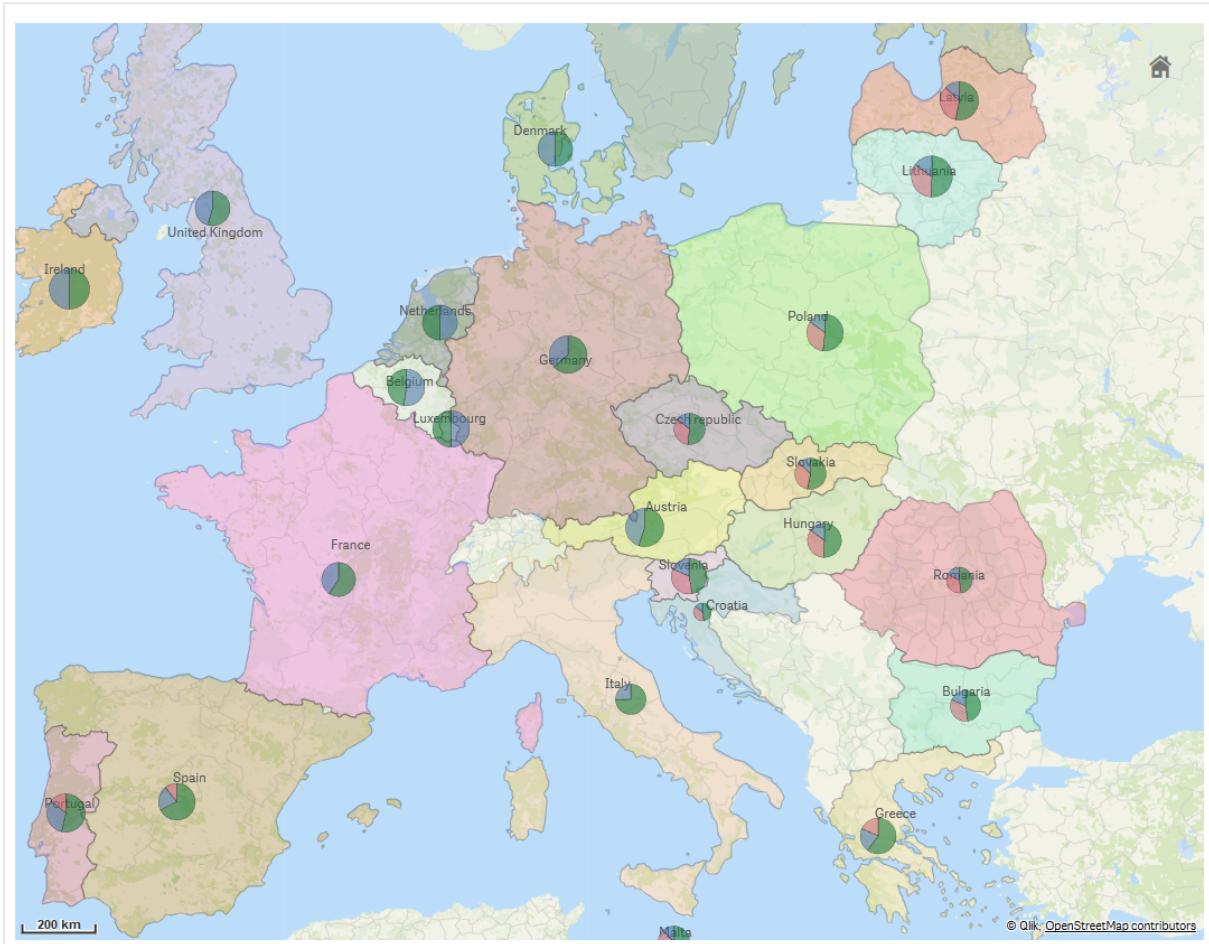
Once the layer is added, you can adjust settings for the layer in the properties panel. .

Chart layers

A chart layer enables you to display small pie charts or bar charts over locations in your map. It uses one dimension to identify the locations and a second dimension to create the pie or bar charts.

You can label the charts and use an expression to control their size. The info bubble displays useful information about the contents of the selected chart.

Map with chart layer displaying source of funds in European Union countries as pie charts. The size of each pie chart shows average absorption rate of funds.



Adding a chart layer

Do the following:

1. From **Layers** in the advanced properties, click **Add layer** and select **Chart layer**.
2. In **Dimensions**, under **Location**, click **Add** and select a field containing point data to use as the locations on your map. Click **Add** and select a field containing data to use as the dimension on your charts.
3. In **Measures**, click **Add** and select a field containing data to use as the measure on your charts.
4. If there are issues with the point locations, adjust the location settings in **Location** in the properties panel.

Limiting location scope in map layers (page 191)

Do the following:

1. Do one of the following:
 - Drag and drop a field containing point data onto the map, select **Add as new layer**, and select **Add as chart layer**.
 - From **Layers** in the properties panel, click **Add layer** and select **Chart layer**.
1. In **Dimensions**, under **Location**, click **Add** and select a field containing point data to use as the locations on your map. Click **Add** and select a field containing data to use as the dimension on your charts.
2. In **Measures**, click **Add** and select a field containing data to use as the measure on your charts.
3. If there are issues with the point locations, adjust the location settings in **Location** in the properties panel.
Limiting location scope in map layers (page 191)

Once the layer is added, you can adjust settings for the layer in the properties panel.

Background layers

Background layers enable you to display a custom base map for your map visualization.

A background layer could, for example, be a map of an airport that then has a point layer with WIFI hotspot locations overlaid on it. If the custom base map format supports transparency, you can overlay it on top of another map. Qlik Sense supports the following custom maps formats as background layers:

- Slippy or tile map services (TMS)
- Web Map Service (WMS)
- Image URL (Image)

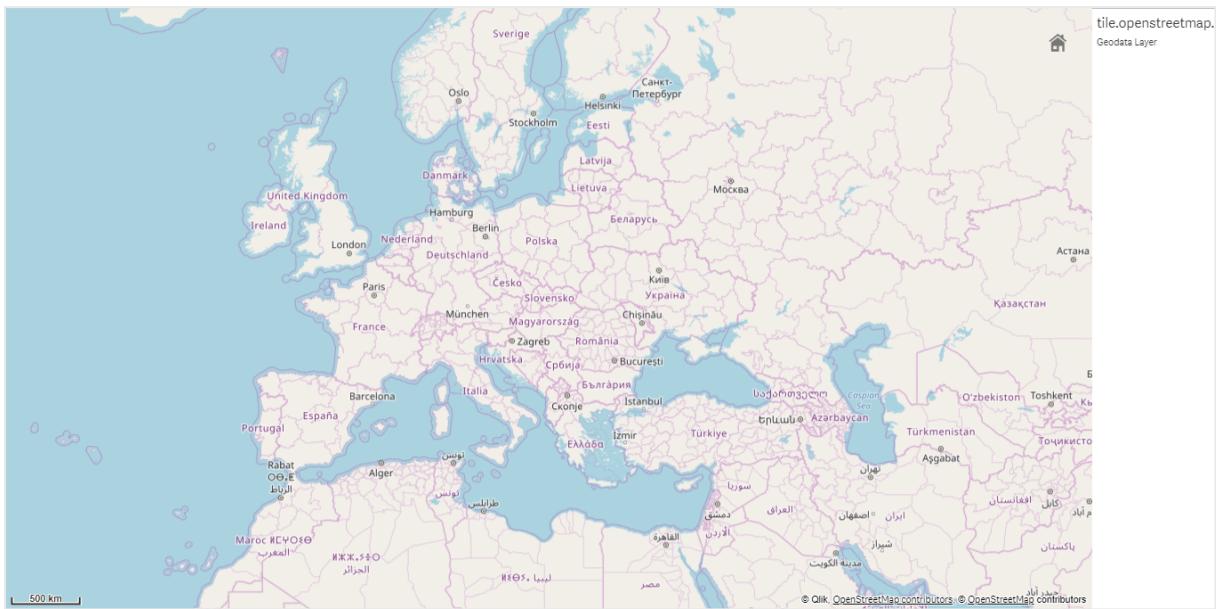
Maps services must be located on their own server. Unlike other layers, the background layer uses no dimensions or measures and only displays an external base map.

If your WMS contains areas with no data, you can set the areas with no data to be transparent. This enables the WMS background layer to be overlaid on another base map or background layer. You can also choose which WMS layers to include in the background layer.

You can insert an image as a background layer. This can be used as a custom base map. You could add an image of a floor plan and use it as a custom base map. An image background layer can also be overlaid on top of another background layer. For example, you could insert an image of a detailed local map as a background layer over top another map. Background layers support the following image types:

- .png
- .jpg
- .jpeg
- .gif

Map with background layer displaying a TMS OpenStreetMap map.



Adding a background layer

Background layer configuration varies depending on the type of background map.

Adding a TMS background layer

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
 2. Select **Background layer**.
 3. After **Format**, select **TMS**.
 4. After **URL**, enter the URL to a tile or slippy map server.
For example, `http://a.tile.opencyclemap.org/cycle/${z}/${x}/${y}.png`.
 5. After **Attribution**, enter the attribution string for the map.
For example, © `OpenCycleMap. Map data © OpenStreetMap contributors.`

Once the layer is added, you can adjust settings for the layer in the properties panel.

Adding a WMS background layer

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
 2. Select **Background layer**.
 3. Under **Format**, select **WMS**.
 4. Click **WMS setup**.

5. Under **WMS server URL**, enter the **URL**.
6. Under **Version**, select the WMS version.
7. Click **Load WMS**.
8. After **CRS**, enter the coordinate reference system used by the WMS map.
9. Select **Transparent** to generate map images that are transparent where there is not data.



This is not supported by all WMS.

10. Under **Image format**, select the WMS image format.
11. Under **Layers**, select the map layers that your WMS server supports.
12. Click **Save**.
13. After **Attribution**, enter the attribution string for the map.
For example, © [OpenCycleMap](http://www.opencyclemap.org/). Map data © [OpenStreetMap](http://www.openstreetmap.org/copyright) contributors.

Once the layer is added, you can adjust settings for the layer in the properties panel.

Adding an image background layer

Images can be used as a custom base map, such as for floor plans. When using an image background layer as a custom base map, set **Base map** in **Map settings** to **None**. Select **Undefined meters** or **Undefined degrees** as the projection. The location data for your other layers must use same coordinate system as this background layer.



When you add an image background layer for a smaller geographic area as a custom base map, add a layer containing data, such as a point layer, at the same time. This lets the map automatically zoom in to the position of your image background layer.



*The lowest unit in the scale bar is 10 meters. If this measurement is too large for your image, you can disable the scale bar in **Presentation** and then proportionally adjust up your image in the background layer to the scale you want. Scale the location data you want to use in other layers as well.*

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
2. Select **Background layer**.
3. Under **Format**, select **Image**.
4. After **URL**, enter the URL of the image.
5. Position your image by entering the coordinates for the top left and bottom right corners of the image. Coordinates must be in the same coordinate type as was selected in **Projection**.



If you are unsure of the coordinates in your image background layer, enable **Show debug info**. This will display the coordinates for the current center of your map.

6. After **Attribution**, enter the attribution string for the image.

For example, © [OpenCycleMap](http://www.opencyclemap.org/). Map data © [OpenStreetMap](http://www.openstreetmap.org/copyright) contributors.

Once the layer is added, you can adjust settings for the layer in the properties panel. For image background layers being used as custom base maps, it is recommended that you change the **Limit zoom levels** and **Limit pan navigation settings** to set the focus on your custom map.

Drill-down layers

When creating a map that has multiple points of data located in a wide geographical area, you can use drill-down dimensions to display your layers in a selection hierarchy.

This enables you to use different layers at different levels within your map, ensuring only the most relevant information is being displayed.

As you make selections in a drill-down layer, the dimension used for locations changes to the next dimension in the drill-down dimension. The drill-down happens across layers that use the same drill-down dimension. If you have an area and point layer that use the same drill-down dimension, you can set which layer displays with which dimension from the drill-down dimension.



If selections cause the current drill-down dimension field to have only one possible value, the next field in the list is used instead.

Drill-down dimensions used for drill-down layers should have the fields in order of highest geographical area to smallest geographical area.

For information on creating drill-down dimensions, see *Creating a drill-down dimension* (page 89).

Do the following:

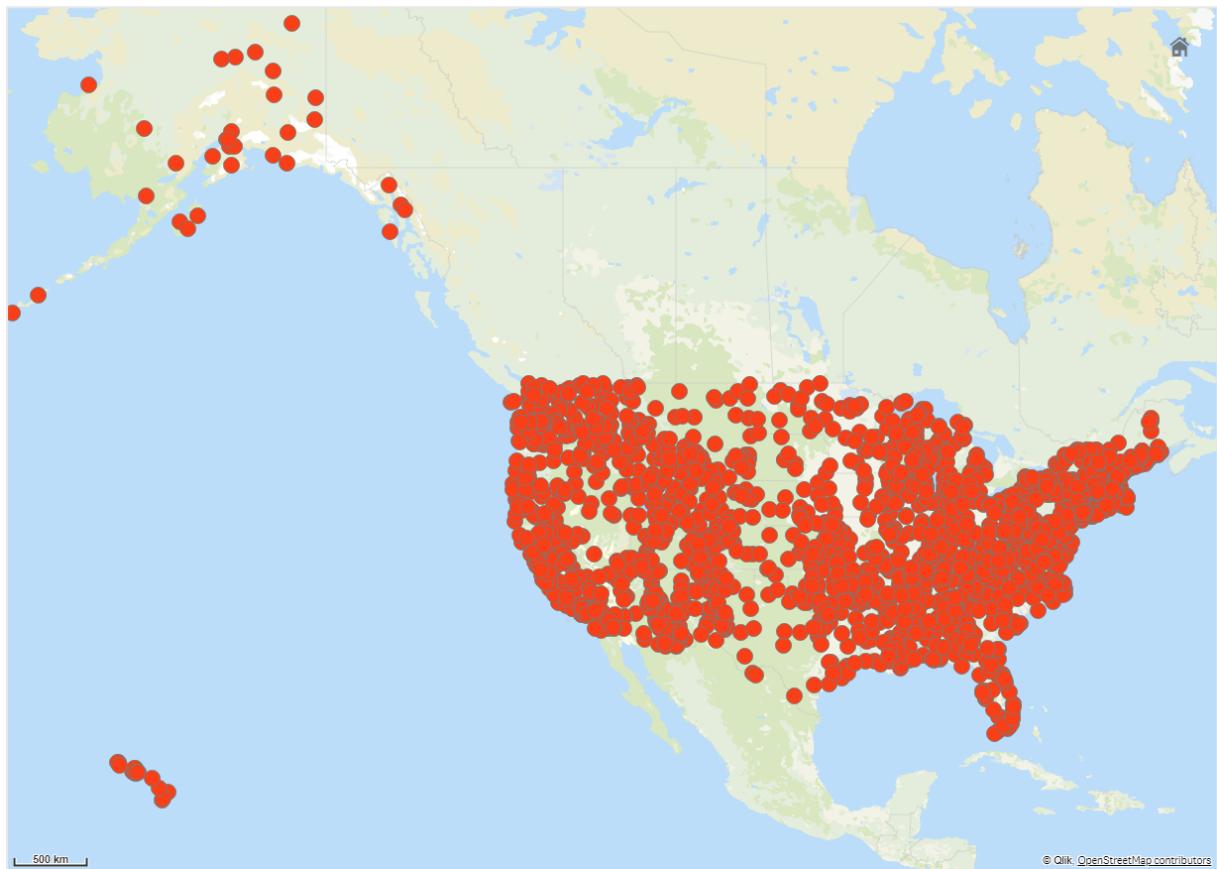
1. From **Layers** in the properties panel, click **Add layer**.
2. Select **Point layer**, **Area layer**, **Line layer**, **Density layer**, or **Chart layer**.
3. Click **Dimensions**, click **Add** and select your drill-down dimension.
4. In your layer, click **Options**.
5. Click **Layer display**.
6. After **Visible drill-down levels**, select which dimensions to display in the layer.

Controlling visible map data with drill-down layers

This example will show you how to build a map with a top-level area layer that drills down into with two point layers.

When creating a map that has multiple points of data located in a wide geographical area, you can use drill-down dimensions to display your layers in a hierarchy. As users make selections in a layer, the dimension displayed in the layer changes to the next dimension in the drill-down dimension. This enables you to use display data at different selection levels within your map, ensuring only the most relevant information is being displayed.

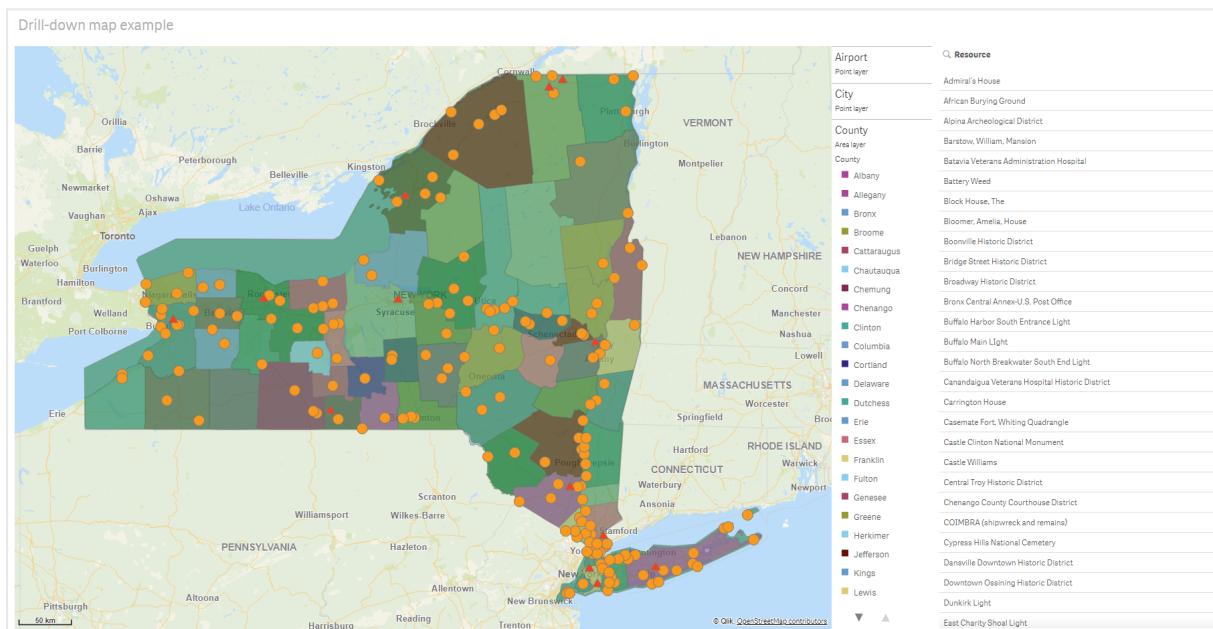
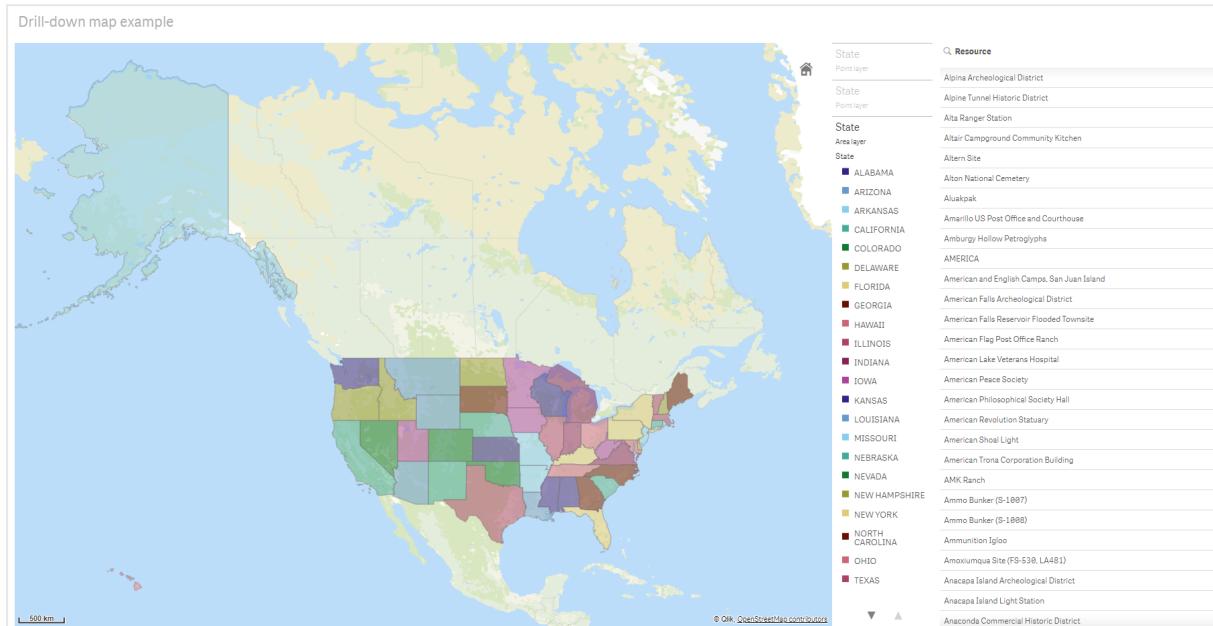
We have a list of all historical sites in the United States National Park Service's National Register of Historic places. When we add the cities that historical sites are located in to a map as a point layer, we get accurate locations for our bubbles. It could be made more understandable with better organization, however.



How then can we organize this information better, especially if we also want to add in another point layer containing airports to help plan how to get to the different sites?

To solve this problem, we will create a map of the United States of America with a layer of states that drills down to the county level. Drilling down to the county level will also show the cities that contain the historical sites as data points, as well as all airports for that state and the surrounding states.

3 Visualizations



Dataset

This example uses two sets of data:

- Federal listings: National Register of Historic Places listed properties from federal agencies ([federal_listed_20190404.xlsx](#))
This data set is available from the National Park Service National Register of Historic Places. It contains data about all registered history places, their location, and the federal agencies associated to them.
 [federal_listed_20190404.xlsx](#).
- Airport data
This table contains data for airports in the United States. It consists of each airport's International Air Transport Association (IATA) code, city, and state or territory.

You must import this data into Qlik Sense either by adding to a spreadsheet that you then import into Qlik Sense or by importing the table from this help page as a web file.

Airport data

Airport data

Airport	AirportCity	AirportState
ABE	Allentown/Bethlehem/Easton, PA	PA
ABI	Abilene, TX	TX
ABQ	Albuquerque, NM	NM
ABR	Aberdeen, SD	SD
ABY	Albany, GA	GA
ACT	Waco, TX	TX
ACV	Arcata/Eureka, CA	CA
ADK	Adak Island, AK	AK
ADQ	Kodiak, AK	AK
AEX	Alexandria, LA	LA
AGS	Augusta, GA	GA
ALB	Albany, NY	NY
ALO	Waterloo, IA	IA
AMA	Amarillo, TX	TX
ANC	Anchorage, AK	AK
APN	Alpena, MI	MI
ART	Watertown, NY	NY
ASE	Aspen, CO	CO
ATL	Atlanta, GA	GA
ATW	Appleton, WI	WI
AUS	Austin, TX	TX
AVL	Asheville, NC	NC
AVP	Scranton/Wilkes-Barre, PA	PA
AZO	Kalamazoo, MI	MI
BDL	Hartford, CT	CT
BET	Bethel, AK	AK
BFL	Bakersfield, CA	CA

Airport	AirportCity	AirportState
BGR	Bangor, ME	ME
BHM	Birmingham, AL	AL
BIL	Billings, MT	MT
BIS	Bismarck/Mandan, ND	ND
BJI	Bemidji, MN	MN
BKG	Branson, MO	MO
BLI	Bellingham, WA	WA
BMI	Bloomington/Normal, IL	IL
BNA	Nashville, TN	TN
BOI	Boise, ID	ID
BOS	Boston, MA	MA
BPT	Beaumont/Port Arthur, TX	TX
BQK	Brunswick, GA	GA
BQN	Aguadilla, PR	PR
BRD	Brainerd, MN	MN
BRO	Brownsville, TX	TX
BRW	Barrow, AK	AK
BTM	Butte, MT	MT
BTR	Baton Rouge, LA	LA
BTV	Burlington, VT	VT
BUF	Buffalo, NY	NY
BUR	Burbank, CA	CA
BWI	Baltimore, MD	MD
BZN	Bozeman, MT	MT
CAE	Columbia, SC	SC
CAK	Akron, OH	OH
CDC	Cedar City, UT	UT
CDV	Cordova, AK	AK
CEC	Crescent City, CA	CA
CHA	Chattanooga, TN	TN

Airport	AirportCity	AirportState
CHO	Charlottesville, VA	VA
CHS	Charleston, SC	SC
CIC	Chico, CA	CA
CID	Cedar Rapids/Iowa City, IA	IA
CLD	Carlsbad, CA	CA
CLE	Cleveland, OH	OH
CLL	College Station/Bryan, TX	TX
CLT	Charlotte, NC	NC
CMH	Columbus, OH	OH
CMI	Champaign/Urbana, IL	IL
CMX	Hancock/Houghton, MI	MI
COD	Cody, WY	WY
COS	Colorado Springs, CO	CO
COU	Columbia, MO	MO
CPR	Casper, WY	WY
CRP	Corpus Christi, TX	TX
CRW	Charleston/Dunbar, WV	WV
CSG	Columbus, GA	GA
CWA	Mosinee, WI	WI
CVG	Cincinnati, OH	KY
DAB	Daytona Beach, FL	FL
DAL	Dallas, TX	TX
DAY	Dayton, OH	OH
DBQ	Dubuque, IA	IA
DCA	Washington, DC	VA
DEN	Denver, CO	CO
DFW	Dallas/Fort Worth, TX	TX
DHN	Dothan, AL	AL
DIK	Dickinson, ND	ND
DLH	Duluth, MN	MN

Airport	AirportCity	AirportState
DRO	Durango, CO	CO
DSM	Des Moines, IA	IA
DTW	Detroit, MI	MI
EAU	Eau Claire, WI	WI
ECP	Panama City, FL	FL
EGE	Eagle, CO	CO
EKO	Elko, NV	NV
ELM	Elmira/Corning, NY	NY
ELP	El Paso, TX	TX
EUG	Eugene, OR	OR
EWN	New Bern/Morehead/Beaufort, NC	NC
EWR	Newark, NJ	NJ
EVV	Evansville, IN	IN
EYW	Key West, FL	FL
FAI	Fairbanks, AK	AK
FAR	Fargo, ND	ND
FAT	Fresno, CA	CA
FAY	Fayetteville, NC	NC
FCA	Kalispell, MT	MT
FLG	Flagstaff, AZ	AZ
FLL	Fort Lauderdale, FL	FL
FNT	Flint, MI	MI
FOE	Topeka, KS	KS
FSD	Sioux Falls, SD	SD
FSM	Fort Smith, AR	AR
FWA	Fort Wayne, IN	IN
GCC	Gillette, WY	WY
GCK	Garden City, KS	KS
GEG	Spokane, WA	WA
GFK	Grand Forks, ND	ND

Airport	AirportCity	AirportState
GGG	Longview, TX	TX
GJT	Grand Junction, CO	CO
GNV	Gainesville, FL	FL
GPT	Gulfport/Biloxi, MS	MS
GRB	Green Bay, WI	WI
GRI	Grand Island, NE	NE
GRK	Killeen, TX	TX
GRR	Grand Rapids, MI	MI
GSO	Greensboro/High Point, NC	NC
GSP	Greer, SC	SC
GTF	Great Falls, MT	MT
GTR	Columbus, MS	MS
GUC	Gunnison, CO	CO
GUM	Guam, TT	TT
HDN	Hayden, CO	CO
HIB	Hibbing, MN	MN
HLN	Helena, MT	MT
HNL	Honolulu, HI	HI
HOB	Hobbs, NM	NM
HOU	Houston, TX	TX
HPN	White Plains, NY	NY
HRL	Harlingen/San Benito, TX	TX
HSV	Huntsville, AL	AL
IAD	Washington, DC	VA
IAH	Houston, TX	TX
ICT	Wichita, KS	KS
IDA	Idaho Falls, ID	ID
ILG	Wilmington, DE	DE
ILM	Wilmington, NC	NC
IMT	Iron Mountain/Kingsfd, MI	MI

Airport	AirportCity	AirportState
IND	Indianapolis, IN	IN
INL	International Falls, MN	MN
ISN	Williston, ND	ND
ISP	Islip, NY	NY
ITO	Hilo, HI	HI
JAC	Jackson, WY	WY
JAN	Jackson/Vicksburg, MS	MS
JAX	Jacksonville, FL	FL
JFK	New York, NY	NY
JLN	Joplin, MO	MO
JNU	Juneau, AK	AK
KOA	Kona, HI	HI
KTN	Ketchikan, AK	AK
LAN	Lansing, MI	MI
LAR	Laramie, WY	WY
LAS	Las Vegas, NV	NV
LAW	Lawton/Fort Sill, OK	OK
LAX	Los Angeles, CA	CA
LBB	Lubbock, TX	TX
LCH	Lake Charles, LA	LA
LEX	Lexington, KY	KY
LFT	Lafayette, LA	LA
LGA	New York, NY	NY
LGB	Long Beach, CA	CA
LIH	Lihue, HI	HI
LIT	Little Rock, AR	AR
LMT	Klamath Falls, OR	OR
LNK	Lincoln, NE	NE
LRD	Laredo, TX	TX
LSE	La Crosse, WI	WI

Airport	AirportCity	AirportState
LWS	Lewiston, ID	ID
MAF	Midland/Odessa, TX	TX
MBS	Saginaw/Bay City/Midland, MI	MI
MCI	Kansas City, MO	MO
MCO	Orlando, FL	FL
MDT	Harrisburg, PA	PA
MDW	Chicago, IL	IL
MEM	Memphis, TN	TN
MFE	Mission/McAllen/Edinburg, TX	TX
MFR	Medford, OR	OR
MGM	Montgomery, AL	AL
MHK	Manhattan/Ft. Riley, KS	KS
MHT	Manchester, NH	NH
MIA	Miami, FL	FL
MKE	Milwaukee, WI	WI
MKG	Muskegon, MI	MI
MLB	Melbourne, FL	FL
MLI	Moline, IL	IL
MLU	Monroe, LA	LA
MMH	Mammoth Lakes, CA	CA
MOB	Mobile, AL	AL
MOD	Modesto, CA	CA
MOT	Minot, ND	ND
MQT	Marquette, MI	MI
MRY	Monterey, CA	CA
MSN	Madison, WI	WI
MSO	Missoula, MT	MT
MSP	Minneapolis, MN	MN
MSY	New Orleans, LA	LA
MTJ	Montrose/Delta, CO	CO

Airport	AirportCity	AirportState
MYR	Myrtle Beach, SC	SC
OAJ	Jacksonville/Camp Lejeune, NC	NC
OAK	Oakland, CA	CA
OGG	Kahului, HI	HI
OKC	Oklahoma City, OK	OK
OMA	Omaha, NE	NE
OME	Nome, AK	AK
ONT	Ontario, CA	CA
ORD	Chicago, IL	IL
ORF	Norfolk, VA	VA
ORH	Worcester, MA	MA
OTH	North Bend/Coos Bay, OR	OR
OTZ	Kotzebue, AK	AK
PAH	Paducah, KY	KY
PBI	West Palm Beach/Palm Beach, FL	FL
PDX	Portland, OR	OR
PHF	Newport News/Williamsburg, VA	VA
PHL	Philadelphia, PA	PA
PHX	Phoenix, AZ	AZ
PIA	Peoria, IL	IL
PIH	Pocatello, ID	ID
PIT	Pittsburgh, PA	PA
PNS	Pensacola, FL	FL
PPG	Pago Pago, TT	TT
PSC	Pasco/Kennewick/Richland, WA	WA
PSE	Ponce, PR	PR
PSG	Petersburg, AK	AK
PSP	Palm Springs, CA	CA
PVD	Providence, RI	RI
PWM	Portland, ME	ME

Airport	AirportCity	AirportState
RAP	Rapid City, SD	SD
RDD	Redding, CA	CA
RDM	Bend/Redmond, OR	OR
RDU	Raleigh/Durham, NC	NC
RHI	Rhineland, WI	WI
RIC	Richmond, VA	VA
RKS	Rock Springs, WY	WY
RNO	Reno, NV	NV
ROA	Roanoke, VA	VA
ROC	Rochester, NY	NY
ROW	Roswell, NM	NM
RST	Rochester, MN	MN
RSW	Fort Myers, FL	FL
SAF	Santa Fe, NM	NM
SAN	San Diego, CA	CA
SAT	San Antonio, TX	TX
SAV	Savannah, GA	GA
SBA	Santa Barbara, CA	CA
SBN	South Bend, IN	IN
SBP	San Luis Obispo, CA	CA
SCC	Deadhorse, AK	AK
SCE	State College, PA	PA
SDF	Louisville, KY	KY
SEA	Seattle, WA	WA
SFO	San Francisco, CA	CA
SGF	Springfield, MO	MO
SGU	St. George, UT	UT
SHV	Shreveport, LA	LA
SIT	Sitka, AK	AK
SJC	San Jose, CA	CA

Airport	AirportCity	AirportState
SJT	San Angelo, TX	TX
SJU	San Juan, PR	PR
SLC	Salt Lake City, UT	UT
SMF	Sacramento, CA	CA
SMX	Santa Maria, CA	CA
SNA	Santa Ana, CA	CA
SPI	Springfield, IL	IL
SPS	Wichita Falls, TX	TX
SRQ	Sarasota/Bradenton, FL	FL
STL	St. Louis, MO	MO
STT	Charlotte Amalie, VI	VI
STX	Christiansted, VI	VI
SUN	Sun Valley/Hailey/Ketchum, ID	ID
SUX	Sioux City, IA	IA
SWF	Newburgh/Poughkeepsie, NY	NY
SYR	Syracuse, NY	NY
TLH	Tallahassee, FL	FL
TOL	Toledo, OH	OH
TPA	Tampa, FL	FL
TRI	Bristol/Johnson City/Kingsport, TN	TN
TTN	Trenton, NJ	NJ
TUL	Tulsa, OK	OK
TUS	Tucson, AZ	AZ
TVC	Traverse City, MI	MI
TWF	Twin Falls, ID	ID
TXK	Texarkana, AR	AR
TYR	Tyler, TX	TX
TYS	Knoxville, TN	TN
VLD	Valdosta, GA	GA
VPS	Valparaiso, FL	FL

Airport	AirportCity	AirportState
WRG	Wrangell, AK	AK
XNA	Fayetteville, AR	AR
YAK	Yakutat, AK	AK
YUM	Yuma, AZ	AZ

Instructions

Once you have loaded the data sets into a new app in Qlik Sense, you can begin building your map. To make the example map, you must complete the following tasks:

1. Create the drill-down dimensions.
2. Add the map to our sheet.
3. Add the *State-County* area layer.
4. Add the *State-City* point layer.
5. Add the *State-Airport* point layer.
6. Add the *Resource* filter pane.

Creating the drill-down dimensions

First, you need to create three drill-down dimensions. This will create the relationships between *State* and the fields *County*, *City*, and *Airport*, enabling the *County*, *City*, and *Airport* layers to become visible after a state from the *State* layer has been selected.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click  to display the master items.
3. Click **Dimensions**.
4. Click **Create new**.
5. Select **Drill-down**.
6. Add the field *State* to the dimension.
7. Add the field *County* to the dimension.
8. After **Name**, type *State-County*.
9. Click **Create..**.
10. Add the field *State* to the dimension.
11. Add the field *City* to the dimension.
12. After **Name**, type *State-City*.
13. Click **Create..**.
14. Add the field *State* to the dimension.
15. Add the field *Airport* to the dimension.
16. After **Name**, type *State-Airport*.

17. Click **Create..**
18. Click ✓ **Done editing.**

Adding the map to the sheet

The next step is to add a map to your sheet..

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. From the assets panel, drag an empty map to the sheet.

Adding the *State-County* area layer

The first layer you add is an area layer to which you add the dimension *State-County*. In addition, you set the country as '*USA*' to ensure we map to the state of Georgia rather than the nation of Georgia.

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
2. Select **Area layer**.
3. In **Dimensions**, click **Add** and select *State-County*.
4. Click **Location**.
5. Set **Scope for locations** to **Custom**.
6. After **Country**, enter '*USA*'.
7. After **Administrative area (Level 1)**, select *State*.
8. Click **Colors**.
9. Set **Colors** to **Custom**, select **By dimension**, and select **100 colors**.
10. Select **Persistent colors**.
11. Adjust the **Opacity** slider to half opacity.
12. Click ✓ **Done editing**.

Adding the *State-City* point layer

The next layer you add is a point layer. You add *State-City* as the dimension and then set the country as '*USA*' and then set *State* as the first-level administrative area, as some counties cities in different states have the same names.

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
2. Select **Area layer**.
3. In **Dimensions**, click **Add** and select *State-City*.
4. Click **Location**.
5. Set **Scope for locations** to **Custom**.
6. After **Country**, enter '*USA*'.
7. After **Administrative area (Level 1)**, select *State*.

8. Click **Colors**.
9. Set **Colors** to **Custom**, select **Single color**, and select a color.
10. Click **Options**.
11. In **Layer** display, set **Visible drill-down levels** to **Custom**.
12. Clear **State**.
13. Click ✓ **Done editing**.

Adding the *State-Airport* point layer

The final layer you add to the map is a point layer to which you add the *State-Airport* field as the dimension. Qlik Sense recognizes IATA codes for placement, ensuring the airports are located in their actual position rather than just in their city.

Do the following:

1. From **Layers** in the properties panel, click **Add layer**.
2. Select **Area layer**.
3. In **Dimensions**, click **Add** and select *State-Airport*.
4. Click **Location**.
5. Set **Scope for locations** to **Custom**.
6. After **Country**, enter 'USA'.
7. After **Administrative area (Level 1)**, select *State*.
8. Click **Size & Shape**.
9. From **Shape**, select **Triangle**.
10. Click **Colors**.
11. Set **Colors** to **Custom**, select **Single color**, and select a color.
12. Click **Options**.
13. In **Layer** display, set **Visible drill-down levels** to **Custom**.
14. Clear **State**.
15. Click ✓ **Done editing**.

Add the *Resource* filter pane

Finally, you can optionally add a filter pane containing the field *Resource*. This provides you with a list of the available historical sites as you make selections within your map.

Do the following:

1. In the assets panel, drag and drop a filter pane into the sheet.
2. Click **Add dimension**.
3. Add a filter pane containing the field *Resource*.

Creating a map with multiple background layers

You can use multiple background layers in a map chart. This enables you to create layered base maps for your map chart data.

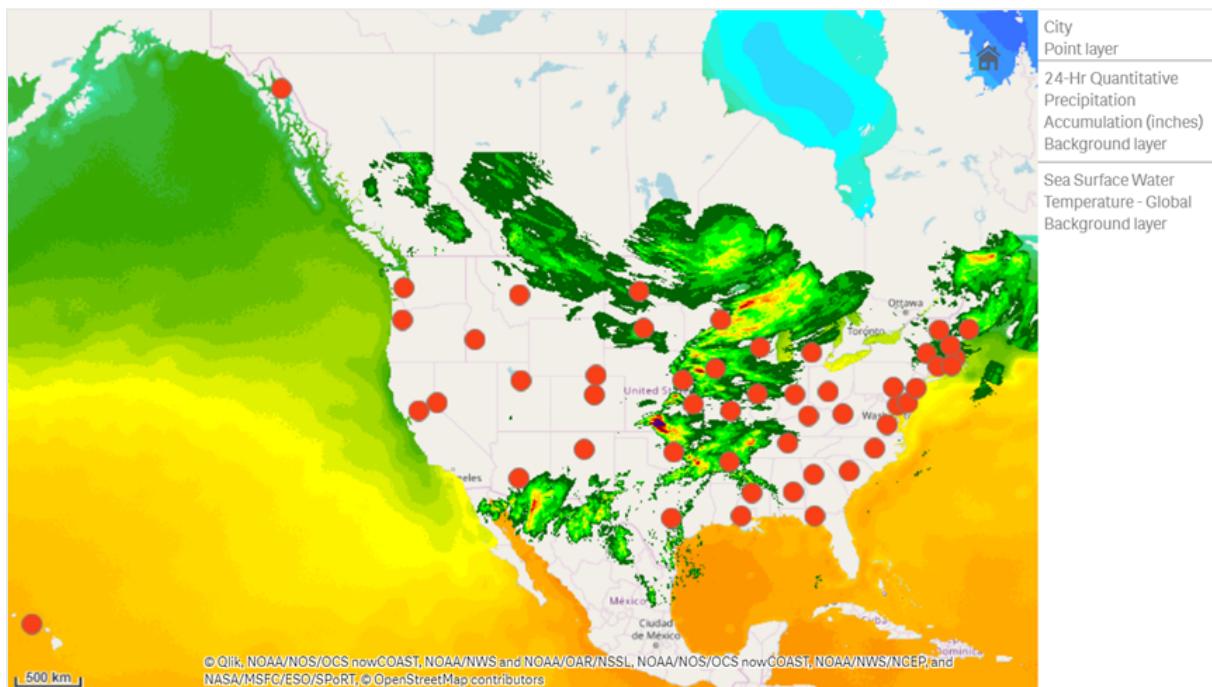
Some WMS sources include layers with transparency. With these sources, you can overlay the WMS background layer on top of a base map or other layers. Images can also be placed over other background layers. This enables you to create more complex base maps for your data.



Whether or not an WMS transparency layer supports transparency is dependent on the WMS source.

In this map example, a custom TMS is used in a background layer. Two WMS background layers are placed on top, using WMS endpoints from the National Ocean and Atmospheric Administration. The first contains sea temperature data. The second contains 24-hour rainfall data. A point layer containing cities is overlaid these background layers.

Map chart with a point layer overlaid on a TMS background layer and two WMS background layers



Dataset

This example uses a single table for data, *Cities*, with an external TMS and WMS data from external sources. You can import the two tables into your app using the web file connector.

Cities

Cities

Cities
Albany, New York
Annapolis, Maryland
Atlanta, Georgia

Cities
Augusta, Maine
Austin, Texas
Baton Rouge, Louisiana
Bismarck, North Dakota
Boise, Idaho
Boston, Massachusetts
Carson City, Nevada
Charleston, West Virginia
Cheyenne, Wyoming
Columbia, South Carolina
Columbus, Ohio
Concord, New Hampshire
Denver, Colorado
Des Moines, Iowa
Dover, Delaware
Frankfort, Kentucky
Harrisburg, Pennsylvania
Hartford, Connecticut
Helena, Montana
Honolulu, Hawaii
Indianapolis, Indiana
Jackson, Mississippi
Jefferson City, Missouri
Juneau, Alaska
Lansing, Michigan
Lincoln, Nebraska
Little Rock, Arkansas
Madison, Wisconsin
Montgomery, Alabama
Montpelier, Vermont

Cities
Nashville, Tennessee
Oklahoma City, Oklahoma
Olympia, Washington
Phoenix, Arizona
Pierre, South Dakota
Providence, Rhode Island
Raleigh, North Carolina
Richmond, Virginia
Sacramento, California
Saint Paul, Minnesota
Salem, Oregon
Salt Lake City, Utah
Santa Fe, New Mexico
Springfield, Illinois
Tallahassee, Florida
Topeka, Kansas
Trenton, New Jersey

How I built this



*This example adds layers so that they are layered on top of each other in the final order. If you add these layers in a different order, you click and drag layers into the correct order you want in **Layers**.*

Do the following:

1. Add the table *Cities* to your app. You can copy and paste the table using manual entry or you can import the table into your app using the web file connector.
2. In a sheet, add a map chart.
3. In **Map settings**, set **Base map** to **None**.
4. Add a background layer and do the following:
 - In **Data**, do the following:
 - For **Format**, select **TMS**.
 - For **URL**, enter [https://a.tile.openstreetmap.org/\\${z}/\\${x}/\\${y}.png](https://a.tile.openstreetmap.org/${z}/${x}/${y}.png).
 - For **Attribution**, enter © OpenStreetMap contributors.

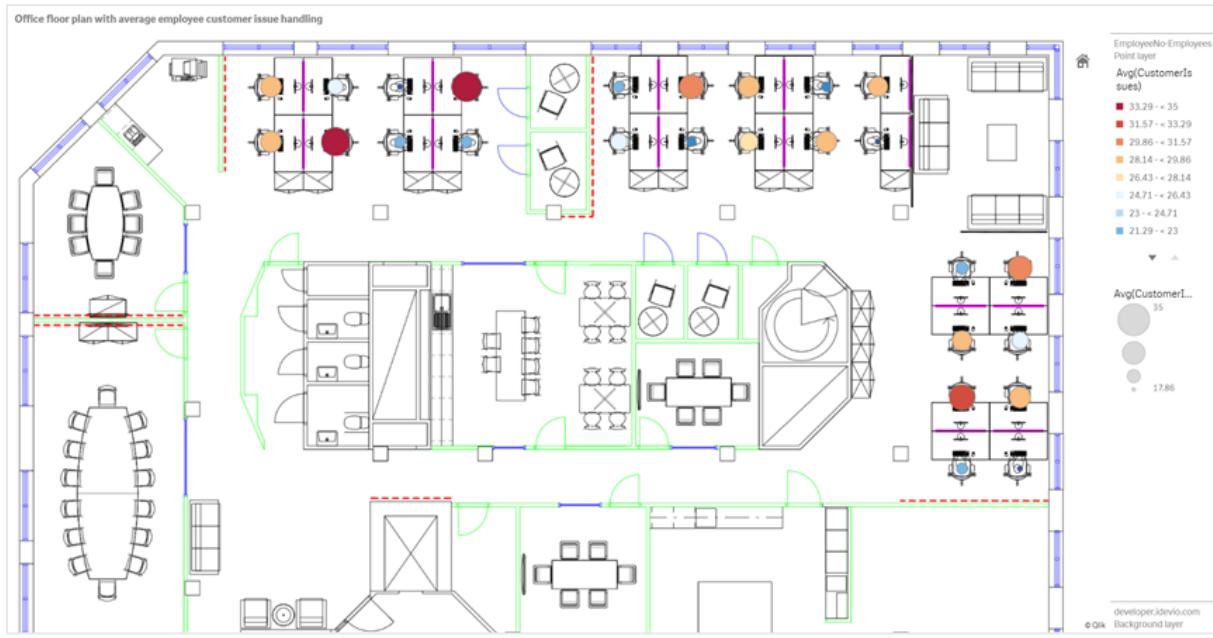
- In **Options**, do the following:
 - Set **Show legend** to **None**.
5. Add a background layer and do the following:
- For **Label**, enter *Sea Surface Water Temperature - Global*.
 - For **Format**, select **WMS** and click **WMS setup**. Do the following:
 - For **WMS server URL**, enter
https://nowcoast.noaa.gov/arcgis/services/nowcoast/analysis_ocean_sfc_sst_time/MapServer/WMServer.
 - For **Version**, select **1.3.0**.
 - Click **Load WMS**.
 - In **Step 2 (Settings)**, select the first **Image** layer.
 - Click **Save**.
 - For **Attribution**, enter *NOAA/NOS/OCS nowCOAST, NOAA/NWS/NCEP, and NASA/MSFC/ESO/SPoRT*.
6. Add a background layer and do the following:
- For **Label**, enter *24-Hr Quantitative Precipitation Accumulation (inches)*.
 - For **Format**, select **WMS** and click **WMS setup**. Do the following:
 - For **WMS server URL**, enter
https://nowcoast.noaa.gov/arcgis/services/nowcoast/analysis_meteohydro_sfc_qpe_time/MapServer/WMServer.
 - For **Version**, select **1.3.0**.
 - Click **Load WMS**.
 - In **Step 2 (Settings)**, under layers, select the third **Image** layer.
 - Click **Save**.
 - For **Attribution**, enter *NOAA/NOS/OCS nowCOAST, NOAA/NWS and NOAA/OAR/NSSL*.
7. Add a point layer and do the following:
- In **Data**, add *Cities* as the dimension.
 - In **Location**, do the following:
 - Set **Scope for locations** to **Custom**.
 - Set **Country** to '**USA**'.
 - In **Colors**, set **Colors** to **Custom** and set **Color** to *f93f17*.

Creating a map with an image background layer

You can use images as custom base maps in map visualizations. You may want to display data over a floor plan.

This example map chart displays an office floor plan with a point layer. Each bubble is positioned over an employee's desk. The weekly average of customer issues handled per day by the employee determines the size and color of each bubble.

Map example with floor plan image background with a point layer showing employees sized by the number of customer issues they have resolved



This example hides the default scale. Images do not have a consistent size scale. Image size in a map background layer is also dependent on what coordinates the user sets for the corners of the image.

Dataset

This example uses two tables and a background image. You can import the two tables into your app using the web file connector.

CustomerIssues

CustomerIssues

Date	EmployeeNumber	CustomerIssues
8/1/2019	1	24
8/1/2019	2	31
8/1/2019	3	21
8/1/2019	4	42
8/1/2019	5	24
8/1/2019	6	40
8/1/2019	7	40
8/1/2019	8	19
8/1/2019	9	23
8/1/2019	10	47

Date	EmployeeNumber	CustomerIssues
8/1/2019	11	38
8/1/2019	12	21
8/1/2019	13	22
8/1/2019	14	15
8/1/2019	15	30
8/1/2019	16	46
8/1/2019	17	41
8/1/2019	18	31
8/1/2019	19	50
8/1/2019	20	27
8/1/2019	21	35
8/1/2019	22	38
8/1/2019	23	37
8/1/2019	24	31
8/1/2019	25	13
8/1/2019	26	11
8/2/2019	1	35
8/2/2019	2	31
8/2/2019	3	33
8/2/2019	4	14
8/2/2019	5	24
8/2/2019	6	33
8/2/2019	7	40
8/2/2019	8	14
8/2/2019	9	44
8/2/2019	10	24
8/2/2019	11	37
8/2/2019	12	39
8/2/2019	13	49
8/2/2019	14	16
8/2/2019	15	42

Date	EmployeeNumber	CustomerIssues
8/2/2019	16	13
8/2/2019	17	45
8/2/2019	18	48
8/2/2019	19	46
8/2/2019	20	18
8/2/2019	21	18
8/2/2019	22	45
8/2/2019	23	47
8/2/2019	24	31
8/2/2019	25	10
8/2/2019	26	21
8/3/2019	1	16
8/3/2019	2	34
8/3/2019	3	15
8/3/2019	4	44
8/3/2019	5	49
8/3/2019	6	18
8/3/2019	7	16
8/3/2019	8	41
8/3/2019	9	27
8/3/2019	10	46
8/3/2019	11	21
8/3/2019	12	49
8/3/2019	13	38
8/3/2019	14	30
8/3/2019	15	48
8/3/2019	16	17
8/3/2019	17	42
8/3/2019	18	48
8/3/2019	19	44
8/3/2019	20	44

Date	EmployeeNumber	CustomerIssues
8/3/2019	21	12
8/3/2019	22	44
8/3/2019	23	17
8/3/2019	24	24
8/3/2019	25	25
8/3/2019	26	33
8/4/2019	1	20
8/4/2019	2	45
8/4/2019	3	32
8/4/2019	4	37
8/4/2019	5	32
8/4/2019	6	50
8/4/2019	7	41
8/4/2019	8	14
8/4/2019	9	37
8/4/2019	10	39
8/4/2019	11	28
8/4/2019	12	35
8/4/2019	13	24
8/4/2019	14	19
8/4/2019	15	25
8/4/2019	16	26
8/4/2019	17	23
8/4/2019	18	45
8/4/2019	19	48
8/4/2019	20	36
8/4/2019	21	40
8/4/2019	22	21
8/4/2019	23	10
8/4/2019	24	42
8/4/2019	25	35

Date	EmployeeNumber	CustomerIssues
8/4/2019	26	26
8/5/2019	1	24
8/5/2019	2	28
8/5/2019	3	44
8/5/2019	4	19
8/5/2019	5	34
8/5/2019	6	37
8/5/2019	7	14
8/5/2019	8	26
8/5/2019	9	38
8/5/2019	10	25
8/5/2019	11	41
8/5/2019	12	35
8/5/2019	13	48
8/5/2019	14	36
8/5/2019	15	36
8/5/2019	16	37
8/5/2019	17	31
8/5/2019	18	44
8/5/2019	19	21
8/5/2019	20	28
8/5/2019	21	13
8/5/2019	22	10
8/5/2019	23	50
8/5/2019	24	35
8/5/2019	25	11
8/5/2019	26	39
8/6/2019	1	26
8/6/2019	2	14
8/6/2019	3	45
8/6/2019	4	27

Date	EmployeeNumber	CustomerIssues
8/6/2019	5	33
8/6/2019	6	21
8/6/2019	7	14
8/6/2019	8	45
8/6/2019	9	41
8/6/2019	10	35
8/6/2019	11	35
8/6/2019	12	13
8/6/2019	13	35
8/6/2019	14	26
8/6/2019	15	40
8/6/2019	16	14
8/6/2019	17	20
8/6/2019	18	23
8/6/2019	19	11
8/6/2019	20	23
8/6/2019	21	31
8/6/2019	22	48
8/6/2019	23	39
8/6/2019	24	50
8/6/2019	25	47
8/6/2019	26	40
8/7/2019	1	49
8/7/2019	2	39
8/7/2019	3	16
8/7/2019	4	28
8/7/2019	5	46
8/7/2019	6	37
8/7/2019	7	36
8/7/2019	8	15
8/7/2019	9	18

Date	EmployeeNumber	CustomerIssues
8/7/2019	10	25
8/7/2019	11	12
8/7/2019	12	37
8/7/2019	13	40
8/7/2019	14	35
8/7/2019	15	11
8/7/2019	16	12
8/7/2019	17	22
8/7/2019	18	16
8/7/2019	19	46
8/7/2019	20	39
8/7/2019	21	41
8/7/2019	22	26
8/7/2019	23	25
8/7/2019	24	34
8/7/2019	25	50
8/7/2019	26	41

EmployeeData

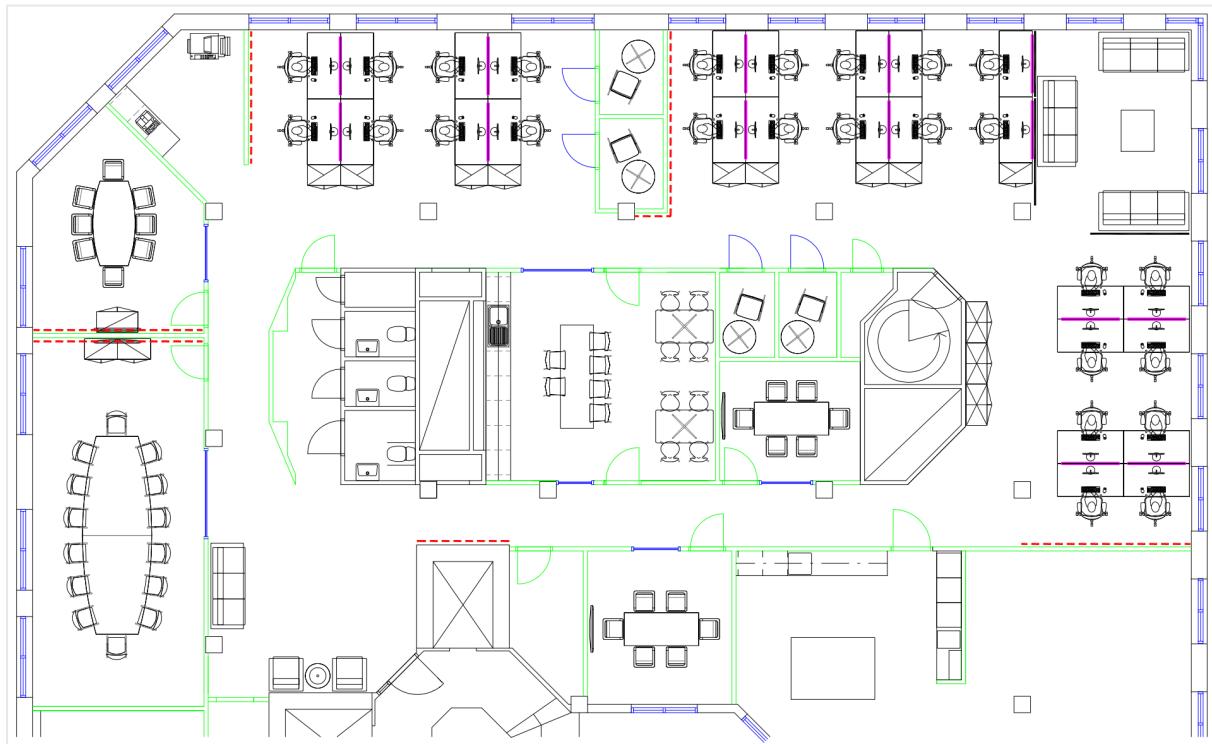
EmployeeData

Employee	EmployeeNumber	Latitude	Longitude
Nehru Pollard	1	2.1	-2.1
Duncan Bell	2	1.675	-2.1
Arthur Miller	3	2.1	-1.6
Latifah Randall	4	1.675	-1.6
Glenna Giles	5	2.1	-1.1
Chaim Gates	6	1.675	-1.1
Elijah Mcgowan	7	2.1	-0.585
Serina Richards	8	1.675	-0.585
Nora Odonnell	9	2.1	0.595
Fiona Craig	10	1.675	0.595
Kirestin Mcguire	11	2.1	1.155

Employee	EmployeeNumber	Latitude	Longitude
Francesca Wilkerson	12	1.675	1.155
Virginia Sanford	13	2.1	1.6
Beau Weeks	14	1.675	1.6
Justin Cook	15	2.1	2.2
Lisandra Sloan	16	1.675	2.2
Brody Ball	17	2.1	2.6
Kirk Welch	18	1.675	2.6
Julian Mcgee	19	0.7	3.25
Geoffrey Wheeler	20	0.7	3.7
Carter Leonard	21	0.14	3.25
Noel Watson	22	0.14	3.7
Damian Everett	23	-0.3	3.25
Justina Frazier	24	-0.3	3.7
MacKenzie Garcia	25	-0.85	3.25
Germane Carey	26	-0.85	3.7

Background image

Example background image



How I built this

Do the following:

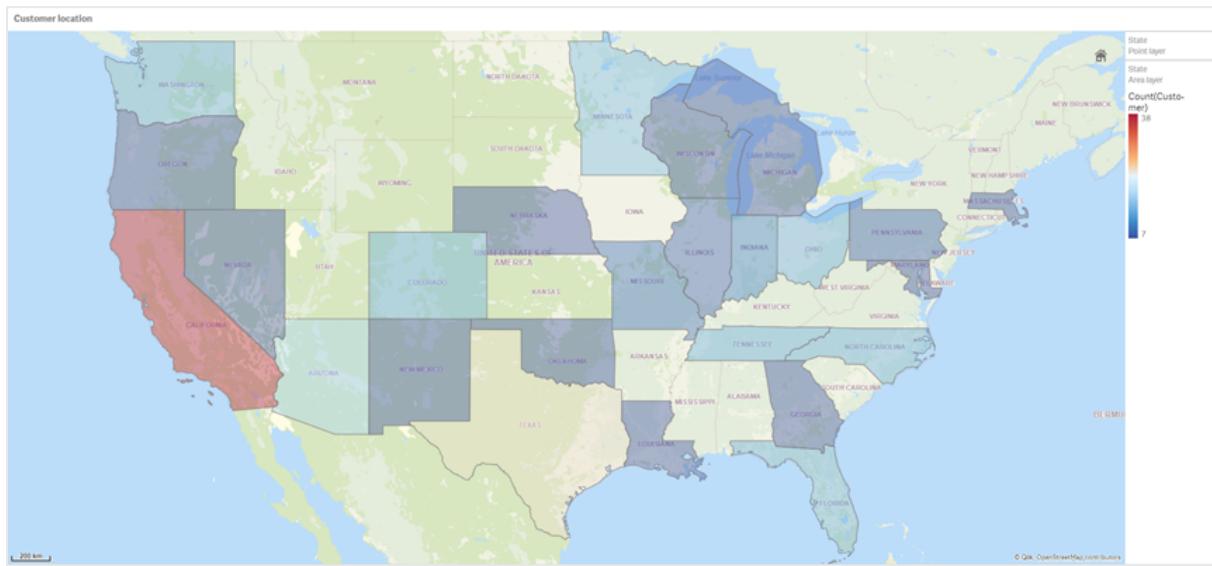
1. Add the data sources to your app and associate the tables using the *EmployeeNumber* fields in each table.
2. In a sheet, add a map chart.
3. In **Map settings**, do the following:
 - Change **Base map** to **None**.
 - Change **Projection** to **User defined (degrees)**.
4. In **Appearance > Presentation**, do the following:
 - Toggle the **Scale bar** to **Off**.
 - Set **Limit zoom levels** to **Custom** and set the zoom to *7x zoom - 8x zoom*.
5. Add a background layer to the map chart and in **Data**, do the following:
 - In **Format**, select **Image**.
 - Set the **URL** to https://help.qlik.com/en-US/sense/Subsystems/Hub/Content/Resources/Images/ui_map-image-background-example.png.
 - Set the **Top left corner** to 2.5 for **Top (latitude)** and -4.1 for **Left (longitude)**.
 - Set the **Bottom right corner** to -2.5 for **Bottom (latitude)** and 4.1 for **Right (longitude)**.
6. Add a point layer to the map chart and do the following:
 - In **Data**, add Employee as the field.
 - In **Location**, select **Latitude and longitude fields**. Set *Lat* as the **Latitude field** and *Long* as the **Longitude field**.
 - In **Size & Shape**, set **Size by***Avg(CustomerIssues)*.
 - In **Color**, do the following:
 - Set **Color** to **Custom** and color **By measure**.
 - Select *Avg(CustomerIssues)* as the measure.
 - Select **Diverging classes** as the color scheme.

Creating a map focused on a region

You can configure your map charts to stay focused on a region of interest, such as a country. You can also set the minimum and maximum zoom levels of your map. This controls how far the Qlik Sense zooms into the map when you make selections.

The following example explains how to make a map with the focus locked on a single region and with a custom maximum zoom level.

The map chart with panning limited to just the region of interest

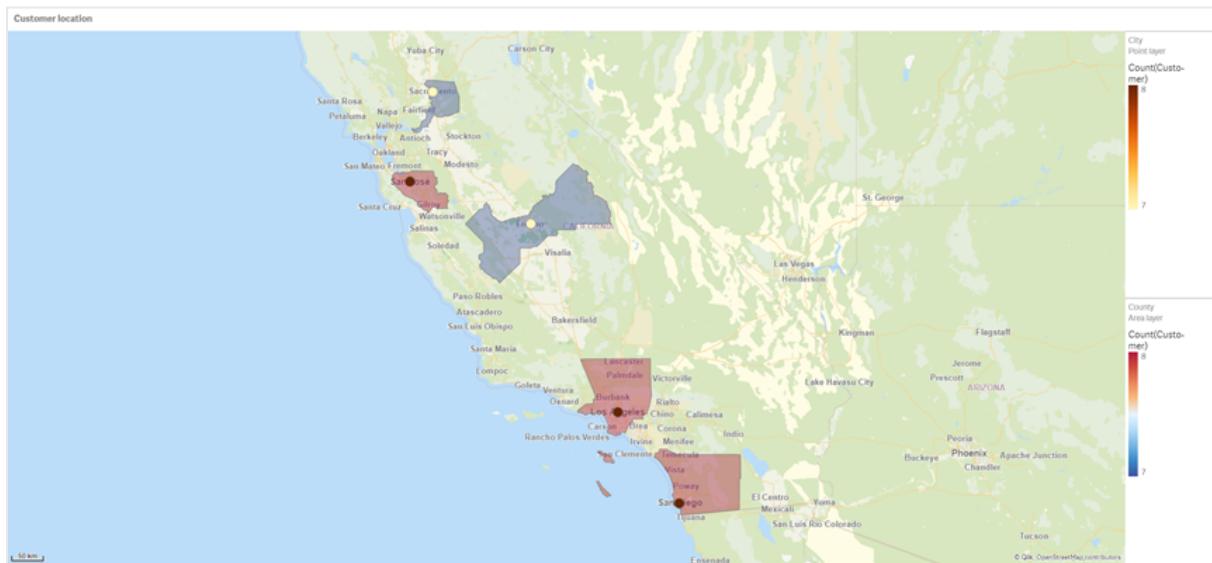


This map has set a pan limit so users cannot pan the focus of the map visualization away from the region of interest. Users can drill down between separate map layers, but they cannot scroll outside of the initial view.

This map visualization uses a point layer and an area layer with two drill-down dimensions. One drill-down dimension enables the area layer to display counties after states are selected. The other drill-down dimension enables the point layer to display cities as points after a state is selected in the area layer.

A zoom limit is applied to the map chart. When a state is selected in the area layer, the map drills down to display the counties in the area layer and the cities in a separate point layer. The map remains zoomed a level appropriate for a general regional overview.

A map chart with a state selection applied



Dataset

This example uses two sets of data. You must import this data into Qlik Sense. You can add the data to a spreadsheet and import it into Qlik Sense. You can also import the table from this help page as a web file.

City data

Sample city data

City	City Code	County	State	Latitude	Longitude
New York	1	New York County	US	40.730599	-73.986581
Los Angeles	2	Los Angeles County	CA	34.053678	-118.242702
Chicago	3	Cook County	IL	41.875555	-87.624421
Philadelphia	5	Philadelphia County	PA	39.952335	-75.163789
Phoenix	6	Maricopa County	AZ	33.446768	-112.075672
San Antonio	7	Bexar County	TX	29.4246	-98.49514
San Diego	8	San Diego County	CA	32.717421	-117.162771
Dallas	9	Dallas County	TX	32.776196	-96.796899
San Jose	10	Santa Clara County	CA	37.34385	-121.883135
Austin	11	Travis County	TX	30.271129	-97.7437
Indianapolis	12	Marion County	IN	39.76838	-86.158045
Jacksonville	13	Duval County	FL	30.332184	-81.655651
Columbus	15	Franklin County	OH	39.96226	-83.000706
Charlotte	16	Mecklenburg County	NC	35.227087	-80.843127
Detroit	17	Wayne County	MI	42.348664	-83.056738
Memphis	18	Shelby County	TN	35.149022	-90.051628
Seattle	19	King County	WA	47.603832	-122.330062
Denver	20	Denver County	CO	39.739154	-104.984703
Boston	22	Suffolk County	MA	42.360482	-71.059568
Nashville	23	Davidson County	TN	36.162226	-86.774342
Baltimore	24	Baltimore City County	MD	39.290861	-76.610807
Portland	25	Multnomah County	OR	45.520247	-122.674195
Las Vegas	26	Clark County	NV	36.166286	-115.149225
Milwaukee	27	Milwaukee County	WI	43.034993	-87.922497
Albuquerque	28	Bernalillo County	NM	35.084103	-106.650985

City	City Code	County	State	Latitude	Longitude
Tucson	29	Pima County	AZ	32.221742	-110.926476
Fresno	30	Fresno County	CA	36.73082	-119.699202
Sacramento	31	Sacramento County	CA	38.581572	-121.4944
Kansas City	32	Jackson County	MO	39.084469	-94.56303
Colorado Springs	33	El Paso County	CO	38.833958	-104.825348
Atlanta	34	Fulton County	GA	33.749099	-84.390185
Omaha	35	Douglas County	NE	41.258732	-95.937873
Raleigh	36	Wake County	NC	35.780402	-78.639078
Miami	37	Miami-Dade County	FL	25.774266	-80.193659
Minneapolis	38	Hennepin County	MN	44.9773	-93.265469
Tulsa	39	Tulsa County	OK	36.152436	-95.990409
Cleveland	40	Cuyahoga County	OH	41.505161	-81.693445
New Orleans	41	Orleans Parish	LA	29.949932	-90.070116

Customer

Sample customer data

Customer	Customer Number	City Code
A Superior System	10000453	1
Beech Aircraft Corporation	10003882	1
Deere and Company	10009863	1
Gailey Enterprises	10012851	1
J.A. Bauer Pottery Company	10017852	1
Old Towne Creations	10020715	1
Real World	10022755	1
ValueClick	10025878	1
A&G	10000457	2
Bell Canada Enterprises	10004207	2
Échange CAC Exchange	10010832	2
Gainunion	10012885	2
J.M. Haggar	10018129	2
OnDeck Systems	10020849	2

Customer	Customer Number	City Code
Reflex Presentations	10022962	2
Wasabi	10026294	2
Aadast	10000471	3
Bendix Corporation	10004255	3
ECl com	10010855	3
Galaxy Marketing Associates	10012907	3
Johnson and Higgins	10018352	3
One Planet Solutions	10020864	3
Relcom	10022978	3
Wayne'sWorld	10026334	3
ABI TruTrac	10000488	5
Best Way! Imaging	10004602	5
Ed Stefanov	10010881	5
Gamacles	10013011	5
Joy Line	10018371	5
Onebox	10020907	5
SageGroup	10023511	5
VEI	10026023	5
AboveNet	10000496	6
Bezeq	10005043	6
EDA Today	10010882	6
Gamma One Conversions	10013039	6
K	10018518	6
Keystroke Quality	10019420	6
Outsource Documents	10020982	6
PAGE	10021240	6
Sarcom	10023703	6
Systems of Missouri	10024915	6
Velos Medical Informatics	10026038	6
Abplus	10000497	7

Customer	Customer Number	City Code
BF Datacom	10005099	7
Edmark	10010923	7
GammaGraphX (GGX)	10013052	7
K&K Enterprises	10018603	7
Overdreams	10021006	7
SAS Institute	10023757	7
Ventana s Group	10026081	7
ABSolute	10000499	8
BH Feldman Consulting	10005236	8
Elbit Oy	10010990	8
Gammel Group	10013061	8
Karickal Exports	10018868	8
Owl's Eye Productions	10021015	8
SAT-SAGEM (usa)	10023780	8
White Oak Interactive	10026365	8
Absolute Magic	10000501	9
Bibb Manufacturing Company	10005376	9
Eloi Companies	10011018	9
Gandalf Systems	10013079	9
KAT Micro Distributing	10018877	9
P.C'S	10021111	9
Science Applications International (SAIC)	10023964	9
Xcert	10026521	9
Abstract	10000502	10
BidCast	10005620	10
Elucidex	10011052	10
Ganymede	10013080	10
Kendrick Jansen	10018950	10
P.C.G. Associates	10021107	10
Screen Digest	10024016	10

Customer	Customer Number	City Code
Xyratex	10026868	10
AC Exchange	10001103	11
Bien Logic	10005688	11
Embedded Support Tools (EST)	10011089	11
Garbee and Garbee	10013127	11
Kennecott Copper Corporation	10018957	11
Pacific Bell	10021160	11
Smith Manufacturing Company	10024477	11
Yurie Systems	10027119	11
AC&E	10001263	12
Acara	10001786	12
Big	10005810	12
Big Picture Technologies	10005919	12
EMC	10011093	12
Garlin Imports	10013137	12
KENROB and Associates	10019066	12
Packet Design	10021232	12
Sterling Armament Company	10024704	12
Zero G	10027370	12
Acacia	10001784	13
Big Mountain Multi	10005861	13
Emergency	10011286	13
Gate9th	10013312	13
Kerite Company	10019194	13
PADL	10021239	13
Sun Microsystems	10024880	13
Acc Tonec	10001818	15
Big Planet	10005922	15
Enterprises	10011355	15
Gatierf Publications	10013341	15

Customer	Customer Number	City Code
kGS	10019469	15
Page Marketing	10021242	15
T & S	10024916	15
Accel Partners	10002114	16
Biz-comm	10006836	16
Equitable Life Assurance Society	10011360	16
GCC Technologies	10013376	16
KillerGraffix	10019502	16
Page Research	10021270	16
T.F.C.	10024918	16
Accent Interactive	10002115	17
Boott Cotton Mills Corporation	10006916	17
Farmland Industries	10011499	17
GDC	10013426	17
Kim Tom Co	10019512	17
PageBoy	10021283	17
T.J.T. International	10024919	17
Accent Systems	10002117	18
Borden Company	10006917	18
Farrell Lines Company	10011546	18
GEAR	10013538	18
Kimball (W.W.) Company	10019514	18
PagePoint	10021286	18
T.M. Denton Consultants	10024924	18
Access Point	10002128	19
Accidental	10002137	19
Boston and Albany Railroad Company	10006919	19
Bre-X	10007117	19
FCS	10011600	19
Federated Co-Operatives Limited	10011732	19

Customer	Customer Number	City Code
GearSource	10013572	19
Gehlken Enterprises	10013670	19
Levi Strauss and Company	10019783	19
Lobster Productions	10019952	19
PageSites	10021297	19
PageWeavers	10021302	19
T3West	10024930	19
Tag Systems	10024942	19
AccessWare	10002136	20
Brazilian Traction, Light and Power Company	10006977	20
Federal Express Corporation	10011623	20
Gebbie Press	10013574	20
Lewis Grocer Company	10019812	20
PageWave	10021300	20
Tadpole	10024940	20
ACCPAC International	10002138	22
Brentano's	10006983	22
FenP Innovators	10011821	22
Greymac Trust Company	10013870	22
Lucky	10020080	22
Pallister Management	10021305	22
Take 3	10025022	22
Accrue	10002139	23
C&C	10007134	23
Fentek Industries	10011842	23
Grove Farm Co., Inc.	10013871	23
Market	10020181	23
Palo Alto	10021331	23
Talarian	10025024	23
Accton	10002140	24

Customer	Customer Number	City Code
C. Hoelzle Associates	10007163	24
Fenwick & West	10011870	24
Grumman Corporation	10013899	24
Maui Island(MIC)	10020193	24
Panasonic	10021350	24
Target	10025052	24
AccuCom	10002142	25
C.O.F.	10007183	25
Ferrari	10011918	25
Guarantee Mutual Life Company	10014043	25
MaxBan	10020227	25
Pangaea Systems	10021380	25
Taroch	10025063	25
ACCUCOMP	10002144	26
C3 Development	10007253	26
FiberPlex	10011979	26
GURUs	10014059	26
MaXpeed	10020239	26
Panix	10021411	26
Tartanchase	10025086	26
AccuLAN	10002149	27
C3I	10007258	27
FICOM	10012023	27
H.H. Robertson Company	10014064	27
MAXSTRAT	10020247	27
Paraclipse	10021494	27
Taskers of Andover, Ltd.	10025158	27
Accumedic Systems	10002150	28
Cabco West	10007268	28
Fifth Generation	10012128	28

Customer	Customer Number	City Code
HarborGroup	10014238	28
Maxtor	10020256	28
Parian Development Group	10021575	28
Tatters	10025213	28
Ace Equipment	10002153	29
Cadtex	10007309	29
Fifth Moon	10012157	29
HarveyOpolis	10014540	29
Maxwell Laboratories	10020266	29
Password Busters	10021641	29
Teamaker	10025239	29
Acer	10002154	30
Cajun	10007448	30
Filene's	10012158	30
HBS	10014802	30
Mb digital marketing	10020281	30
Patrick Townsend & Associates	10021677	30
Teammax	10025241	30
ACES Research	10002155	31
CalComp	10007457	31
Filetron	10012161	31
Hederman Brothers	10015011	31
McGraw-Hill Publishing Company	10020306	31
Patton Enterprises	10021736	31
Teamsolve	10025242	31
ACI	10002159	32
Caleidoscopio S.r.l.	10007468	32
Emergent	10011345	32
FIND/SVP	10012305	32
GateMaster	10013332	32

Customer	Customer Number	City Code
Hekimian Laboratories	10015253	32
Mead Corporation	10020323	32
Paul Budde	10021750	32
Todd Shipyards Corporation	10025267	32
ACME Laboratories	10002161	33
Calypso	10007536	33
Finder	10012328	33
Helfand Enterprises	10015267	33
Medics	10020332	33
PayMaxx	10021794	33
Transylvania Company	10025288	33
Acsior	10002412	34
Camelot IT	10007591	34
Finest Host	10012369	34
Helius	10015325	34
Melville Shoe Corporation	10020340	34
Payton Group International	10021800	34
UBF	10025304	34
Action Systems	10002501	35
Camros	10007866	35
Finisar	10012399	35
Hercules Incorporated	10015686	35
Mersey Docks	10020364	35
Pick Professionals	10021802	35
UK Total.net	10025353	35
ActivCard	10002520	36
Champion International	10009606	36
Firehole	10012514	36
hesketh.com	10015789	36
Nancy Leffingwell Enterprises	10020449	36

Customer	Customer Number	City Code
Picka	10021804	36
UKnames.net	10025359	36
Active	10002526	37
Dan River Mills, Inc.	10009633	37
FireTrans	10012600	37
Hewlett Packard	10015886	37
Nbase	10020491	37
PLAINFIELD ROOFING AND SHEET METAL	10026438	37
Ultimate Group	10025392	37
Aberdeen	10000486	38
Ad-vantage	10003687	38
Besicorp	10004516	38
Data General Corporation	10009639	38
ECS Associates	10010865	38
Galaxy Systems	10012927	38
G-Com International	10013397	38
Hidden Oak	10015956	38
Jones and Laughlin Steel Corporation	10018363	38
NBTel	10020492	38
One World	10020868	38
QualityLogic	10022166	38
Safe	10023495	38
Ultra-Image	10025418	38
Vectrix	10026009	38
Ad Agency	10003554	39
Day and Zimmermann, Inc.	10009645	39
G.R. Barron	10012761	39
Hill and Knowlton, Inc.	10016113	39
NBX	10020515	39
Quallaby	10022167	39

Customer	Customer Number	City Code
Underwriters Laboratories	10025467	39
Associates	10003749	40
Dayton Rubber Company	10009650	40
Gadzoox Microsystems	10012792	40
iBEAM	10016780	40
Nematron	10020662	40
Quantum	10022201	40
Universal Access	10025608	40
Beckman Instruments, Inc.	10003857	41
De Postel	10009669	41
Gaecom	10012799	41
Id	10017290	41
Oki	10020672	41
Rdlabs	10022746	41
Userland	10025737	41

How I built this

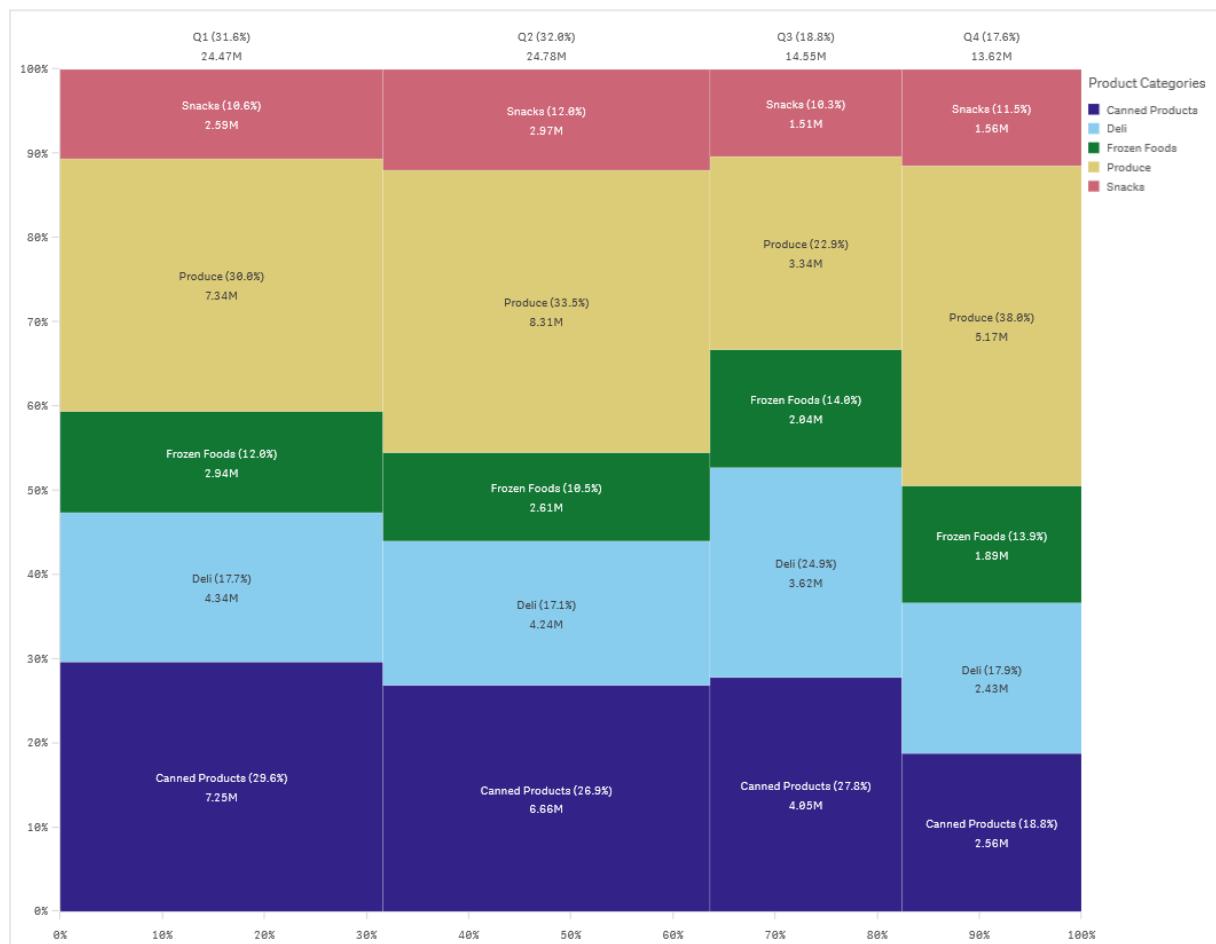
Do the following:

1. Import the data into **Data manager**.
2. Make the recommended associations between the two tables in **Data manager**.
3. Create two drill-down dimensions, one of *State* and *County* and one of *State* and *City*.
4. Add a map and in **Presentation**, set a **Custom** zoom level with a maximum zoom of **7x zoom**.
5. Add a point layer with *State -City* as the dimension. Set the following properties:
 - In **Location**, set the scope to **Custom** and enter 'USA' in **Country**.
 - In **Size & Shape**, set **Size by** to *Count(Distinct(Customer))*.
 - In **Colors**, color by measure using *Count(Distinct(Customer))*.
 - In **Options>Layer display**, set **Visible drill-down levels** to **Custom** and select *City*.
6. Add an area layer with *State-County* as the dimension. Set the following properties:
 - In **Location**, set the scope to **Custom**, enter 'USA' in Country, and *State* in **Administrative area (Level 1)**.
 - In **Colors**, color by measure using *Count(Distinct(Customer))* and set the **Opacity** slider to 40%.
7. Center the region in the map and then set **Limit pan navigation** to **Custom** and click **Set pan limit**.

Mekko chart

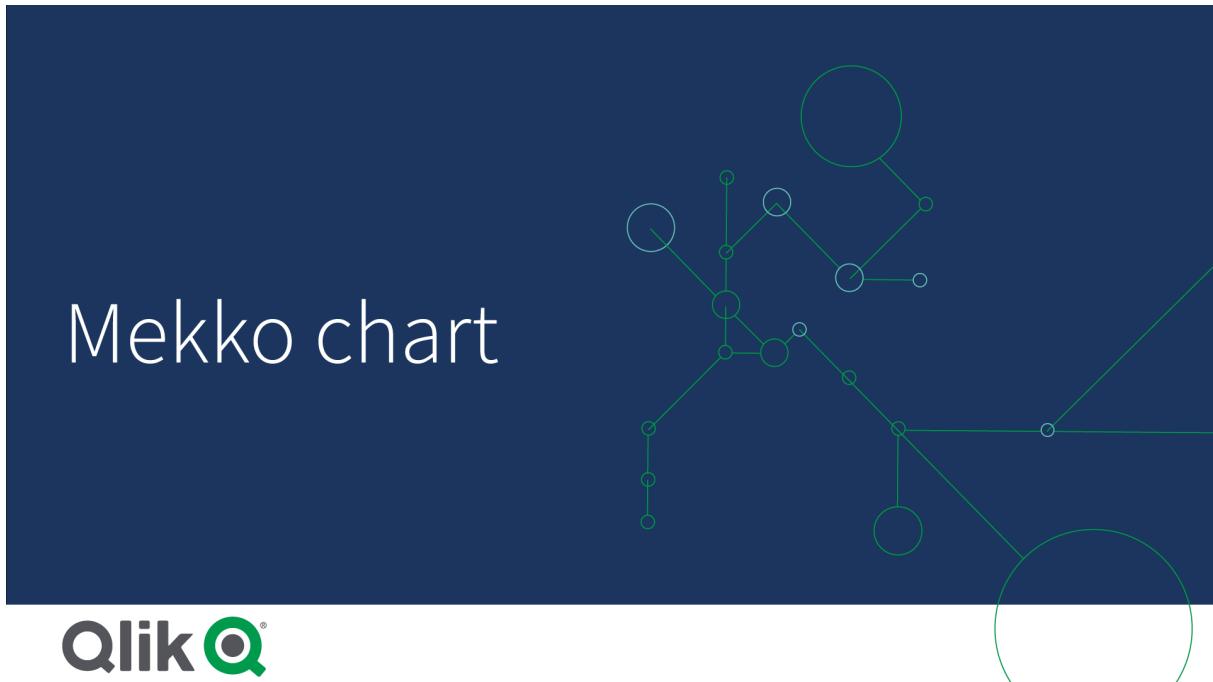
The mekko chart is suitable for comparing groups, while being able to compare category items contained within these groups. The dimension axis shows the groups, while the measure axis shows the normalized percentage value for each category item. The size of each group shows its value.

In this image, the dimensional groups represent different yearly quarters. The width of each quarter shows the normalized percentage value of the sum of sales, for that quarter. Each quarter is divided in several product categories, such as Produce, Canned Products, and Frozen Foods. The height of each product category corresponds to the normalized percentage value of the sum of sales of that product category, within that quarter. This allows you to easily compare between product categories inside a specific quarter, and between different quarters.



Sorting is automatic according to size. By default, the coloring is by dimension, with 12 colors, but that can be changed in the properties panel. You can decide which dimension to color by. In this example, the coloring is by product category to clearly distinguish the different categories and how their contribution to the sales change in each quarter.

If the data set contains negative values, a text message is shown stating that the negative values cannot be displayed.



When to use it

Use a mekko chart when you need to visualize normalized percentage values of grouped data. Mekko charts should primarily be used with values that can be aggregated.

Advantages

The mekko chart is easy to read and understand. You get a good comparison between groups, and a comparison of the impact of categories within each group.

Disadvantages

The mekko chart does not work so well with many dimension values due to the limitation of the axis length.

Mekko charts are not good when there is a big difference in the magnitude of the measure values. Nor is a mekko chart the right choice when mixing absolute and relative values.

Negative values cannot be displayed in mekko charts.

Creating a mekko chart

You can create a simple mekko chart on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty mekko chart to the sheet.
2. Click **Add dimension** and select a dimension or a field. This dimension defines the grouping.

3. Add a second dimension in the order of hierarchy level. The second dimension you add defines the categories.
4. Click **Add measure** and select a measure or create a measure from a field.

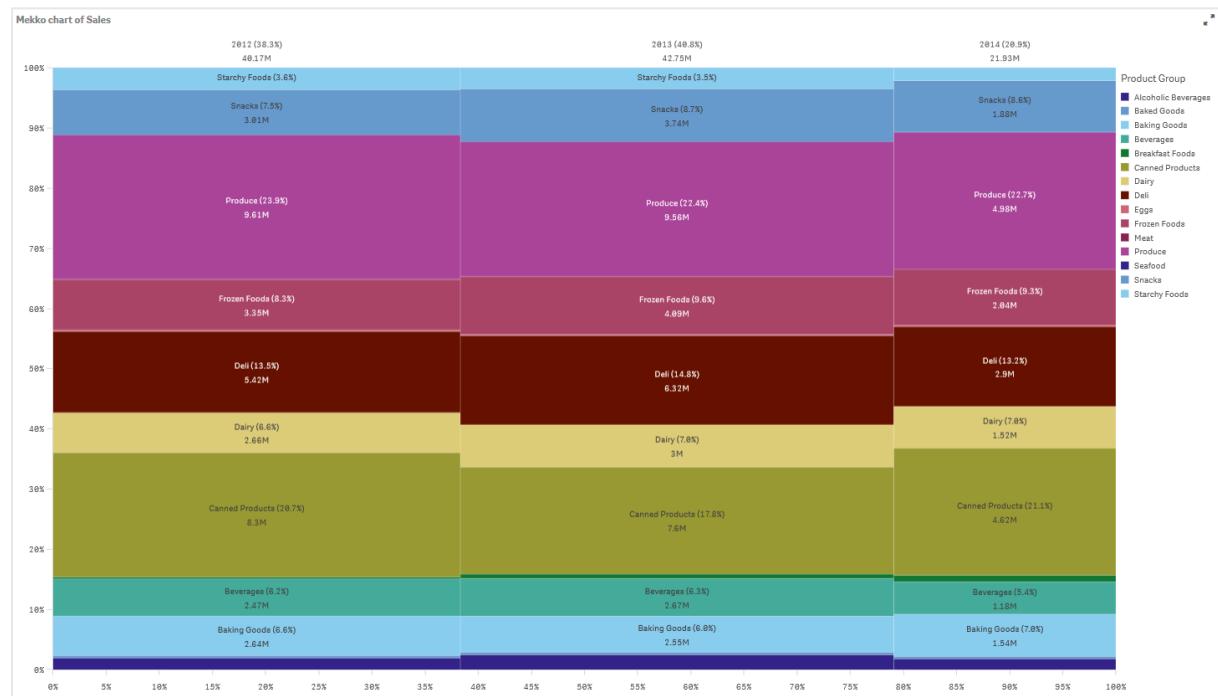
In a mekko chart you need two dimensions and one measure.

Each bar corresponds to the first dimension, divided into smaller categories based on the second dimension. The values of the measures determine the relevant height of the categories inside each bar, as well as the size of the bars.

When you have created the chart, you may want to adjust its appearance in the properties panel.

Comparing groups and group categories against a measure with a mekko chart

This example shows how to make a mekko chart to compare the sales between different years as well as the impact specific product groups have on a year's sales, and how that compares to other years.



Dataset

In this example, we will use two data files available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial, and the files are available in the *Tutorials source* folder:

- *Sales.xls*
- *Item master.xls*

To download the files, go to [Tutorial - Building an App](#).

Add the two data files to an empty app, and make sure that they are associated by *Item Number*.

The dataset that is loaded contains sales data. The *Item master* table holds the information about the items ordered, such as product groups.

Measure

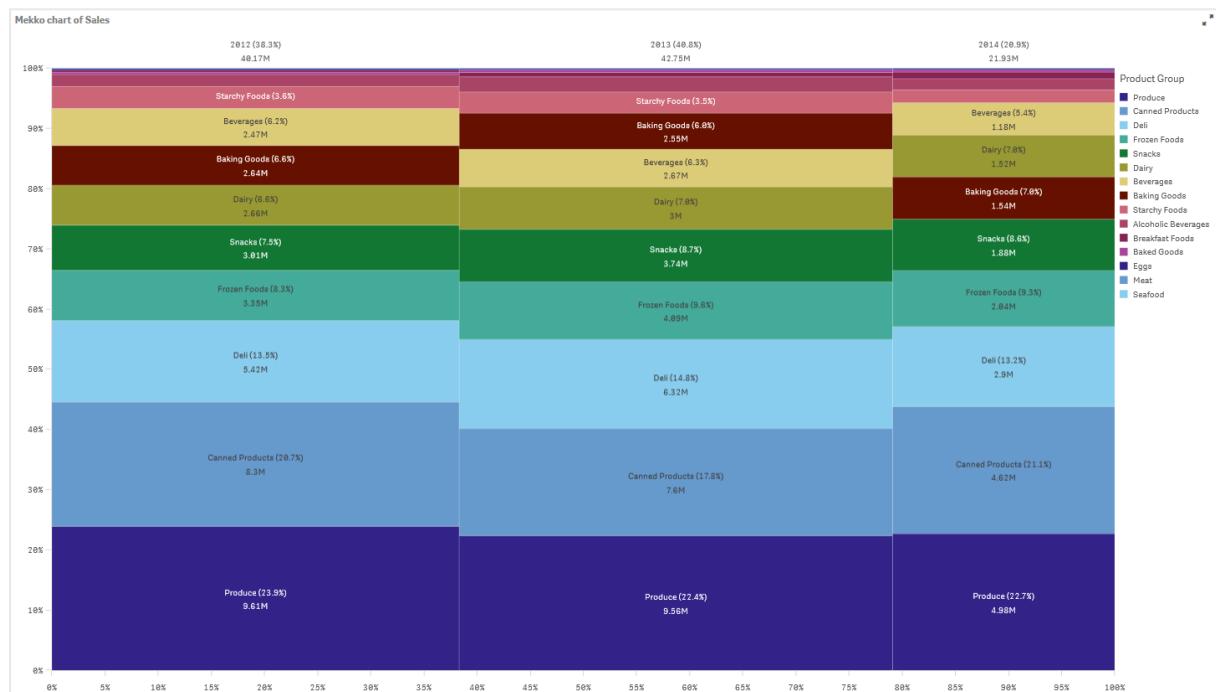
We use the sales volume as the measure, by creating a measure in Master items with the name *Sales*, and the expression `sum(sales)`.

Visualization

We add a mekko chart to the sheet and set the following data properties:

- **Dimension:** Date.autoCalendar.Quarter (year).
- **Dimension:** Product Group (product group)
- **Measure:** *Sales*; the measure that was created as a master item.

The following mekko chart is created:



The size of each column represents the volume of sales for each year. The size of the different sections inside each column represent the volume of sales for each specific product group during that year.

Notice how the sum of all three yearly percentages adds up to 100%. The same applies to the percentages of the product groups section of each year. This is because the percentages are normalized, so the size of each section is relevant to the total.

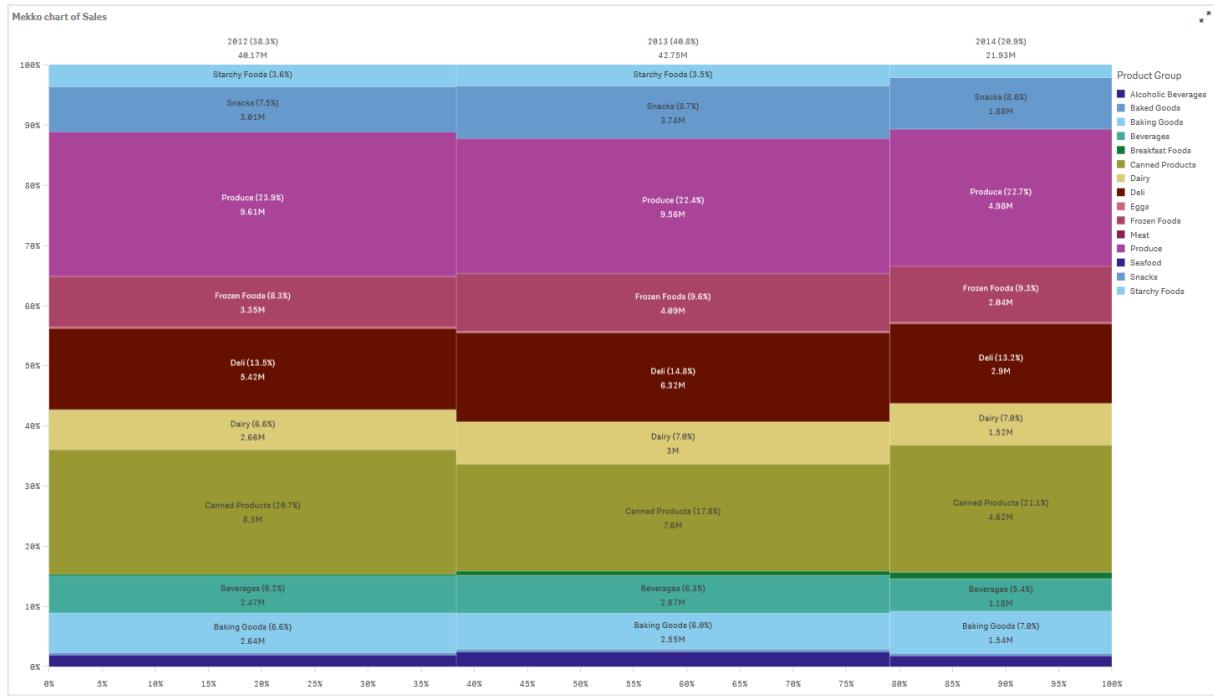
We want to have a better overview of how each product group's sales change from one year to the next. This is not obvious for product groups such as Baking Goods, Dairy, or Beverages which do not sit side by side from one column to the next. We want to display a different sorting, one based on the product groups.

This can be changed under **Sorting** in the properties panel.

Set the sorting order to the following:

1. Date.Year
2. Product Group
3. Sum(Sales)

The chart becomes as follows:



Discovery

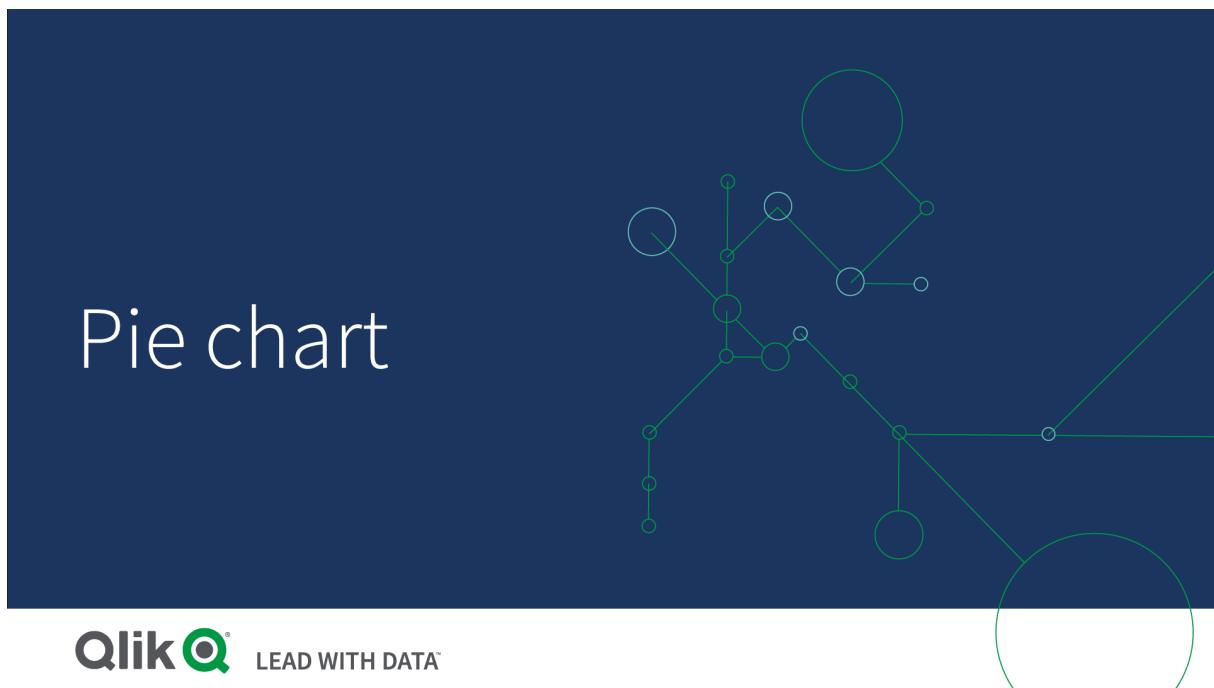
The mekko chart visualizes the normalized percentage of sales per year for different product groups, as well as the normalized percentage of sales of each year. The visualization is sorted in order of product group, per year. You can hover the mouse pointer over a product group and view the details.

In the mekko chart we can see that Produce has the highest sales volume throughout the three years.

Pie chart

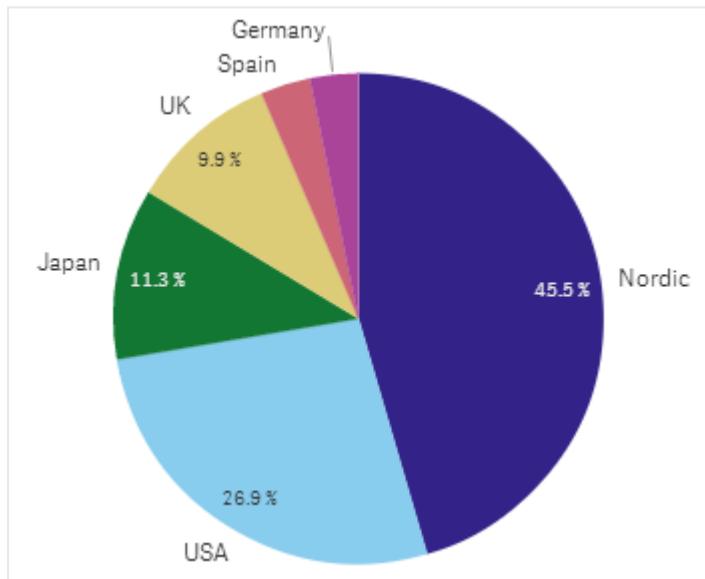
The pie chart displays the relation between values as well as the relation of a single value to the total. You can use a pie chart when you have a single data series with only positive values.

In the pie chart, the dimensions form sectors of the measure values. A pie chart can have one dimension and up to two measures. The first measure is used to determine the angle of each slice in the chart.



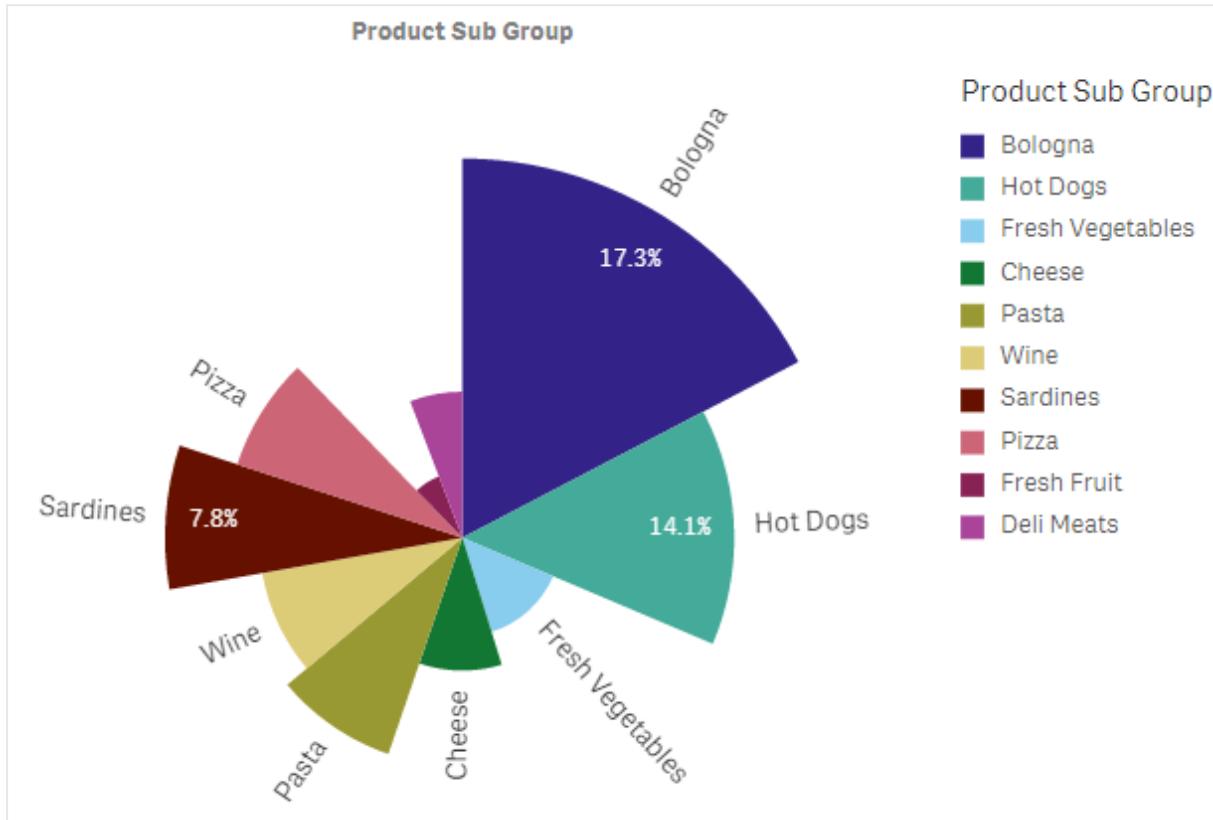
Qlik  LEAD WITH DATA™

Sales per region in a pie chart.



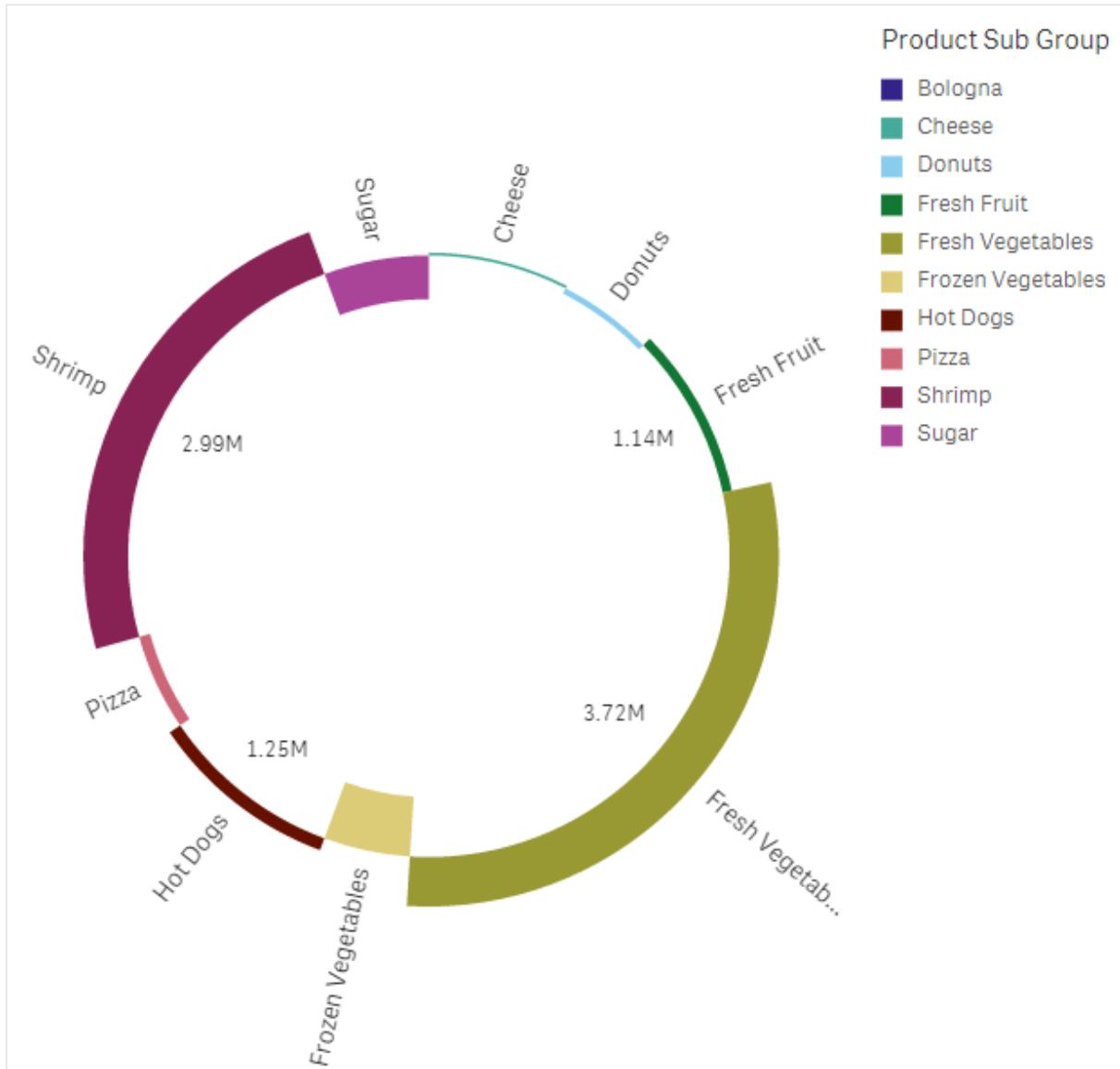
Optionally, a second measure can be used to determine the radius of each pie slice. This style of pie chart is also known as a rose chart.

Sales by product sub group in a pie chart with the average sales per invoice sales determining slice radius



In the pie presentation of the pie chart, negative values in the radius measure are not supported and will be excluded. Negative values in the radius measure are supported in the donut presentation and will point in towards the center of the pie chart.

Sales by product sub group in a pie chart with radius measure containing a comparison of sales from the previous year



When to use it

The primary use of a pie chart is to compare a certain sector to the total. The pie chart is particularly useful when there are only two sectors, for example yes/no or queued/finished.

We do not recommend that you compare the results of two pie charts with each other.

Advantages

The pie chart provides an instant understanding of proportions when few sectors are used as dimensions. When you use 10 sectors, or less, the pie chart keeps its visual efficiency.

Disadvantages

It may be difficult to compare different sectors of a pie chart, especially a chart with many sectors.

The pie chart takes up a lot of space in relation to the values it visualizes.

Creating a pie chart

You can create a pie chart on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty pie chart to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.

The following settings are used by default in a pie chart:

- The top 10 sectors are presented in descending size order, clockwise.
- Colors are presented by dimension.
- Value labels are presented in percent.

After you have created the pie chart, you may want to add a radius measure or adjust its appearance and other settings in the properties panel.

Pivot table

The pivot table presents dimensions and measures as rows and columns in a table. In a pivot table you can analyze data by multiple measures and in multiple dimensions at the same time.

You can rearrange measures and dimensions by pivoting rows and columns.



When to use it

The pivot table is useful when you want to include several dimensions or measures in a single table, and then want to reorganize them to see different subtotals.

Advantages

The pivot table is very powerful when you want to analyze multiple dimensions and measures at once, and then reorganize them to get a different perspective on your data. You can expand the rows you are interested in while keeping the rows in the rest of the table collapsed.

Disadvantages

The pivot table may seem a bit complicated, and does not give insights at a glance.

Creating a pivot table

You can create a new pivot table on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty pivot table to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.

You can adjust appearance and other settings in the properties panel.



Column width is automatically set to keep columns together for improved readability. You can adjust the width of the dimension column by dragging the header divider. The width of individual measure columns will still be automatically set. Double-click the header divider to reset to automatic column width.

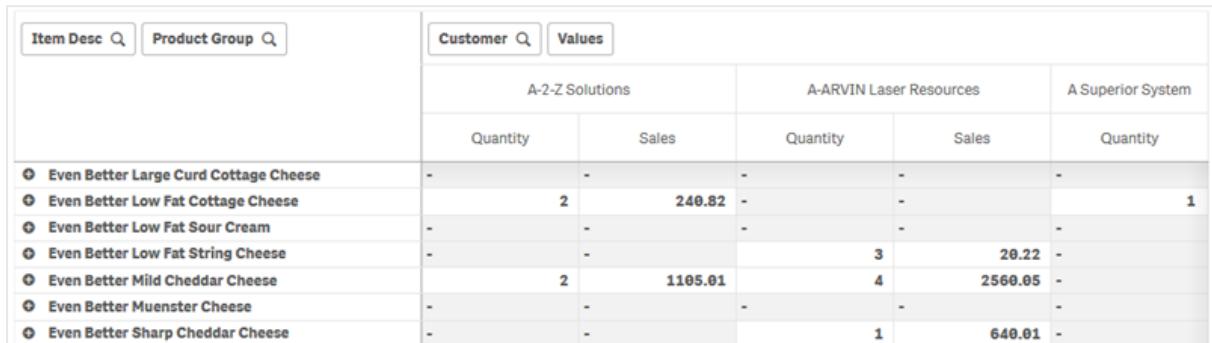
Pivoting your data in the table

When you want to rearrange your data, drag the items to a new column or row.

In the following pivot table, the dimension *Customer* has been dragged to the position after *Product Group*. The dimension *Item Desc* has been moved to the position before *Product Group*. As a consequence, the dimensions are now sorted by *Item*, primarily. Focus has shifted from *Customer* to *Item Desc*. You can find out the quantities and sales for each customer by clicking . This will expand a dimension.

Item Desc	Product Group	Customer	Values	
			Quantity	Sales
● American Beef Bologna			166	4346.12
● American Chicken Hot Dogs			173	15115.88
● American Cole Slaw			156	3979.37
● American Corned Beef			1771	211676.74
● American Foot-Long Hot Dogs			52	2267.24
● American Low Fat Cole Slaw			16	-37.75

By moving the dimension *Customer* from rows to columns, you retain focus on the dimension *Item Desc*, but you also get the distribution of items per customer. The move has made the pivot table more information dense.



The screenshot shows a pivot table interface with two search bars at the top: 'Item Desc' and 'Product Group'. Below the search bars are two buttons: 'Customer' and 'Values'. The main area of the table has three columns representing different customers: 'A-2-Z Solutions', 'A-ARVIN Laser Resources', and 'A Superior System'. Each customer row contains two columns: 'Quantity' and 'Sales'. The data rows list various cheese items, such as 'Even Better Large Curd Cottage Cheese', 'Even Better Low Fat Cottage Cheese', etc., with their corresponding quantities and sales figures across the three customers.

Item Desc	Product Group	Customer				
		Values		A-2-Z Solutions		A-ARVIN Laser Resources
		Quantity	Sales	Quantity	Sales	Quantity
Even Better Large Curd Cottage Cheese		-	-	-	-	-
Even Better Low Fat Cottage Cheese		2	240.82	-	-	1
Even Better Low Fat Sour Cream		-	-	-	-	-
Even Better Low Fat String Cheese		-	-	3	20.22	-
Even Better Mild Cheddar Cheese		2	1105.01	4	2560.05	-
Even Better Muenster Cheese		-	-	-	-	-
Even Better Sharp Cheddar Cheese		-	-	1	640.01	-

Measure grouping

As you may have noticed, *Quantity* and *Sales* are not presented as separate measures in the top column row. Next to the dimension *Customer*, you find an item called *Values*. When you use more than one measure, they are automatically grouped together forming a measure group, *Values*. This group can be added to the rows section or the columns section. The measure group cannot be edited or selected in the table. You cannot split up the measure item and use one measure as a row and another as a column.

Pivoting your data in the properties panel

In the properties panel, you can add measures and dimensions to the pivot table, and also pivot rows or columns.

Data

In the data pane, you can add dimensions and measures. You can move items between rows and columns. You can also change item order inside rows or columns. When you use more than one measure, they are grouped and a *Values* item is created.

 **Chart suggestions** 

Data

Dimensions

Row

Item Desc > 

Product Group > 

Customer > 

Add

Column

Values 

Add

Measures

Values

Quantity > 

Sales > 

Add

Sorting

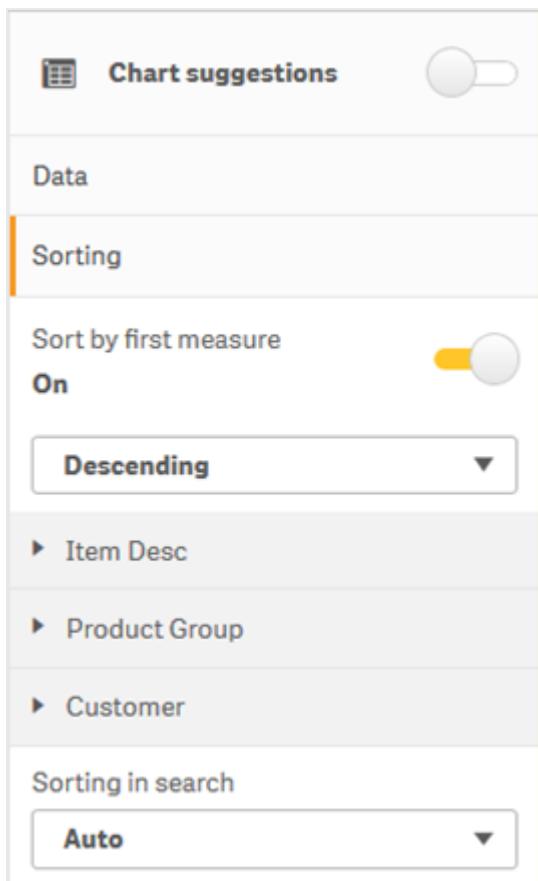
Add-ons

Appearance

Hide properties 

Sorting

On the sorting pane, you can change the internal order of dimensions and measures.

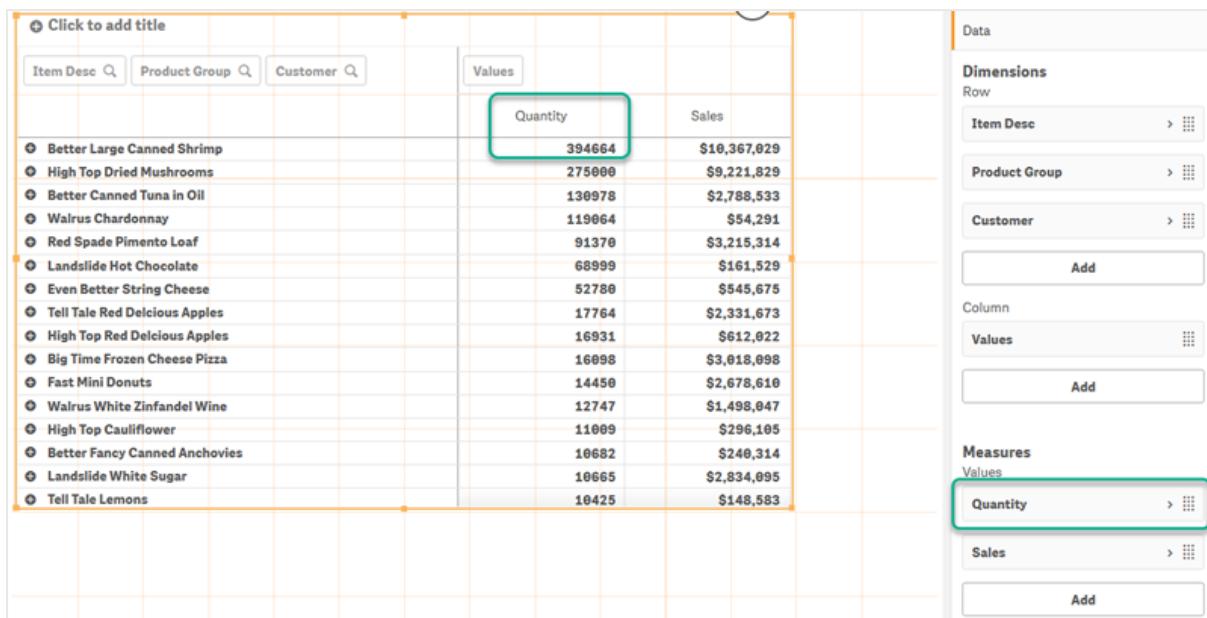


Sort by first measure

If you have more than one measure, a measure group is created. **Sort by first measure** will sort dimension values by the numeric value of the first measure. This sort order will affect all dimensions, and take precedence before any other sort order defined under the dimensions.

For example, you have two measures: *Quantity* and *Sales*. In the properties pane, under **Data > Measures**, *Quantity* is listed first. If you **Sort by first measure** in descending order, your table is sorted starting with the dimension with the highest *Quantity*.

Pivot table in Edit mode. Sort by first measure has been toggled on.



Limitations:

- This option is only supported if subtotals are calculated. You can calculate subtotals by doing one of the following:
 - In the properties pane, go to **Data**, and click on a dimension. Toggle on **Show totals**.
 - In the properties pane, go to **Appearance > Presentation**. Toggle on **Indent rows**.
- This option is not supported for calculated dimensions.
- This option is only supported if all dimensions are in the Row section and all measures are in the Column section.

Sort by expression

When sorting by expression, the expression is only applied to the first dimension of a pivot table. Subsequent dimensions are sorted in the same order as the first dimension.

Global grouping

Global grouping lets you create a limited data set, and within that data set, single out values that you want to focus on. For example: the best quarters, the top sales people, or the worst selling products.

Example:

In the following pivot table, no limitation is applied. The values are sorted on *Sales*, descending. The list is long and the values for 2013 are not shown.

Year ▾	
Sales Rep Name ▾	sum(Sales)
2014	\$41,006,958.72
Judy Thurman	\$6,037,992.86
Stewart Wind	\$4,717,671.77
Lee Chin	\$3,535,768.74
Cheryle Sincock	\$1,791,498.68
Brenda Gibson	\$1,750,292.96
John Greg	\$1,443,128.30
Martha Richard	\$1,388,402.75
Amalia Craig	\$1,200,853.57
David Laychak	\$1,170,791.14
Karl Anderson	\$957,467.35
Max Blagburn	\$940,446.81
David Howard	\$850,575.53
Angelen Carter	\$810,618.88
Amanda Honda	\$704,245.66
Amelia Fields	\$635,124.63
Donna Brown	\$603,055.39
Peggie Hurt	\$525,843.84
Craig Amundson	\$495,495.93
Micheal Williams	\$469,046.29
Grand Total	\$41,006,958.72

In the following pivot table, a limitation has been applied to the (inner) dimension *Sales Rep Name*, so that only the top five sales representatives for the years 2013 and 2014 are shown.

Year	Sales Rep Name	Sum(Sales)
2014		\$41,006,959.00
	Judy Thurman	\$6,037,993.00
	Stewart Wind	\$4,717,672.00
	Lee Chin	\$3,535,769.00
	Cheryle Sincock	\$1,791,499.00
	Brenda Gibson	\$1,750,293.00
2013		\$38,657,267.00
	Stewart Wind	\$5,669,097.00
	Judy Thurman	\$4,951,304.00
	Lee Chin	\$3,685,579.00
	John Greg	\$2,104,622.00
	Cheryle Sincock	\$1,353,069.00

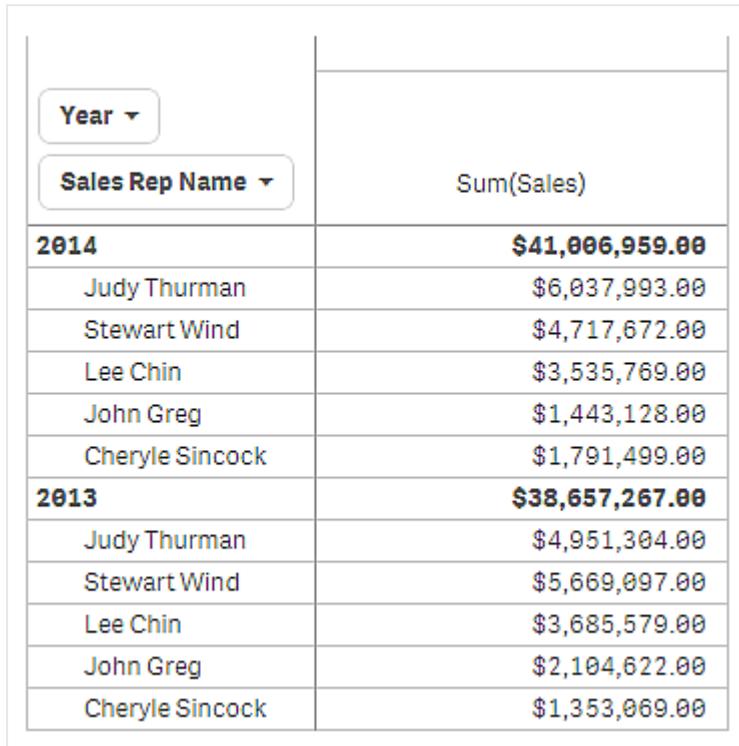
The next step is to select global grouping in the properties panel. The option **Global grouping** is only available after you have applied a limitation on the dimension.

When global grouping is selected, the limitation on the top five sales representatives is applied again, but this time the dimension *Year* is ignored. The five sales representatives with the highest sales (either in 2013 or 2014) are the only ones that will be presented in the final pivot table.

The following image shows the six highest results for 2014 and 2013. The top four results are from 2014, but the fifth (John Greg) is from 2013. Because five other sales representatives have higher sales than Brenda Gibson (who was number five in 2014), she is removed.

Judy Thurman	\$6,037,993.00
Stewart Wind	\$4,717,672.00
Lee Chin	\$3,535,769.00
John Greg	\$2,104,622.00
Cheryle Sincock	\$1,791,499.00
Brenda Gibson	\$1,750,293.00

The following image shows the pivot table with global grouping applied. The pivot table only contains the sales results for the top five sales representatives. Even though *Brenda Gibson* had a better result in 2014 than John Greg, his result for 2013 qualified him for the top five list.



The screenshot shows a pivot table with the following structure:

Year	Sales Rep Name	Sum(Sales)
2014		\$41,006,959.00
	Judy Thurman	\$6,037,993.00
	Stewart Wind	\$4,717,672.00
	Lee Chin	\$3,535,769.00
	John Greg	\$1,443,128.00
	Cheryle Sincock	\$1,791,499.00
2013		\$38,657,267.00
	Judy Thurman	\$4,951,304.00
	Stewart Wind	\$5,669,097.00
	Lee Chin	\$3,685,579.00
	John Greg	\$2,104,622.00
	Cheryle Sincock	\$1,353,069.00

Creating a bookmark with an expanded pivot table

By default, if you create a bookmark that contains a pivot table, the pivot table will be shown collapsed. If you expanded any rows using , they will not be shown. However, you can choose to show the pivot table as expanded.

Do the following:

1. Click  in the toolbar.
2. Click **Create new bookmark**.
Change the name and description, if desired.
3. Toggle on **Save layout**.
4. Click **Save**.

Comparing straight tables and pivot tables

You can see the efficiency of a pivot table if you compare it to a regular table that has the same data.

Straight table

In the following table, you have:

- Three dimensions: Customer, Product Group, and Item Desc
- Two measures: Quantity and Sales

The table shows food product sales. If you want to rearrange the data to simplify analysis, your options are limited. You can change the order of the columns, but that does not improve the overview. You can also set the sorting order, either in the sorting section in the properties panel, or by clicking the dimension columns. However, the problem persists. The customers, product groups, and items are all displayed more than once. It is not possible to get a good summary of the data.

Customer	Product Group	Item	Quantity	Sales
Totals			1,818,294	\$104,852,674.81
A-2-Z Solutions	Alcoholic Beverages	Good Light Wine	2	\$337.58
A-2-Z Solutions	Alcoholic Beverages	Pearl Chardonnay	8	\$513.89
A-2-Z Solutions	Alcoholic Beverages	Pearl Light Beer	1	\$60.10
A-2-Z Solutions	Alcoholic Beverages	Walrus Light Wine	7	\$34.69
A-2-Z Solutions	Baked Goods	Colony Pumpernickel Bread	2	\$9.54
A-2-Z Solutions	Baked Goods	Colony Wheat Bread	1	\$74.73
A-2-Z Solutions	Baked Goods	Great Blueberry Muffins	3	\$149.02
A-2-Z Solutions	Baking Goods	BBB Best Apple Butter	6	\$211.35
A-2-Z Solutions	Baking Goods	BBB Best Apple Preserves	2	\$276.20
A-2-Z Solutions	Baking Goods	BBB Best Extra Chunky Peanut Butter	1	\$617.40
A-2-Z Solutions	Baking Goods	BBB Best Grape Jam	1	\$33.75
A-2-Z Solutions	Baking Goods	BBB Best Pepper	4	\$328.97

Pivot table

We add a pivot table to the sheet and use the same info:

- Three dimensions: Customer, Product Group, and Item Desc
- Two measures: Quantity and Sales

	Customer	Product Group	Item Desc	Values	
				Quantity	Sales
A-2-Z Solutions				1418	\$196,298
A-ARVIN Laser Resources				25	\$4,053
A Superior System				868	\$103,728
A&B				891	\$92,121
A&G				133	\$12,503
A&R Partners				156	\$30,392
A1 Datacom Supply				5830	\$259,600
a2i				14	\$452
A2Z Solutions				454	\$69,977
AA-Wizard				917	\$94,209
Adast				881	\$351,243

Discovery

As you can see, the pivot table presents the data in a much more condensed way. Compared to the regular table, the number of rows has been halved and the number of columns is three instead of five.

One of the advantages of a pivot table is the interchangeability: the ability to move row items to columns and column items to rows. You can rearrange the data and have several different views of the same data set. You can move dimensions and measures to bring forward data of interest and hide data that is either too detailed, or irrelevant to the analysis.

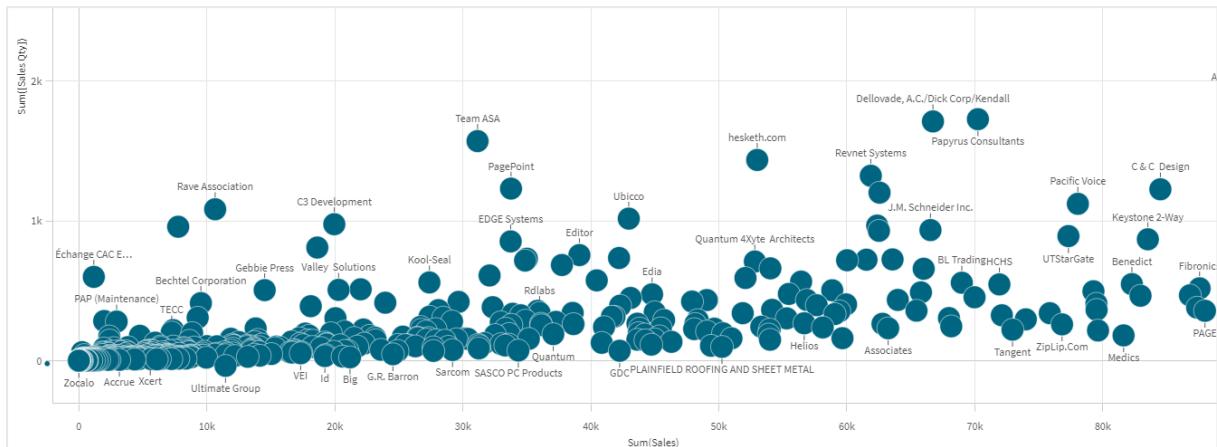
The pivot table shows the dimensions *Customer*, *Product Group*, and *Item Desc*, and the measures *Quantity* and *Sales*. In this view, you have a summary of quantity and sales for each customer. If you want to know which items and product groups that the customers bought, you can expand the customer fields by clicking . A icon indicates that a field can be further expanded and present more details, while a icon indicates that the field can be collapsed, to reduce the number of fields and details.

Scatter plot

The scatter plot presents pairs of values from two or three measures.

This is useful when you want to show data where each instance has two numbers, for example, the relationship between Sales and Quantity per Customer. In the scatter plot below, a third measure (Cost) is used to generate the bubble size.

Scatter plot displaying the relationship between Sales and Quantity per Customer.



The scatter plot presents values from different measures over one dimension as a collection of points. In most charts, you find your dimension on one of the axes, but for a scatter plot, the dimension is represented by the points in the chart, and the measures are found on each of the two axes. When a third, optional, measure is used, its value is reflected in the bubble size. If you are analyzing large data sets and view compressed data, the density of the data points is reflected by color.



When to use it

The scatter plot helps you find potential relationships between values, and to find outliers in data sets. The scatter plot is useful when you want to show data where each instance has at least two metrics, for example, average life expectancy and average gross domestic product per capita in different countries.

Advantages

The scatter plot is a great way to visualize the correlation of two or more measures at the same time. The third measure is an efficient way of differentiating between values and simplifying the identification of, for example, large countries, large customers, large quantities, and so on.

Disadvantages

The scatter plot may be difficult to understand for an inexperienced user, because it has measure value on both axes, and the third, optional, measure adds complexity to the interpretation. Make sure a novice can interpret the scatter plot correctly. Using descriptive labels is a good way to make the visualization easier to interpret.

Values may be placed on top of each other and are then not visible until you zoom in.

Creating a scatter plot

You can create a scatter plot on the sheet you are editing.

In a scatter plot you need one dimension and at least two measures. You can have maximum one dimension and three measures, where the third measure is visualized as bubble size.

Do the following:

1. From the assets panel, drag an empty scatter plot to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.
4. Click **Add measure** and select a measure or create a measure from a field.
5. Optionally, if you want bubble size to be set according to a third measure:
Click **Add measure** and select a measure or create a measure from a field.

When you have created the scatter plot, you may want to adjust its appearance and other settings in the properties panel.

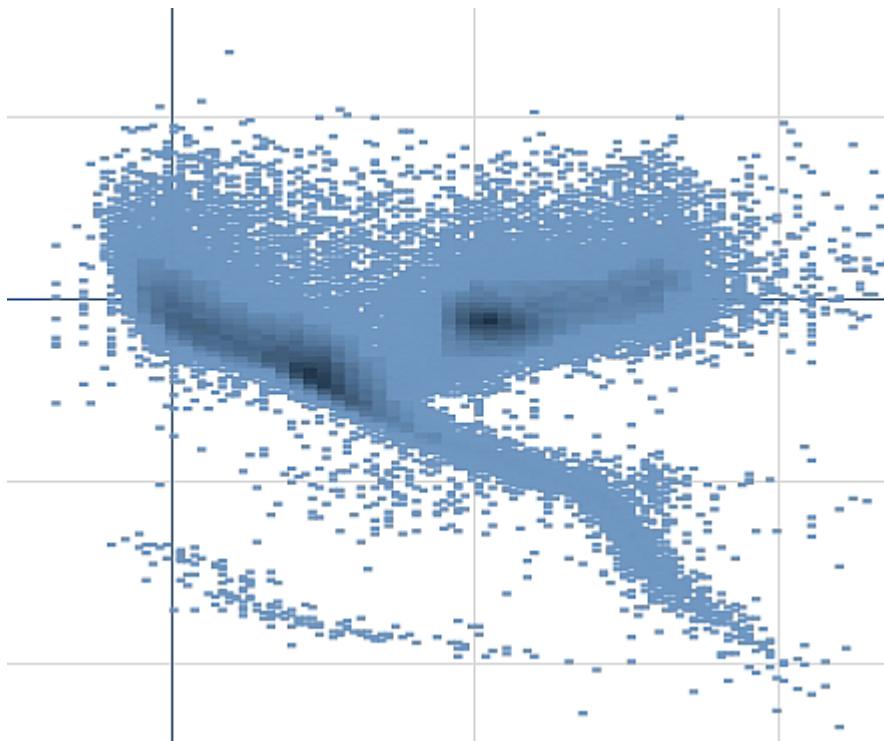
Display limitations

Large data sets in scatter plots

You can set the maximum number of visible bubbles in a scatter plot if you are using advanced edit mode. Go to **Presentation > Max visible bubbles**. The default is 2,500. The maximum is 50,000. If you set a number less than 1,000, the scatter plot will behave as if the maximum is 1,000 visible bubbles.

If the number of displayed data points is less than **Max visible bubbles**, the data will be shown as individual bubbles. If there are more data points than the number set in **Max visible bubbles**, you will see an overview of your dataset as a table with colored boxes. This switch between compressed view and bubble view is done automatically. If there are more than 5,000 visible bubbles, then bubble labels and out of bound bubbles will not be shown.

Scatter plot with compressed data in a bubble view.



Zooming and panning

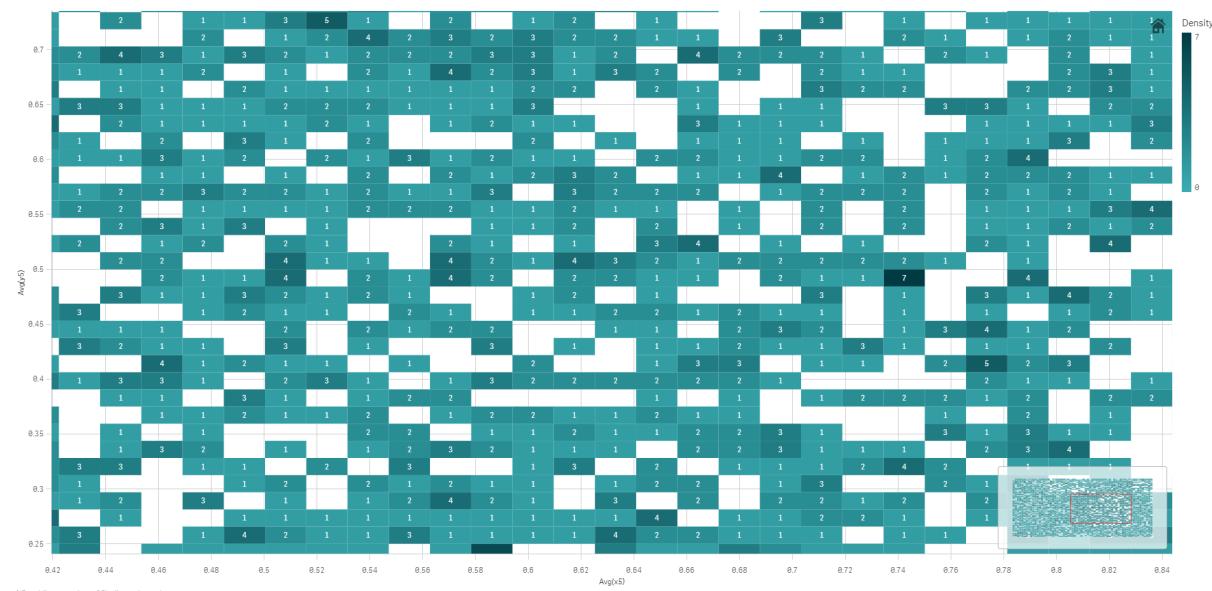
In a scatter plot, you can zoom and pan around in your data. The interaction differs depending on what device you are using. If you are zooming in you can see where in the data set you are located by looking at the mini chart in the bottom right corner. If you zoom in on large data sets you will be able to see the data shown as boxes with values inside. The values represent the number of points in each box.

You cannot make selections when the scatter plot is rendering during a pan or zoom.

Zooming and panning is not possible when you have made a selection in the compressed data view.

You can change the compression resolution in the visual exploration menu or in the properties panel.

Scatter plot with compressed data in a compressed view.



* Providing overview of 5k dimension values.

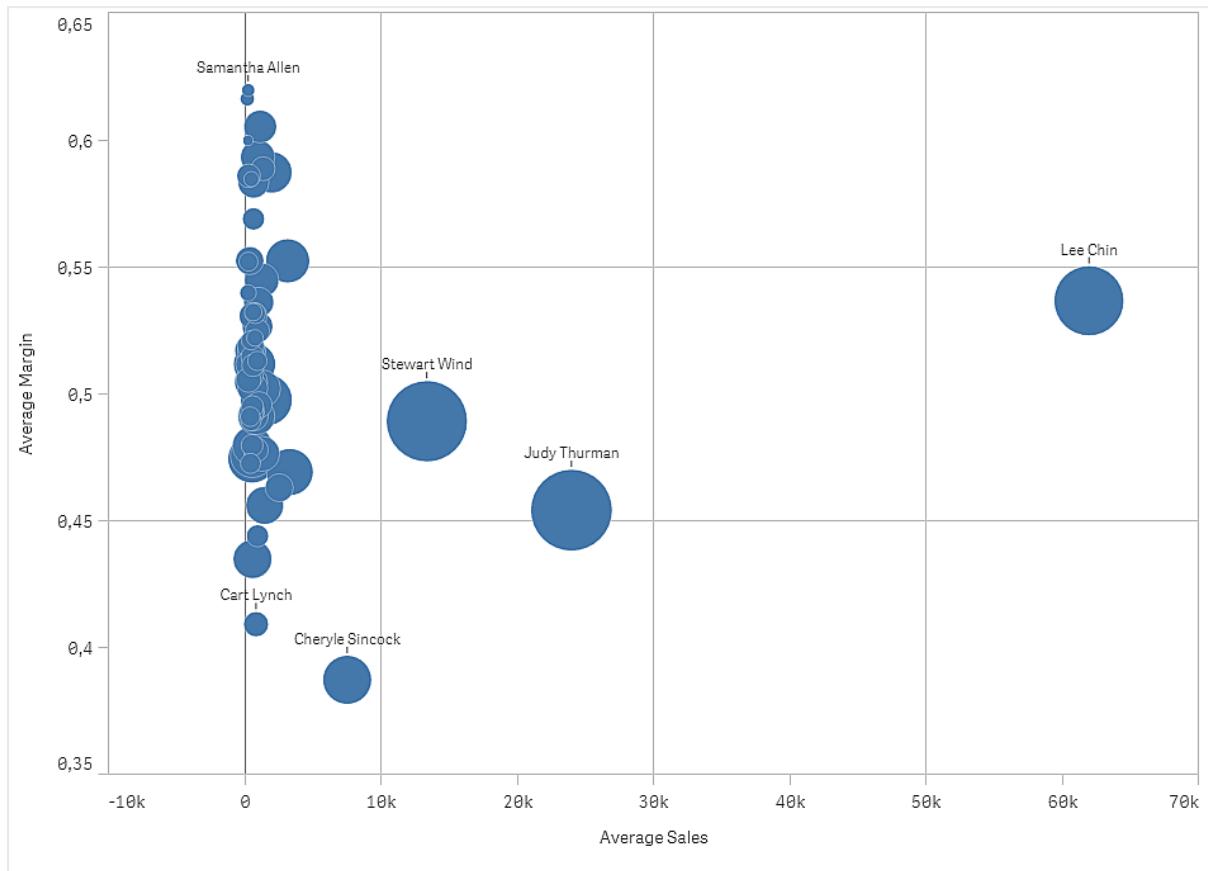
Coloring

Compressed scatter plots are always auto-colored by the primary data color. That means that any custom color definition made will not affect compressed data. The density of the data points is reflected by color. More dense data will be displayed in a darker color.

However, if you zoom or make selections so that the number of displayed data points is reduced to less than **Max visible bubbles**, the data will be colored as defined.

Correlating two measures with a scatter plot to find outliers

This example shows how to make a scatter plot to correlate two measures. We want to compare how the average sales volume correlates with the average margin for a group of sales persons, and find the outliers.



Dataset

In this example, we'll use two data files available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial, and the files are available in the *Tutorials* source folder:

- *Sales.xls*
- *Sales rep.csv*

To download the files, go to [Tutorial - Building an App](#).

Add the two data files to an empty app, and make sure that they are associated by *Sales Rep ID - Sales Rep Number*.

The dataset that is loaded contains sales data. The *Sales rep* table holds the information about the sales persons.

Measures

We need two measures that we create in Master items:

- *AverageSales* with the expression `Avg(sales)`. This is the average of the sales value for all orders.
- *AverageMargin* with the expression `Avg(Margin/sales)`. This is the average of the sales margin for all orders.

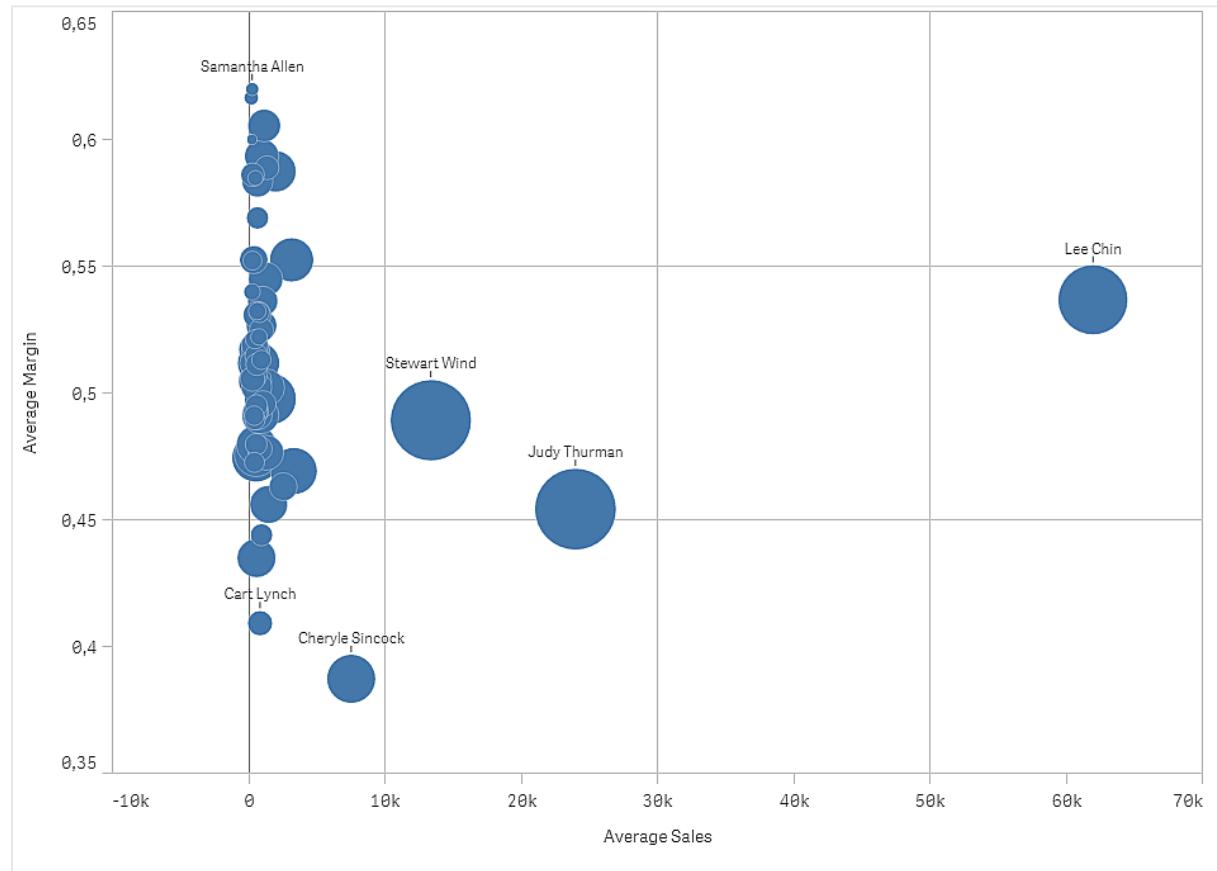
Visualization

We add a scatter plot to the sheet and set the following data properties:

- **Dimensions > Bubble:** Sales Rep Name (sales person)
- **Measures > X-axis:** AverageSales
- **Measures > Y-axis:** AverageMargin

A scatter plot is created, with a bubble for each sales person.

But we also want to have information about the total sales for each sales person, by adding the third measure Avg(sales). The size of each bubble reflects the total sales for each sales person.



Discovery

The scatter plot visualizes the average sales versus average margin for each sales person, and we can see which sales persons distinguish themselves in performance. You can hover the mouse pointer over a sales person and view the details.

In the chart we can see that Lee Chin has the highest average sale value. Stewart Wind has the largest total sales volume, followed by Judy Thurman. Cheryle Sincock has a significantly smaller average margin than other sales persons, while outperforming most of them in average sales volume.

Table

The table shows several fields simultaneously, where the content of each row is logically connected. Typically, a table consists of one dimension and several measures.

Customer		Sales	Quantity	Margin (%)	# of Invoices	Average Sales per Invoice
Totals		\$104,852,674.81	1,816,372	4127.8%	38,314	\$2,736.67
A-2-Z Solutions		\$196,298.49	1,418	3841.7%	58	\$3,384.46
A-ARVIN Laser Resources		\$4,053.05	25	3792.6%	13	\$311.77
A Superior System		\$103,728.12	868	4074.5%	167	\$621.13
A&B		\$92,120.60	891	4202.9%	18	\$5,117.81
A&G		\$12,502.61	133	4708.0%	12	\$1,041.88
A&R Partners		\$30,392.45	156	3409.9%	6	\$5,065.41
A1 Datacom Supply		\$259,599.52	5,830	4025.7%	111	\$2,338.73
a2i		\$451.64	14	5983.7%	9	\$50.18
A2Z Solutions		\$69,977.36	454	4121.1%	94	\$744.44
AA-Wizard		\$94,209.44	917	4660.6%	41	\$2,297.79

You only make selections in the dimension columns. All dimension columns have a search icon in the header.

When to use it

Use a table, when you want to view detailed data and precise values rather than visualizations of values. Tables are good when you want to compare individual values. Drill-down group dimensions are very efficient in tables. Within a limited space, you can drill down to the next level of detail and analyze the updated measure values.

Advantages

You can filter and sort the table in different ways. Many values can be included in a table, and when you drill down in a table, you make good use of the limited space on a sheet. A table is excellent when you want to see exact values rather than trends or patterns.

Disadvantages

If the table contains many values, it is difficult to get an overview of how values are related. It is also hard to identify an irregularity within the table.

Creating a table

You can create a new table on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty table to the sheet.
2. Click **Add dimension** and select a dimension or a field.
3. Click **Add measure** and select a measure or create a measure from a field.

When you have created the table, you may want to adjust its appearance and other settings in the properties panel.



Column width is automatically set to keep columns together for improved readability. You can adjust the width of a column by dragging the header divider. Double-click the header divider to reset to the default width.

Aligning data

If **Text alignment** is set to **Auto**, column data is aligned according to data type: text values are left-aligned and number values, including date related values, are right-aligned. If you set it to **Custom**, you can align the data to the left, center, or right.

Sorting the table

You can adjust the sorting of the table in several ways:

- Column sorting: adjust the order of the dimensions and measures from left to right
- Row sorting: adjust the sorting priority order of the rows
- Internal sorting: use the internal sorting order of dimensions and measures
- Interactive sorting: during analysis you can click on a column header to sort the table

Column sorting

By default, the order in which columns are sorted is set by the order in which dimensions and measures are added to the table. If you add the measure *Sales* first, it is presented first (leftmost) in the table. The next dimension or measure that is added is presented in the second column, and so on. The column sorting order can be changed in the properties panel, under **Columns**.

Row sorting

By default, rows are sorted by the first added dimension or measure, numeric values descending, text values ascending. A small arrow under the column header shows by which column the table is sorted.

You can change the row sorting in the properties panel, under **Sorting**. Drag the dimensions and measures to change the sorting priority order. In many cases, sorting is not only affected by the first dimension or measure in **Sorting**, but also the following ones.

Example:

In the following screenshot, the rows are first sorted by *Customer*, then by *Month*, and then by *Product Type*. As you can see, the columns *Customer* and *Month* have several rows with the same values (*A-2-Z Solutions* and *Month*). The rows in *Product Type* are ordered alphabetically, but only those that were sold in January to the customer *A-2-Z Solutions* are displayed.

Customer	Month	Product Type	Sales
Totals		\$104,852,674.81	
A-2-Z Solutions	Jan	Baking Goods	\$248.83
A-2-Z Solutions	Jan	Beer and Wine	\$129.25
A-2-Z Solutions	Jan	Breakfast Foods	\$68.29
A-2-Z Solutions	Jan	Canned Soup	\$45.24
A-2-Z Solutions	Jan	Carbonated Beverages	\$187.42
A-2-Z Solutions	Jan	Dairy	\$8,262.54
A-2-Z Solutions	Jan	Specialty	\$686.59
A-2-Z Solutions	Feb	Beer and Wine	\$24.60
A-2-Z Solutions	Feb	Breakfast Foods	\$270.72
A-2-Z Solutions	Feb	Canned Soup	\$91.80

By changing the sorting order, so that secondary sorting is by *Product Type*, followed by *Month*, all *Product Type* items sold to the customer *A-2-Z Solutions* are presented in alphabetical order, whereas only the months when they were sold are displayed under *Month*.

Customer	Product Type	Month	Sales
Totals			\$104,852,674.81
A-2-Z Solutions	Baking Goods	Jan	\$248.83
A-2-Z Solutions	Baking Goods	Jul	\$1,318.04
A-2-Z Solutions	Baking Goods	Nov	\$396.00
A-2-Z Solutions	Beer and Wine	Jan	\$129.25
A-2-Z Solutions	Beer and Wine	Feb	\$24.60
A-2-Z Solutions	Beer and Wine	Apr	\$129.25
A-2-Z Solutions	Beer and Wine	Jun	\$60.10
A-2-Z Solutions	Beer and Wine	Jul	\$129.25
A-2-Z Solutions	Beer and Wine	Oct	\$400.65
A-2-Z Solutions	Beer and Wine	Nov	\$10.09
A-2-Z Solutions	Beer and Wine	Dec	\$63.07
A-2-Z Solutions	Bread	Jul	\$158.56
A-2-Z Solutions	Bread	Oct	\$74.73

Internal sorting

Each dimension and measure has a default (**Auto**) internal sorting order, which can be changed. Under **Sorting**, click the item you want to change and click the button to switch to **Custom** sorting. Changes made to the internal sorting of an item may not have any effect if the sorting is in conflict with an item with higher priority.

Interactive sorting

During analysis, you can set which column to sort on by clicking the column header. The first click sorts the table according to the default sorting of the selected item. A second click reverses the sorting order.

Interactive sorting is session based and is not saved. If you want your changes to the sorting to be persistent, you need to make the changes in the properties panel.

Displaying totals

By default, the totals of numeric values are displayed under the column names. In the properties panel, you can change this to display the totals at the bottom of a column, or not at all.

Displaying more data

You can freeze the first column from scrolling, and select to wrap multiline text in headers and cells separately. These settings are changed in the property panel under **Appearance > Presentation**. You can also disable horizontal scrolling, and enable the column picker feature. This lets app consumers change table

column order.

Adding a trend indicator to a measure

You can add a trend indicator to a measure column. This will show a symbol next to the measure value. You can define the ranges that determine which symbol is displayed and in which color it is displayed. You enable the indicator by setting **Representation** to **Indicator** in the measure properties.

Setting the indicator limits

You need to add the limits for the ranges you want to use for showing indicators with **Add limit**. You can set a limit value in three ways.

- Use the slider.
- Type a value in the text box.
- Set an expression that returns the limit value.

When you have added the limits, you can select the color and the symbol of the indicator for each defined range.

Styling the indicator

You can style the way the indicator is displayed.

- You can show both the indicator and the measure value by selecting **Show values**.
- You can set the value color to the same as the indicator color with **Apply color to value**.
- You can display the indicator to the right or to the left of the value with **Indicator position**.

Example

In this example, we added a trend indicator to the Sales measure to indicate which values are below the target value. The indicator limits are:

- For values below 3000000 a red flag is displayed.
- For values in the range of 3000000 to 3500000 a yellow flag is displayed.
- For values above 3500000 a green check-mark is displayed.

Year	Month	Sales
2012	jan.	1773749,81
2012	feb.	3867568,01
2012	mars	3892194,86
2012	apr.	3660633,9
2012	maj	3191647,98
2012	juni	4259259,66
2012	juli	2519872,65
2012	aug.	3799274,06
2012	sep.	3739097,87
2012	okt.	3036455,81
2012	nov.	3528099,04
2012	dec.	2905448,63

Adding a mini chart to a measure

You can add a mini chart to a measure column. This will show a small chart visualization instead of the measure value. You can define the dimension that determines what data is displayed and in which color it is displayed. You enable the indicator by setting **Representation** to **Mini chart** in the measure properties.



The mini chart popup only shows the value of the measure, and is not broken down by individual dimension point values.

Adding mini chart
to a measure



Setting the mini chart type

After specifying which dimension the mini chart will be based on, you must select a **Mode**.

- **Bars** creates a bar chart.
- **Dots** creates a dot chart.
- **Sparkline** creates a sparkline chart. You can show dots at each data point along the sparkline chart by selecting **Show dots**.
- **Positive/negative** creates a chart with each value represented by a dot above or below the zero.

When you have selected the mode, you can specify the **Y-axis** of the Mini chart at the bottom of the mini chart options.

Styling the mini chart

You can set the color of bars or lines for **Bars**, **Dots**, and **Sparkline**.

- You can optionally set the **Max value color** and **Min value color**, which will highlight the highest and lowest visible chart values.
- You can optionally set the **Highlight first** and **Highlight last** colors, which will highlight the first and last visible chart values.

You can set the positive and negative color for a **Positive/negative** mini chart.

Display limitations

Number of rows and columns

In a table, you can have millions of rows and virtually any number of columns with dimensions and measures. But because huge tables are impractical and hard to manage, the limit for what is practical is far less than the theoretical maximum. In most cases, it is desirable to see all the columns without scrolling horizontally.

Tables with content of mixed sizes

In a table you can have both columns where the content fits on one row within the cell, and columns containing wrapped multiline text. In some cases you will see a shift in alignment and number of rows when the multiline column is scrolled in and out of view. When the view only contains content that fits on one row, the table will adjust and show all content on single line rows, which means more rows are displayed.

We recommend that you disable multiline text wrapping in these cases to avoid confusion for the user.

Searching in tables

In a table, you can search the dimension columns, and make selections in the resulting list.

Do the following:

1. Click  in the dimension column that you want to search in.
A selection popup is displayed with a list of all values of the field. This includes values that are excluded by selections. Excluded values are dark gray.
2. Type your search string.

While you type, the list is filtered to only display matching items.

3. Make a selection by clicking or drawing.
4. Confirm your selection.



You can confirm the selection of all matching items by pressing Enter.

The new selection is active and reflected in all associated visualizations.



You can remove the search string by clicking or pressing Esc. The search string is always removed when you press Enter.

Selections in tables

You can make selections in a table by clicking or drawing in the dimension columns.

Measure values cannot be selected. When you make a selection, it is always the dimension values that you select. You can only make selections in one column at a time.

Table with three selected fields in the dimension Customer.

Customer	Sales	Margin (%)	# of Invoices	Average Sales per Invoice
A-2-Z Solutions	\$158.56	3929.7%	1	\$1,277.65
A-ARVIN Laser Resources	3643.0%	1	\$248.83	
A Superior System	1730.2%	1	\$1,318.04	
A&B	5072.5%	2	\$198.00	
A&G	8056.5%	1	\$129.25	
A&R Partners	3650.4%	1	\$24.60	
A1 Datacom Supply	8056.5%	1	\$129.25	
a2i	2360.4%	2	\$200.32	
A-2-Z Solutions	3805.7%	1	\$10.09	
	2746.2%	1	\$63.07	
				\$158.56

To deselect a row, click it. To confirm a selection, click or click outside the visualization. You can also press Enter. To cancel, click or press Esc. If you confirm, the selection is reflected in all visualizations associated with the table.

You cannot select dimension values that are null. Null values in a table are presented as dashes (-). Rows without valid dimension values will not be included in the selection.

Accumulating values over a dimension in a table

This example shows how to use a table to compare data which accumulate over a dimension.

Year	Month	Sales	Accumulation of sales
Totals		64891921.17	-
2012	Jan	1773749.81	1773749.81
2012	Feb	3867568.01	5641317.82
2012	Mar	3892194.86	9533512.68
2012	Apr	3660633.9	13194146.58
2012	May	3191647.98	16385794.56
2012	Jun	4259259.66	20645054.22
2012	Jul	0	20645054.22
2012	Aug	0	20645054.22
2012	Sep	0	20645054.22
2012	Oct	0	20645054.22
2012	Nov	0	20645054.22
2012	Dec	0	20645054.22
2013	Jan	4574043.41	4574043.41
2013	Feb	3333839.69	7907883.1
2013	Mar	4266053.47	12173936.57
2013	Apr	2498575.88	14672512.45
2013	May	3533538.09	18206050.54
2013	Jun	4115434.48	22321485.02
2013	Jul	0	22321485.02
2013	Aug	0	22321485.02
2013	Sep	0	22321485.02
2013	Oct	0	22321485.02
2013	Nov	0	22321485.02
2013	Dec	0	22321485.02
2014	Jan	4114861.14	4114861.14
2014	Feb	3198717.63	7313578.77
2014	Mar	3789271.2	11102849.97
2014	Apr	3575328.84	14678178.81
2014	May	3541237.39	18219416.2
2014	Jun	3705965.73	21925381.93

Dataset

In this example, we will use a data file available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial. The file is available in the *Tutorials* source folder: *Sales.xls*

To download the file, go to [Tutorial - Building an App](#).

Add the data file to an empty app. The dataset that is loaded contains sales data.

Measure

We use the sales volume as the measure that we create in Master items:

- *Sales* with the expression `sum(sales)`. This is the sum of the sales volume.

Visualization

We add a table to the sheet and set the following data properties:

- **Dimension:** Year (`Date.Year`).
- **Dimension:** Month (`Date.Month`).

- **Measure:** *Sales*; the measure that was previously created.

The following table is created, with columns showing the year, the month, and the sum of sales for each month.

Year	Month	Sales
Totals		104852674.81
2012	Jan	1773749.81
2012	Feb	3867568.01
2012	Mar	3892194.86
2012	Apr	3660633.9
2012	May	3191647.98
2012	Jun	4259259.66
2012	Jul	2519872.65
2012	Aug	3799274.06
2012	Sep	3738097.87
2012	Oct	3038455.81
2012	Nov	3528099.04
2012	Dec	2905448.63
2013	Jan	4574043.41
2013	Feb	3333839.69
2013	Mar	4266053.47
2013	Apr	2498575.88
2013	May	3533538.09
2013	Jun	4115434.48
2013	Jul	2696221.99
2013	Aug	3792981.81
2013	Sep	4887106.08
2013	Oct	2917027.48
2013	Nov	3647345.62
2013	Dec	3291822.6
2014	Jan	4114861.14
2014	Feb	3198717.63
2014	Mar	3789271.2
2014	Apr	3575328.84
2014	May	3541237.39
2014	Jun	3705965.73

Make sure to set the **Sorting** with *Year > Month > Sales*.

Accumulation

To have our sales data accumulate over one dimension, we need to set an additional data property:

- **Measure:** *Sales*; the measure that was previously created.

We add this measure twice to reuse it for accumulation. To do this we need to apply a modifier to our *Sales* measure.

Do the following:

1. Under **Measure: Sales** set the **Modifier** to Accumulation. This will set the measure to accumulate over one dimension.
2. Set the **Modifier>Dimension** to *Month*. This sets the dimension over which the accumulation will take place.
3. Make sure **Across all dimensions** is turned off. We want the accumulation to restart at the beginning of each year.



In Accumulation there is the option to select **Show excluded values**. When turned on, your visualization includes any dimensional values that do not have data. This ensures that all values, including months with no sales data, are counted in the accumulation.

Our table becomes as follows, with the last column showing sales accumulating from one month to the next, for each year.

Year	Month	Sales	Accumulation of sales
Totals		184852674.81	-
2012	Jan	1773749.81	1773749.81
2012	Feb	3867568.01	5641317.82
2012	Mar	3892194.86	9533512.68
2012	Apr	3660633.9	13194146.58
2012	May	3191647.98	16385794.56
2012	Jun	4259259.66	20645054.22
2012	Jul	2519872.65	23164926.87
2012	Aug	3799274.06	26964200.93
2012	Sep	3739097.87	30703298.8
2012	Oct	3836455.81	33739754.61
2012	Nov	3528099.04	37267853.65
2012	Dec	2908448.63	40173302.28
2013	Jan	4574043.41	4574043.41
2013	Feb	3333839.69	7907883.1
2013	Mar	4266083.47	12173936.57
2013	Apr	2498575.88	14672512.45
2013	May	3533558.09	18206050.54
2013	Jun	4115434.48	22321485.02
2013	Jul	2696221.99	25017707.01
2013	Aug	3792981.81	28810688.82
2013	Sep	4087106.08	32897794.9
2013	Oct	2917027.48	35814822.38
2013	Nov	3647345.62	39462168
2013	Dec	3291822.6	42753990.6
2014	Jan	4114861.14	4114861.14
2014	Feb	3198717.63	7313578.77
2014	Mar	3789271.2	11102849.97
2014	Apr	3575328.84	14678178.81
2014	May	3541237.39	18219416.2
2014	Jun	3705965.73	21925381.93

It is good practice to have the title of your charts represent their content. So consider changing the titles of the columns of your table to reflect that the last column contains an accumulation of sales.

Our data for the year 2014 is insufficient as it stops at *June*. In order to achieve a better comparison between the accumulated sales of all three years we will choose a more appropriate range of months, from *January* to *June*.

Do the following:

1. Click **Done**.
2. Select all months from *January* to *June* and confirm your selection.

Our table becomes as follows, with the sales accumulating from *January* to *June*, inside each year.

Year	Month	Sales	Accumulation of sales
Totals		64891921.17	-
2012	Jan	1773749.81	1773749.81
2012	Feb	3867568.01	5641317.82
2012	Mar	3892194.86	9533512.68
2012	Apr	3660633.9	13194146.58
2012	May	3191647.98	16385794.56
2012	Jun	4259259.66	20645054.22
2012	Jul	0	20645054.22
2012	Aug	0	20645054.22
2012	Sep	0	20645054.22
2012	Oct	0	20645054.22
2012	Nov	0	20645054.22
2012	Dec	0	20645054.22
2013	Jan	4574043.41	4574043.41
2013	Feb	3333839.69	7907883.1
2013	Mar	4266053.47	12173936.57
2013	Apr	2498575.88	14672512.45
2013	May	3533538.89	18266050.54
2013	Jun	4115434.48	22321485.02
2013	Jul	0	22321485.02
2013	Aug	0	22321485.02
2013	Sep	0	22321485.02
2013	Oct	0	22321485.02
2013	Nov	0	22321485.02
2013	Dec	0	22321485.02
2014	Jan	4114861.14	4114861.14
2014	Feb	3198717.63	7313578.77
2014	Mar	3789271.2	11102849.97
2014	Apr	3575328.84	14678178.81
2014	May	3541237.39	18219416.2
2014	Jun	3705965.73	21925381.93

Discovery

The table shows the sales volume of each month, grouped into years. By having the sales accumulate inside each year, we get a better understanding of the volume of sales for each year. We have made a selection of the months from *January* to *June*, to compare the same range of months between the three years. In the last column of the table we can see that the accumulated sales for 2014 so far have been higher than the accumulated sales for 2012, but not as high as the ones for 2013.

Using relative numbers in a table to calculate contribution

This example shows how to use the relative numbers modifier to calculate contribution in a one dimensional table.

Product Group	Sales comparison table				
	Year	Sales per year	Contribution to total sales of current selection	Contribution to total sales from all years	Contribution to sales of each year
Totals	\$ 20,520,054	100%	20%	20%	
	2012	\$ 8,296,002	40%	8%	21%
	2013	\$ 7,602,738	37%	7%	18%
	2014	\$ 4,621,314	23%	4%	21%

Dataset

In this example, we will use a data file available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial. The file is available in the *Tutorials* source folder: *Sales.xls*

To download the file, go to [Tutorial - Building an App](#).

Add the data file to an empty app. The dataset that is loaded contains sales data.

Measure

We use the sales volume as the measure that we create in Master items:

- *Sales* with the expression `sum(sales)`. This is the sum of the sales volume.

Visualization

We start by adding a filter pane to the sheet and set the following data properties:

- **Dimension:** Product Group.

We also add a table to the sheet and set the following data properties:

- **Dimension:** Year (Date.Year).
- **Measure:** *Sales*; the measure that was previously created.

The following table is created, with columns showing the year, and the sum of sales for each year.

Sales comparison table	
Year	Sales per year
Totals	\$ 104,852,675
2012	\$ 40,173,302
2013	\$ 42,753,991
2014	\$ 21,925,382

Make sure to set the **Number formatting** to **Money**, and the **Format pattern** to `$ #,##0;- $ #,##0`.

It is good practice to have the title of your charts represent their content. Additionally, consider changing with the title of each column to reflect what it represents. The first column that we added is the *Year*, and the second column contains the *Sales per year*.

Relative numbers

We could use the filter pane to select specific product groups and see their yearly sales and their total sales. That would not provide a good comparison to their contribution. Instead we will use relative numbers to get percentages as an indication of the product group contribution. By changing the parameters upon which the relative numbers calculation is based, we will get different contributions and gain more insight.

Contribution of a year to the sales of product group

To see how much each year has contributed to the total sales of a specific product group when we make a selection, we need to add a third column:

Do the following:

- In the properties panel right-click on *Sales per year* and select **Duplicate**.

To use this duplicate measure as a relative number we will apply a modifier.

Do the following:

- Under **Measure**: *Sales per year* set the **Modifier** to Relative numbers. This will set the measure to act as a relative number.
- Set the **Modifier>Selection scope** to *Current selection*. This sets the modifier to be calculated relative to any selection made.
- The **Modifier>Dimensional scope** is set to *Disregard dimension* by default. This sets the modifier to be calculated relative to the total.
- Set the **Number formatting** to **Custom**, and the **Format pattern** to `#,##0%`.
- Change the **Label** to something meaningful such as *Contribution to total sales of current selection*.

Our table becomes as follows, with the last column showing the contribution of each year to the total sales of the selected product group.

		Sales comparison table	
	Year	Sales per year	Contribution to total sales of current selection
Totals		\$ 104,852,675	100%
	2012	\$ 40,173,302	38%
	2013	\$ 42,753,991	41%
	2014	\$ 21,925,382	21%

Contribution of a product group to total sales

To see how much each year's sales of a specific product group have contributed to the total sales of all product groups, we need to add a fourth column:

Do the following:

- In the properties panel right-click on *Contribution to total sales of current selection* and select **Duplicate**.

We will apply the Relative numbers modifier again, but with different properties this time.

Do the following:

1. Set the **Modifier>Selection scope** to *Disregard selection*. This sets the modifier to be calculated disregarding any selection made.
2. The **Modifier>Dimensional scope** is already set to *Disregard dimension*. This sets the modifier to be calculated relative to the total.
3. Change the **Label** to something meaningful such as *Contribution to total sales from all years*.

The **Number formatting** and the **Format pattern** are already set, since we duplicated the previous measure.

Our table becomes as follows, with the last column showing the contribution of the selected product group to the total sales from all three years.

Sales comparison table				
Year	Sales per year	Contribution to total sales of current selection	Contribution to total sales from all years	
Totals	\$ 104,852,675	100%	100%	
2012	\$ 40,173,302	38%	38%	
2013	\$ 42,753,991	41%	41%	
2014	\$ 21,925,382	21%	21%	

Since we have not made any selections yet, the last two columns have the same values.

Contribution of a product group to yearly sales

To see how much the sales of a specific product group have contributed to that year's sales compared to all the other product groups, we need to add a fifth column:

Do the following:

- In the properties panel right-click on *Contribution to total sales from all years* and select **Duplicate**.

We will apply the Relative numbers modifier again, but with different properties this time.

Do the following:

1. Set the **Modifier>Selection scope** to *Disregard selection*. This sets the modifier to be calculated disregarding any selection made.
2. The **Modifier>Dimensional scope** is already set to *Respect dimension*. This sets the modifier to be calculated relative to each dimensional value.
3. Change the **Label** to something meaningful such as *Contribution to sales of each year*.

Our table becomes as follows, with the last column showing the contribution of the selected product group to the total sales from all three years.

Product Group		Sales comparison table			
		Year	Sales per year	Contribution to total sales of current selection	Contribution to total sales from all years
Alcoholic Beverages		Totals	\$ 104,852,675	100%	100%
Baked Goods		2012	\$ 40,173,302	38%	38%
Baking Goods		2013	\$ 42,753,991	41%	41%
Beverages		2014	\$ 21,925,382	21%	21%
Breakfast Foods					
Canned Products					
Dairy					

Since we have not made any selections yet, the last column shows the yearly contribution of all product groups for each year.

Making a selection

We can now start making selections to change our relative numbers to something that provides more insight.

Do the following:

1. Click ✓ **Done editing** in the toolbar.
2. Select *Canned Products* from the product group filter pane.

Our table becomes as follows.

Product Group		Sales comparison table			
		Year	Sales per year	Contribution to total sales of current selection	Contribution to total sales from all years
Canned Products	✓	Totals	\$ 20,520,054	100%	20%
Alcoholic Beverages		2012	\$ 8,296,002	40%	8%
Baked Goods		2013	\$ 7,602,738	37%	7%
Baking Goods		2014	\$ 4,621,314	23%	4%
Beverages					
Breakfast Foods					
Dairy					

Discovery

The table shows the relative sales for each year. By having the different contribution columns, using the relative numbers as a measure modifier, we get a better understanding of the contribution of each product group to the total sales. From the *Contribution to total sales of current selection* column we can see that 40% of the *Canned Product* sales happened in 2012, with a dramatic drop in 2014. The *Contribution to total sales from all years* column shows that 8% of the total sales from all three years came from the 2012 sales of *Canned Product*. The *Contribution to sales of each year* column also tells us that for 2012 the *Canned Product* sales contributed 21% of the sales of that year, and the same applies to the sales of 2014.

Text & image

The text & image visualization complements other visualizations by offering options to add text, images, hyperlinks, and measures.

You can format and color the text and align the paragraphs. The background image has sizing and positioning options. You can also set the responsive behavior for text and images.



The text & image visualization is only available in the advanced edit mode.

When to use it

The text & image visualization is intended for presentation purposes, and does not support selections. However, the measures in the text & image visualization are updated when selections are made. Some typical uses:

- Use it on the first sheet of an app for essential information.
- Display a company image, or use a background image together with formatted text and measure values to present figures in a compelling way.
- Link to sites with additional information.
- Use the responsive behavior to ensure that the visualization renders well on all devices.

Advantages

The text & image visualization contrasts with the other visualizations. You have many options for making the text & image visualization stand out next to more regular charts.

Disadvantages

You are limited to a few measure values and rather short texts, otherwise the text & image visualization will be cluttered.

Creating a text & image

You can create a text & image visualization on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty text & image chart to the sheet.
2. Click the text & image chart to open the editing toolbar.
3. Add and format text, images, hyperlinks or measures to the text & image chart.



If you double-click a text & image chart in the assets panel, it is added to the sheet immediately.

Editing a text & image

In the text & image visualization you can add and format text, images, measures, and links in various ways.

When you are editing a sheet and the text & image visualization does not have focus, you need to click twice to open the editing toolbar. In the editing toolbar, you can format text properties such as color, size, and style, and also align the text. Additionally, you have options for adding links and images.

Creating a link

You can mark a text section and use it for a link.

If you do not add a prefix, <http://> is added automatically, assuming that you are adding a web address.

Do the following:

1. Select the text section that you want to use for the link.
2. Click  in the editing toolbar to open the link dialog.
3. Enter the web address that you want to link to.
4. Click .

Removing a link

You can remove a link from a text section.

Do the following:

1. Click the link so that the cursor is somewhere inside it.
2. Click  in the editing toolbar to open the link dialog.
3. Click .

The link is removed, but not the text.

Adding an image

You can add an image through the editing toolbar. You can use one of the default images, or an image of your own.

Do the following:

1. Click  in the editing toolbar.

The **Media library** opens.

The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

For Qlik Sense Desktop: You can place images in the following folder on your computer:

C:\Users\<user>\Documents\Qlik\Sense\Content\Default. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.

2. Click on a folder in the media library, for example **In app** or **Default**.
3. Select the image that you want to add.
4. Click **Insert**.



Alternatively, right-click the image file you want to add and select **Insert**.



In the properties panel, you can add a background image, which, for example, can be used when you want to insert text in the image. The images added through the editing toolbar are not background images.

Adding a measure

You can add a measure in the following ways:

- By dragging a field from the assets panel and adding it as a measure.
- By dragging a measure from **Master items**.
- By adding a measure (existing or new) from the properties panel.

When you are editing the measure, it is displayed as a token, which can be styled and moved around in the visualization. You can also apply number formatting to it. When you leave the editor, the measure value is displayed. Values that cannot be aggregated are shown as a hyphen (-).

Deleting a measure

You can delete a measure in the following ways:

- Place the cursor before the token and press Delete.
- Place the cursor after the token and press Backspace.
- In the properties panel, right-click the measure and select **Delete** in the dialog.
- In the properties panel, click the measure and click **Delete** .

Treemap

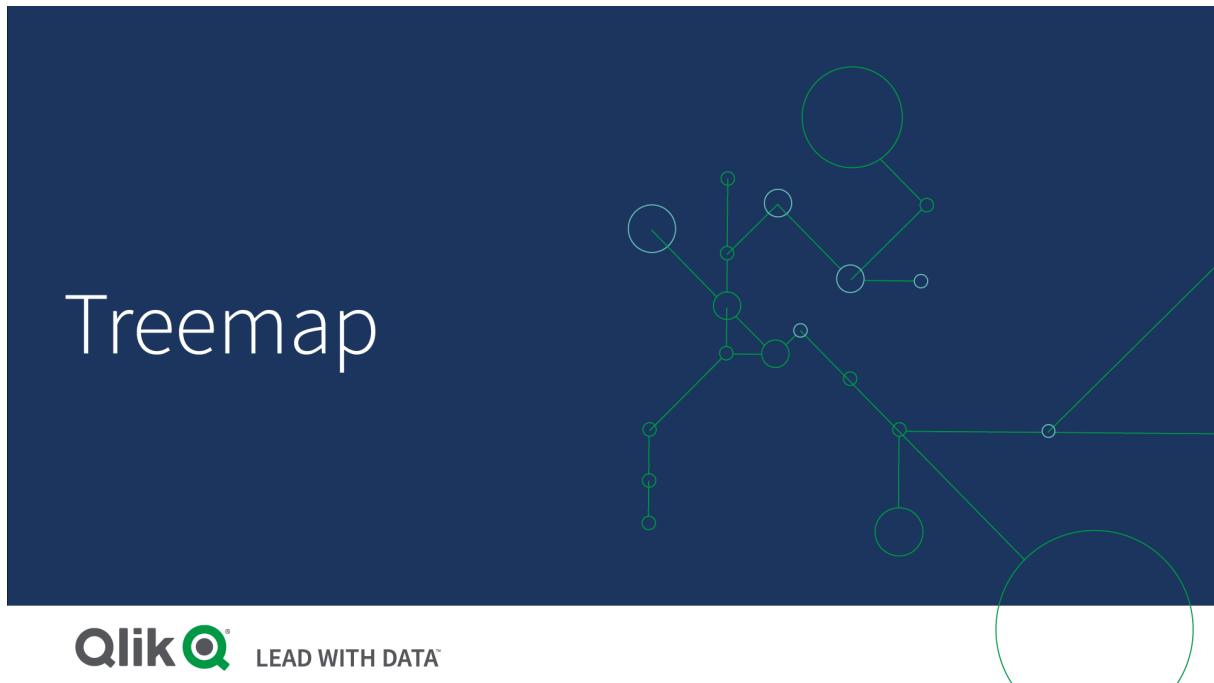
Treemaps display hierarchical data by using nested rectangles, that is, smaller rectangles within a larger rectangle.



In this image you have several product groups, such as Produce, Canned Products, and Frozen Foods. Each product group consists of a large rectangle. You can regard the product groups as branches of the tree. When you select a product group, you drill down to the next level, the product type, for example, Vegetables, Meat, and Dairy. You can regard the product types as sub-branches of the tree. The branches have leaves. A leaf node's rectangle has an area proportional to a specified dimension of the data. In this example, the items Ebony Squash, Bravo Large Canned Shrimp, Red Spade Pimento Loaf, and so on, are the leaves. The leaf nodes are colored to show a separate dimension of the data.

Sorting is automatic according to size. By default, the coloring is by dimension, with 12 colors, but that can be changed in the properties panel. When you have more than one dimension, you can decide which dimension to color by. In this example, the coloring is not by dimension, but by expression ($\text{Avg}(\text{Margin})$), a calculated measure, and by using this expression, you can see which items have the highest average margin. The darker the color, the higher the average margin.

If the data set contains negative values, a text message is shown stating that the negative values cannot be displayed.



When to use it

Use a treemap when space is constrained and you have a large amount of hierarchical data that you need to get an overview of. Treemaps should primarily be used with values that can be aggregated.

Advantages

Treemaps are economical in that they can be used within a limited space and yet display a large number of items simultaneously.

When there is a correlation between color and size in the tree structure, you are able to see patterns that would be difficult to spot in other ways, for example, when a certain color is particularly relevant.

Disadvantages

Treemaps are not good when there is a big difference in the magnitude of the measure values. Nor is a treemap the right choice when mixing absolute and relative values.

Negative values cannot be displayed in treemaps.

Creating a treemap

You can create a treemap on the sheet you are editing.

Do the following:

1. From the assets panel, drag an empty treemap to the sheet.
2. Click **Add dimension** and select a dimension or a field. This should be the highest level in the hierarchy. It will be displayed as the main group in the chart.
3. Click **Add measure** and select a measure or create a measure from a field. The measure will define the size of a rectangle.
4. Add more dimensions in the order of hierarchy level . The last dimension you add defines the rectangles. The other dimensions define the grouping of the rectangles.

In a treemap you need at least one dimension and one measure, but to make full use of the treemap it is preferable to have two or three dimensions. You can only have one measure, but up to 15 dimensions. We do not recommend using more than three dimensions as the treemap may become unmanageable.

When you have created the treemap, you may want to adjust its appearance and other settings in the properties panel.

Display limitations

When displaying large amounts of data in a treemap , there may be cases when not each dimension value within a rectangle is displayed with correct color and size. These remaining values will instead be displayed as a gray, striped area. The size and total value of the rectangle will still be correct, but not all dimension values in the rectangle will be explicit.

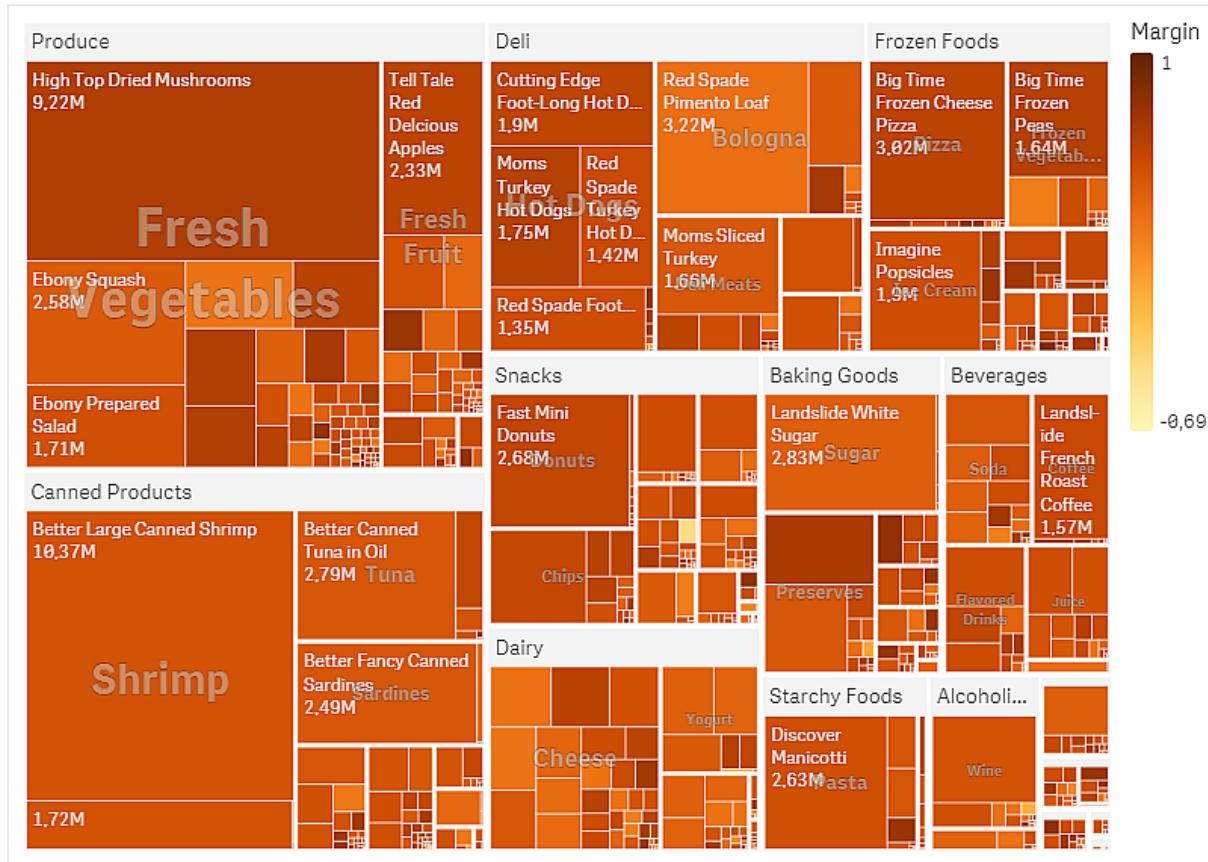
To remove the gray areas, you can either make a selection or use dimension limits in the properties panel.

Displaying hierarchical data with a treemap

This example shows how to make a treemap to view hierarchical data. You can display a large number of values in limited space and detect patterns that could be hard to spot with other charts.

We will look at sales numbers for different products which are categorized in four levels.

- Product group
- Product sub group
- Product item



Dataset

In this example, we'll use two data files available in the Qlik Sense Tutorial - Building an App. Download and expand the tutorial, and the files are available in the *Tutorials* source folder:

- *Sales.xlsx*
- *Item master.xlsx*

To download the files, go to [Tutorial - Building an App](#).

Add the two data files to an empty app, and make sure that they are associated by *Item Number*.

The dataset that is loaded contains sales data for food and beverage products. The *Item Master* table holds the information about the product categories.

Visualization

We add a tree map to the sheet and set the following dimensions and measures:

- **Dimensions > Group:** Product Group (product group)
- **Dimensions > Group:** Product Sub Group (product sub group)
- **Dimensions > Rectangle:** Item Desc (product item description)
- **Measures > Size:** Sum(sales) (sum of sales)

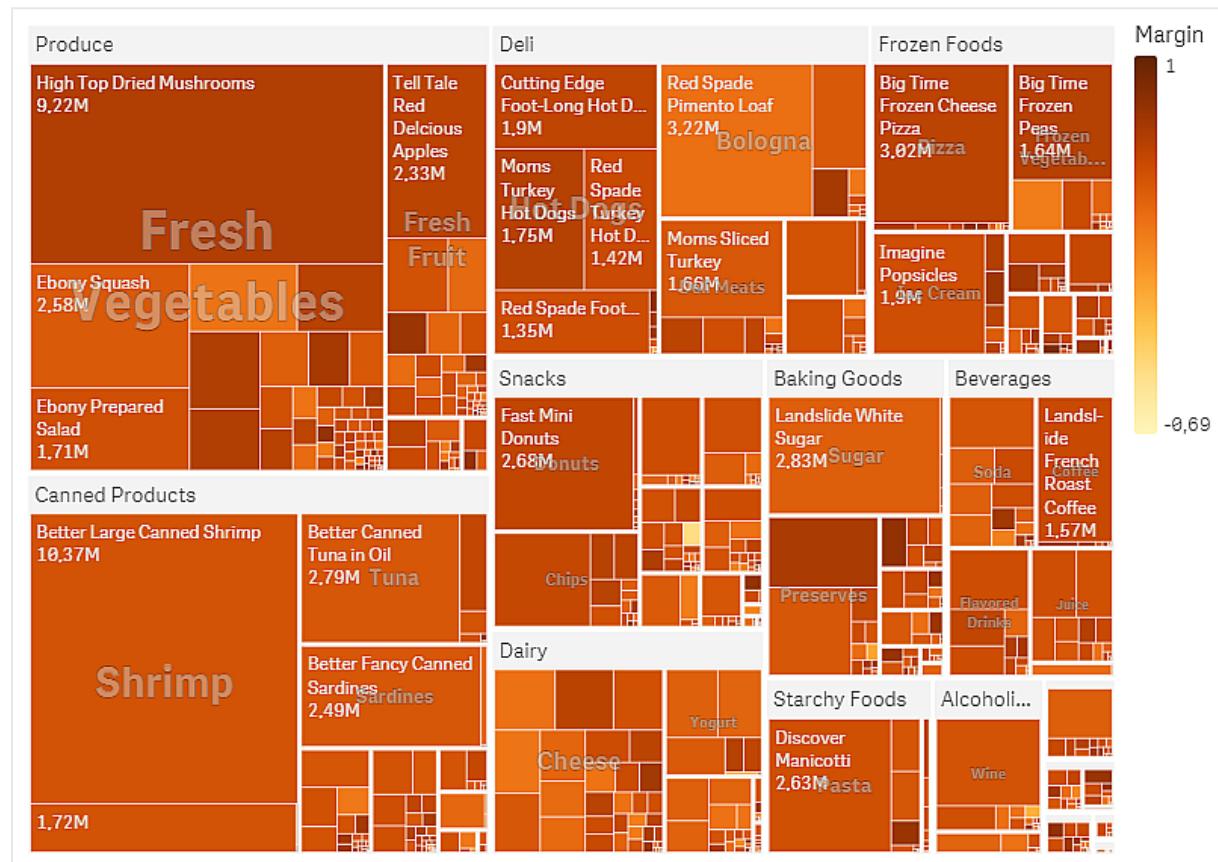
A treemap is created, with a rectangle for each product item. Rectangles are sized according to sales and grouped according to Product Group and Product Sub Group.

But we also want to add some more information to the chart. The rectangles are colored by the product group dimension, but we want color them by margin to also see which items are most profitable. You can do this in **Appearance > Colors and legend**.

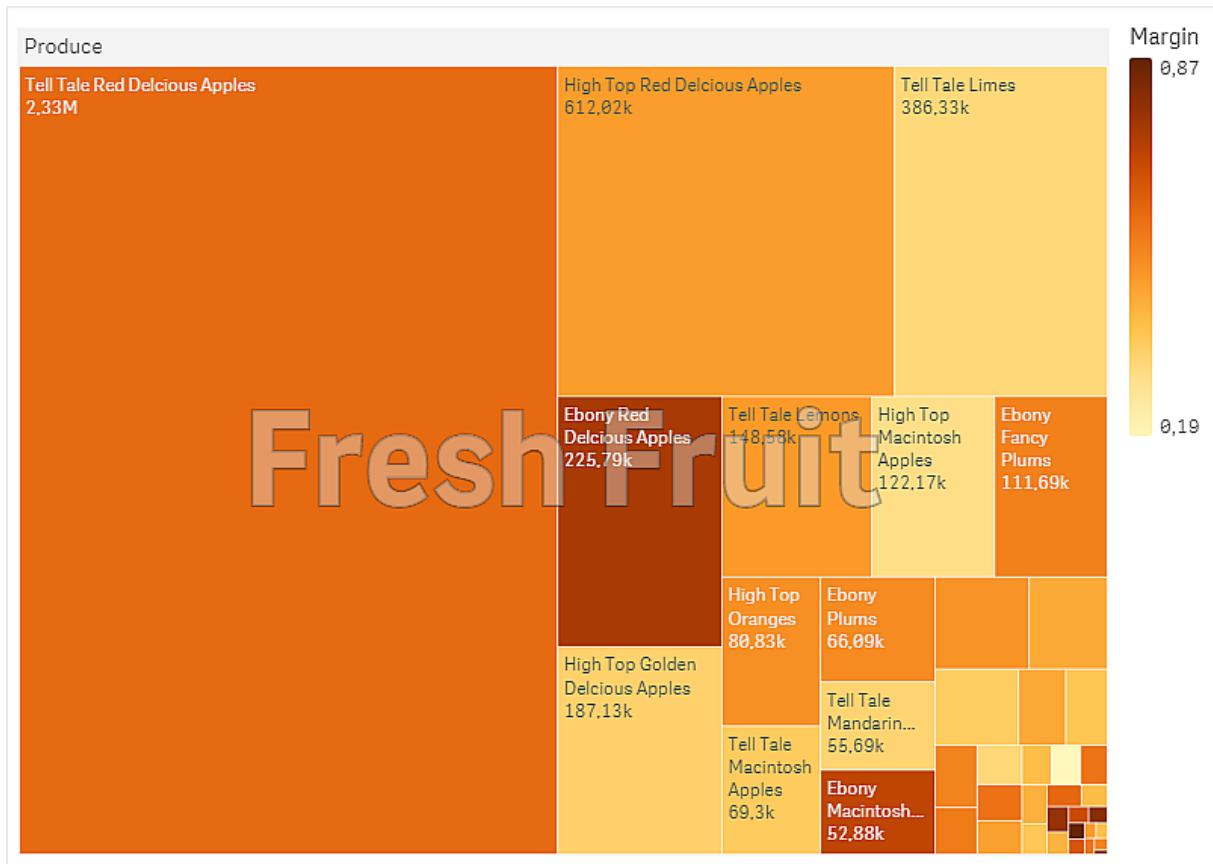
- Set **Colors to Custom**.
- Set **Color by measure** to Avg(Margin/Sales)

We also want to see the sales values in the chart. You can turn them on with **Value labels Appearance > Presentation**.

Discovery



We can see in the chart that Produce has the largest sales, followed by Canned Products. The coloring by margin allows you to identify product items or groups that stand out. You can drill down in the tree by selecting a product group.



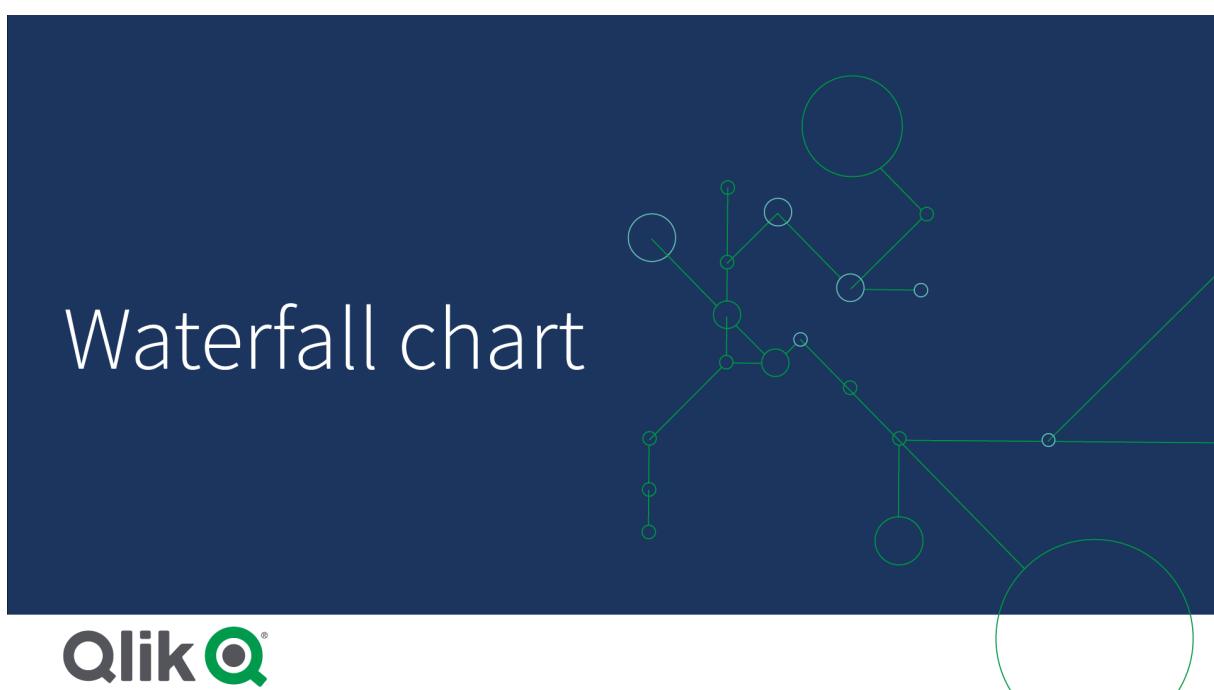
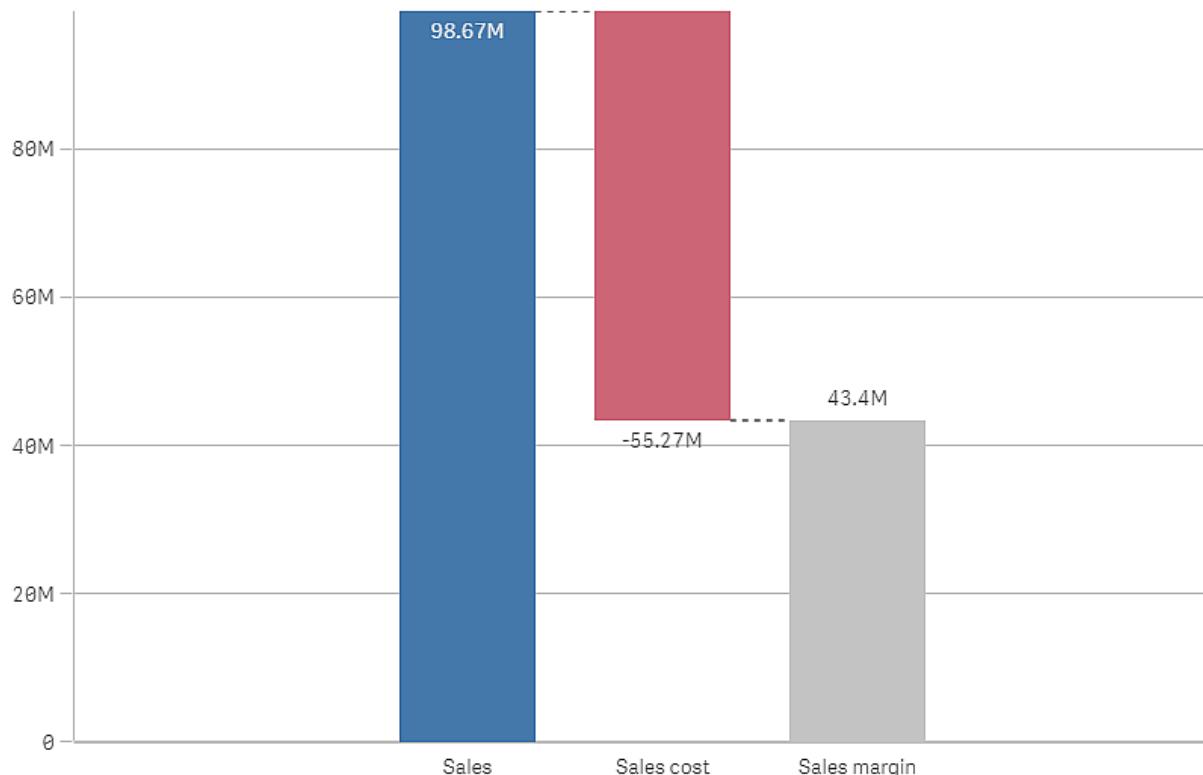
Here we have selected Produce, then Fresh Fruit. Tell Tale Red Delicious Apples are the number one sellers, but margins are higher on Ebony Red Delicious Apples.

Waterfall chart

The waterfall chart is suitable for illustrating how an initial value is affected by intermediate positive and negative values. The starting and the final values are represented by whole bars, and intermediate values by floating bars. You can also show subtotals in the chart.

Waterfall chart displaying Sales, Sales cost and Sales margin.

Sales



When to use it

The waterfall chart is suitable for illustrating how an initial value is affected by intermediate positive and negative values. One example of this is an income statement, when you want to show the positive and negative contributions of different accounts.

Advantages

The waterfall chart provides a quick understanding of the transition of a value.

Disadvantages

The waterfall chart is not relevant for detailed analysis of the data as you can't make selections in the chart or expand the data.

Creating a waterfall chart

You can create a waterfall chart on the sheet you are editing.

In a waterfall chart you need to use one measure for each bar in the chart. The order of the measures defines the order of the bars in the chart. For each measure, you need to define how it affects the previous value. You can add up to 15 measures in a waterfall chart.

Do the following:

1. From the assets panel, drag an empty waterfall chart to the sheet.
2. Add the first measure.
This is the first bar of the chart. By default, it will use the measure operation **Add**, and show a positive value.
3. Add a second measure.
This is the second bar of the chart. If you want to show this measure as a negative contribution, change **Measure operation** to **Subtract**.
4. Continue to add measures, setting **Measure operation** to **Add** or **Subtract** in the advanced properties panel depending on how you want them to contribute.
5. Add subtotals. There are two ways of adding subtotal bars to the chart:
 - If you have a data field containing subtotal data, add a measure with the subtotal data and select **Subtotals** as **Measure operation**.
 - If you don't have a data field containing subtotal data, you can add an automatically calculated subtotal by selecting the **Subtotals** check box of the measure before where you want the subtotal bar.

When you have created the waterfall chart, you may want to adjust its appearance and other settings in the properties panel.

Defining your measures

You can use the **Measure operation** option of each measure to set how it affects the previous value.

- **Add**

The measure value adds to the previous bar. If this is the first measure, a whole bar is shown starting at 0.

- **Subtract**

The measure value subtracts from the previous bar.



If the data already contains a negative sign, the result of subtraction will be a positive change.

- **Subtotals**

The measure value is considered a subtotal.



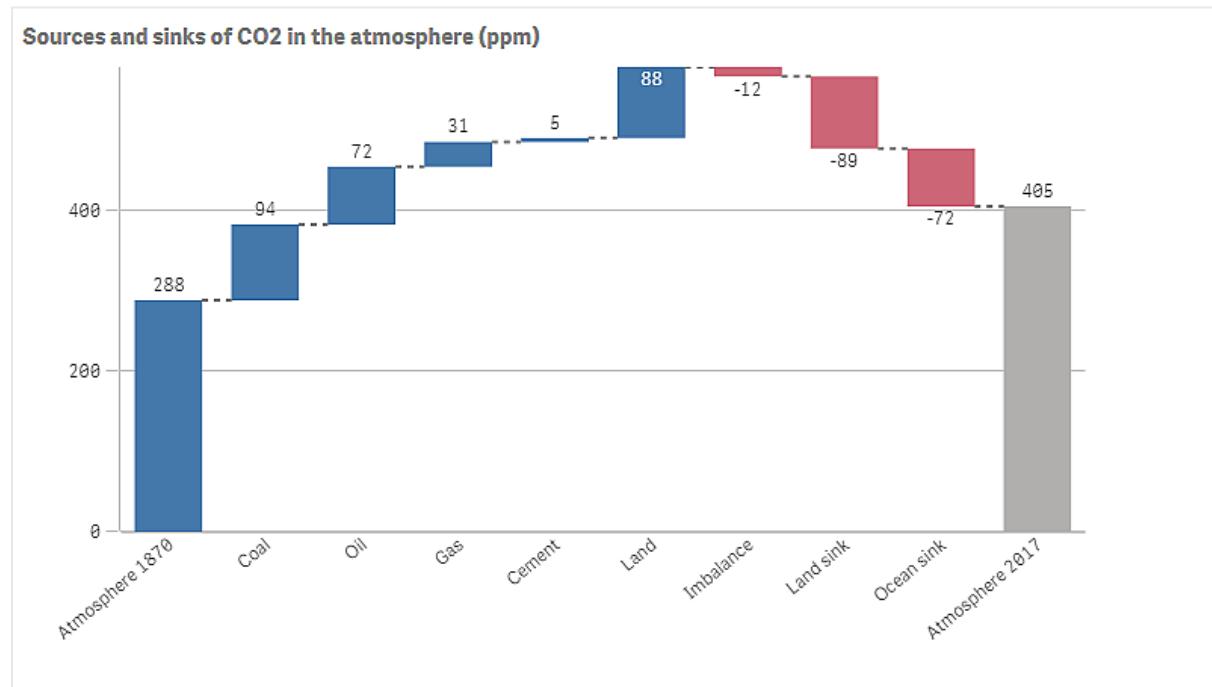
*If you do not have sub-totals as a field, you can add subtotals automatically by enabling **Subtotals** in the measure before you want the subtotal.*

In the waterfall chart shown above, the first bar, Sales, is defined as **Add**. The second bar, Sales cost, is defined as **Subtract**, and the third bar, Sales margin, is defined as **Subtotals**.

Visualizing positive and negative contributions to the result with a waterfall chart

This example shows how to make a waterfall chart to visualize how positive and negative values contribute to the final result.

We want to see the positive contributions to CO₂ in the atmosphere, such as oil or gas, in relation to negative factors, such as land sink.



Dataset

In this example, we will use a simple dataset that you can copy to a text file to use as data source. It contains the atmosphere CO2 level in ppm from 1870, as well as the positive and negative contributors to CO2 in the atmosphere in the time between 1870 and 2017.

```
Atmosphere 1870,Coal,oil,Gas,Cement,Land,Imbalance,Land sink,Ocean sink,Atmosphere 2017  
288,94,72,31,5,88,-12,-89,-72,405  
(Source: CDIAC/GCP/NOAA-ESRL/UNFCCC/BP/USGS)
```

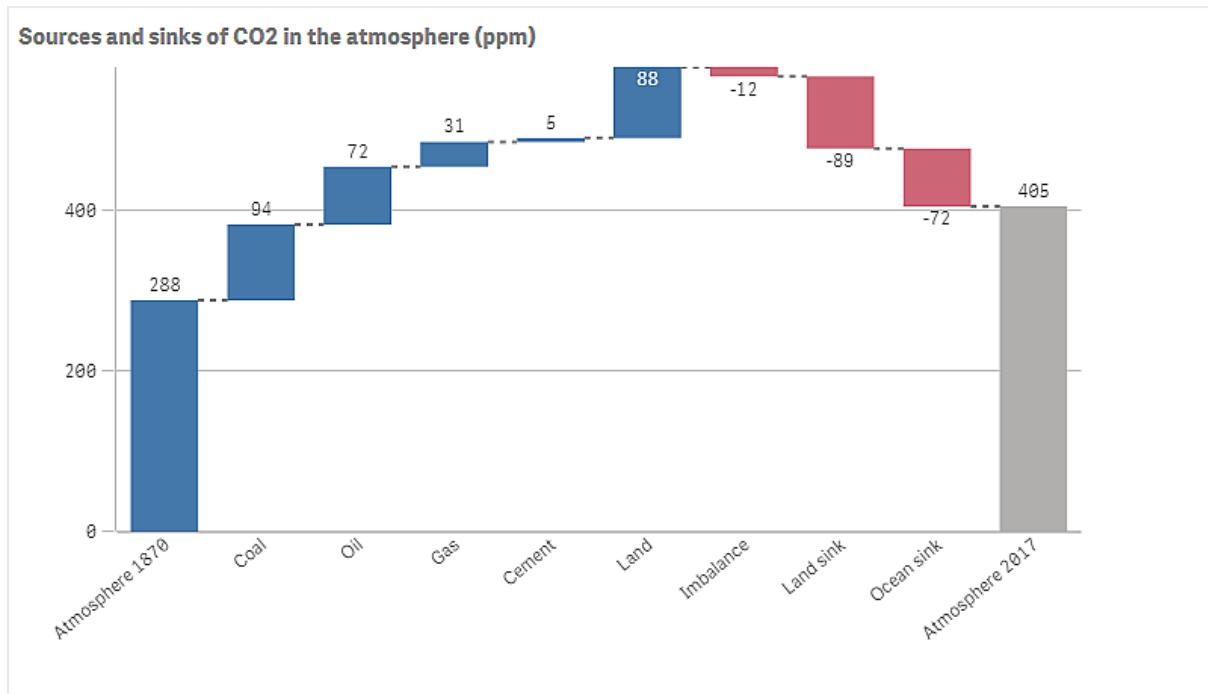
Add the text file as data source in an empty app, and load the data. Make sure that you use comma as field delimiter.

Visualization

We add a waterfall chart to the sheet and add the fields as measures in the following order. Edit the label for each measure to remove the `Sum()` part.

- Sum(Atmosphere1870)
- Sum(Coal)
- Sum(Oil)
- Sum(Gas)
- Sum(Cement)
- Sum(Land)
- Sum(Imbalance)
- Sum(Land_sink)
- Sum(Ocean_sink)

To add a bar that shows the calculated result, the CO2 level for 2017, open the final measure and select **Subtotals**. You can set **Subtotal label** to *Atmosphere 2017*.



Discovery

As you can see in the chart, the CO₂ level in the atmosphere is higher in 2017 than in 1870. Coal is the main contributor of CO₂. Some of the increase is offset by carbon sinks that absorb CO₂, for example, the ocean. As we can see, this is unfortunately not enough to stop the increase of CO₂ in the atmosphere.

Button

You can use buttons to:

- add quick links for easy selection and navigation in your app
- reload data

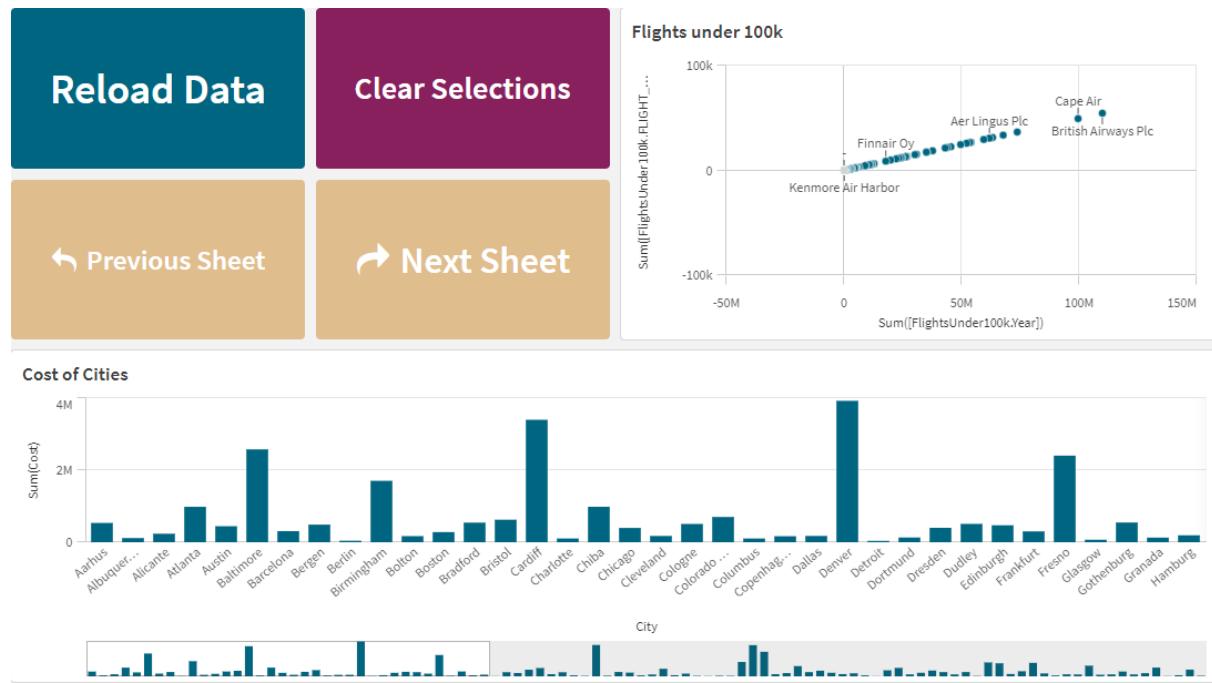
When to use it

The button is useful when you want to help the app user by providing:

- predefined selections or selection controls
- navigation links to other sheets, stories or websites
- reload data functionality in an app

You can reload data in an app that you own. App users need the appropriate access rights.

Example of buttons in use on a sheet.



Creating a button

You can create a button on the sheet you are editing. This shows how to create a button that makes a selection and then navigates to another sheet. It is also possible to create a button that just performs the action, or that navigates.

Do the following:

1. In the **Assets** panel under **Charts**, drag a **Button** object to the sheet.
2. Click **Add action** under **Actions** in the properties panel.
3. Select the action you want to use.
For some actions you need to provide details for the action. For example, for the **Select values in a field** action, you need to select a field, and which value to select in the field.
4. Select which navigation option to use under **Navigation**. You can navigate to another sheet, a story or a website.
5. Set the label of the button in **Label** under **General** in the properties panel.

You will now have a button that makes a data selection and then navigates to the place in the app you selected.

Setting the action

You can add one or more actions to be performed when the button is clicked. For some actions you need to provide details for the action.

The actions are performed in the order they are listed under **Actions**. You can change the order of an action by dragging it.



You do not have to add an action if you just want the button to perform navigation.

Apply bookmark

You can apply the selection that is defined in a bookmark that you choose.

Clear all selections

You can clear all selections in all states in the app. You can optionally overwrite locked selections.

Clear selections in other fields

You can clear selections from all fields except the one you specify. You can optionally overwrite locked selections.

Move forwards in your selections

You can move one step forwards in your selection history.

Move backwards in your selections

You can move one step backwards in your selection history.

Clear selections in field

You can clear all selections from a field that you specify.

Lock all selections

You can lock all selections in the app.

Lock a specific field

You can lock selections in a field that you specify.

Unlock all selections

You can unlock all selections in the app.

Unlock a specific field

You can unlock selections in a field that you specify.

Select all values in a field

You can select all values in a field that you specify. You can optionally overwrite locked selections.

Select values in a field

You can select a list of values in a field that you specify. Separate the values to select with a semi colon. You can optionally overwrite locked selections.



It is not possible to use fields with date, timestamp, or money data type.

Select values matching search criteria

You can select all values that match the search results from a search criteria that you specify. You need to specify the search criteria as a string. You can optionally overwrite locked selections.

- If you want to use an expression, you need to enclose it in single quotes, for example, `=Sum([Sales Amount]) > 200000`.
- If you want to search for a partial string, you need to use wild cards (*, ?, ^). If you do not use wild cards, only strings that match exactly are selected.

Select alternatives

Select all alternative values in a field that you specify. You can optionally overwrite locked selections.

Select excluded

Select all excluded values in a field that you specify. You can optionally overwrite locked selections.

Select possible values in a field

Select all possible values in a field that you specify. You can optionally overwrite locked selections.

Toggle field selection

You can set the button to toggle between the current selection, and a selection that adds selections defined by a search string. You can use wild cards in the search string. If you want to define a list of values you need to use the format `(A|B)`, where A and B are values to select.

Set variable value

You can assign a value to a variable.

Reload data

You can run the load script to reload data.

You can also select **Partial reload**.

Refresh dynamic views

Refreshes the displays of all dynamic objects on the sheet based on the current selection.

Navigation

You can choose to navigate to another sheet, a story, or a website when the button is clicked. You do not have to specify an action if you want to create a simple navigation button. A button can only have one navigation option at a time.



Navigation is not supported in stories.

Go to next sheet

Navigate to the next sheet sequentially.

Go to previous sheet

Navigate to the previous sheet sequentially.

Go to last sheet

Navigate to the last sheet.

Go to first sheet

Navigate to the first sheet .

Go to a sheet

Navigate to a specific sheet. Enter the name of the desired sheet, or select it from the list.

Go to a sheet defined by sheet ID

Navigate to a specific sheet. Enter the sheet ID of the desired sheet. You can find the ID of a sheet in the URL of the browser with the sheet open. The sheet ID follows after /sheet/ followed by /.

Example: Sheet URL

`qlik.com/sense/app/8f8ffa0e-3fde-48a5-a127-59645923a1fc/sheet/7300b241-f221-43db-bb8b-8c8cc1887531/state/analysis`

The ID of the sheet is 7300b241-f221-43db-bb8b-8c8cc1887531 in this example.

Go to a story

Navigate to a specific story. Enter the name of the desired story, or select it from the list.

Open a website or email

Open a specified website or email client. Enter the URL to navigate to. Optionally, select **Open in same window** to have the website or email client open in the same window.

Generate on-demand app

Creates an on-demand app based on the current selection. You must select an on-demand app navigation link . The status of the app is displayed as the app is created.

Styling the button

You have a number of styling options available under **Appearance**.

- You can set the label text under **General**.
- You can set the size, color and style of the font under **Font styling**.
- You can set the background to a color or an image under **Background**.
If you use a background image, you have options to size and position the image.
- You can add an icon to the button under **Icon**.
- You can adjust the border of the button under **Borders**.
You can set the corner radius and the width of the border.
You can also set the color of the border, either a single color or with an expression.

Disabling and enabling the button

You can set a calculation condition to enable the chart under **Enable chart**. If the condition is evaluated to 0, the button is disabled.

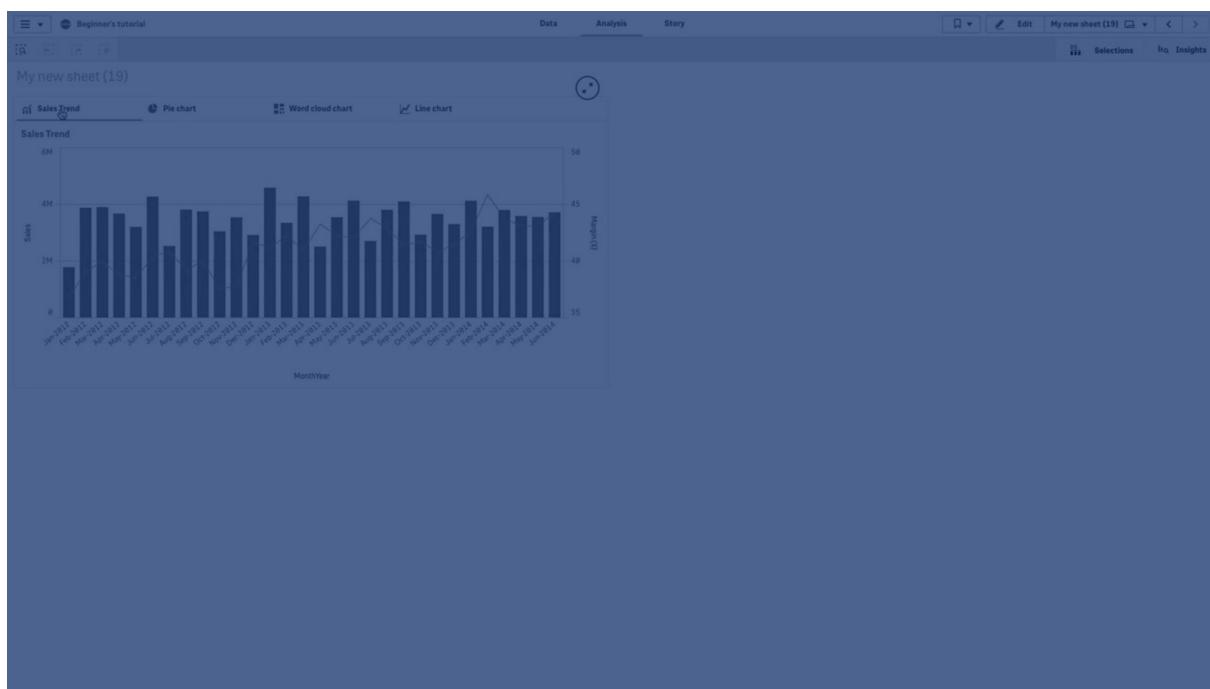
Converting from the **Button for navigation** in the Dashboard bundle

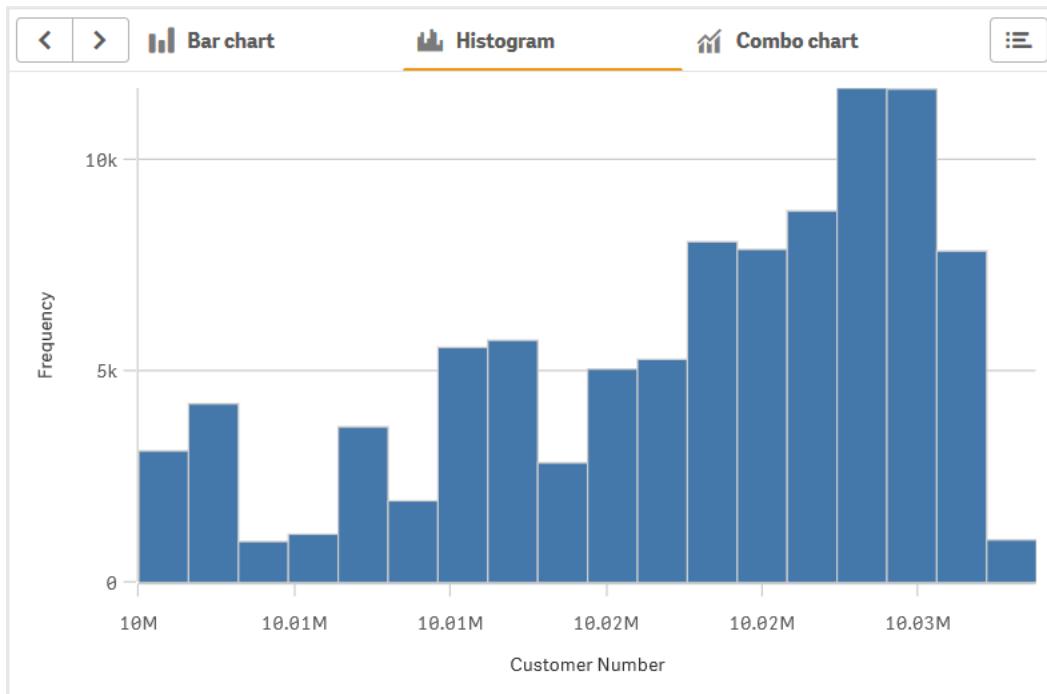
You can convert an existing **Button for navigation** control into a button. There are some limitations when you do the conversion.

- The navigation option to switch to edit mode (**Switch to edit mode**) is not supported.
- The action to select a value and then lock the field (**Select a value and lock field**) is not supported.
This will be converted into two consecutive actions, **Select values in a field** and **Lock a specific field**.

Container

The container is an object that lets you add visualizations in a limited space. You can also show or hide the visualizations inside the container based on conditions.





When to use it

The container is useful when you want to quickly switch between different visualizations on a dashboard with limited space. You can also use a container to show different visualizations based on:

- Which user is accessing the chart.
- The value of a variable.
- The possible number of values in a field, by using the **GetPossibleCount()** function in the condition expression.

Displaying different charts based on selections with a container (page 304)

Creating a container

You can create a container on the sheet you are editing.

Do the following:

1. From the assets panel, drag a **Container** object to the sheet.
2. Click **Add** under **Content** in the properties panel.
3. Select a master visualization in **Master items**, or create a new visualization in **Charts**.
Alternatively: you can add content to your container by dragging available visualizations from your sheet, or from your **Master items**, directly on the container.
4. Drag the charts in the property panel to set the tab order.

You now have a container with a tab for each visualization you added. You can switch between tabs to show different visualizations.

Adding show conditions

You can add show conditions on the tabs you have created.

Do the following:

1. Click on the chart you want to add a show condition to, under **Content** in the property panel.
2. Add a show condition for the chart in **Show condition**. Typically you would use an If() function.
3. Add another show condition to a different chart.

The charts will now be shown or hidden depending on the result of the conditions you added. If the condition of a chart results in True, it is shown, and if it is False, it is hidden.

Changing the default active tab

By default, the first tab is displayed when you view a container. You can change this in the property panel.

Do the following:

1. Go to **Appearance > Container** in the property panel.
2. Use the **Default tab** dropdown to select which chart displays when users view a sheet. The dropdown lists the available charts in the same order they are arranged in the container.
3. If you want to see the new default tab setting take effect, close and re-open your browser.

Creating a bookmark with a container object

By default, if you create a bookmark that contains a container object with selections, the first tab will be active when the bookmark is selected. To save a different default active tab for the container, you can toggle on **Save layout** when creating the bookmark with that tab open. If you set that bookmark as the default bookmark by right-clicking and selecting **Set as default bookmark** in the bookmark list, the sheet will open to the selections and tab that were active when the bookmark was created.

Do the following:

1. Click  in the toolbar.
2. Click **Create new bookmark**.
Change the name and description, if desired.
3. If you want a tab other than the default tab to be the active tab, toggle on **Save layout** with that tab open. This will override the **Default tab** setting in the property panel.
4. Click **Save**.

Display limitations

- You cannot use a master visualization that has a container inside another container.
- You cannot add the same master visualization twice in a container.
- You cannot create a chart inside a container by dropping measures or dimensions on the container.

Displaying different charts based on selections with a container

This example shows how to display different chart content based on user selections using a container.

- When a single product group is selected a chart for product sub groups is displayed.
- When more than one product group is selected a chart for product groups is displayed.

Dataset

In this example, we will use two data files from the Qlik Sense Tutorial - Building an App. To download the files, go to [Tutorial - Building an App](#). Download and unzip the tutorial, and find the files in the *Tutorials* source folder:

- *Sales.xls*
- *Item master.xls*

Create a new app, and add the two data files. Make sure that they are associated by *Item Number*.

The dataset that is loaded contains sales data. The *Item master* table holds the information about the items ordered, such as product groups.

Measures

We need to create two measures in Master items:

- Sales volume: with the name *Sales*, and the expression `Sum(Sales)`.
- Sales margin in percent: with the name *Margin %*, and the expression `Avg(Margin/Sales)*100`.

Visualizations

We need two different master visualizations based on product group selections. One of them with product group as dimension, the other with product sub group as dimension.

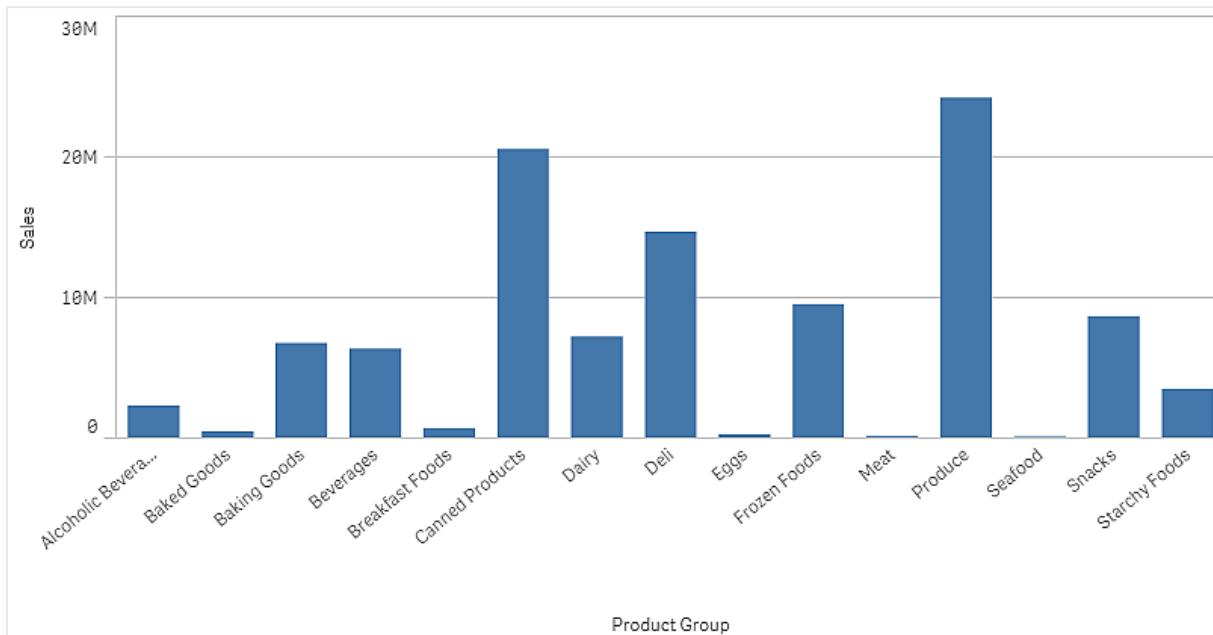
Product group visualization

This is the visualization we want to show when more than one product group is selected.

We add a combo chart to the sheet and set the following data properties:

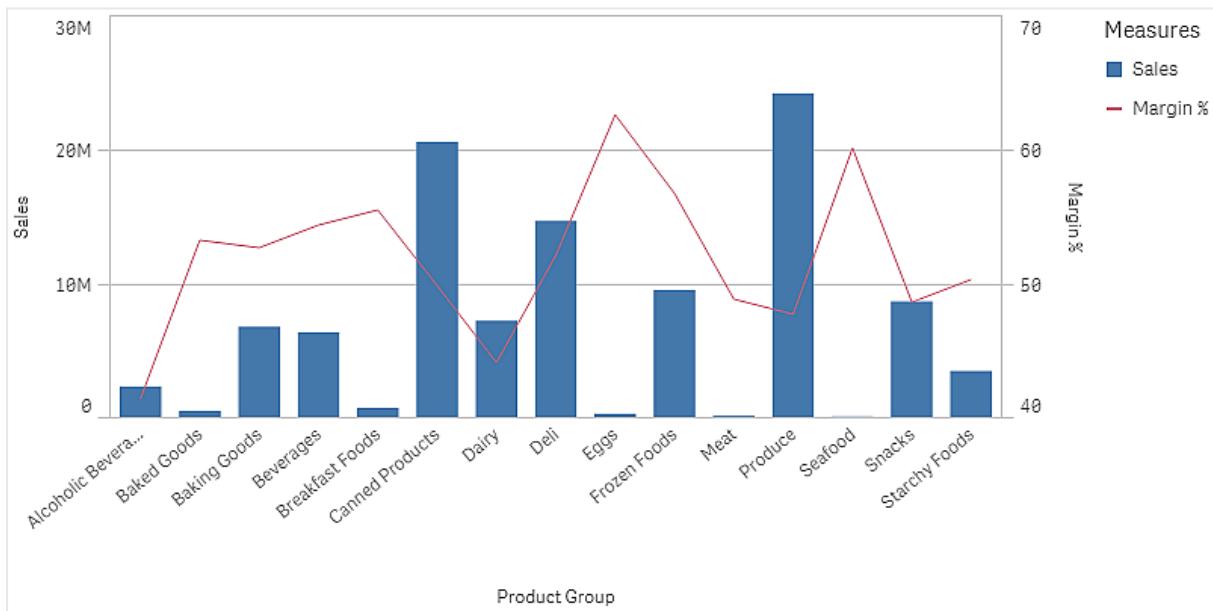
- **Dimension:** Product Group (product group).
- **Measure:** *Sales* (the master measure you created).

The following chart is created, with a bar showing the sales for each product group. It is a bar chart at this stage.



But we also want to show the sales margin, which has a different scale than the sales volume. Sales volume is in the scale of millions, while the margin is a percentage between 0 and 100. If we add margin as a bar next to sales volume, it would be too small to distinguish.

In the properties pane, go to **Measures > Height of line**. Use the drop down to add *Margin %* as a measure.



Product sub group visualization

This is the visualization we want to show when a single product group is selected.

Make a copy of the product group visualization, and change the dimension to Product Sub Group.

Master visualizations

You need to create two master visualizations to use in the container.

- Add the product group visualization as a master visualization with name Product Group - sales and margin.
- Add the product sub group visualization as a master visualization with name Product Sub Group - sales and margin

You can now delete the two visualizations you created earlier, they are saved as master visualizations.

Container

You need to add a container to the sheet. Add the two visualizations to the container.

- Product Group - sales and margin
Set **Show condition** to `=GetPossibleCount([Product Group])>1`
- Product Sub Group - sales and margin
Set **Show condition** to `=GetPossibleCount([Product Group])=1`

You can also hide the selection tabs of the container under **Appearance>Container>Tabs**.

To demonstrate how it works, you can add a filter pane with Product Group.

Discovery

You can now make selections in Product Group. When a single product group is selected, the chart displays data for product sub groups of that product group. Otherwise, the chart displays data for product groups.

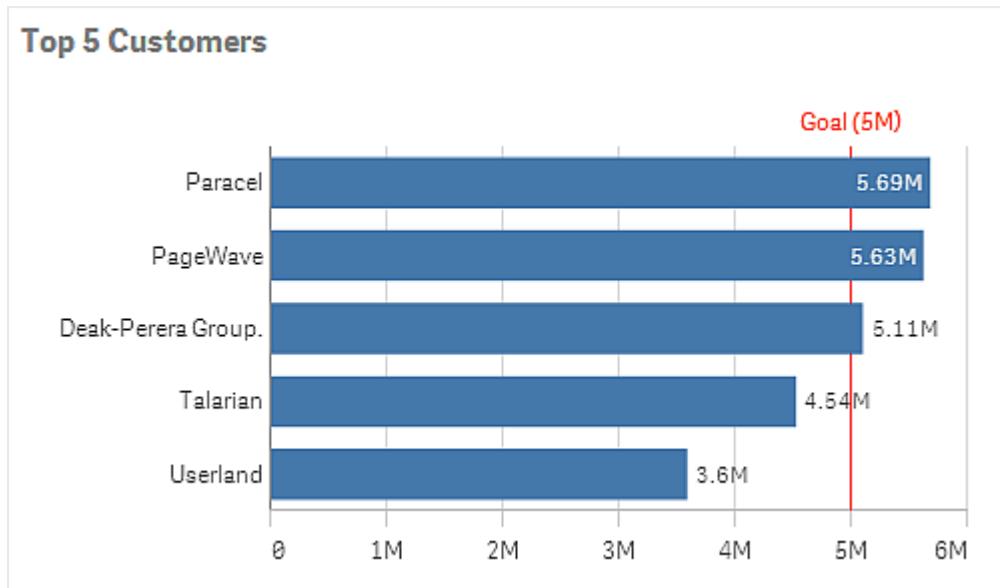
Reference lines

A reference line is a line intersecting the chart area from a given point on the measure axis.



You can use a reference line to indicate a certain level of chart data. The reference line is only drawn if it falls within the current range of the measure axis. You can have several reference lines in the same chart.

Bar chart Top 5 customers with a reference line at 5M.



Reference lines are available in the following visualization types:

- Bar chart
- Box plot
- Combo chart
- Distribution plot
- Gauge
- Histogram
- Line chart
- Scatter plot
- Waterfall chart

Reference line expression

You can either set the reference line expression to an absolute numeric value, or enter an arbitrary numeric expression.

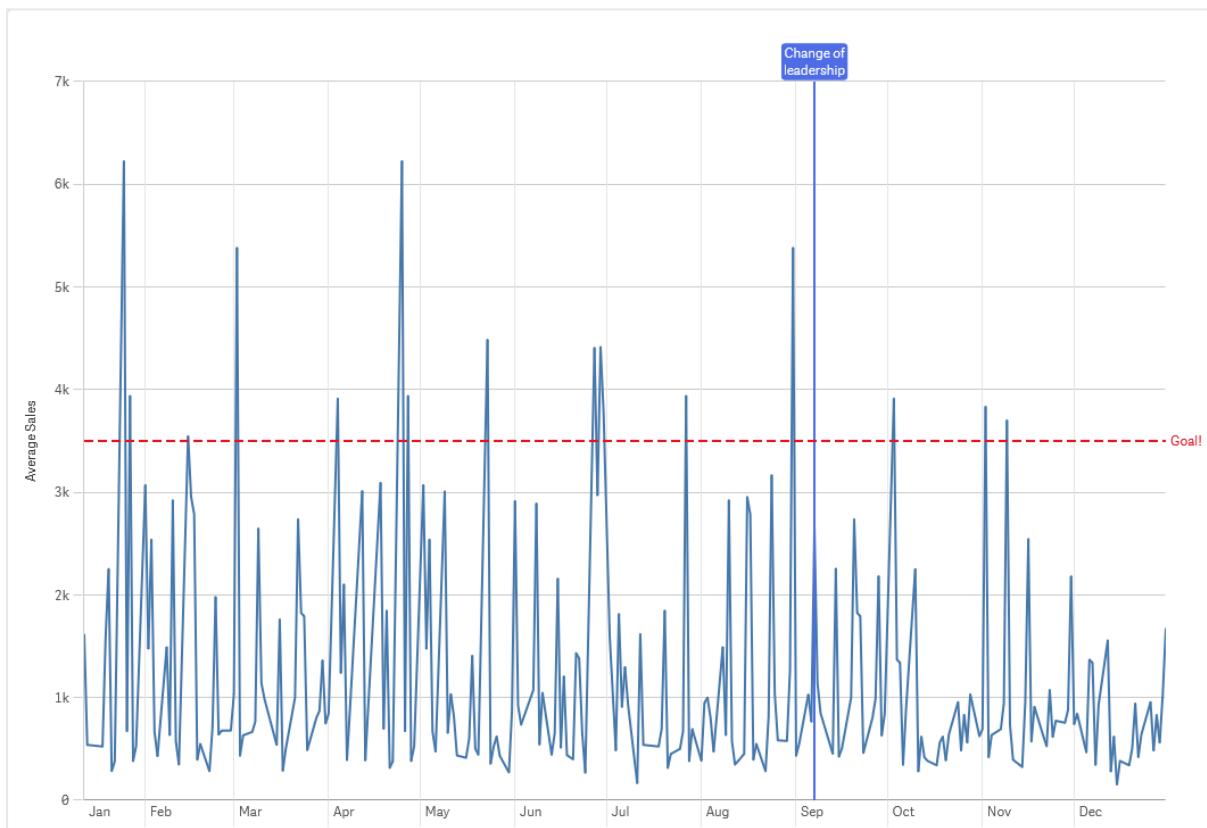
Dimensional reference lines

You can also add reference lines along the dimension axis. Both continuous and discrete axes are supported. It is possible to enter text values for discrete dimensional axes, and numerical values or expressions for continuous axes. On a time axis it is possible to enter an expression that gives a time-based result like a date or a month.

3 Visualizations



Line chart average sales per month, with a reference line at 3.5k and a dimensional reference line in September.



Dimensional reference lines are available in the following visualization types:

- Bar chart
- Combo chart

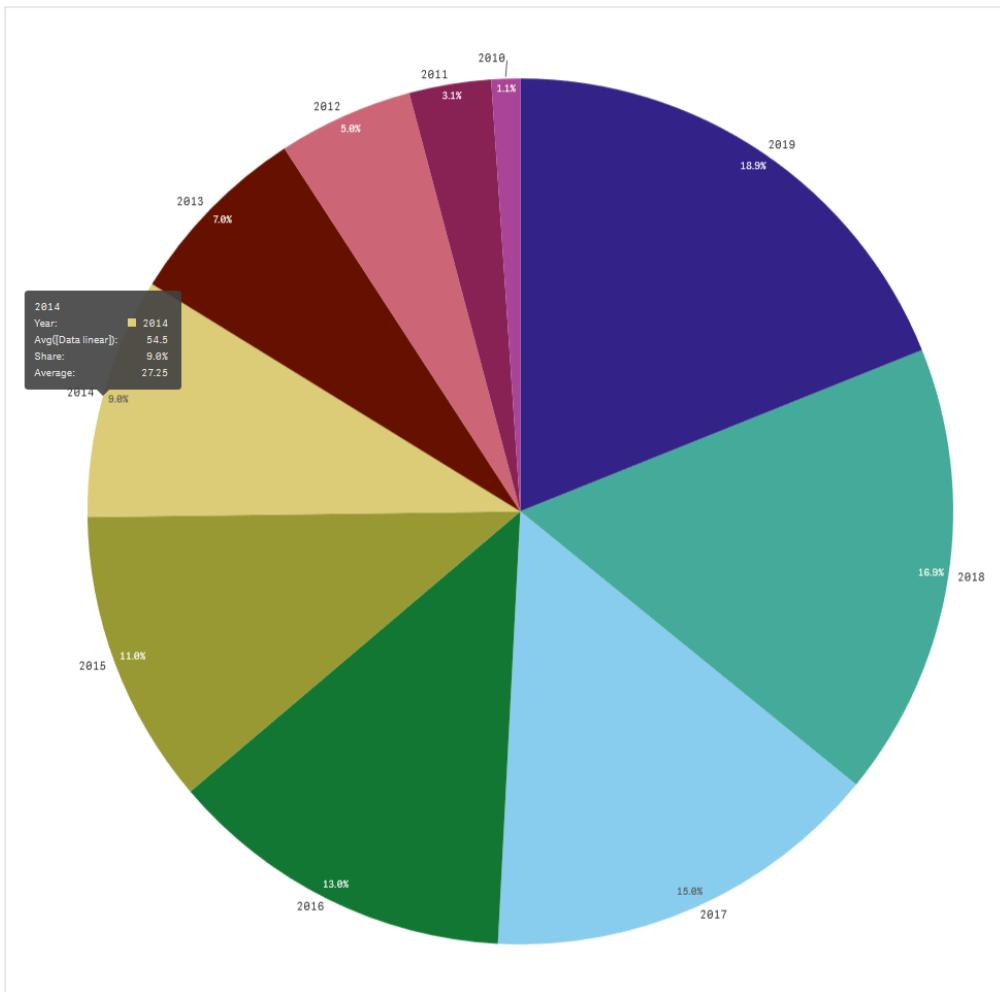
- Line chart

Custom tooltips

Create a custom tooltip to add extra information to a visualization. You can include measures, a master visualization chart, or an image. Tooltips display when you hover over a visualization.



Custom tooltip over a pie chart slice



Custom tooltips are available with the following visualization types:

- Bar chart
- Bullet chart
- Combo chart
- KPI
- Line chart
- Map
- Pie chart
- Scatter plot
- Tree map

Creating a custom tooltip

Creating a custom tooltip

Tooltips can contain measures, a chart, or an image. Add a measure from a field or using an expression. Select a chart from the master visualizations available in the app. Choose an image from the media library or by URL. You can also hide the basic rows that appear in a tooltip by default.

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Select the chart to edit.
3. In the properties panel, click **Appearance**, and then click **Tooltip**.
4. Set the toggle to **Custom**.
5. Optionally, enter a **Title** and **Description** for the tooltip.



You can remove basic tooltip rows that appear by default by clicking **Hide basic rows**.

6. If you want to add a measure, under **Measures**, click **Add measure**. Select a measure from a field using **From a field**, or use an expression. The existing expression displays by default, or you can create a custom expression. Click  to open the expression editor. You can change the label of the tooltip using **Label** as well as its formatting using **Number formatting**.
7. If you want to add a chart, under **Chart**, click **Add chart**.
 - From the **Master items** list, select the master visualization to add to the tooltip.
 - Under **Chart size**, select the size of the chart container in the tooltip.
 - **Small**. Sets the width and height of the container to 140 pixels.
 - **Medium**. Sets the width and height of the container to 200 pixels.
 - **Large**. Sets the width and height of the container to 340 pixels.



After you add the chart, you can edit it by clicking **Edit master item**. See *Editing a master visualization* (page 87).

8. If you want to add an image, under **Images**, Click **Add an image**.
 - Under **Type**, select whether to add the image from your **Media library** or from a URL.
 - **Media library**: Select the image from the media library.
 - **URL**: Enter a URL.
 - Under **Size**, select the size of the image container in the tooltip.
 - **Small**. Sets the width and height of the container to 50 pixels.
 - **Medium**. Sets the width and height of the container to 200 pixels.
 - **Large**. Sets the width and height of the container to 340 pixels.
 - **Original**. Qlik Sense fits the image into the container. If the image is larger than 340 pixels, it is scaled down.

Limitations of charts in custom tooltips

Charts have the following limitations with custom tooltips:

- Charts will not appear in custom tooltips when selecting visualizations on touch devices.
- Treemap chart must have only one dimension to use the chart in a custom tooltip.

- Chart in tooltip is not supported in Storytelling.
- Using container and trellis container charts inside a custom tooltip is not supported.

Null values in visualizations

Data is sometimes missing or cannot be calculated, because the fields contain values that are null or not a number (NaN). In the visualizations, null and NaN values are displayed in different ways, according to the following table.

How Null and NaN values are displayed

Visualization type	Null values in dimensions	Nan values in measures
Bar chart	-	- (when labels are enabled in the properties panel, otherwise empty)
Box plot	-	No representation
Combo chart	-	A combination of the NaN value for the bar and the line.
Distribution plot	-	No representation
Filter pane	No representation	N/A
Gauge	N/A	-
Histogram	-	No representation
KPI	N/A	-
Line chart	-	Empty
Map	No representation	Gray
Pie chart	-	Empty
Scatter plot	-	Empty
Table	-	-
Text & image	N/A	-
Treemap	-	Empty

Dashboard bundle

Dashboard bundle is a set of controls that you can use to enhance navigation and selection in your Qlik Sense app. The controls are optional. You do not have to install or enable them to use Qlik Sense.

Enabling Dashboard bundle

You can install the Dashboard bundle when you install Qlik Sense. If you need to adjust your installation, see: [Modifying an object bundles installation](#).

Dashboard bundle controls

Dashboard controls are in the asset panel under **Custom objects**.

The following controls are included:

- *Animator* (page 316)
You can animate changes in your visualizations over a period of time.
- *Date range picker* (page 317)
You can select a single date or a range of dates from a calendar.
- *NL Insights* (page 319)
Displays natural language insights about selected fields.
- *On-Demand reporting control* (page 323)
You can add a button that creates a Qlik NPrinting report using the current selections in the app.
- *Variable input control* (page 327)
You can set the value of a variable.
- *Video player* (page 329)
You can add a video to your sheet.

Deprecated controls

These controls have been deprecated by the addition of a native control. Existing instances of these controls will still function, but new ones cannot be added. We recommend that you replace deprecated controls with the native control.

Share button control

If you need to configure existing instances of this control, please refer to the last version of the help before deprecation.

Show/hide container

Deprecated by **Container**.

If you need to configure existing instances of this control, please refer to the last version of the help before deprecation.

Tabbed container

Deprecated by **Container**.

If you need to configure existing instances of this control, please refer to the last version of the help before deprecation.

Navigation button

Deprecated by **Button**.

If you need to configure existing instances of this control, please refer to the last version of the help before deprecation.

Limitations

When you use objects from Dashboard bundle there are some limitations compared to built-in controls. The following limitations are valid for all controls from Dashboard bundle:

- The user interface of the visualization is not localized to the language that Qlik Sense is using.
- Right-to-left reading order is not supported.
- Accessibility features are not supported.
- It is not possible to edit objects from an object bundle supplied by Qlik with Dev Hub.

The following table shows which additional features are supported, or not supported, for all bundled objects:

- **Qlik NPrinting**
There is an *On-Demand reporting control* (page 323). However, visualizations made in some objects cannot be used in Qlik NPrinting reports.
- **Download**
Downloading as image, PDF, or Excel.
- **Storytelling**
Using a snapshot of a visualization created with an object in a story.
- **Alternate states**
Making different selections on the same dimension, and comparing the selections in a single visualization or in two or more visualizations side by side.
- **Qlik Sense Mobile Client Managed offline**

Dashboard bundle capability support

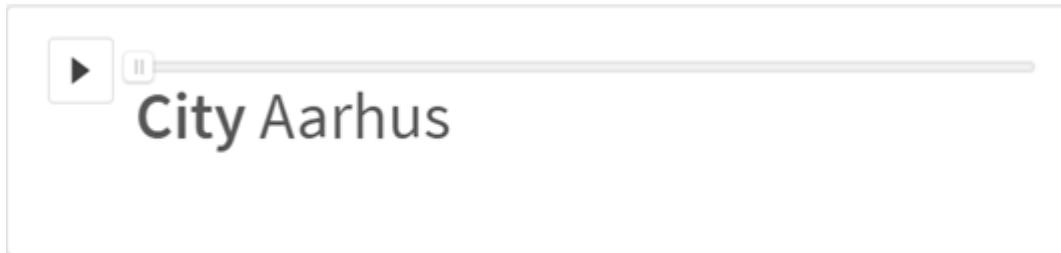
Control	Qlik NPrinting	Download	Storytelling	Alternate states	Qlik Sense Mobile Client Managed offline
Animator	Not applicable	Excel only	Not applicable	Yes	Yes
Date picker	Not applicable	Not applicable	Not applicable	Yes	Yes
NL Insights	Not applicable	Image and PDF only	Yes	No	Not applicable
On-demand reporting	Yes	Not applicable	Not applicable	Not applicable	Not applicable
Variable input	Not applicable	Not applicable	Not applicable	Not applicable	Yes
Video player	Not applicable	Not applicable	Yes	Yes	Not applicable

Animator

The animator control animates the changes in values in your visualizations over a range of values. For example, you could view changes in your visualizations over a period of time. It is included in the Dashboard bundle.

The animator control adds a button and progress slider to the sheet. When pressed, animator cycles through the values of its dimension. Users can use the slider to manually pick values from the dimension to display.

Animator



When to use it

The animator is useful when you want to see your visualizations change over a specified range, such as over a period of time. For example, you could use the animator control to view how visualizations in your sheet change month to month over a period of a year. Animator can be used to animate any dimension values. You could, for example, animate how your sales visualizations change city to city by selecting a dimension containing city values to use with animator.

The selections animator makes while running through values are applied to your current selections.

Creating animators

You can create an animator on the sheet you are editing. The dimension you select to use with animator contains the range of values you want animated. By default, every value in the dimension will be used as a step. You can customize the number of steps used in the animation as well as how quickly animator moves between steps. Animator runs through dimension values by the order in **Sorting**.

Do the following:

1. In the assets panel, open **Custom objects > Dashboard bundle** and drag a **Animator** object to the sheet.
2. Select a field to add as the dimension.

Once you have created the animator, you may want to adjust its animation options and other settings.

Setting animator options

Animator options are set in **Animator Options** in the **Properties** pane.

You can control how fast an animator transitions between values in the selected dimension in **Time Between Steps**. **Time Between Steps** requires values be set in milliseconds.



*It is not recommended to set **Time Between Steps** lower than 300 milliseconds.*

You can set the total steps the animator runs through in **Total Steps**. The animator will divide the total steps evenly across the dimension values. For example, if there are 10 values in the time dimension selected and Total Steps is set to 5, animator will go to values 2, 4, 6, 8, 10). If Total Steps is set to 0, the animator will use every value.

If you want the animation to show the total building over time for your selected values, select **Aggregate**. When **Aggregate** is selected, each step is aggregated to the previous step.

By default, the animator will run through steps in a loop. You can control this with the **Loop** option.

You can control whether or not the dimension and current value displays in the animator as it runs with **Show Dimension Label** and **Show Dimension Value**.

Limitations

Animator has the following limitations:

- Animator does not support drill-down dimensions as the dimension.
- Animator does not support animating trellis charts.

Date range picker

The date range picker (**Date picker**) lets you select a single date or a range of dates from a calendar. It is included in Dashboard bundle.

Date picker has two modes:

- In single date mode, you select from a simple calendar.
- In date interval mode, you can select a range from the calendar, or any of the predefined ranges that are available.

Dates associated to data appear in black in the calendar. Dates with no associated data appear in grey. Users can select any date.

When to use it

The date range picker is useful when you have one or more date fields that you need to use to filter your selections. Unlike a filter pane, a date range picker will only display date fields.

Creating a date range picker

You can create a date range picker on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects** > **Dashboard bundle** and drag a **Date picker** object to the sheet.
2. Select the date field to use as filter in **Date field**.

The list contains only fields that are recognized as a date field by Qlik Sense.



If you want to use a timestamp field, you need to convert it to a date. Use the expression `=date(floor(myTimestamp_Field))` instead of `myTimestamp_Field`.

3. Select if you want to be able to pick a single date or a date interval in **Single date / interval**.

Date interval lets you pick a range of dates.

Single date lets you pick a single date only.

When you have created the date range picker, you may want to adjust its appearance and other settings.

Setting the available date range

You can set the range of dates available to pick by setting the option **Advanced setup** to **On**. You can use expressions or explicit date values. The expressions used below refer to a field named *DateField*.

- Set the first date of the calendar with **Min date**.

Default value is `=Min({1} [DateField])`. This expression returns the first date in *DateField* in the full data set.

- Set the last date of the calendar with **Max date**.

Default value is `=Max({1} [DateField])`. This expression returns the last date in *DateField* in the full data set.

- Set the date to show when the date range picker is opened with **Start date**.

Default value is `=Min([DateField])`. This expression returns the first date in *DateField* in the current selection.

Setting locale of the calendar

You can set the locale of the calendar to get local names for days and months. Change the setting **Locale** under **Calendar Settings > Language and labels** to the two-letter code of the locale you want to use. For example, you can change the default value of *en* for English to *de* for German.

Using predefined ranges

In date interval mode there are a number of predefined ranges available to pick if **Show predefined ranges** is set to **On**. You can customize the text labels for the predefined range selections with the settings under **Calendar Settings > Predefined ranges**:

- **Custom Range** represents the option where you select a date interval freely from the calendar. Default value is **Range**.
- **Today** represents the date value of today. Default value is **Today**.
- **Yesterday** represents the date value of yesterday. Default value is **Yesterday**.
- **Last \$ days** represents the two options for showing the last 7 or 30 days. \$ is replaced by 7 or 30 in the string. Default value is **Last \$ days**.
- **This** represents the current period. By default **Month** is selected. You can select:
 - **Day**
 - **Month**
 - **Quarter**

- **Year**
- **None**
- **Last** represents the previous period. By default **Month** is selected. You can select:
 - **Day**
 - **Month**
 - **Quarter**
 - **Year**
 - **None**

You can specify how many of the previous periods to include in **Last number of**.

You can include the current period by selecting **Include current**.

Customizing text labels

You can customize the text labels that are shown when the date range picker is used.

You change the text prompt shown when the date range picker is minimized with the setting **Default Text** under **Calendar Settings > Language and labels**. The default setting is 'Select date range'.

Limitations

For information about general limitations, see *Limitations (page 315)*.

NL Insights

The NL Insights control displays natural language insights and information about selected dimensions and measures. This allows you to effectively share insights about your data with your app's users.

NL Insights is available as a part of the Dashboard bundle. NL Insights uses Insight Advisor to generate natural language information about your selected dimensions and measures. NL Insights updates the natural language insights as selections are made in the app.

NL Insights displaying natural language insights

The screenshot shows the Qlik Sense interface with the following details:

- Top Bar:** Qlik logo, three dots, Prepare Data load ed..., Analyze Sheet (highlighted in green), Narrate Storytelling, Tutorial - Business Logic, Ask Insight Advisor.
- Left Sidebar:** Insight Advisor (selected), search icon, refresh icon, no selections applied.
- Right Sidebar:** Bookmarks, Sheets, navigation icons, Edit sheet.
- Sheet Content:** "My new sheet" title. The sheet contains four cards:
 - Ranking:** The total Gross Profit is 543.1k.
 - The top City is Cunewalde with Gross Profit that is 10.7% of the total.
 - The top 19 City represents 79.3% of Gross Profit.
 - Correlation:** Correlation: Cost of Sale and Gross Profit have a 90.94% correlation.
 - Ranking:** The total Freight is 163.6k.
 - The top Supplier is New Balls with Freight that is 8% of the total.
 - The top 18 Supplier represents 78.3% of Freight.
 - Ranking:** The total Cost of Sale is 2.14M.
 - The top ProductName is Minni Pälsii with Cost of Sale that is 22.3% of the total.
 - The top 29 ProductName represents 80% of Cost of Sale.

By default, NL Insights generates insights for all analysis types supported for the selected fields. The classification of fields in your app logical model determines which fields are available as dimensions or measures.

NL Insights includes analyses based on the data selected. By default, all possible analyses for the selected data are included. You can remove unwanted analyses from NL Insights. NL Insights can offer the following analysis types:

- Calculated measure
- Ranking
- Ranking (grouped)
- Breakdown (geospatial)
- Breakdown
- Overview
- Relative importance
- Year to date
- Trend over time
- Comparison
- Correlation
- Process control (means)

Analyses are not included if the selected dimensions or measures do not support that analysis type. You can remove unwanted analysis types from **NL Insights** in the properties panel. To learn more about Insight Advisor analyses, see *Insight Advisor analysis types (page 427)*.

You can control the verbosity of the natural language insights. Verbosity can be full or brief. Full verbosity groups natural language insights by analysis type. Brief displays all natural language insights as a list. You can optionally choose to have your natural language insights display as bulleted lists.

NL Insights displaying natural language insights with brief verbosity

The screenshot shows the Qlik Sense interface with the 'Analyze Sheet' tab selected. The top navigation bar includes 'Prepare Data loaded...', 'Analyze Sheet' (highlighted in green), 'Narrate Storytelling', 'Tutorial - Business Logic', 'Ask Insight Advisor', 'Bookmarks', 'Sheets', and 'Edit sheet'. Below the navigation is a toolbar with icons for search, refresh, and other functions. The main area is titled 'My new sheet' and contains four sections of NL Insights results:

- Top-left section:
 - The total Gross Profit is 543.1k.
 - The top City is Cunewalde with Gross Profit that is 10.7% of the total.
- Top-right section:
 - Correlation: Cost of Sale and Gross Profit have a 90.94% correlation.
- Bottom-left section:
 - The total Freight is 163.6k.
 - The top Supplier is New Balls with Freight that is 8% of the total.
- Bottom-right section:
 - The total Cost of Sale is 2.14M.
 - The top ProductName is Minnki Pälsii with Cost of Sale that is 22.3% of the total.



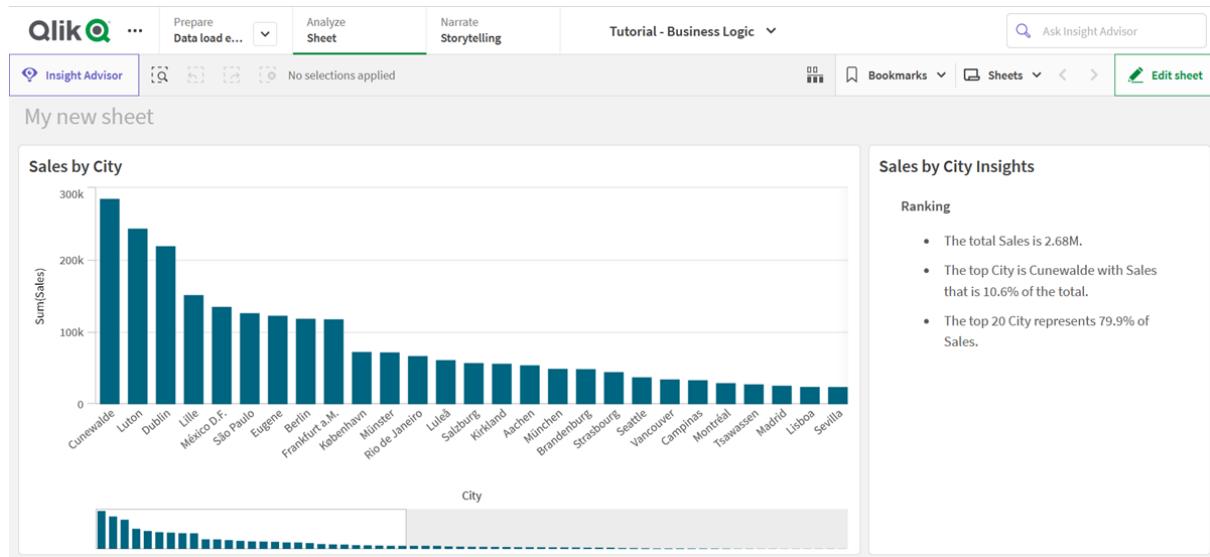
NL Insights has the same language supports as natural language insights. For information, see Supported languages (page 420).

When to use it

Use NL Insights when you want to include a narrative or analytical summary of your data. For example, on a sheet focused on sales, you might want narratives summarizing key ranking, correlation, and comparison analyses for sales to complement your other visualizations.

It is also helpful to pair NL Insights with another chart containing the same dimensions and measures. This help provides additional context and information for the chart. For example, you could add the same dimension and measure as a bar chart to a NL Insights to provide a ranking analysis of the data in the bar chart.

Using NL Insights with a bar chart to provide more context for Sales by City



Creating NL Insights

Do the following:

- In the advanced edit mode assets panel, open **Custom objects > Dashboard bundle** and drag **NL Insights** to the sheet.
- In **Data**, under **Measures**, add the measures to use in the natural language insights. You can add up to three measures.
- In **Data**, under **Dimensions**, add the dimensions to use in the natural language insights. You can add up to three dimensions.
- In **Presentation**, under **Analysis types**, optionally select the analyses to include in your insights.
- In **Presentation**, under **Verbosity**, select how detailed insights should be.
- In **Presentation**, under **Style**, select if the insights should be bulleted or in paragraphs.

Limitations

NL Insights has the following limitations:

- New master items added to the app are not available in NL Insights until you refresh.
- Do not add more than three NL Insights per sheet in an app.
- NL Insights does not support:
 - Alternate states
 - Calculation conditions
 - Custom tooltips
 - Download data
 - View data
 - Master item colors

- Sorting
- Snapshots for data storytelling
- You cannot use NL Insights in trellis containers.

On-Demand reporting control

The On-Demand reporting control adds a button to create Qlik NPrinting reports on demand. It is included in Dashboard bundle.

Requirements

- Qlik NPrinting Server (September 2019 or later) with a Qlik NPrinting app that is connected to the Qlik Sense app that you are designing. This Qlik NPrinting app contains your report templates.
- All users generating On-Demand reports need to be added as users in Qlik NPrinting, with a security role that supports running On-Demand reports.
- Users must be logged into Qlik Sense via NTML (Windows) authentication in order to generate reports or create reporting buttons.

When to use it

The On-Demand reporting control is useful when you want the user to be able to print predefined Qlik NPrinting reports within Qlik Sense, using their selections in the app as a filter.

Installing the Dashboard bundle

You can install the Dashboard bundle when you install Qlik Sense.

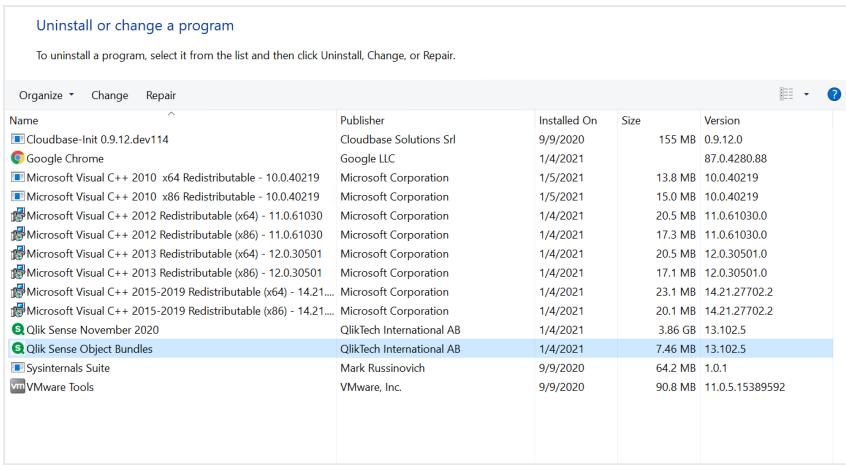
You can add or remove object bundles from your Qlik Sense deployment at any moment. If you have a multi-node installation, object bundles are installed on the central node.



*You can see which extensions are installed in your deployment by checking the **Extensions** section in the Qlik Management Console (QMC).*

Do the following:

1. In **Windows Control Panel**, open **Programs and Features**.
2. In the list of programs, double-click the object bundle that you want to modify.



3. The Object Bundle Setup Wizard opens. Click **Next**.
4. Select **Change**.
5. On the **Custom setup** screen, click on the bundle icon to select how to modify the bundle installation:
 - If the bundle is installed, select **Entire feature will be unavailable** to uninstall it.
 - If the bundle is not installed, select **Entire feature will be installed on local hard drive** to install it.

Then, click **Next**.

6. Click **Change**.
Once the modification is complete, you are required to manually restart the Qlik Sense Repository Service.
7. Click **Finish** to close the Object Bundle Setup Wizard.
8. Manually restart the Qlik Sense Repository Service to apply the changes.

You can verify that the changes have been correctly applied by checking the **Extensions** section in the QMC.

Adding a server as a trusted origin

If you are creating On-Demand reports in Qlik Sense or a QlikView Web Server hosted QlikView AccessPoint, you must add these servers as a trusted origin in Qlik NPrinting web console.

For each server path used to access the Qlik NPrinting web console, you must add a trusted origin. For example, if your server can be accessed through a local domain, a shortened host, or fully qualified URLs, a trusted origin must be added for each of those server paths.

How you enter the server paths in the **Address** field of the trusted origin depends on your security configuration.

Configuration 1: SSL security not setup with Qlik NPrinting web console

A trusted origin is required for each server path used to log onto the Qlik NPrinting web console. For example, if your server is *qlikserver1* and has an IP address of 192.168.0.101, you would add the following trusted origins:

- *http://qlikserver1*
- *http://qlikserver1.domain.local*

- <http://192.168.0.101>

Configuration 2: SSL security setup with Qlik NPrinting web console and third-party

A trusted origin is required for each server path used to log onto the Qlik NPrinting web console. URLs must use HTTPS. For example, if your server is *qlikserver1* and has an IP address of 192.168.0.101, you would add the following trusted origins:

- <https://qlikserver1>
- <https://qlikserver1.domain.local>
- <https://192.168.0.101>

Adding a trusted origin

Do the following:

1. In Qlik NPrinting, click **Admin > Settings**.
2. Click **On-Demand Settings**.
3. Click **Add trusted origin**.
4. After **Name**, enter a name for the trusted origin.
5. After **Description**, enter a description for the trusted origin.
6. After **Address**, enter the URL for the QlikView Server.
The URL must be the base URL (scheme and host) only. For example, <http://qlikserver1> rather than <http://qlikserver1/qlikview/index.htm>.
7. Click **Create**.

Configuring On-Demand settings on Qlik NPrinting Server

You must configure these settings on your Qlik NPrinting Server:

- You must have a connection to a QlikView Server or a Qlik Sense server.
QlikView Server and QlikView Cluster connections are supported with On-Demand. Local connections are not supported. See: [Creating connections](#).
- If you are connecting to a QlikView Server or Cluster with a QlikView Server Extranet License, the following options must be enabled when configuring the connection:
 - Connection requires authentication
 - Apply user section access for reports
- You must enable your Qlik NPrinting reports for On-Demand.
- You can limit On-Demand report creation through security roles. See: [Role based security](#).



On-Demand is compatible with Chrome, Firefox, and Explorer. It will not run in Microsoft Edge, because the certificates will not download.

Enabling Qlik NPrinting reports for On-Demand creation

Qlik NPrinting reports must be manually enabled for On-Demand creation. Reports must be assigned to the QlikView document or Qlik Sense app with which you want to generate On-Demand reports. On-Demand creation can be enabled during the creation of a new report.

Do the following:

1. In Qlik NPrinting web console, click **Reports**.
2. Open a report or create a new report.
3. Select an app from the **App** drop-down list is connected to the QlikView document or Qlik Sense app for which you want to generate a report.



On-Demand requests only work with reports based on a single connection. MultiDoc reports are not supported.

4. Select the **Enable On-Demand** check box.
5. Click **Save**.

The report template is now available.

Accepting the Qlik NPrinting web console certificate

The default installation certificate for Qlik NPrinting web console may be registered as invalid by your browser. If your browser registers the Qlik NPrinting web console certificate as invalid, you must accept the certificate before you can use On-Demand in QlikView AccessPoint or Qlik Sense.



If you switch to a different browser or clear your browser's cache, you will have to re-accept the Qlik NPrinting certificate. On-Demand works in Chrome, Firefox, and Explorer. It does not run in Microsoft Edge, because the certificates will not download.

Do the following:

1. Open the Qlik NPrinting web console.
2. Depending on your browser, do one of the following:
 - Choose to continue to the Qlik NPrinting web console.
 - Add a security exception and continue to the Qlik NPrinting web console.
3. Log into your Qlik NPrinting web console.

You can now use Qlik NPrintingOn-Demand using this browser.

Creating an On-Demand reporting button

You can create an On-Demand reporting button on the sheet you are editing.

Do the following:

1. From the assets panel, under **Custom objects** > **Dashboard bundle** drag **On-demand reporting** to the sheet.
2. In the property panel, under **NPrinting Connection** > **NPrinting server URL**, set the server connection. The URL must end with /.
For example: <https://<server name>:4993/>
3. Under **NPrinting App**, select the Qlik NPrinting app that contains the report template.

4. Under **NPrinting connection**, select the Qlik NPrinting connection.
As the default, the list only shows Qlik NPrinting connections to the Qlik Sense app you are working in. You can use the **App/Connection filter** toggle to display connections not associated with the current Qlik Sense app.
5. Under **Report Configuration > Choose Report**, select the report you want to connect to the button.
6. In **Default Export Format**, select a default export format.
7. Under **Apearances > Button Label**, set the button label. For example: *Generate report*.

You have now created an On-Demand reporting button.

Generating a report

You can generate a report that is filtered according to the current selections in the Qlik Sense app. You need to be in analysis mode.



You can only filter on fields that are included in the Qlik NPrinting report template, or the report will fail. It is not possible to filter on selections using an expression.

Do the following:

1. Click on the button that you created.
The **Export** dialog opens.
2. There are two buttons to choose from:
 - a. **Quick Report**: generates a single report with the default output format.
 - b. **New Report**: choose from a list of reports that have been made available to you. You can also choose the output format.
3. When the report is generated a download button is activated. Click the **Download** button to download the report.

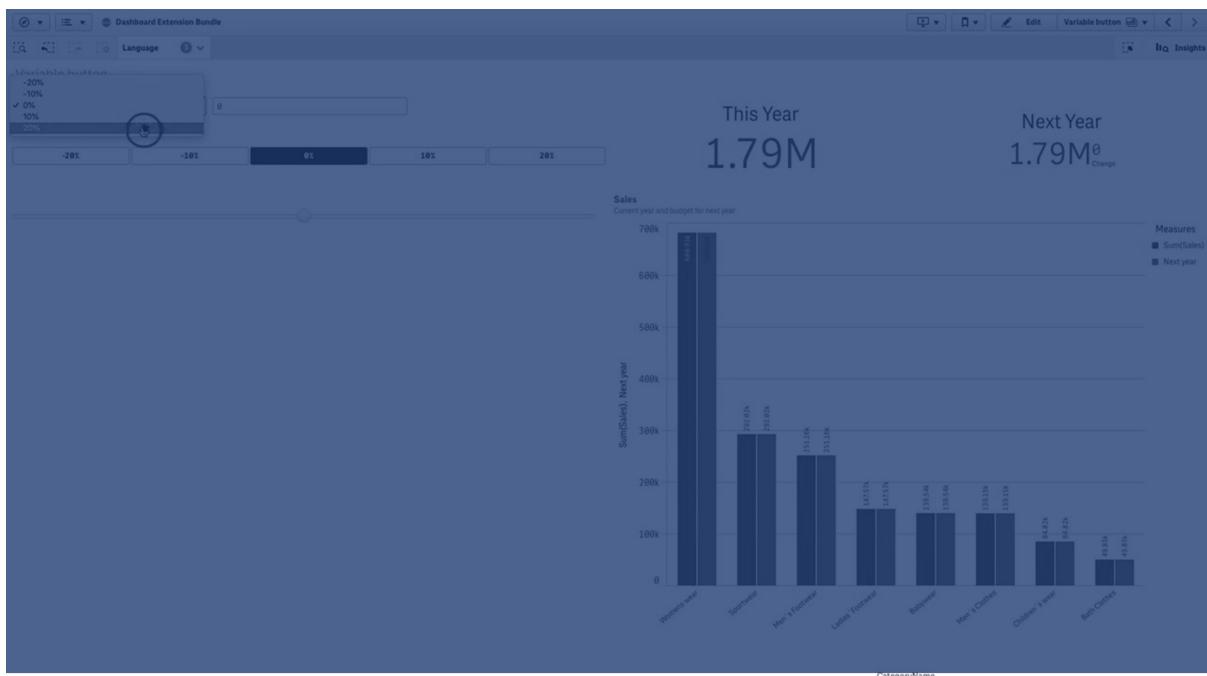
Limitations

For information about general limitations, see *Limitations (page 315)*.

- If the Qlik NPrinting report template contains a filter, it is not possible to make conflicting selections in the Qlik Sense app, or the report will fail.
- When you use the On-Demand reporting control in a mashup, you need to use Windows authentication. Header authentication is not supported.
Additionally, if the mashup contains multiple apps, you can only use the app where on-demand originates from to filter on-demand reports.
-

Variable input control

You can use the variable input control to set the value of a variable. It is included in Dashboard bundle.



When to use it

If you have visualizations with expressions that contain a variable, you can use the variable input control to let the user control the variable value.

Creating a variable input control

You can create a variable input control on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects** > **Dashboard bundle** and drag a **Variable input** object to the sheet.
2. Select the variable to use in **Name** under **Appearance** > **Variable** in the property panel.
3. Select how you want to input data in **Show as**:
 - **Buttons** lets you add a number of buttons with one button for each defined variable value. You can select how to display the buttons in **Display**, select **Row** to show them in a horizontal row or **Column** to show them in a vertical column. You can define the buttons in two different ways, fixed (**Fixed**) or (dynamic) **Dynamic** with the **Fixed or dynamic values** setting under **Values**. If you select to use fixed values, you need to add each button with **Add Alternative** and define a value (**Value**) and a label (**Label**) for each button. If you use dynamic values, you define the buttons with a string in **Dynamic values**. Use | to separate buttons, and ~ to separate value from label. For example, 'Germany~GER|France~FRA' will create two buttons labelled GER and FRA. The first will change the variable value to Germany, and the second changes the value to France. You do not need to specify labels if you want to use values as labels.
 - **Drop down** adds a drop down with one item for each defined variable value.

You can define the items in two different ways, fixed (**Fixed**) or (dynamic) **Dynamic** with the **Fixed or dynamic values** setting under **Values**.

If you select to use fixed values, you need to add each item with **Add Alternative** and define a value (**Value**) and a label (**Label**) for each item.

If you use dynamic values, you define the items with a string in **Dynamic values**. Use | to separate items, and ~ to separate value from label. For example, 'Germany~GER|France~FRA' will create two items labelled GER and FRA. The first will change the variable value to Germany, and the second changes the value to France. You do not need to specify labels if you want to use values as labels.

- **Input box** provides a simple input box that will update the variable value.
- **Slider** creates a slider that updates the variable value. You define the minimum setting with **Min** and the maximum setting with **Max** under **Values**. You can also set the step to use with **Step**. If you select **Slider label**, the selected value is displayed when you drag the slider. The variable value is updated when you stop dragging the slider, but you can select **Update on drag** if you want the variable value to be updated while you drag. This can be useful when you want visualizations based on the variable to update dynamically when dragging the slider. Use this option with caution, as the constant redrawing of charts can be annoying.

Limitations

For information about general limitations, see *Limitations (page 315)*.

- The slider label can only display numeric values. This means, if the value is a date the numeric value of the date is displayed. If the value is a percentage value, the percentage character is not displayed.

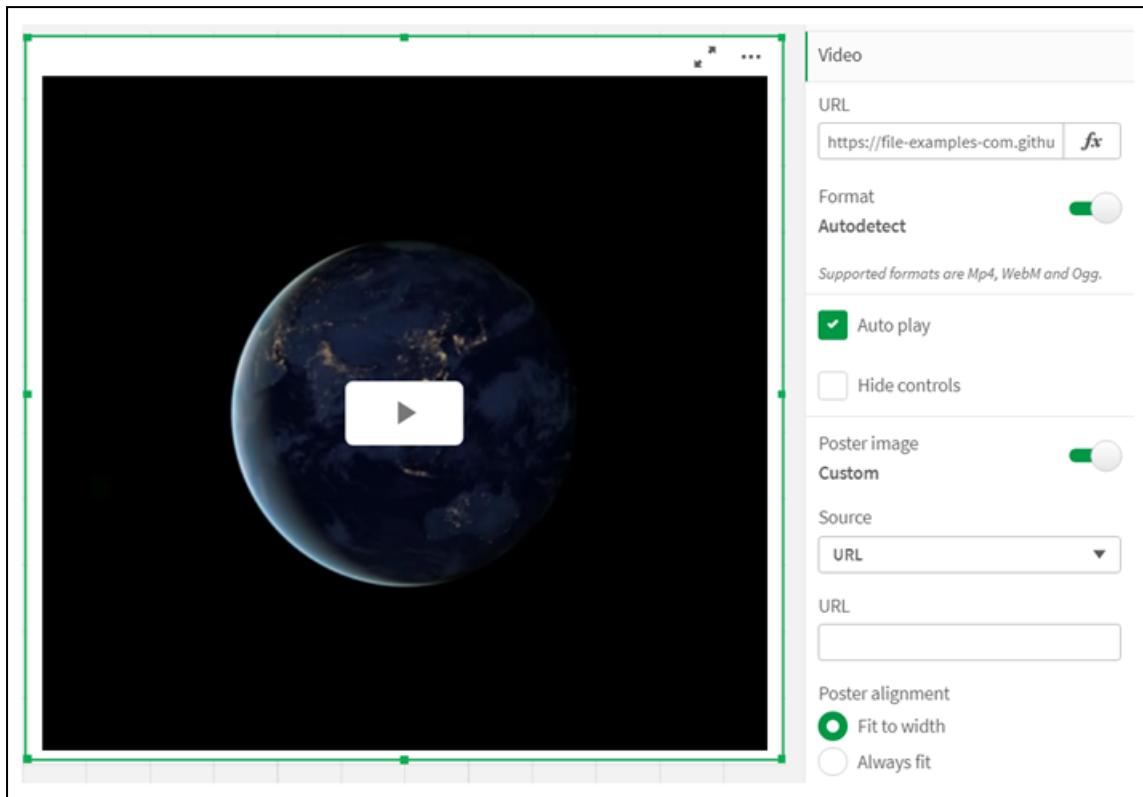
Video player

The video player lets you play videos in different formats from external sources. It is included in the Dashboard bundle.

The following video formats are supported.

- MP4
- OGG
- WEBM
- YouTube

Video player



When to use it

The video player is useful when you want to include a video in your application. For example, you want to show a demo or a training video, add a tutorial, or present corporate news.

Adding videos

You can add video player object to a sheet that you are editing. Add or paste a video link into the video object. The video is streamed from a source; it is not uploaded to Qlik Sense. By default, the video format of the link will be auto-detected, the video will not auto-play, and video controls will be available to app consumers.

Do the following:

1. In the assets panel, open **Custom objects > Dashboard bundle** and drag a **Video player** object to the sheet, or click **Add to Sheet**.
2. Add a video as a URL in the video player object field **Add URL**, or in the properties panel under **Video > URL**. You can format the URL field as an expression.
3. **Autodetect** format is default setting and it will adjust accordingly based on the URL. You can also turn off **Autodetect**, and then select the correct supported video format.

Once you have added the video, you may want to adjust its options.

Setting video options

Video options are set in the **Properties** pane.

Click **Auto play** if you want the video to automatically start when viewed in analyzer mode. **Auto play** is turned off by default. When **Auto play** is selected, the video starts but it is muted. The app user has to select **Unmute** to turn the sound on.

Click **Hide Controls** to turn off video controls in analyzer mode. By default, video controls are visible.

You can add a **Poster image**. By default, the poster image is turned off. For **Source**, select an external URL that directs to an image, or add an image from your media library.

If a poster image is selected, the selected image is displayed before the video starts playing. **Poster alignment** specifies how the poster image will align inside the video player.

You cannot play the video in edit mode. The video restarts when you switch between analyzer mode and edit mode.

In analyzer mode, you start the video with the play button. You can pause, switch between mute and unmute, and view the video in full screen. You can also share the video player object, and **Take snapshot**.

In edit mode, the right-click context menu provides options for Qlik Sense. In analyzer mode, the context menu provides options for YouTube.

The app developer (edit mode) or app consumer (analyzer mode) must allow YouTube videos to play in Qlik Sense.

Limitations

The video player has the following limitations:

- Safari does not support OGG and WEBM formats.
- Qlik Sense Desktop cannot play MP4 format due to limitations in Chromium.
- Qlik Sense Mobile Client Managed on iOS can only play MP4 compressed in H.264 or MPEG-4 format, but not other MP4, OGG and WEBM formats due to limitations in iOS.
- Qlik Sense Mobile Client Managed offline is not supported for video playing.
- iPhone adds native video controls. It does not use video player controls, therefore **Hide Controls** will not turn off controls.
- Android does not support OGG format.
- When exporting or printing a sheet that includes a video player object, the video player object will not be included.
- Qlik NPrinting does not support the video player object.

Visualization bundle

Visualizations are an important means of conveying information from massive data. The Visualization bundle is a set of charts that can be used to enhance and increase your Qlik Sense app's charting capacity. The charts are optional. You do not have to install or enable them to use Qlik Sense.

Enabling Visualization bundle

You can install the Visualization bundle when you install Qlik Sense. If you need to adjust your installation, see: [Modifying an object bundles installation](#).

Visualization bundle charts

Visualization bundle charts are in the asset panel under **Custom objects**.

The following charts are included:

- *Bar & area chart* ([page 334](#))
Create bar charts and area charts and enhance them with transitions and connectors.
- *Bullet chart* ([page 334](#))
A bullet chart is a gauge that can also show a target marker and a qualitative range to show performance.
- *Funnel chart* ([page 338](#))
A funnel chart is a visual representation of the connected stages of a linear process.
- *Grid chart* ([page 345](#))
A chart that displays comparative data and with the values represented as colors.
- *Heatmap chart* ([page 348](#))
A chart that displays comparative data and with the values represented as colors.
- *Multi KPI chart* ([page 353](#))
A chart that shows KPI for multiple dimension values to quickly understand and track performance.
- *Network chart* ([page 363](#))
Creates a cluster diagram representing a graphical chart of a computer network.
- *P&L pivot chart* ([page 372](#))
Creates a pivot table that you can style, for example for profit and loss reporting.
- *Radar chart* ([page 377](#))
Creates a two-dimensional chart using radial axes to show the scoring of a measure in one dimension or another.
- *Sankey chart* ([page 382](#))
A flow chart diagram chart visually emphasizing major transfers or flows within defined system boundaries.
- *Straight table* ([page 386](#))
A table that allows app developers to create tables that can be customized by users who do not have edit permissions.
- *Trellis container* ([page 396](#))
Creates a trellis chart based on a master visualization.
- *Variance waterfall chart* ([page 399](#))
Shows the variance between two measures over the different values of a dimension.
- *Word cloud chart* ([page 402](#))
A cloud chart of words with their size based on measure value.

Limitations

Visualization bundle visualizations have more limitations than built-in visualizations, such as bar charts. The following limitations are valid for all visualizations from Visualization bundle:

- The user interface of the visualization is not localized to the language that Qlik Sense is using.
- Right-to-left reading order is not currently supported.
- Accessibility features are not currently supported.
- It is not possible to edit objects from an object bundle supplied by Qlik with Dev Hub.

The following table shows which additional features are supported, or not supported, for all bundled objects.

- **Printing**
Printing a visualization made in an object .
- **Qlik NPrinting**
There is an *On-Demand reporting control* (page 323). However, visualizations made in some objects cannot be used in Qlik NPrinting reports.
- **Download**
Download as image, PDF, or Excel.
- **Storytelling**
Using a snapshot of a visualization created with an object in a story.
- **Alternate states**
Making different selections on the same dimension, and comparing the selections in a single visualization or in two or more visualizations side by side.



Exported radar charts will not include the chart legend.

Visualization bundle capability support

Chart	Printing	Qlik NPrinting	Download	Storytelling	Alternate states
Bar & area	Yes	Yes	Yes	Yes	Yes
Bullet	Yes	Yes	Yes	Yes	Yes
Funnel	Yes	Yes	Yes	Yes	Yes
Grid chart	Yes	Yes	Yes	Yes	Yes
Heatmap	Yes	Yes	Yes	Yes	Yes
Multi KPI	No	No	Yes	Yes	Yes
Network	Yes	No	Yes	Yes	Yes
Org chart	Yes	Yes	Yes	Yes	Yes
P&L pivot	Yes	No	Yes	Yes	Yes
Radar	Yes	Yes	Yes	Yes	Yes
Sankey	Yes	Yes	Yes	Yes	Yes

Chart	Printing	Qlik NPrinting	Download	Storytelling	Alternate states
Straight table	Yes	Yes	Yes	Yes	Yes
Trellis container	No	No	No	No	No
Variance waterfall	Yes	Yes	Yes	Yes	Yes
Wordcloud	Yes	No	Yes	Yes	Yes

Bar & area chart

New bar & area charts are no longer supported as of May 2021. If you need to configure existing instances of this chart, please refer to the last version of help before support ended: . Existing bar & area charts will continue to function, but new ones cannot be added. It is recommended that you replace existing bar & area charts with bar, line, and combo charts. Many features of the bar & area chart have been added to these charts.



Bar chart (page 142)

Line chart (page 184)

Combo chart (page 162)

Bullet chart

The bullet chart (**Bullet chart**) displays a gauge with extended options. Bullet charts can be used to visualize and compare performance of a measure to a target value and to a qualitative scale, such as poor, average, and good. The bullet chart is included in the Visualization bundle.



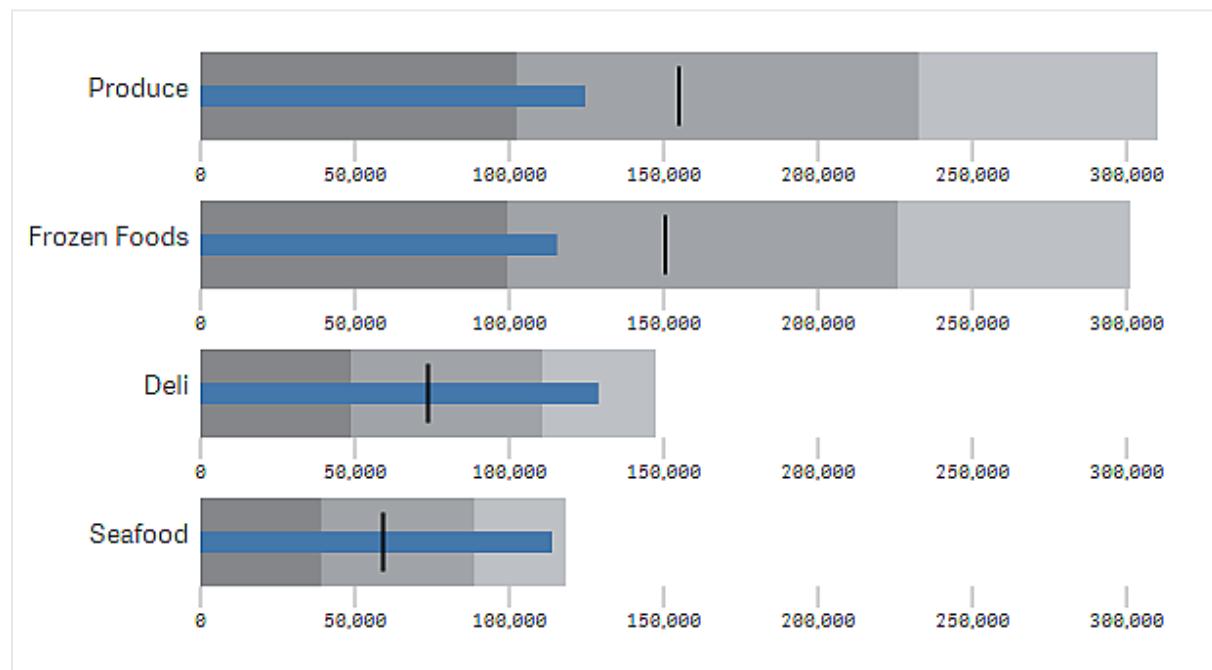
We recommend that you use the [Bullet chart](#) instead of this object. The bullet chart in the Visualization bundle will be deprecated in a later release. Existing instances of this chart will still function, but new ones cannot be added.

A bullet chart can contain from one to three measures:

- The first measure (**Measure**) is the actual value, represented by the bar.
- The second measure (**Marker**) defines a target value, which is represented by a vertical marker line.
- The third measure (**Range**) defines a qualitative range displayed behind the bar. This consists of three ranges.

You can also add a dimension. This will show one gauge for every dimension value. If you do not define a dimension, the chart will show a single gauge.

A bullet chart with one gauge for each value of the dimension (product group).





Bullet chart

When to use it

Bullet charts let you compare and measure performance with more enriched information than a common gauge. This is helpful when comparing performance according to a target and a simple performance rating. For example: you can show how sales relate to a target value, and in context of poor, good, and stretched performance.

Creating a bullet chart

You can create a bullet chart on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Bullet chart** object to the sheet.
2. Click the **Add measure** button to select the value measure of the chart.
Once the first measure is selected the bullet chart is displayed.
3. To add a target value, click **Add** under **Measures**. You can define a fixed value or use a measure with target values.
4. To add performance ranges, click **Add** under **Measures**. This will define the maximum performance range value. You can set the internal range limits later.
5. To show one gauge for every dimension value, click **Add** under **Dimensions**.

The bullet chart is now displayed with the dimensions and measures you selected.

Changing the appearance of the chart

You can customize the appearance of your bullet chart.

Changing the color scheme

You can change color scheme of the value bar, the marker and the range.

Do the following:

- Set value bar color in **Appearance > Measure bar > Change bar color** in the property panel.
- Set marker color in **Appearance > Marker > Change marker color** in the property panel.
- Set range color in **Appearance > Range > Change range color** in the property panel. The color you select is used for the highest range. The two lower ranges are displayed in darker shades of that color.



It's a good idea to select a light range color to avoid poor contrast between the ranges. You should also make sure that the range color is less visually dominant than the value bar.

Changing the range limits

You can also customize the limits of the ranges under **Appearance > Range** in the properties panel. The full range is defined by the value of the third measure of the chart.

- **Set middle range (%)** defines the upper limit of the middle range.
- **Set lower range (%)** defines the upper limit of the lower range.

Changing the scale of the axis

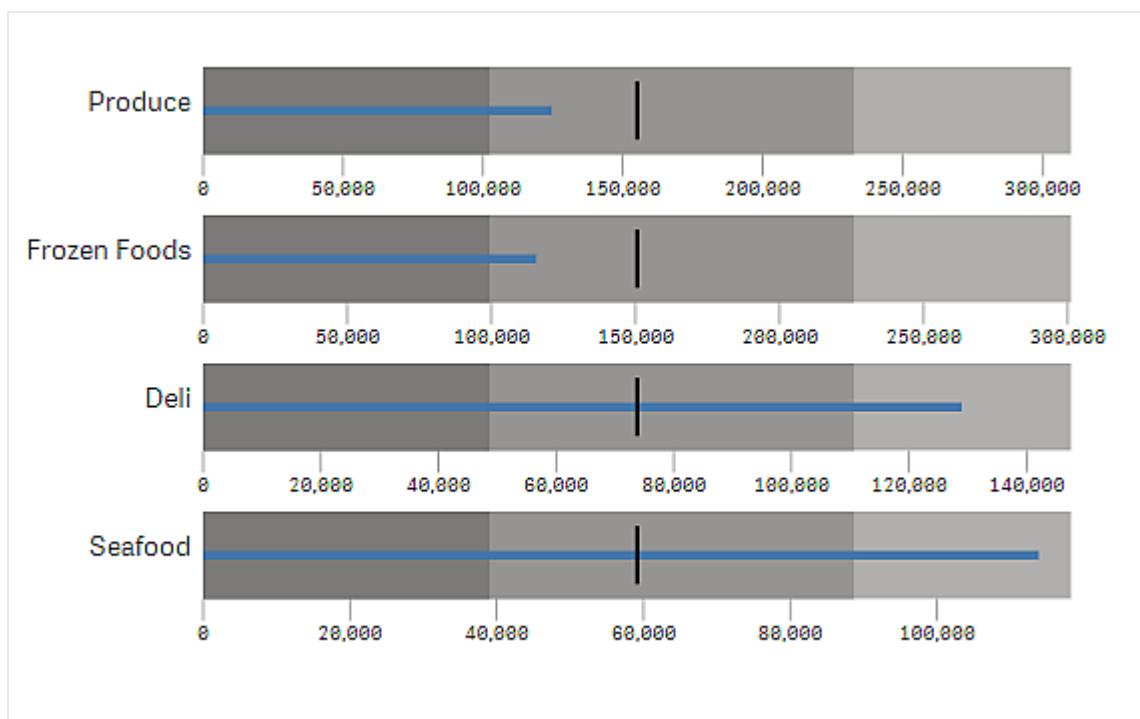
If you use a dimension to show several gauges, you can select how to show the scale of the axis with **Axis > Consistent for all axis dimension values**.

- If you want each dimension gauge to use the same scale, enable this option. If the range measure depends on the dimension value, the range bars will have different lengths. This is useful when you want to be able to compare the actual values.
- If you want each range bar to be equally long, disable this option.

Example:

In this example, **Consistent for all axis dimension values** is disabled. This makes it easier to compare the relative performance of each product group.

*A bullet chart with **Consistent for all axis dimension values** disabled.*



Limitations

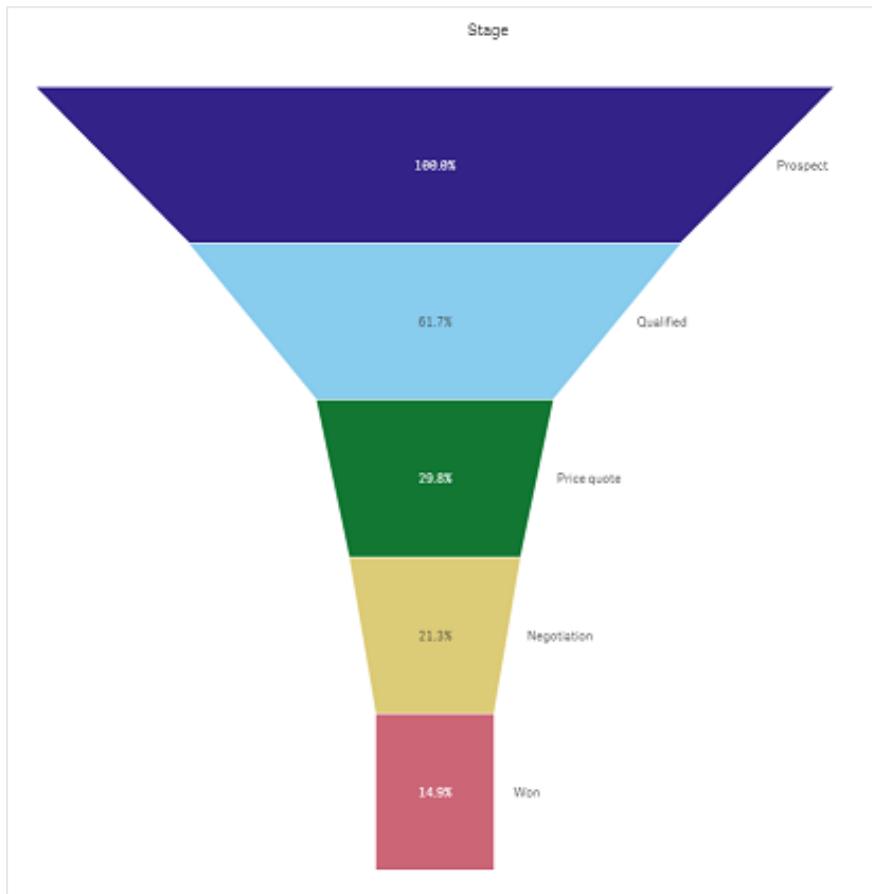
For information about general limitations, see *Limitations* (page 333).

- You can not make selections in a bullet chart.
- If you want to change the number format, you need to set the same format in **Number formatting** for all three measures.

Funnel chart

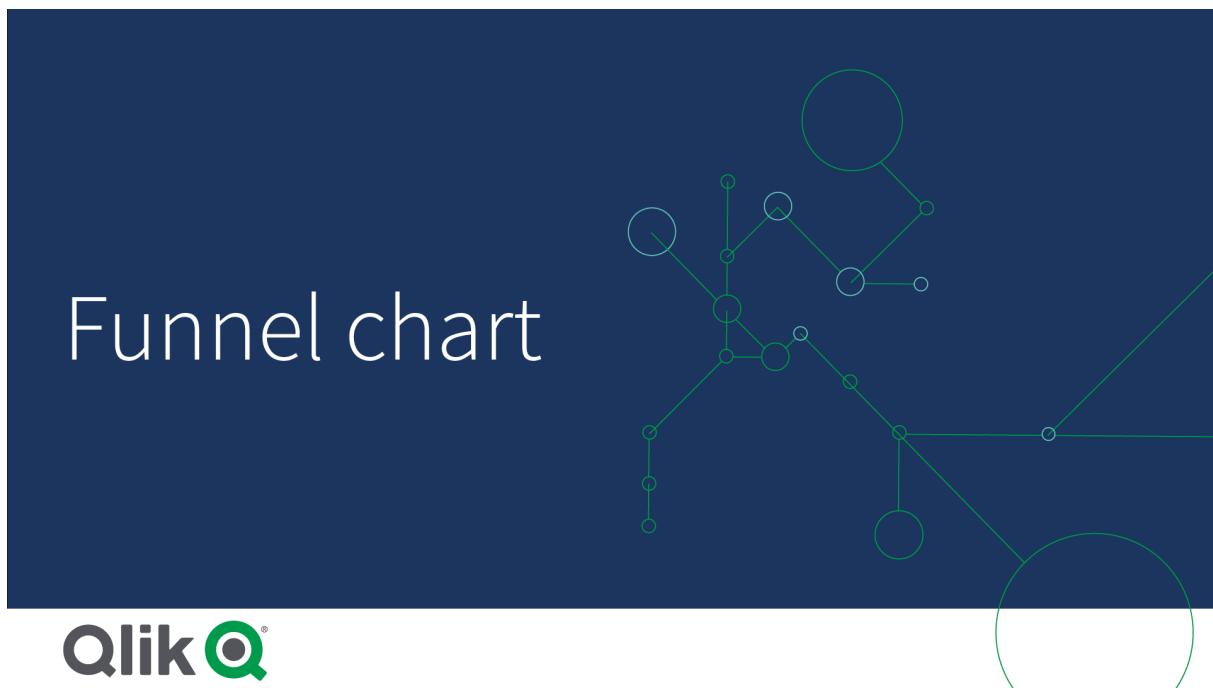
The Funnel chart (**Funnel chart**) lets you add a sequential chart showing the connected stages of a process. Each stage will decrease and should contain a subset of the previous stage. The decrease is gradual, giving the chart an ever narrower funnel.

A funnel chart displaying the conversion rates of prospects to customers in a sales process.



Funnel charts show values across multiple stages in a process. They can represent anything that is decreasing in size. The funnel chart shows a process that starts at 100% and ends with a lower percentage. Each chart segment represents the value of a specific item and can influence the size of other segments. The Funnel chart is included in the Visualization bundle.

- The chart requires one dimension and one measure.
- Unlike bar charts, funnel chart segments are centered to create a funnel shape.
- A chart with increasing stages instead of decreasing is a pyramid graph.



When to use it

The funnel chart is useful to illustrate the stages of a process and the overall decrease of each step, for example:

- Represent a sales process showing the amount of potential revenue for each stage.
- Illustrate the number of sales prospects at each stage in a sales pipeline, i.e. the process from prospective customer to made purchase.
- Identify potential problem areas and bottlenecks of a sales process.
- Communicate a sales process to new team members and vendors.
- Illustrate website visitor trends – from visitor homepage hits to other areas, for example downloads, etc.
- Show order fulfillment with initiated orders at the top, followed by for example, orders in delivery, delivered, canceled and returned.
- Showing the flow of information from top secret to unclassified.
- Representing knowledge areas from general knowledge to expert knowledge.

Creating a funnel chart

You can create a funnel chart on the sheet you are editing.

- The dimension decides how it should be grouped in segments. By default, dimension values are displayed in descending order by measure value.
- The measure is the value that decides the size of each segment.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Grid chart** object to the sheet.
2. Click the top **Add dimension** button and select the target dimension (usually target market) of the chart.
3. Click the **Add measure** button to select the measure (what is to be measured) of the chart. Once dimensions (dimension label) and measure (value label) have been selected the funnel chart displays automatically (in color) in the chart field.
4. Click **Done** to revert to the main display field.
5. Click applicable region under **Region**. The chart field will update displaying chosen parameters and details.

Dimension, measure and region details will continue to display in the chart field even when reverting to **Edit** mode.

Changing the appearance of the chart

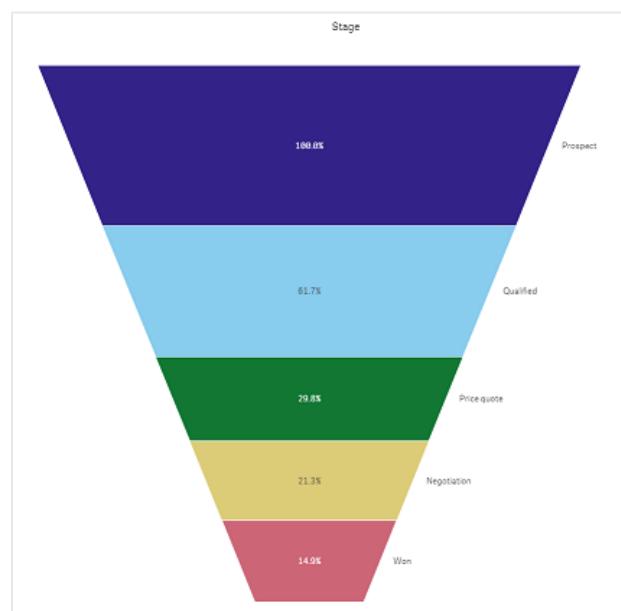
You can customize your chart with one or more features.

Funnel mode

You can set the shape of the funnel with **Appearance > Presentation > Funnel Mode**.

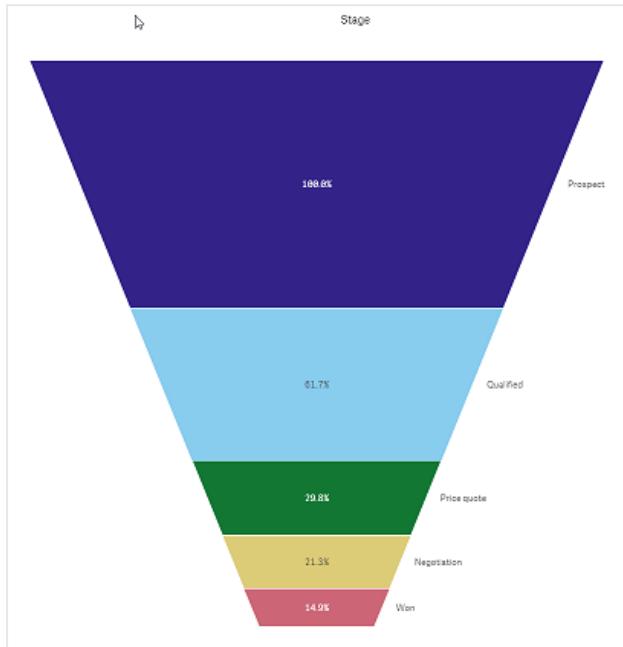
Area

The area of each item is proportionate to the measure. Only the height of the individual segment is affected - not the overall chart or contents.



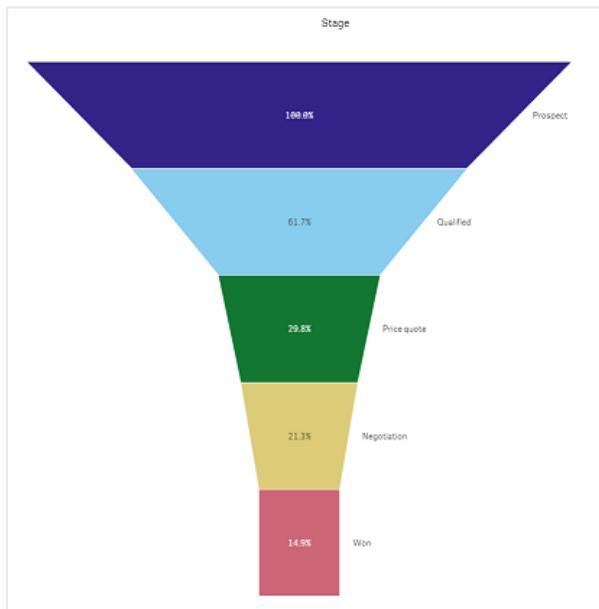
Height

The height of each item is proportionate to the measure. Only the height of the individual segment is affected - not the overall chart or contents.



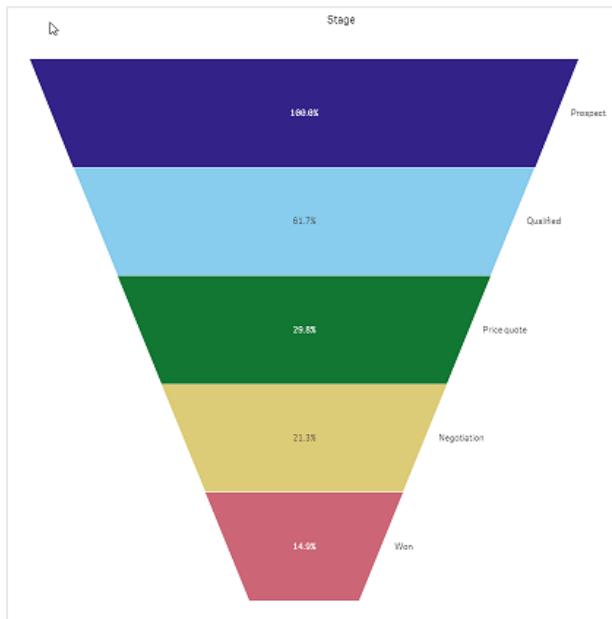
Width

The width of the upper edge is proportionate to the maximum value of the measure. The top segment is always 100% and the following segments are relative in size to the first. The lowest segment is rectangular. This affects the shape of the funnel and each segment has its individual slope.



Ordering

The measure only orders the segments with largest value at the top. The ordering is fixed so the shape of the funnel is not affected.

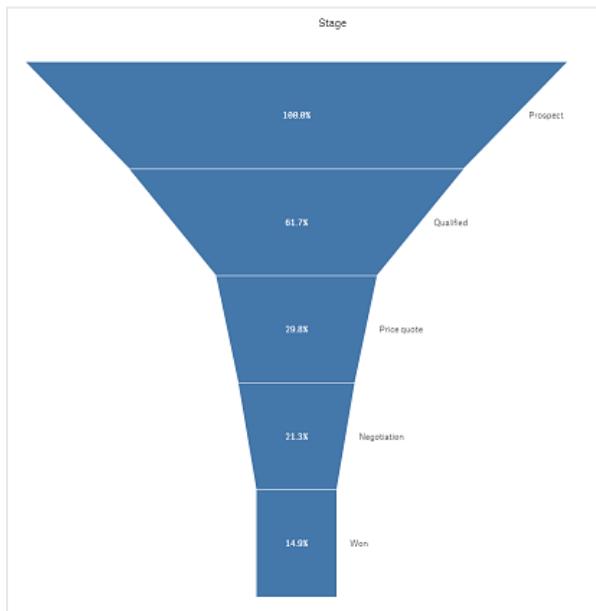


Colors

You can choose how to color the funnel chart by changing the **Appearance > Colors** setting from **Auto** to **Custom**.

Single color

You can use a single color for the entire funnel chart by selecting **Single color** in **Colors**.



Color by dimension

You can also color your entire funnel chart by dimension by selecting **By dimension** in **Colors**. There are two gradient color schemes available:

- **12 colors**
- **100 colors**

You can also select to keep persistent colors with **Persistent colors**.

Color by measure

You can color your funnel chart by measure by selecting **By measure** in **Colors**. The measure of the chart is selected as default, but you can change it under **Select measure**. This measure must be unlinked before a new can be selected. You can also change the measure using an expression in the Expression editor (**fx**).

You can choose between four color schemes. It is also possible to reverse the colors.

- Sequential gradient
- Sequential classes
- Diverging gradient
- Diverging classes

You can also set a color range for your chart based on your measure by setting **Range** to **Custom**. Set the range in **Range > Min** and **Range > Max**. You can use a number or an expression that evaluates as a number.

Color by expression

You can color your funnel chart by measure by selecting **By expression** in **Colors**. You can color it in two ways.

- The expression evaluates as a valid CSS color to color the chart. **The expression is a color code** should be checked.
- The chart is colored by the expression value according to a color scheme. **The expression is a color code** should be unchecked.

You can also set the range of the colors.

Sorting

Funnel chart elements are automatically sorted from largest to smallest. You can change the sort order in the properties pane. Go to **Sorting**, and drag your dimensions and measurements into the desired order.

Styling and formatting

The position and order of labels for dimension and measure values can be displayed in different ways. You can for example hide the dimension value or display measures as values or as a percentage.

Dimension label

You can choose if you want to show the dimension label or not with **Appearance > Presentation > Dimension label**

Value labels

You can choose how to show value labels for each segment by setting **Appearance > Presentation > Value labels** to **Custom**.

- **None** hides the value labels.
- **Share** displays the values as percentage.
- **Values** displays the actual measure values.

Limitations

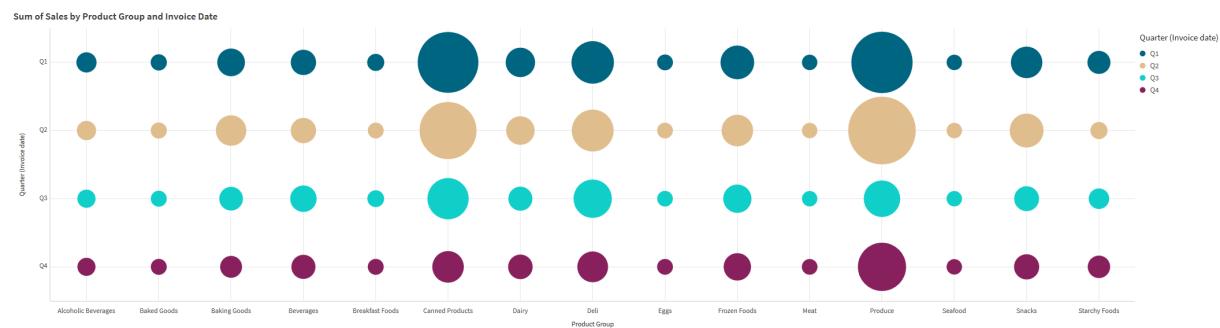
For information about general limitations, see *Limitations* (page 333).

Grid chart

The grid chart (**Grid chart**) uses symbols of varying size sorted in a grid. The grid chart is included in the Visualization bundle.

Grid charts must have two dimensions and one measure. The dimension values are the x and y axes. The measure is the metric that determines the size of the symbol in each crossing.

Grid chart with circles. Circle color is set by dimension (Quarter) and circle size is based on measure (Sum of Sales)



When to use it

A grid chart is useful when you need to quickly find measure values that stand out from other values. For example: you want to see which products sell well in some months, but poorly in others.

Grid charts do not work well when the measure has:

- Many duplicate values.
- Values that are numerically close to each other.

Creating a grid chart

You can create a grid chart on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Grid chart** object to the sheet.

2. Click the **Add Dimension** button to select the x axis value of the chart.
3. Click the second **Add Dimension** button to select the y axis value of the chart.
4. Click the **Add Measure** button to select the metric being weighed.

The grid chart is now displayed with the dimensions and measures you selected.

Sorting

Grid chart elements are automatically sorted alphabetically or numerically from smallest to largest on the x axis. You can change the sort order of either axes in the property panel.

Do the following:

1. In the property panel, go to **Appearance>Sorting**.
2. Click on a dimension.
3. Toggle **Sorting** from **Auto** to **Custom**.
4. You can choose any of these options:
 - **Sort by expression: Ascending** or **Descending**.
 - **Sort numerically: Ascending** or **Descending**.
 - **Sort alphabetically: Ascending** or **Descending**.

Changing the appearance of the chart

You can customize the appearance of your grid chart.

Grid chart with stars. Star color and size is set by measure (Gross Sales).



Customizing the symbols

You can change the shape and size of the symbols from the **Appearance>Presentation** tab in the property panel:

- Use the **Symbol size** slider to change size.
- Use the **Symbol** dropdown to change symbol type.

Placing labels on symbols

You can place the individual values of symbols directly onto the chart. From the **Appearance > Presentation** tab, toggle the **Labels** from **Off** to **Auto**.

Using a grid chart as a heat map

You can change the layout of a grid chart to a heat map from the **Appearance > Presentation** tab in the property panel. From the **Layout** dropdown menu, choose:

- **Standard**: The chart will show individual symbols.
- **Heatmap**: The chart will display data in a grid with a legend ordered by color.

For a visual demo about using a grid chart as a heatmap chart, see [Creating a heatmap using a grid chart](#).

Changing the color scheme

You can change the color scheme of the symbols from the **Colors and legend** tab. Set **Colors** to **Custom**. From the dropdown menu, choose:

- **Single color**: You can choose any color.
- **By dimension**: Under **Color scheme**, you can change how many different colors are used.
- **By measure**: Under **Color scheme**, you can alter the color gradient. Click the **Reverse colors** check box to flip the gradient order. The colors used are based on your app theme.
- **By expression**: You can have a color code in the expression. Or, deselect the **The expression is a color code** check box, and you can alter the color gradient. Click the **Reverse colors** check box to flip the gradient order. The colors used are based on your app theme.

Changing the range limits

You can customize the limits of the ranges if you color by measure or expression. The full range is defined by the value of the dimensions. Go to **Appearance > Colors and legend > Range** in the properties panel:

- **Min**: Defines the lower limit of the range.
- **Max**: Defines the upper limit of the range.

Changing the scale of the axes

You can customize the presentation of the x and y axes. Go to **Appearance > X-axis: [dimension]** or **Appearance > Y-axis: [dimension]** in the properties panel. You can set:

- **Labels and title**: Select what labels and title to display.
- **Label orientation**: Select how to display the labels. The following options are available:
 - **Auto**: Automatically selects one of the other options depending on the space available on the chart.
 - **Horizontal**: Labels are arranged in a single horizontal line.
 - **Tilted**: Labels are stacked horizontally at an angle.
 - **Layered**: Labels are staggered across two horizontal lines.

To view examples of label orientation, see *X-axis and Y-axis (page 458)*.

- **Position**: Select where to display the dimension axis.
- **Number of axis values**:

- **Auto:** The number of visible bars is determined by the number of dimensions and measures used.
 - **Max:** The number of visible bars is set to maximum.
 - **Custom:** When custom is selected, you can directly set the upper limit to the number of visible bars using **Maximum number**, or by entering an expression. Click **Expression** to open the Expression editor.
- **Show grid lines:** Select whether to display grid lines.

Limitations

For information about general limitations, see *Limitations* (page 333).

- The **Number of axis values** setting in the property panel has a limit of 55 per axis.
- Sometimes when you set dimension limits on a grid chart, chart labels will incorrectly display more items than requested.

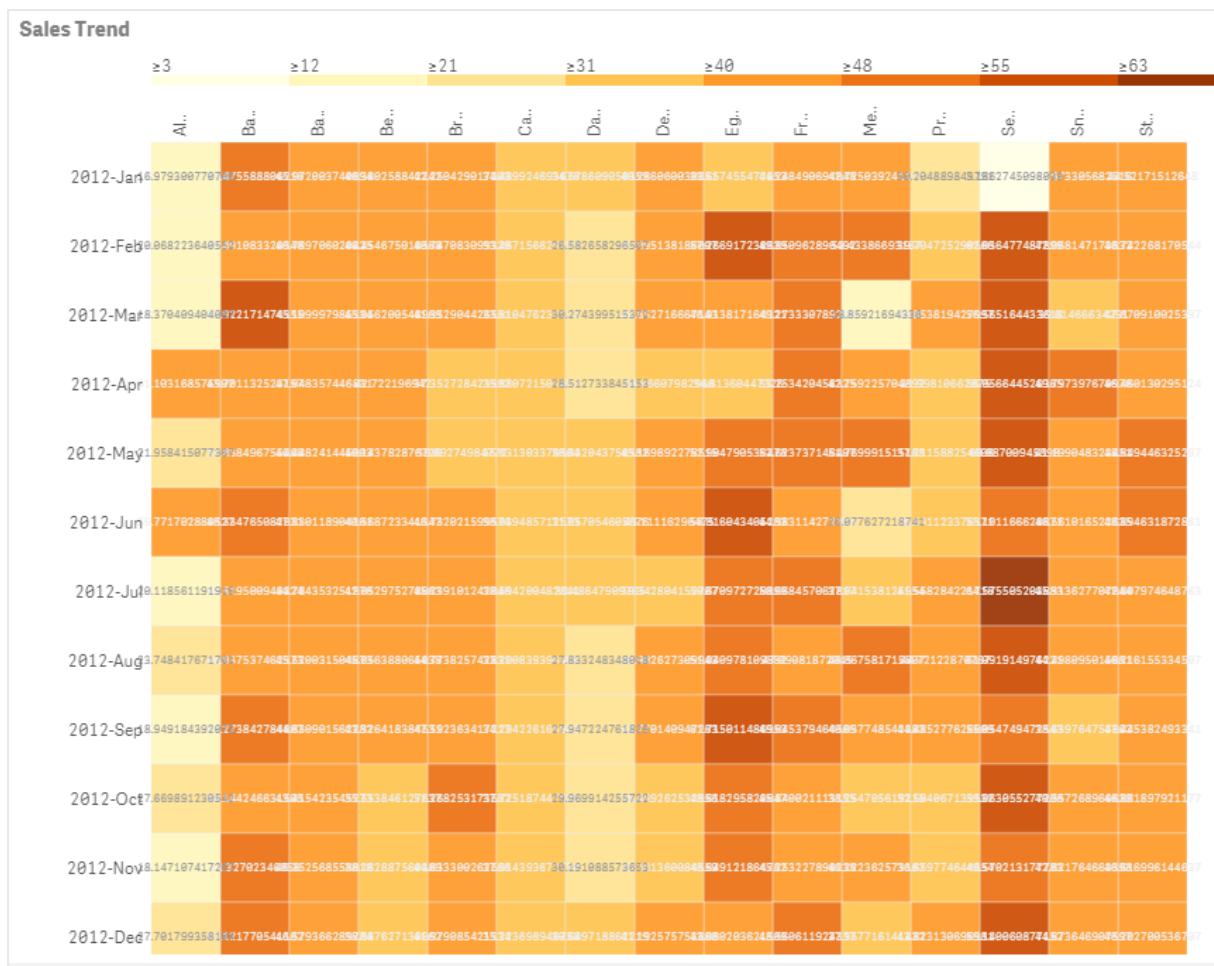
Heatmap chart

A Heatmap chart (**Heatmap chart**) displays comparative data where the values are represented as color patterns in a chart. You can convey information instantly by combining elements from several sources into one. It is included in the Visualization bundle.



As of February 2022, the Heatmap extension will no longer be supported. Apps that use the Heatmap extension at that time will continue to function. However, the extension will no longer be available in the Assets panel.

Heatmap chart with grid layout, data, and labels.



A heatmap can display large amounts of data in their entirety because values are replaced by colors. This condensed color-coded format provides an easy-to-understand overview of data.

Heatmaps require two dimensions and one measure. A second measure is optional. The chart displays in a tabular format with color-coded tiles. The highest and lowest values show in each dimension column. The values in between are shown in a color gradient, centered upon the average.

When to use it

A heatmap chart displays a visual summary of large amounts of comparative data. The information is presented in color patterns, and is communicated almost instantly in a single chart. A heatmap is useful when:

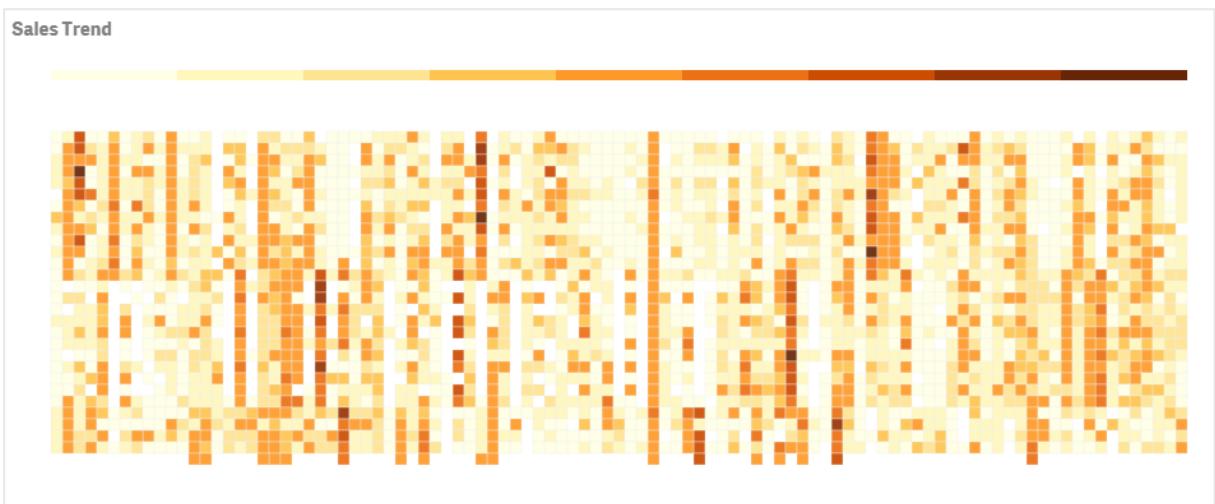
- Comparing performance between companies, markets, or investments.
- Identifying the level of performance between departments of a business.
- Setting investment priorities and highlighting areas of concern.
- Presenting vast statistical data and data sets.
- Measuring user interaction with websites.
- Rating and categorizing places, people, performances, or jobs.

Creating a heatmap chart

You can create a heatmap chart on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Grid chart** object to the sheet.
2. From the assets panel, under **Custom objects > Visualization bundle** drag a **Heatmap chart** object to the sheet.
3. Click the top **Add dimension** button and select dimension.
4. Click the lower **Add dimension** button and select second dimension variable.
5. Click the **Add measure** button to select the measure of the chart.



Changing the appearance of the chart

Using the lasso selection tool

The lasso selection tool lets you make a two-dimensional selection of a specific area you want to take a closer look at, by tracing a border around it.

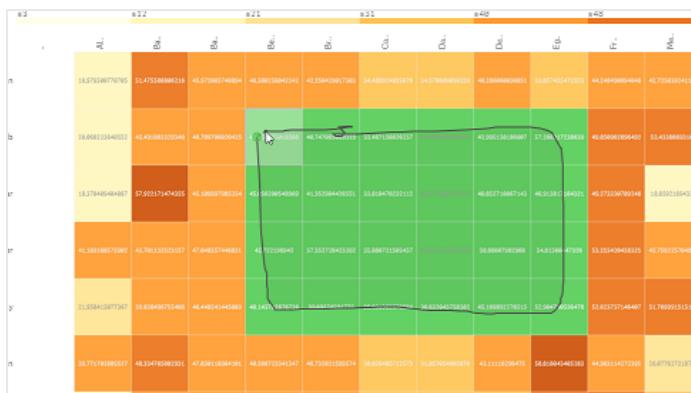
That outline then disappears and you can trace another lasso in the desired area.

Do the following:

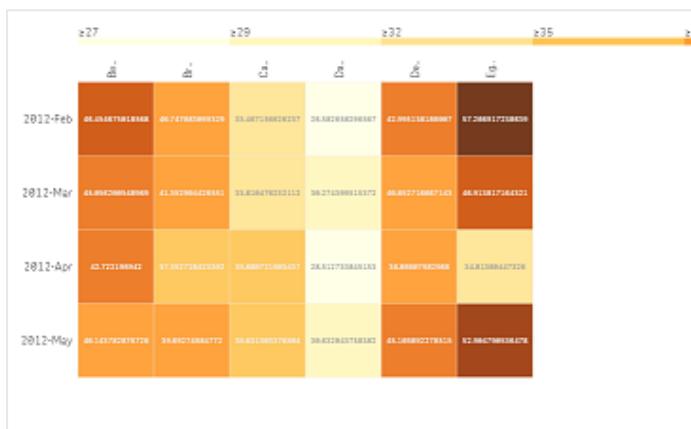
1. Open your heat map in the main display field. If you are working in edit mode, click **Done**.
2. Click the cursor on a tile and, while pressing the mouse button down, outline of the area of the chart you want to take a close look at.
3. End your outline where it began. The chart will automatically zoom in and display only the outlined area.

Examples:

A border is traced around an area using the lasso tool, coloring it green.



Once selection is complete, only the zoomed-in outlined area show in the chart.



Using dimension selection option

You can select a column or row of tiles for either dimension by clicking on the dimension label, or a selection of two dimensions by clicking on a tile. Once selected, only the chosen row, column or tile displays in the chart.

Changing the color scheme

You can change color scheme of your chart.

Do the following:

1. Click **Appearance > Design** in the property panel.
2. Select a color scheme under **Color schema**.



Changing label colors

You can set your own label color under **Appearance > Label color** in the properties panel. Click the label color box and select a color from the gradient color circle. You can also click the easel symbol below the color circle, and select color or enter a color code string in the field next to the easel symbol. The colors should be valid CSScolors.

Toggling the legend

The legend provides a thin gradient line of color description at the top of the chart. You can hide the legend by turning off **Appearance > Design > Legend** in the properties panel.

Adjusting tile opacity

You can adjust tile opacity by moving the slide button of the tile opacity slider under **Appearance > Design > Tile opacity** in the properties panel. Setting opacity to 1 allows the setting to show colors clearly, giving tiles a more individually distinct appearance.

Using a mean in color scale

In a heatmap you can calculate and display the mean of data set. It is used as middle value in a color scale. You can toggle this feature on or off under **Appearance > Options > Use mean in scale** in the properties panel.

Enter a scale value by expression for mean value. This allows the system to choose color range that defines a median color scale. You can also insert a scale string into the expression in the Expression editor (**fx**) under **Appearance > Options > Mean scale value** in the properties panel. When no mean is used, the mean scale value should be set to 0.

Setting a fixed scale

You can set minimum, maximum, and mean values to define a fixed color scale independent from a data set. To do this, move the **Fixed scales** slide button to the right under **Appearance > Options > Fixed scale** in the properties panel. Then enter a minimum value under **Min scale value**, a maximum value under **Max scale value**, and a mean value under **Mean scale value**. You can also insert a scale string into the expression in the Expression editor (**fx**).

Toggling titles

This option hides the name of the heatmap chart. Click **Appearance > General** in the properties panel, and toggle the **Show titles** slide button.

Changing label sizes on chart axes

You can also customize the size of the labels on both the y-axis and the x-axis of the chart. Click **Appearance** in the properties panel, and enter the label size you want under **Y-axis label size** and under **X-axis label size**.

Setting minimum horizontal size

You can set a minimum size of how the chart displays horizontally with **Appearance > Options > Minimum horizontal size** in the properties panel. You can also insert a string into the expression in the Expression editor (**fx**).

Limitations

For information about general limitations, see *Limitations (page 333)*.

- The heatmap does not support sorting by expression.

Multi KPI chart

The Multi KPI chart (**Multi KPI**) is a visualization that allows you to show multiple KPI values for different dimension values. The values can be individually customized using various conditional formatting settings. This enables an easy view and tracking of goals. It is included in the Visualization bundle.

- The chart shows KPIs using measures and one dimension.
- Up to 15 measures and 80 values can display simultaneously.
- All KPI values can be grouped or displayed individually.
- Each value can be independently customized using for example, colors, icons, labels, font sizes, alignments, styles, links to different sheets, etc.
- The chart supports adding graphics, embedding objects into a chart and to display measures infographically.



When to use it

The multi KPI chart is useful when you want to easily view, understand and track the performance of your goals. It is also helpful when you want to customize individual KPI values using conditional formatting. You can link KPIs to separate sheet and insert objects to represent information or data.

Creating a multi KPI chart

You can create a multi KPI chart on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Multi KPI** object to the sheet.
2. Click the **Add measure** button and select the main measure of the KPI chart. The main measure and a KPI value for the chosen measure is displayed.
3. Click **Add** under **Data > Dimensions** in the property panel and select dimension.

When you have selected measure and dimension, a multi KPI chart is displayed.

Example:

A chart with the measure (Margin %) shown for different values of the dimension (City).

Washington	Santander	Hannover	Detroit	Bristol	Miami	Newcastle	Liverpool	Valladolid
Margin (%)								
66	53	53	51	51	51	51	50	50

Adding additional measures

You can add additional measures to your chart in property panel under **Data > Dimensions**. The chart updates to reflect the added measures. Up to 15 measures and 80 values can display simultaneously. The main measure is always the top measure listed for each dimension. When you add more measures, they appear under the initial KPI value in the order they are entered.

Example:

A chart with a two measures (Margin %, Quantity) grouped per dimension (City).

Washington	Santander	Hannover	Detroit	Bristol
Margin (%) 66 Quantity 771	Margin (%) 53 Quantity 1k	Margin (%) 53 Quantity 1k	Margin (%) 51 Quantity 315	Margin (%) 51 Quantity 13k

Customizing the KPIs

You can customize your KPI values and measures with one or more conditional settings. You can add several KPIs together, group them and link to different sheets. You can also configure KPI values independently by differentiating them with one or more conditional settings such as text, color, icons, graphics, etc.

For conditional settings options, see *Changing a measure/label color (page 356)*.

Limitations

For information about general limitations, see *Limitations (page 333)*.

Multi KPI charts cannot be used in Qlik NPrinting reports.

Customizing your KPIs

With the Multi KPI chart (**Multi KPI**) values and measures can be customized using one or more conditional settings. You can add several KPIs together, group them, and link to different sheets. KPI values can also be customized independently by differentiating them using one ore more CSS-properties, such as text color, fonts, sizes, icons, labels, graphics.

KPI values can also be customized independently by differentiating them using one ore more css-properties such as text color, background color, fonts, sizes, icons, labels, graphics.

KPI values using icons, different colors, fonts, Master Visualizations



Coloring a Multi KPI

The color option enables all KPI values (measures and labels) to be differentiated by colors. You can change the colors of text, icons and background color. The colors should be valid css-colors.

You can choose from the following options:

- The color palette with a set of predefined colors.
- The color circle (click the easel symbol in the color palette and choose individually).
- Enter a color code string in the field next to the easel symbol.

Changing a measure/label color

Do the following:

1. Click **Data > Measures** in the property panel.
2. Click applicable measure and select to color the value (**Value color**) or the label (**Label color**).
3. Select color from the color palette that opens.



Changing background color

Do the following:

1. You can also set your own color by clicking the easel symbol in the color palette and select color or enter a color code string in the field next to the easel symbol. The colors should be valid css-colors.
2. Click **Appearance > Styles** in the property panel.
3. Click the **Background color** box.
4. Select color from the color palette that opens and the chart updates.

Changing sizes of a Multi KPI

The size of dimensions, measures, labels and icons can be configured using predefined sizes available from a drop-down menu:

- **Dimension labels:** Select the size of the labels from the drop-down menu under **Appearance > Dimensions > Size** in the property panel.

- **Measure labels:** All measures can have individual sizes.

Do the following:

1. Select the size of the measures in the property panel under **Data > Measures**.
2. Select **measure** and click the **Override parameters** box.
3. Enter the label type as a string or in the expression under **Label** and select the size from the options in the drop-down menu under **Size**.

Icons: All icons can be differentiated by applying different colors.

Do the following:

1. Go to **Data > Measures** in the property panel.
2. Select **measure** and choose icon from the icon option.
3. Then choose **icon size** from the drop-down menu under **Icon**.

Changing the fonts of a Multi KPI

You can change the font-family of all labels if you go to **Data > Measures** in the property panel and select applicable measure. Under **Font style** enter a label string and press Enter.

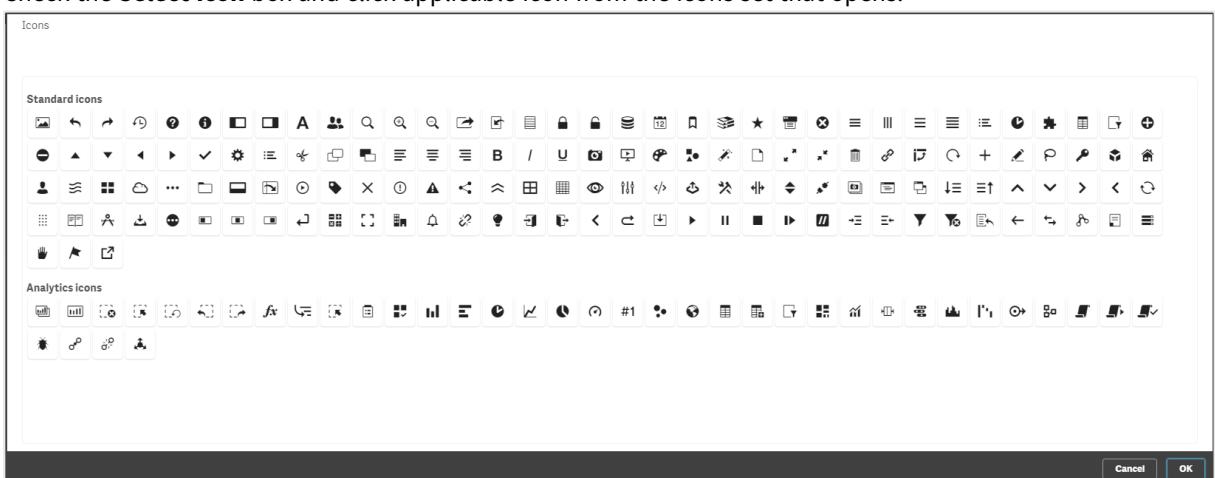
You can also change the font using an expression in the Expression editor (**fx**) or apply individual font css properties entered through **Appearance > Styles > Styles (CSS)** in the property panel.

Changing icons

You can set a predefined icon to represent a value and/or label by selecting from the icon set. The icons can then be further customized by differentiating using the colors, fonts, labels and sizes options.

Do the following:

1. Click **Data > Measures** in the property panel.
2. Check the **Select icon** box and click applicable icon from the icons set that opens.



3. Click **OK**. The pop-up will close and your chart update.
4. Choose whether to configure your label or value by selecting **Label** or **Value** under **Icon position**.

You can also change a value/label to an icon by entering an icon string under **Icon** (**Data > Measures** > applicable measure in the property panel) and then press Enter. Thereafter click **Label** or **Value** under **Icon position** depending on which you want to configure. It is also possible to change an icon using an expression in the Expression editor (**fx**). The icons should be valid CSSicons.

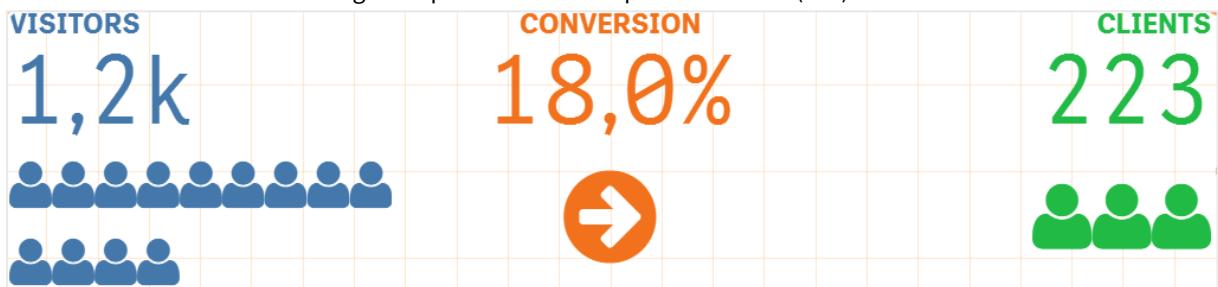
Using infographic mode

The infographic mode allows you to show each measure graphically as an appropriate number of icons.

Do the following:

1. Click the applicable measure under **Data > Measures** in the property panel and click the **Icon** button.
2. Select an icon to represent a measure from the predefined icon set (**Icons**) that opens.
3. Click the **Infographic mode** button. The resulting KPI chart updates and graphically shows the measures with applicable number of icons (max. 100 icons per measure).

You can also select an icon using an expression in the Expression editor (**fx**).



Embedding a master visualization

To graphically illustrate the KPI object, you can embed a master visualization.

Do the following:

1. In the in the property panel go the **Data > Measures** and click **Add**.
2. Click the **Expression editor** (**fx**) button.
3. Enter the string ='Drag and Drop here' into the **Expression editor** and click **Apply**.
4. Drag and drop a master visualization object into the value region of each measure placeholder.

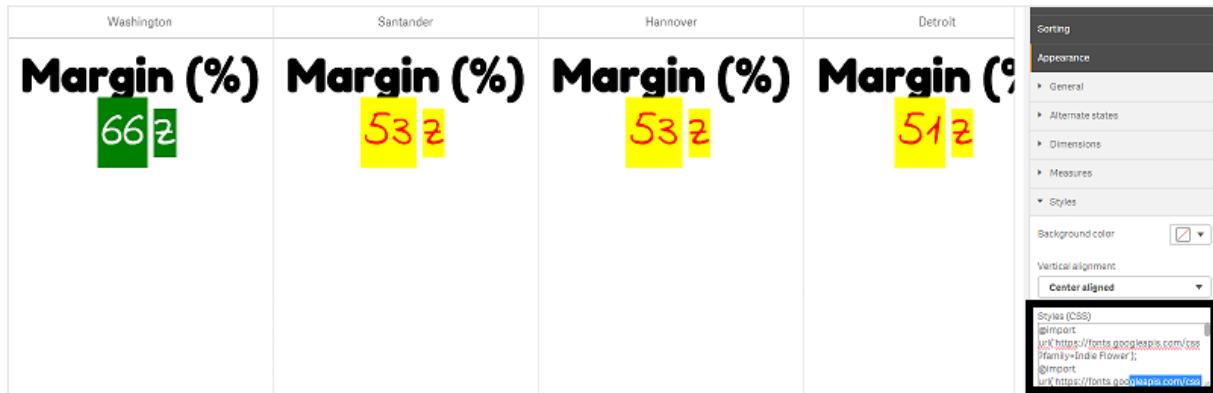
You can also change insert the object ID string into the expression in the Expression editor (**fx**) under **Visualization** for each measure (**Data > Measures** > applicable measure in the property panel).



Using Styles (CSS) properties

You can further configure your measures using non-predefined Styles (CSS) property for each measure. This can be icons, changing color of text/icons, label, font, background color, size, etc. Copy and paste the styles into the **Styles (CSS)** field (under **Appearance > Styles** in the property panel).

For icons, enter appropriate classes in the **Icon** field of the chosen measure under **Data > Measures** in the property panel. You can also enter the icon string using an expression in the Expression editor (**fx**).



Formatting your KPIs

You can format the chart layout to be presented in different ways.

Alignment

You can configure the main chart text alignment by selecting applicable box under **Alignment** of chosen measure under **Data > Measures** in the property panel.

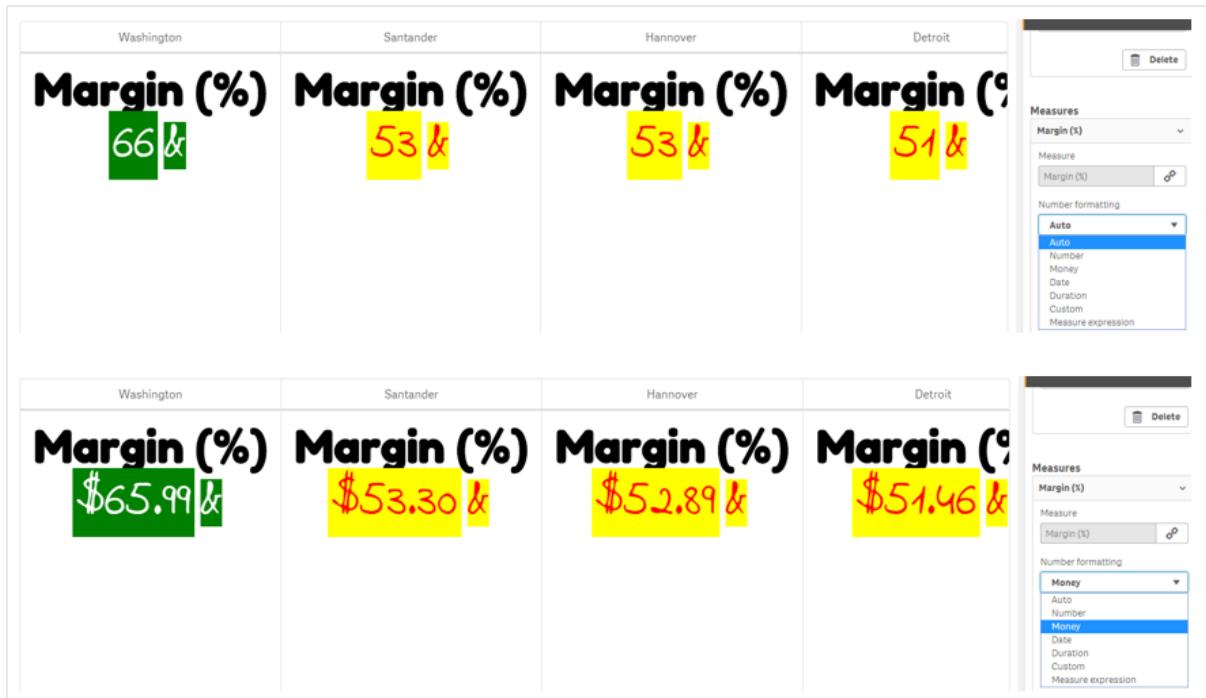
All KPIs can also be vertically aligned. This alignment can be adjusted to align top, bottom and center or as stretched. Select applicable alignment from the **Vertical alignment** menu under **Appearance > Styles** in the property panel. It is also possible to center align labels by checking the **Center align labels** in the property panel under **Appearance > Dimensions**.

Number formatting

You can also custom format the KPI number values. Different formatting can be applied to the same values. The chart updates to reflect the changed number type.

Do the following:

1. Click **Data > Measures** in the property panel and select applicable measure.
2. Select applicable number formatting from the **Number formatting** menu.



Format pattern

The text of a specific measure can be changed or removed.

Do the following:

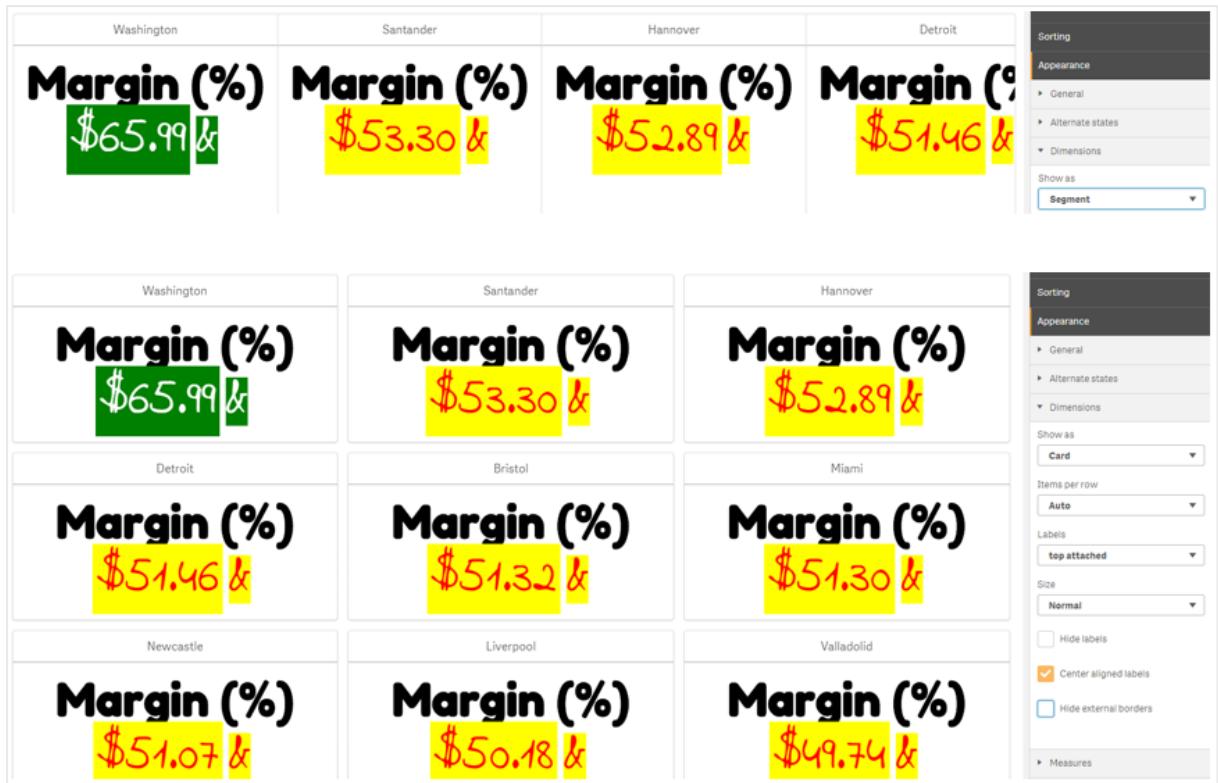
1. Click **Data > Measures** in the property panel.
2. Click applicable measure and enter/remove the applicable format string under **Format pattern** and press Enter.

Segment/card

It is also possible to configure the layout of the KPI chart where dimensions and values display in a segment or card layout.

Do the following:

1. Click **Appearance > Dimensions** in the property panel.
2. Choose **Segment** or **Card** layout from the **Show as** menu. The chart will update.



The segment layout can be configured horizontally or vertically under **Appearance > Dimensions > Orientation** in the property panel. Click either the **Horizontal** or the **Vertical** buttons.

Items per row

It is also possible to decide the number of items (KPIs) per line in a chart in the property panel under **Appearance > Measures**. Use the default Auto or select applicable number from the **Items per row** menu (between 1-8).

Borders

You can hide external and internal borders of the chart. By default all borders are visible. To configure borders go to **Appearance > Dimensions** in the property panel and check , as applicable, **Hide external borders** or **Hide internal borders**.

Value/Label layout and formatting

The position and order of labels for both dimension and measure values can be altered and displayed in different ways. Labels can be hidden or displayed vertically or horizontally. By default **Horizontal** is selected.

Dimension label layout

Do the following:

1. Click **Appearance > Measures** in the property panel.
2. Click either **Horizontal** or **Vertical** under **Labels orientation** to select how the labels should display.

Measure value layout

Measure value layout can be set in two different ways.

Do the following:

1. Existing parameters:

- a. Click **Appearance > Measures** in the property panel.
- b. Click either **Horizontal** or **Vertical** under **Labels orientation** to select how the labels should display.

2. New parameters:

- a. Click **Data > Measures** in the property panel.
- b. Click applicable measure and check the **Override parameters** box.
- c. Enter the label string under **Label** and press Enter.

Label order

You can also adjust the order in which measure value labels appear.

Do the following:

1. Click **Appearance > Measures** in the property panel.
2. Click either **Label, Value** or **Value, Label** under **Labels orientation** to select how the labels should display.

Icon order

You can decide where to show icons - before or after a value. In the property panel under **Appearance > Measures** go to **Icon order** and click either the **Icon, Value** or **Value, Icon** buttons to select which displays first in the chart.

Hide Labels

- **Measure labels:** This option hides the measure label of each KPI. To hide the labels click **Data > Measures** in the property panel and select applicable measure. Then check the **Hide labels** box.
- **Dimension labels:** This option hides the dimension label of each KPI. To hide the labels click **Appearance > Dimensions** in the property panel. Then check the **Hide labels** box.

Hide Values

This option hides the value for each KPI. To hide the value, click **Data > Measures** in the property panel. Then check the **Hide values** box.

Group KPI values

All KPI values can be grouped or displayed individually per measure. Go to the **Data > Measures** in the property panel. Choose applicable measure and check the **Group by dimension** box.

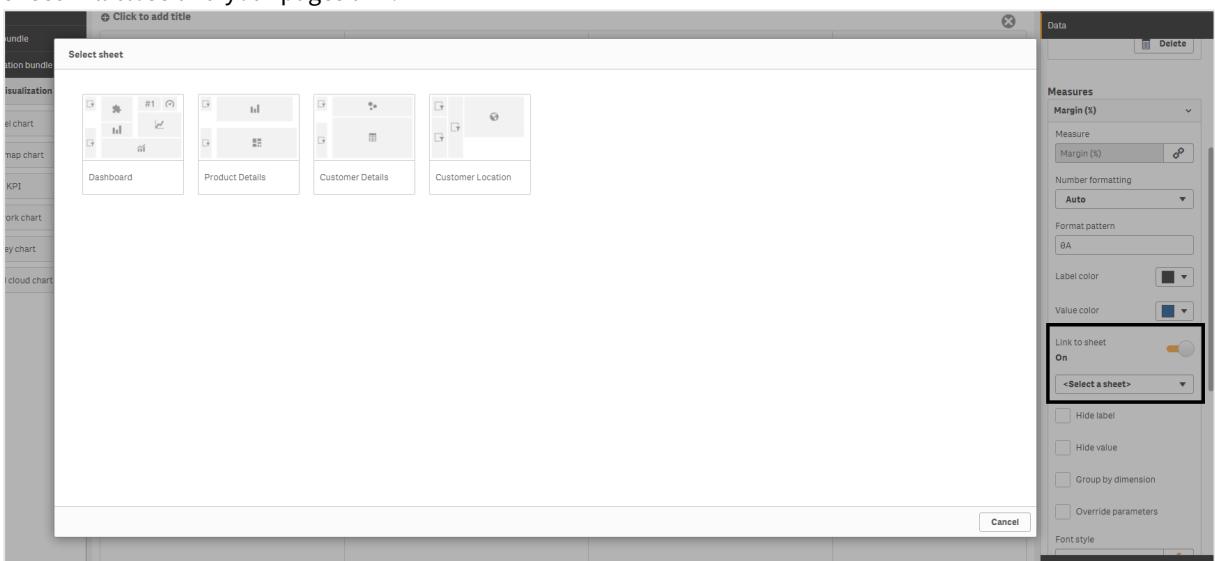
Another option is to enter a dimension value string under **Dimension value** and press Enter. The chart will update. You can also enter the dimension value using an expression in the Expression editor (**fx**). The values should be valid CSS values.

Link to a separate sheet

Each measure can be linked to another sheet.

Do the following:

1. Click **Data > Measures** in the property panel and select applicable measure.
2. Move the **Link to sheet** slider to the right to turn the option on.
3. Click the **Select sheet** button and from the **Select sheet** pop-up click the page you want a link to. The sheet will close and your pages link.



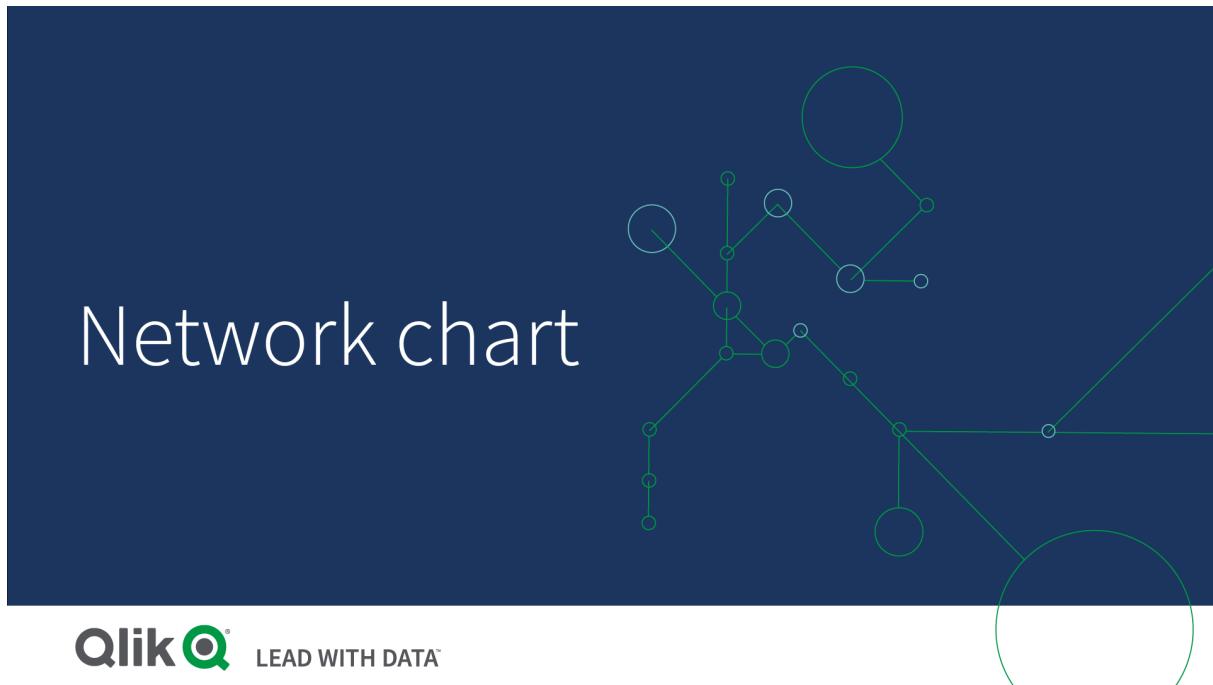
Limitations

For information about general limitations, see *Limitations (page 333)*.

Network chart

The network chart (**network chart**) lets you draw a network of connected nodes and edges from a data set to a sheet. A network chart shows how information flows, how components interact, and where components exist in the network. The network chart is included in the Visualization bundle.

A network chart can provide a broad overview or be very detailed. Nodes represent system components, and edges show the movement of information from one component to another. Network nodes are connected to the nodes they interact with the most. This visualization uses different styles, colors, sizes, and images to represent different levels of a network.



Qlik  LEAD WITH DATA™

What it contains

You need to define three dimensions, with a fourth optional dimension:

- **Node identifier:** This dimension controls which nodes are presented in the chart.
- **Node label:** This dimension sets the label of each node.
- **Node parent:** This dimension sets the parent of a node, and controls the relationships between nodes. It needs to contain the value of the node identifier of the parent to connect to.
- **Node group (optional):** You can use this dimension to group nodes. All nodes in the same group will have the same color.

You can use up to three measures to enhance the diagram. All measures are optional, but you need to add them in the following order:

1. **Tooltip:** You can set a measure value that is displayed in a tooltip when hovering over a node.
2. **Node size:** You can set the size of the node according to a measure.
3. **Edge size:** You can set the width of the lines between nodes according to a measure.



You need to add a tooltip before you can set node size. You can set edge size after adding a tooltip and node size.

When to use it

Network chart diagrams can illustrate computer or telecommunications networks. They show the components of a network and how they interact. For example, a group of connected computers, printers, modems, hubs, and routers. This type of chart is helpful when:

- Planning the structure of a network.
- Coordinating updates to an existing network.
- Reporting and troubleshooting network problems .
- Keeping track of components.
- Documenting detailed network documentation.

Formatting your data

A network chart requires data that is structured consistently according to a network data model where each record can have multiple parents and children. Each record needs to contain at least:

- A field that identifies the node, the node identifier. Node identifier values must be integer values, starting from 0 and in sequential order.
- A field with the name of the node.
- A field that defines the parent node. This value needs to be the node identifier of another node. If this field is empty, and no other record refers to this node, a disconnected node is created.



It is also possible to use a hierarchical data model where each node has a single parent. This will create a tree-shaped chart.

Here is some example data that you can save in a text editor and load in a new app. You need to load both example files. The example shows passenger flows between different airports.

- ID is the identifier of an airport node.
- Name is the name of an airport node. This is used as label of the node.
- LinkTo contains the node identifier of the parent node.
- Group states the group of a node. This can be used to color the nodes according to group.
- Volume is the passenger flow volume between ID and LinkTo. This can be used as a measure in edge size.
- NodeVolume is the total passenger flow volume for a node. This is loaded in a separate table as the chart cannot aggregate the volumes automatically.

Example 1: *Airports1.csv*

```
ID;Name;LinkTo;Group;volume
0;Frankfurt;;0;
1;London;0;1;5
2;Madrid;0;1;4
2;Madrid;1;1;8
3;Warsaw;0;1;7
4;Arlanda;0;1;1
3;Warsaw;1;1;5
4;Arlanda;1;1;6
5;Tunis;0;2;8
5;Tunis;2;2;4
6;Berlin;0;1;6
6;Berlin;4;1;4
7;Rome;0;1;6
7;Rome;6;1;3
```

```
8;San Francisco;0;3;2  
9;New York;0;3;9
```

Example 2: Airports2.csv

```
ID,NodeVolume  
0,48  
1,24  
2,16  
3,12  
4,11  
5,12  
6,13  
7,9  
8,2  
9,9
```

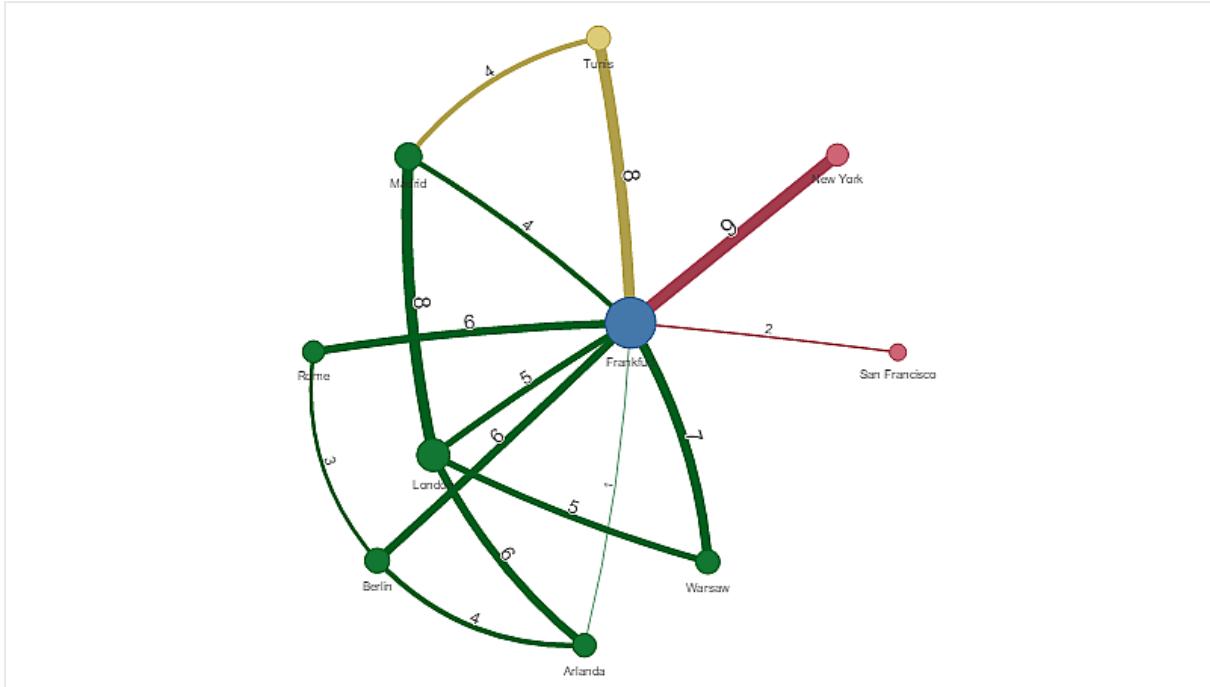
Creating a network chart

This example of a network chart illustrates how different airports are connected worldwide based on base data set. It uses the example data set from [Formatting your data](#). Prepare by saving the example data set in a text file, creating a new app and loading the example data set.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Network chart** object to the sheet.
2. Click the top **Add dimension** button and select *ID* as the node identifier.
3. Click the second **Add dimension** button and select *Name* as the node label.
4. Click the third **Add dimension** button and select *LinkTo* as the parent node.
5. Click **Data** in the properties panel. Click the **Add** button under **Node group** and select *Group* as the group dimension.
6. Under Measure, click the **Add** button under **Tooltip**, and select **Volume > Sum(NodeVolume)**.
7. Click the **Add** button under **Node size**, and select **Volume > Sum(NodeVolume)**.
8. Click the **Add** button under **Edge size**, and select **Volume > Sum(Volume)**.

The chart displays:



Changing the appearance of the chart

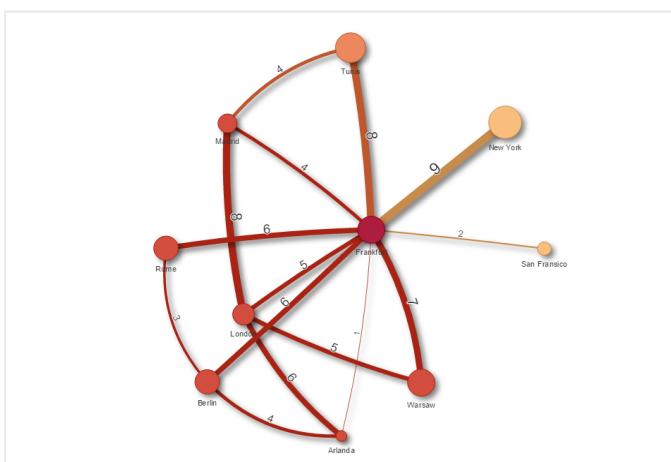
You can customize your chart with one or more features.

Configuring edge type

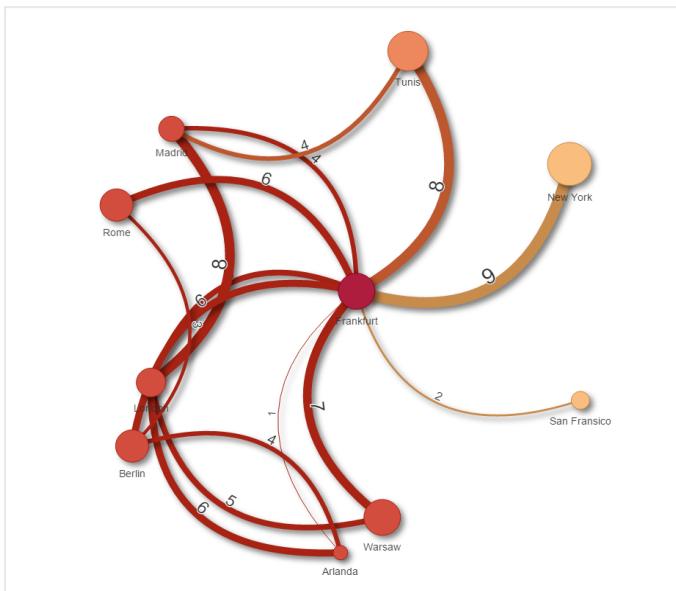
You can configure the shape of the curve between chart nodes under **Settings > Edge Type** in the properties panel. Select the shape option of the chart curves from the menu.

Examples:

A chart with dynamic edge type (curves).



The same chart with curved CW edges.



Toggling edge value

The edge value represents value of the curves between chart nodes and their width. This option hides the numerical value of each curve. To hide the value move the slide button under **Settings > Display edge value** in the properties panel to the left to turn the option off.

Configuring edge label

The edge label value is the numerical value of each chart curve. You can change the position of these or hide the labels.

- **Hide edge label value:** This option hides the edge label of the network chart. Toggle the slide button under **Settings > Display edge value** in the properties panel.
- **Edge label position:** This option decides where on each chart curve the edge value displays, for example: above or below. Select applicable option from the menu.

Configuring node shape

You can customize the shape of the nodes, for example: dot, square, diamond, or triangle. Go to **Settings > Node shape** in the properties panel, and select the node shape from the menu.

Toggling shadow option

This option lets you turn off the shadow effects behind chart curves and nodes (used to highlight background). Toggle the slide button under **Settings > Display shadow** in the properties panel.

Sorting

Sorting is set to Auto with the system choosing the sorting order as the default (under **Sorting > chosen dimension or measure** in the properties panel) for all dimensions and measures. You can change them individually by expression.

Do the following:

1. Open the dimension or measure menu under **Sorting** in the properties panel.
2. Move **Sorting** slide button to the left to turn the option from Auto to Custom.
3. Click the **Sort by expression** check box.
4. Enter an order string under **Expression** and press Enter. You can also change the color using an expression in the Expression editor (**fx**).
5. Choose start order by selecting **Ascending** or **Descending** for the menu below.

Number formatting

It is possible to format the measure value. Different formatting can be applied to the same value, for example: money, data, or duration. The chart updates to reflect the changed number type.

Do the following:

1. Click **Data > Measures** in the properties panel and click chosen measure.
2. Select applicable number formatting form the **Number formatting** menu.
3. Enter details in the panel fields. These display when choosing an option other than Auto when configuring the chart.

Limitations

For information about general limitations, see *Limitations (page 333)*.

- Network chart visualizations cannot be used in Qlik NPrinting reports.
- The maximum size of the data set displayed in the network chart is 1400 rows. If the selected data set is larger, nodes and links can be omitted from the chart.
Use a smaller data set or employ selections to limit the data set.

Org chart

You can use the org chart (**Org chart**) to create organization charts from data with a tree structure. You can navigate through the hierarchy by expanding and collapsing the tree. The org chart is included in the Visualization bundle.

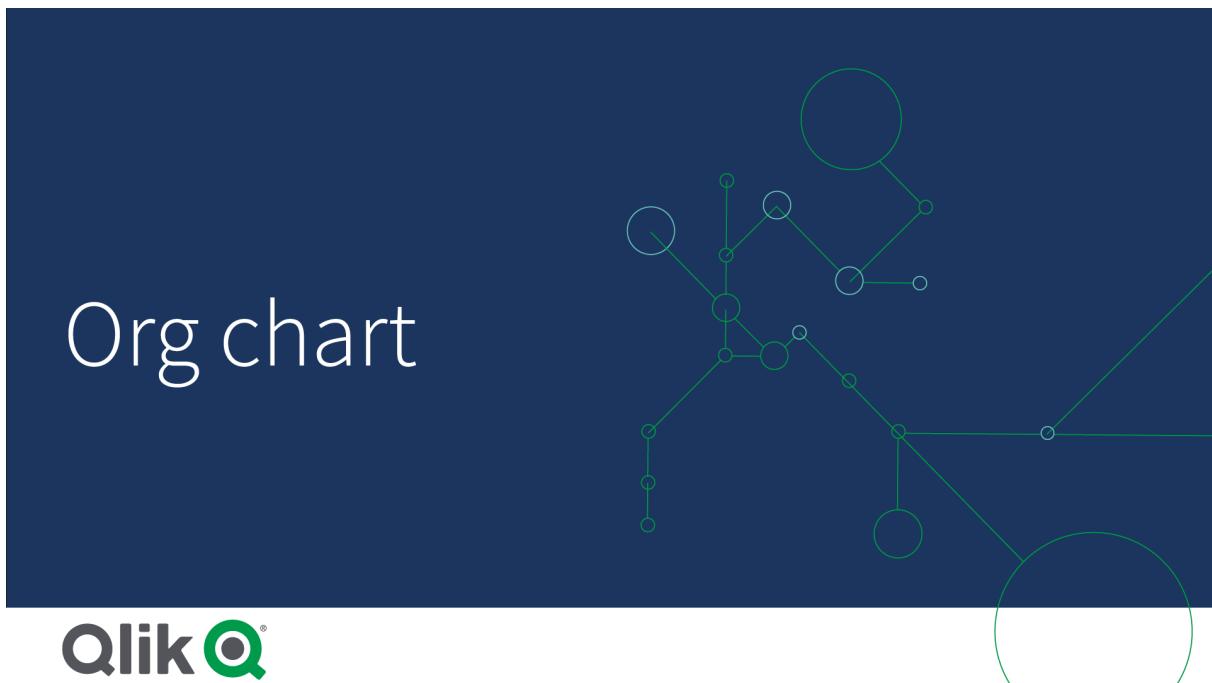
You need to use a data set with two dimensions that define the tree structure.

- The first dimension needs to be the ID of the node. This would typically be an employee ID in a traditional organization chart.
- The second dimension needs to be the ID of the parent node. This would typically be the employee ID of a manager.
This connects to the first dimension to create a tree structure.
If this value is empty or points to a node that does not exist, the node becomes a root node at the top of the tree.

Example data set for an org chart

EmployeeID	ManagerID	Name	Title
A101		Mary Bell	CEO
A102	A101	John Bialik	Executive secretary
O101	A101	Lee Mayer	COO
I101	A101	Wendy Sanderson	CIO
T101	A101	Asim Nawrat	CTO
T102	T101	Emily Diaz	VP Products
T103	T101	Christine Nemic	VP R & D

You can also add a measure that is used as a card description.



Preparing the dataset

There are some things you need to consider when preparing the dataset for an org chart.

- You need to use a dataset with less than 33000 rows.
If you use a larger dataset, the excess rows will be ignored. This can result in a tree with incorrect structure. A message is displayed: **The maximum data limit is reached. The tree may display incorrectly.**
- Make sure that the data does not contain circular references.
Circular references can result in nodes being omitted, or creation of multiple root nodes. One of these messages is displayed:
Data contains circular references, nodes are omitted.
No root node, check your data for circular references.

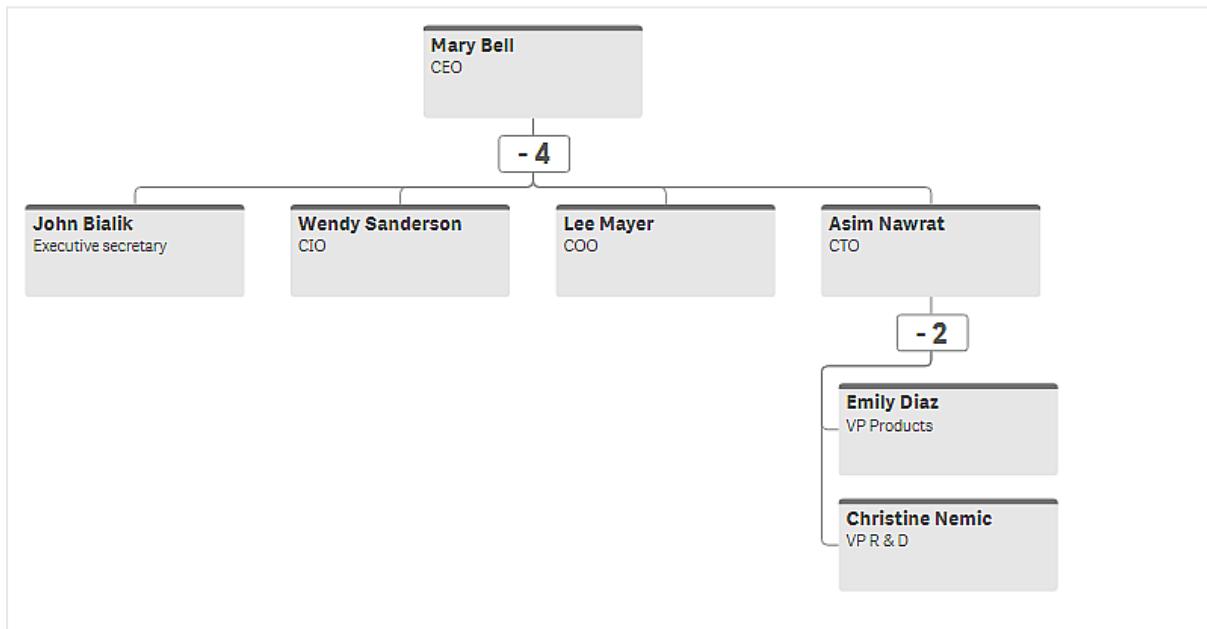
- If a node has more than 99 children, the excess children nodes will not be rendered.
A message is displayed: **Maximum number of child nodes reached, nodes are omitted.**
- You should avoid having a large number of rows that do not have a parent node. These will be shown as root nodes. The maximum number of root nodes is 99.

Creating an org chart

You can visualize the example data set using an org chart. Every employee is shown as a card in a tree structure that you can expand and collapse.

Do the following:

1. Add an **Org chart** to your sheet.
2. Set the first dimension to be EmployeeID.
3. Set the second dimension to be ManagerID.
4. Add labels to the cards. Expand the EmployeeID dimension to see the label properties.
 - a. Set **Card title** to Name.
 - b. Set **Card sub-title** to Title.



Changing the appearance of the chart

You can customize the appearance of your org chart.

Adding a measure to set a description

You can display a description by adding a measure. This will replace the value set in the property **Card description**.

Changing the presentation mode

You can set how the org chart displays in **Appearance > Presentation > Presentation mode**. Org charts can either display the entire tree, or they can collapse and expand. If an org chart is set to **Expand/collapse**, you can also set the org chart to automatically resize to fit every time it is expanded or collapsed.

Changing the colors

You can change the color of the card background in two different ways:

- Expand the first dimension and set an expression that returns a color or a color code under **Card background color**. You can refer to data fields in the expression.
- Set a background color under **Appearance > Presentation > Background color**. You can set all cards to a single color, or use an expression. If you use an expression you cannot refer to data fields.

You can also set the color of the text with **Appearance > Presentation > Font color**.

Changing card borders

You can choose if org chart cards have borders and top bars under **Card appearance**. If you choose to include borders, you can set a border color in **Appearance > Presentation > Border color**. You can set borders to a single color or use an expression. If you use an expression, you cannot refer to data fields.

Selections in the org chart

You can select cards in the org chart. Selecting a card will include all child nodes below a card in the selection.

For information about general limitations, see *Limitations (page 333)*.

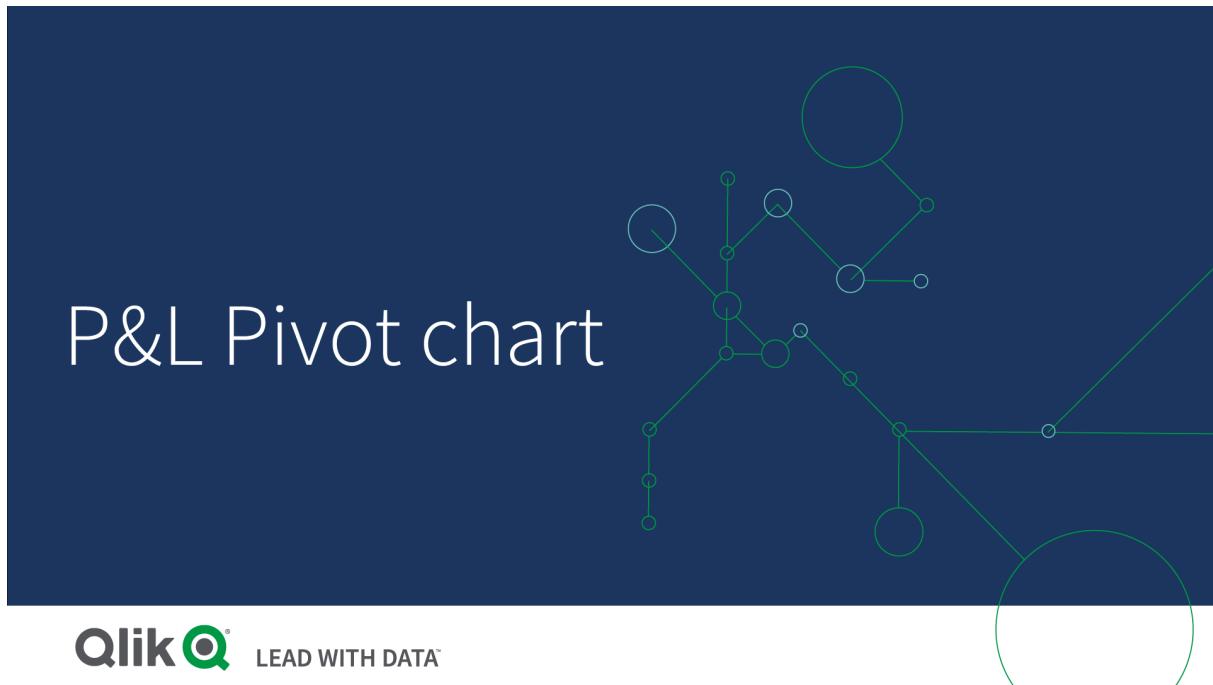
P&L pivot chart

You can use the **P&L pivot** chart to create a pivot table with a layout for profit and loss reporting. You can color cells according to performance and style the chart with custom fonts and alternating row colors. You can also download the chart to Excel including formatting. The **P&L pivot** chart is included in the Visualization bundle.

Account Desc	France				Italy			
	Balance...	Bal LY (k)	% Var	Diff (k)	Balance...	Bal LY (k)	% Var	Diff (k)
Revenues								
Gross sales revenues	33,329	29,663	0	3,666	37,495	33,371	0	4,124
Less return & allowances	346	242	0	104	389	272	0	117
Net sales revenues	32,983	29,905	0	3,078	37,106	33,643	0	3,463
Cost of goods sold								
Direct materials	-6,320	-6,636	-0	316	-7,110	-7,466	-0	356
Direct labor	-6,100	-5,917	0	-183	-6,863	-6,657	0	-206
Manufacturing overhead								
Indirect labor	-5,263	-5,000	0	-263	-5,921	-5,625	0	-296
Depreciation, manufacturing equip.	-360	-7	49	-353	-405	-8	49	-33
Other mfr overhead	-4,000	-4,400	-0	400	-4,500	-4,950	-0	450
Net mfr overhead	-9,623	-9,407	0	-216	-10,826	-10,583	0	-243
Net costs of goods sold								
	-22,043	39,312	-2	-61,355	-24,798	44,226	-2	-69,624
Gross profit								
	10,940	-9,407	-2	20,347	12,308	-10,583	-2	22,820
Operating expenses								
Selling expenses	0	0	0	0	0	0	0	0
Sales salaries	-4,200	-3,990	0	-210	-4,725	-4,489	0	-236
Warranty expenses	-730	-15	49	-715	-821	-16	49	-58

P&L pivot chart dimensions and measures

Dimensions	Measures	Result
1 dimension	up to 9 measures	A table with one row for each dimension value and one column for each measure.
2 dimensions	up to 8 measures	A pivot table with one row for each value of the first dimension and one column for each measure pivoted using the second dimension.



Coloring cells to show performance

You can color cells to show performance according to a scale of **Poor**, **Fair** and **Good**. All rows and columns are colored by default, but you can choose which columns and rows to color if you want to.

Do the following:

1. Make sure that **Enabled** is set to **On** under **Appearance > Color by condition**.
2. Select which rows to color by performance.
Set **Color all rows by condition** to **Specified rows**.
Add rows to color by name (dimension value) with **Add row to color**.
3. Select which measures to color by performance.
Set **Color all measures to Specified measures**.
Add a list of measures by number in **Measure indices** with the first measure of the chart numbered zero. Separate the measures with a comma.

Example: Color the first, third, and fifth measure.

0,2,4

4. Set performance limits and colors.
You can set the range limits for **Poor** and **Fair**.

- All cells with a value lower than the **Poor** range limit are displayed with the background color and text color set for **Poor**.
- All cells with a value lower than the **Fair** range limit, but higher than **Poor**, are displayed with the background color and text color set for **Fair**. You should set the **Fair** range limit higher than **Poor**.
- All other cells are displayed with the background color and text color set for **Good**.

Styling the chart using a style template

You can create a layout for the chart, for example to show a profit and loss report, using a style template.



You need to be able to add and load a CSV file to the app.

Do the following:

1. Create a style template as a CSV file. Use the style template format described below.
2. Load the style template to your app as one field. When you add the file, do not use semicolon as field separator, each row should be loaded as one field.
3. Set **Style template field** under **Appearance > Table format** to the name of the template field you added.

You can load several style templates in your app, and change the layout with **Style template field**.

Style template format

The style template is created as a comma separated text file (CSV) using UTF-8 encoding.

The style template rows need to align to the data in your first dimension. You need to refer to a dimension value in the template. The dimension value should be first in every row. You do not need to specify all rows/dimension values in the template. The style template can contain maximum 5000 rows.

Each row in the template should be in the following format. It is not required to use a header row.

DimensionValue;Bold;Background;FontStyle;TextColor;Align;FontSize;Comment

- DimensionValue
The dimension value of the row that you want to style.
- Bold
Set to `<bold>` if you want bold text.
- Background
Set a background color. You can use `<dark>`, `<night>`, `<soft>`, `<red>`, `<orange>`, `<violet>`, `<blue>`, `<green>` or a color code in RGB format, for example `rgb(183,219,255)`. The default background color is white.
- FontStyle
You can set the font style to `<italic>` or `<oblique>`.
- TextColor
You can set the color of the text to `<white>`. The default background color is black.
- Align
You can center align the text with `<center>`. The default alignment is left for text and right for numeric values.
- FontSize
You can set the font size to `<large>`, `<medium>` (default) or `<small>`.
- Comment
You can use the `<comment>` tag to replace all zero values with a space. This is useful when you want to include a sub header row without values.

You can also use the style tags in any order, and exclude tags that are not used. These rows will give the same result:

```
Operating expenses;<bold>;<italic>;;;  
Operating expenses;<italic>;<bold>
```

Style template example for profit and loss reporting

```
Cost of goods sold;<bold>;RGB(225,226,226);;;;  
Extraordinary items after tax;<bold>;RGB(193,216,47);;;<center>;<large>;  
Extraordinary items;<bold>;<italic>;<center>;;<comment>  
Financial revenue & expenses;<bold>;<italic>;<center>;;<comment>  
General & administrative expenses;<bold>;<italic>;<center>;;<comment>  
Gross profit;<bold>;RGB(193,216,47);;;<center>;<large>;  
Income before tax & extraordinary items;<bold>;RGB(193,216,47);;;<large>;  
Manufacturing overhead;<bold>;<italic>;<center>;;<comment>  
Net costs of goods sold;<bold>;RGB(225,226,226);;;;  
Net gain on sale of land;<bold>;RGB(193,216,47);;;<center>;<large>;  
Net Income (Profit);<bold>:#191970;;<white>;<center>;<large>;  
Net mfr overhead;<bold>;RGB(225,226,226);;;;  
Net sales revenues;<bold>;RGB(225,226,226);;;;  
Operating expenses;<bold>;<italic>;;;  
Operating income before taxes;<bold>;RGB(193,216,47);;;<large>;  
Other general & admin expenses;<bold>;rgb(128, 191, 255);<white>;<center>;<large>;  
Revenues;<bold>;<italic>;<center>;;<comment>  
total general & admin expenses;<bold>:#efefef;;;;  
total operating expenses;<bold>;rgb(128, 191, 255);<white>;;;;  
Total selling expenses;<bold>;RGB(225,226,226);;;;
```

To use this template, you need a data file where the first dimension contains values that correspond to the first item of each row, for example *Cost of goods sold*.

P&L pivot chart styled with the layout template in the example

Account Desc	France				Italy			
	Balance...	Bal LY (k)	% Var	Diff (k)	Balance...	Bal LY (k)	% Var	Diff (k)
Revenues								
Gross sales revenues	33,329	29,663	0	3,666	37,495	33,371	0	4,1
Less return & allowances	346	242	0	104	389	272	0	1
Net sales revenues	32,983	29,905	0	3,078	37,106	33,643	0	3,4
Cost of goods sold								
Direct materials	-6,320	-6,636	-0	316	-7,110	-7,466	-0	3
Direct labor	-6,100	-5,917	0	-183	-6,863	-6,657	0	-2
Manufacturing overhead								
Indirect labor	-5,263	-5,000	0	-263	-5,921	-5,625	0	-2
Depreciation, manufacturing equip	-360	-7	49	-353	-405	-8	49	-3
Other mfr overhead	-4,000	-4,400	-0	400	-4,500	-4,950	-0	4
Net mfr overhead	-9,623	-9,407	0	-216	-10,826	-10,583	0	-2
Net costs of goods sold								
	-22,043	39,312	-2	-61,355	-24,798	44,226	-2	-69,6
Gross profit								
	10,946	-9,407	-2	20,347	12,308	-10,583	-2	22,8
Operating expenses								
Selling expenses	0	0	0	0	0	0	0	0
Sales salaries	-4,200	-3,990	0	-210	-4,725	-4,489	0	-2
Warranty expenses	-730	-15	49	-715	-821	-16	49	-8

Limitations

For information about general limitations, see *Limitations (page 333)*.

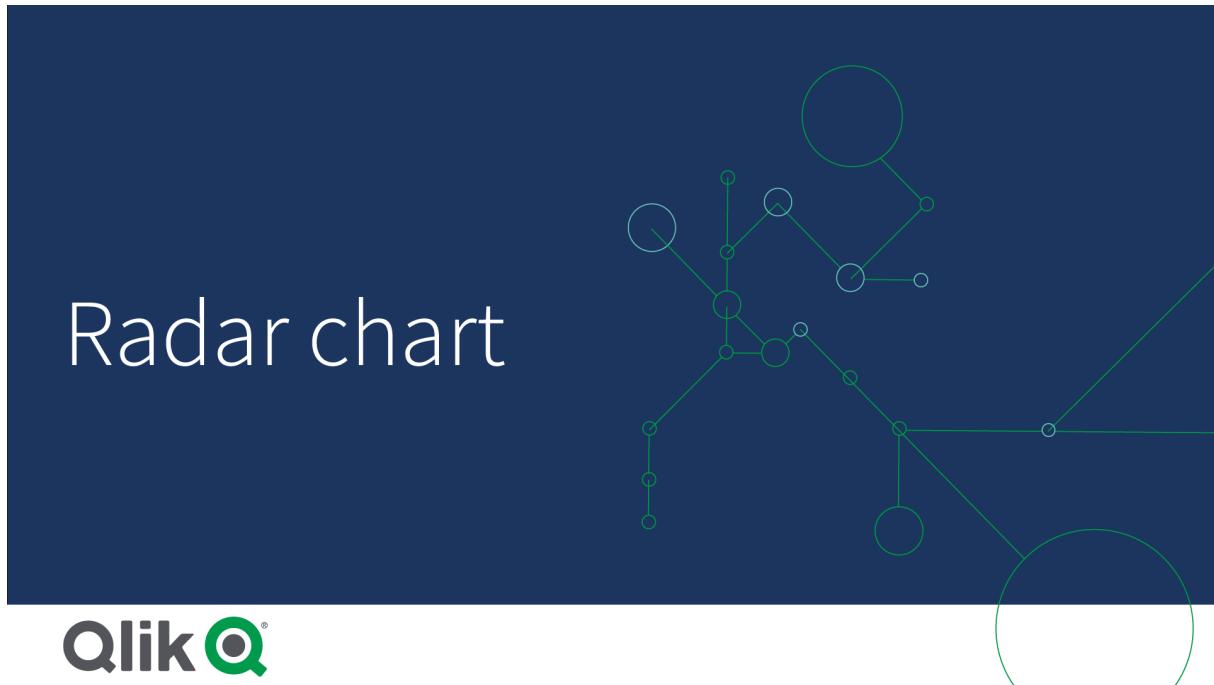
- There is a limitation for data transfer between Qlik Engine and **P&L pivot**. This limits each request for data to 10000 elements, for example, 1000 rows and 10 columns. **P&L pivot** can make automatic requests for more data using pagination.
You can set the limit of how much data to request with **Pagination > Max pagination loops**. The default value is 20000 elements (**20k cells**) and the maximum value is 40000 elements.
You can also modify the error message displayed to the user when the data limit is exceeded. The recommended workaround when the data limit is exceeded is to apply filters to the data to limit the data transfer.
- It is not possible to convert a **P&L pivot** chart to another visualization or convert another visualization to a **P&L pivot** chart.

Radar chart

The Radar chart (**Radar chart**) displays a two-dimensional chart using radial axes to chart one or more groups of values over multiple variables. Radar charts can be used to visualize and compare performance to a set standard or to a group's performance. The Radar chart is included in the Visualization bundle.

Radar charts requires two dimensions and one measure. The y-axis goes from the center to the perimeter and the x-axis is the perimeter of the chart. Each value represents the distance from the center of the chart, and displays on axes starting from the center. The center of the chart represents the minimum value, and the edge the maximum value.

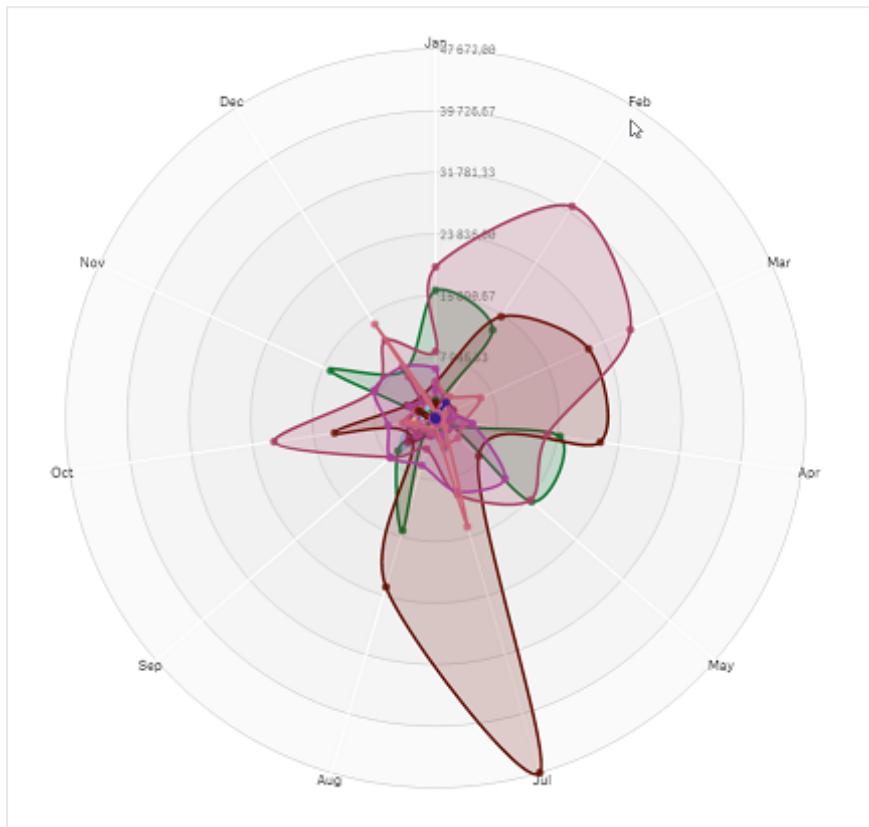
A radar chart can show multiple series, each separately connected with their values. The data only goes around the chart once. Adding values only increases the chart's granularity.



Understanding a radar chart

A radar chart consists of evenly spaced segments (axes) arranged radially around a point. There is one segment for each data value of the second dimension. The value of each measure is shown by a node on applicable axis. A line connects the values that belong to the same series creating the chart's star-like shape.

A radar chart with one axis for each month of the Date.Month dimension.



When to use it

Radar charts let you compare and measure data with an arbitrary number of variables. This is helpful when comparing something's features or performance over several metrics. For example: before buying a computer, you can compare different computers across several features, such as memory storage, processing, and screen size. A radar chart is also useful when:

- Measuring quality improvements and performance.
- Comparing allocated amount versus actual spending in an organization.
- Identifying outliers, commonality, and clusters of data with similar values and features.
- Charting an athlete's strengths and weaknesses.
- Comparing results of small-to-moderate-sized multivariate data sets.

Creating a radar chart

You can create a radar chart on the sheet you are editing.

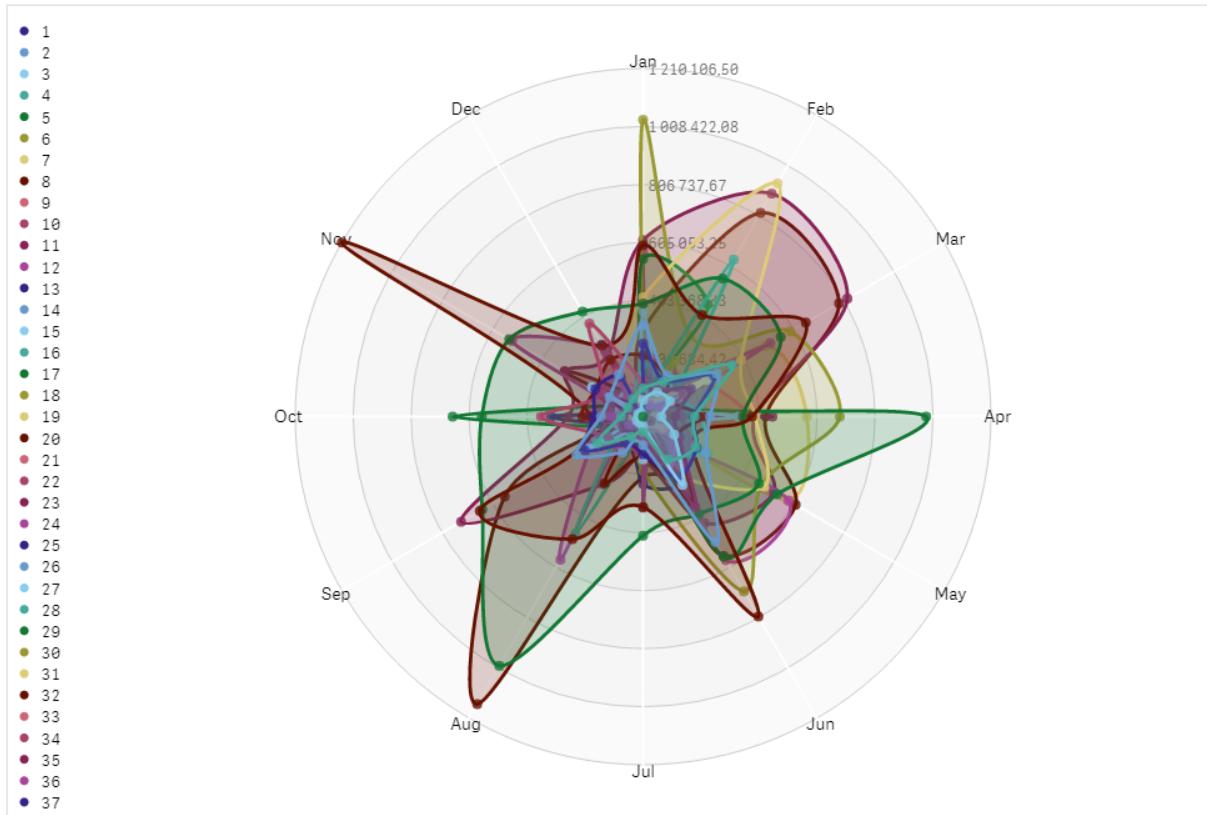
Do the following:

1. In the assets panel, open **Custom objects** > **Visualization bundle** and drag a **Radar chart** object to the sheet.
2. Click the top **Add dimension** button and select the first dimension category to be measured (x-axis).

3. Click the second **Add dimension** button to select the second dimension (y-axis).
4. Click the **Add measure** button to select the measure of the chart.

Once dimensions and measure have been selected the radar chart displays automatically (in color) in the chart field.

A radar chart with two dimensions and one measure.



Changing the appearance of the chart

You can customize your radar chart with one or more features.

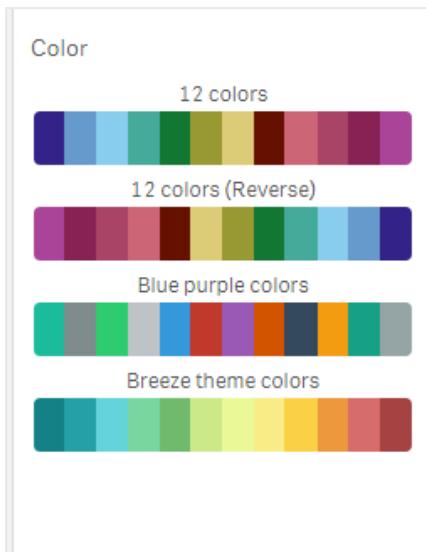
Changing the color scheme

You can change color scheme of your chart by choosing from four predefined options.

Do the following:

1. Click **Appearance > Design** in the property panel.

2. Select a color scheme under **Color**.



Changing the stroke type

You can also customize the shape line connecting each node under **Appearance > Design > Stroke type** in the properties panel. Select the shape from the menu.

Toggling the legend

The legend provides a small text description of each node of the chart. The legend expands to show its text, and the chart shrinks to accommodate the legend. To toggle the legend, move the slide button under **Appearance > Design > Legend** in the properties panel.

Limiting dimensions

You can set limits on your dimension values. To change limitations and terms go to **Data > Dimensions** in the properties panel. Click dimension, and under **Limitation** choose a limitation from the menu.

Toggling titles

This option can hide the name of the radar chart. To toggle the name, click **Appearance > General** in the properties panel. Move the **Show titles** slide button.

Number formatting

The default setting for **Number formatting** for measures and dimensions is **Auto**. You can change this setting to **Number**, and then select a number format.

Limitations

Radar charts have the following limitations:

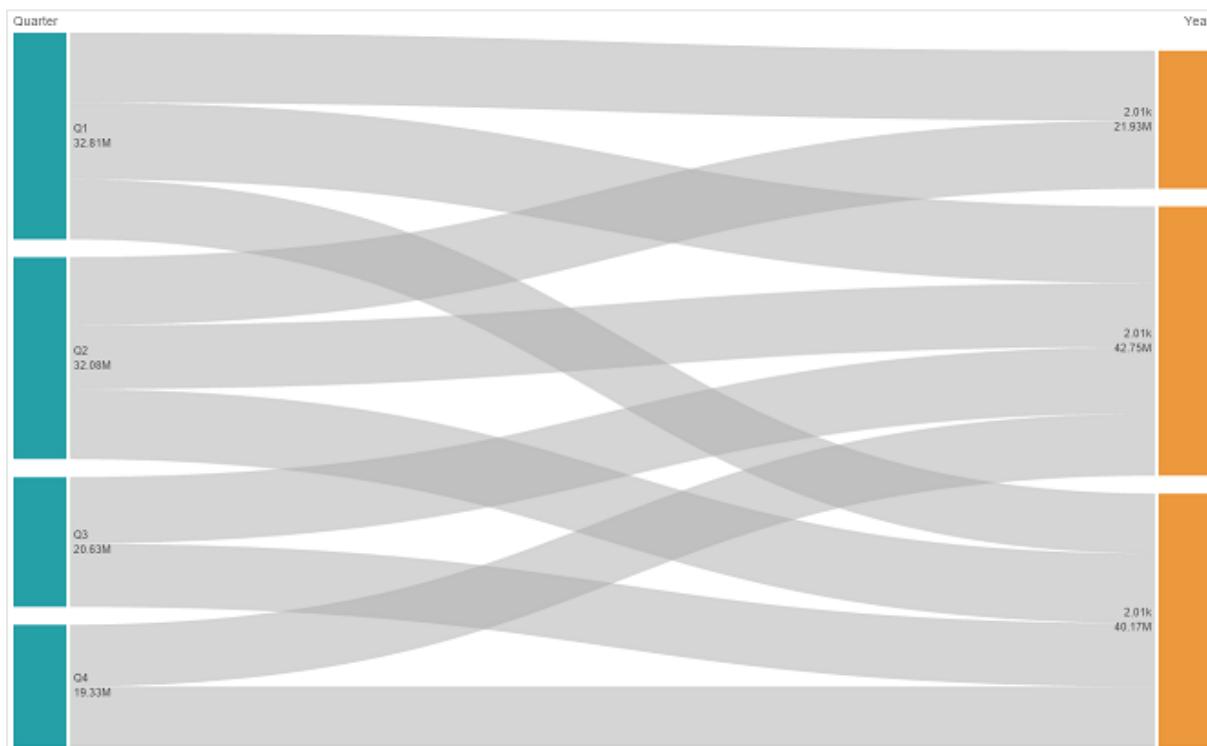
- Radar charts can only process 100 unique values per dimension.
- Exported radar charts will not include the chart legend.
- For information about general limitations, see *Limitations* (page 333).

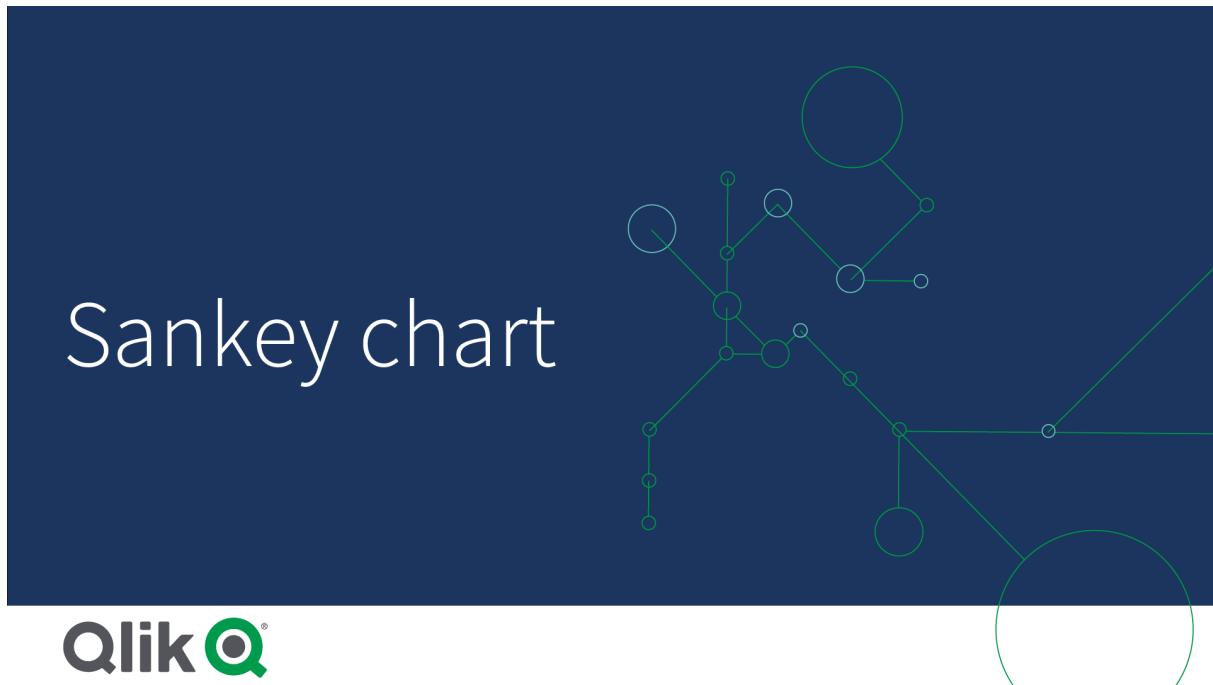
Sankey chart

The Sankey chart (**Sankey chart**) lets you add a specific type of flow chart to the sheet you are editing. The chart visually emphasizes major transfers or flows within defined system boundaries. The width of the chart arrows is shown proportionally to the flow quantity. The Sankey chart is included in the Visualization bundle.

- A minimum of two dimensions and one measure is required. You can use up to five dimensions, but only one measure.
- The dimensions do not need to be of equal size on each side of the diagram.
- You can use the dimension values to set the color of the flows in the chart.
- Link colors can be based on source or target anchor.

A chart with a source dimension (Quarter) and the target dimension (Year).





When to use it

The sankey chart is useful when you want to locate the most significant contributions to an overall flow. The chart is also helpful when you want to show specific quantities maintained within set system boundaries.

Creating a sankey chart diagram

You can create a sankey chart on the sheet you are editing.

Do the following:

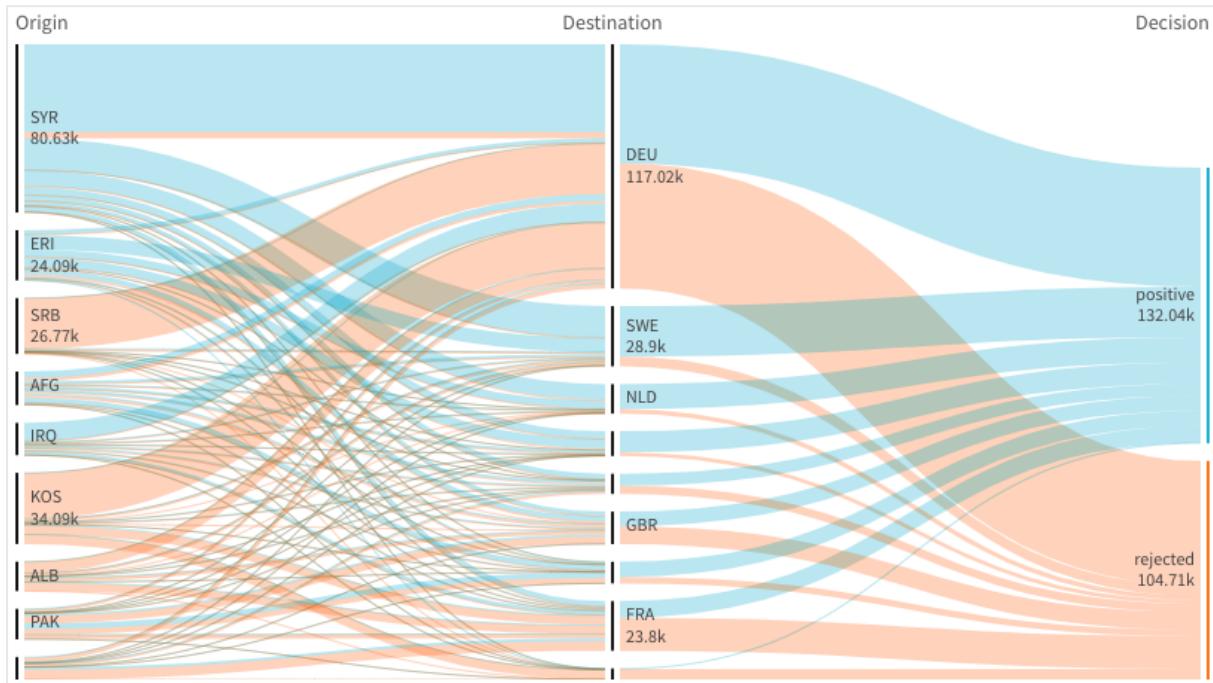
1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Sankey chart** object to the sheet.
2. Click the top **Add dimension** button and select the source dimension for the flow of the chart (appears to the left).
3. Click the second **Add dimension** button to select the target dimension for the flow of the chart (appears to the right).
4. Click the **Add measure** button to select the measure of the chart.

Once dimensions and measure have been selected the sankey chart diagram displays automatically (in color) in the chart field.

Adding additional dimensions

You can add up to five dimensions to your chart in property panel under **Data > Dimensions**. The chart updates to reflect the added dimensions. The dimensions are displayed from left to right, with the first entered dimension always being the source dimension. The target dimension is always displayed to the right. When you add more dimensions, they are added to the right in the order they are entered.

A chart with three dimensions: the source dimension (Origin), the target dimension (Decision) and one additional (Destination).



Sorting

Sankey chart elements are automatically sorted from largest to smallest flow. You can change the sort order in the property pane.

Do the following:

1. Click **Sorting** under **Appearance** in the property panel.
2. Toggle **Sorting** from **Auto** to **Custom**.
3. You can toggle **Sort numerically**:
 - Toggle on: Sort numerically by **Ascending** or **Descending**.
 - Toggle off: Drag your dimensions and measurements into the desired order.

Changing the appearance of the chart

You can customize your chart with one or more features. Your chart automatically updates.

Link colors

The colors of chart links are based on either the source or target anchors. To apply the source or target anchor color to chart links either use the string ='SOURCE' or ='TARGET'. You can also select a separate color by entering a color code string. The color should be a valid CSS color.

Do the following:

1. Click **Presentation** under **Appearance** in the property panel.
2. Enter the applicable string under Link color.
3. Press Enter and the chart updates.

You can also change the link colors using an expression in the Expression editor (**fx**). It is also possible to color a link that has its intensity based on the Margin % of the dimension values it represents.

Example:

Enter the string =rgb(round(Avg ([Margin %])*255), 100, 100) where Margin % is a value between 0-1 and the link will display as red in the chart.

Link opacity

You can adjust link opacity by moving the slide button of the link opacity slider under **Appearance > Link opacity** in the property panel. Also, setting opacity to 1 (furthest right) allows the setting to drop a shadow, giving links a more individually distinct appearance.

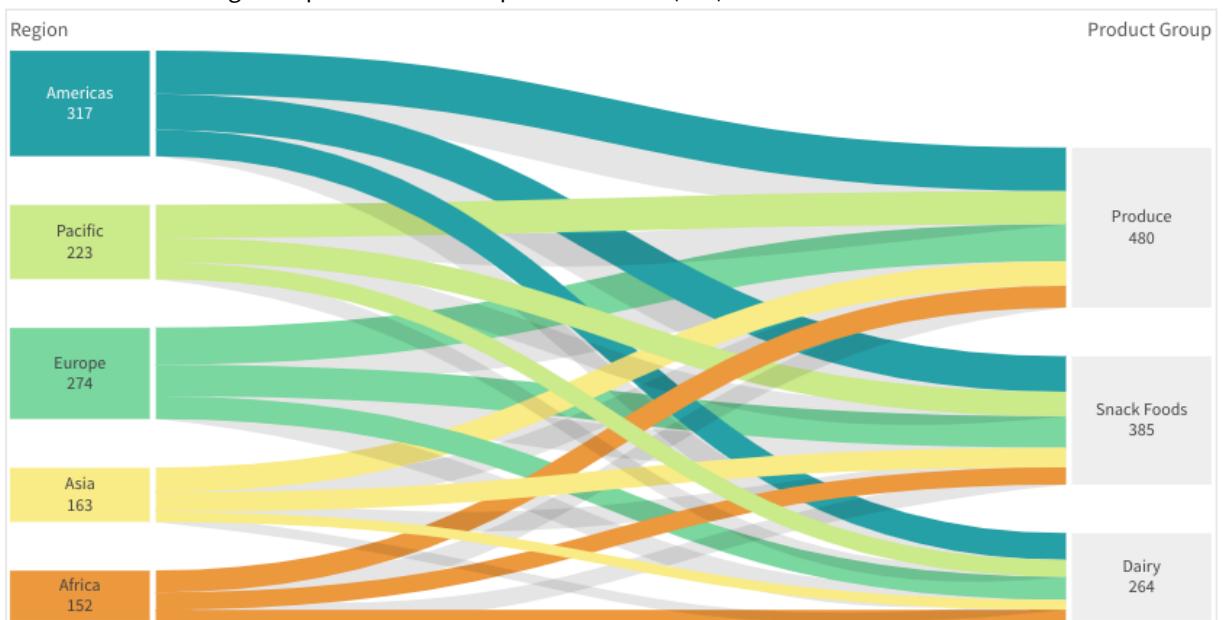
Node colors

You can change the node colors of each dimension value. The color should be a valid CSScolor.

Do the following:

1. Select applicable dimension under **Data > Dimensions** in the property panel.
2. Enter the color code string under Node color and press Enter. The chart will update.

For example: To use the color Aqua (#00ffff) set the color code string to ='#00ffff'. You can also set the node colors using an expression in the Expression editor (**fx**).



Node padding and width

You can set both the vertical distance between nodes ("node padding") and the horizontal width of chart nodes ("node width").

Do the following:

1. Click **Presentation** under **Appearance** in the property panel.
2. Move applicable slide button of the Node padding and/or Node width sliders to adjust node settings.

Limitations

For information about general limitations, see *Limitations (page 333)*.

Straight table

The **Straight table** shows several fields simultaneously, where the content of each row is logically connected. A table can consist of as many dimensions and measures as needed. The Straight table is included in the Visualization bundle.

App creators can add many fields simultaneously, customize the table at the column-level, add alternative dimensions and measures, set column width, apply pagination and turn on chart exploration.

Chart exploration allows users that do not have edit rights to customize the original straight table when they are in analysis mode. These users can add or remove columns, change sort order, re-arrange columns, and make selections. They can then share, download, subscribe, or bookmark the new table layout. The changes made in chart exploration mode by a user are not seen by other users analyzing the sheet.



This page is about the Visualization bundle straight table. For information on the native table object, see *Table (page 268)*.

Straight table in a sheet with chart exploration turned on

The screenshot shows a Qlik Sense interface with a 'Chart exploration' sidebar on the left. The sidebar includes a search bar and a list of selected dimensions: Customer, Region, Sales Rep Name, Date, Sum(Sales), and Sum(Cost). The main area displays a straight table titled 'Total sales and cost' with the following columns: Customer, Region, Sales Rep Name, Date, Sum(Sales), and Sum(Cost). The table contains numerous rows of data, each with a different color background. At the bottom of the table, there are pagination controls: 'Rows per page: 100', 'Select page: 1', and a range '1 - 100 of 24907'.

When to use it

Use a straight table when you want to view detailed data and precise values rather than visualizations of values. Tables are good when you want to compare individual values. Drill-down group dimensions are very efficient in tables. Within a limited space, you can drill down to the next level of detail and analyze the updated measure values. Use this table when you want users to be able to create custom tables in analysis mode.

Advantages

You can filter and sort the table in different ways. Many values can be included in a table, and when you drill down in a table, you make good use of the limited space on the sheet. A table is excellent when you want to see exact values rather than trends or patterns. Tables are an easy way to export data into other programs.

Disadvantages

If the straight table contains many values, it is difficult to get an overview of how values are related. It is also hard to identify an irregularity within the table.

Best practices for optimizing tables

If a table has too many dimensions and measures, it may load slowly. As a best practice, add the majority of your fields and expressions as Alternative columns. This encourages users to add only the fields they need using chart exploration.

You can also increase performance by ensuring a table has 10 columns or less.

Tables that extract fields from a single data source will perform better than tables that contain fields from multiple data sources.

Creating a straight table

You can create a straight table on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects** > **Qlik Visualization bundle** and drag a **Straight table** object to the sheet.
2. Click **Add columns** to add items to the table. A dropdown will open:
 - **Fields & master items**: Opens a searchable dialog box displaying every field and master item. You can choose to add any item as a dimension or measure.
 - **Custom expression**: Opens a dialog box where you can type in an expression, or open the expression editor **fx**.
3. In the properties panel, under **Data**, click **+** to add more dimensions or measures to the table.

When you have created the table, you may want to adjust its appearance and other settings in the properties panel.



By default, column widths are all set to one value, so that the sum of them equals the width of the chart. You can adjust the width of a column by dragging the header divider. Double-click the header divider to reset to the default width.

Working with table items

Do the following:

1. Under **Columns** and **Alternative columns**, click the checkbox next to any item and then **...** to perform actions such as cut, paste, and delete. Click the checkbox next to **Columns** or **Alternative columns** to select all items in the list. Use the arrow icons to move items between each section.



Alternative columns are columns that users can choose to add to the table when using Using chart exploration (page 394).

2. To change the column order, click **...** next to a field or expression and drag the item .

3. Click on an item name to open its individual properties. Here you can change the label, set a background color for the column, control text alignment, set column width, and more.



*If **Text alignment** is set to **Auto**, column data is aligned according to data type: text values are left-aligned and number values, including date related values, are right-aligned. If you set it to **Custom**, you can align the data to the left, center, or right.*

Data tab in the properties panel when a straight table is selected. The Customer field was clicked, so related dimension properties are displayed on the left.

The screenshot shows the Qlik Sense properties panel with the 'Data' tab selected. On the left, under 'Field', the 'Customer' dimension is selected. The 'Label' field also contains 'Customer'. A checked checkbox for 'Include null values' is present. Under 'Presentation', there are sections for 'Show column if' (empty), 'Background color expression' (empty), 'Text color expression' (empty), 'Text alignment' set to 'Auto', and 'Column width' set to 'Auto'. On the right, the 'Columns' section lists several fields: Customer (selected), Region, Sales Rep Name (checked), Date, Sum(Sales) (checked), and Sum(Cost). Below this is a sorting section with up and down arrows and an 'Alternative col...' button. At the bottom, there are buttons for 'Customer Number' and 'Invoice Date'.

Sorting the table

You can adjust the sorting of the table in several ways:

- Column sorting: Adjust the order of the dimensions and measures from left to right .
- Row sorting: Adjust the sorting priority order of the rows.
- Internal sorting: Use the internal sorting order of dimensions and measures.
- Interactive sorting: During analysis, you can click on a column header to sort the table.

Column sorting

By default, the order in which columns are sorted is set by the order in which dimensions and measures are added to the table. If you add the measure *Sales* first, it is presented first (leftmost) in the table. The next dimension or measure that is added is presented in the second column, and so on. The column sorting order can be changed in the properties panel, under **Columns**.

Row sorting

By default, rows are sorted by the first added dimension or measure, numeric values descending, text values ascending. A small arrow under the column header shows by which column the table is sorted.

You can change the row sorting in the properties panel, under **Sorting**. Drag the dimensions and measures to change the sorting priority order. In many cases, sorting is not only affected by the first dimension or measure in **Sorting**, but also the following ones.

Example:

In the following screenshot, the rows are first sorted by *Customer*, then by *Month*, and then by *Product Type*. As you can see, the columns *Customer* and *Month* have several rows with the same values (*A-2-Z Solutions* and *Month*). The rows in *Product Type* are ordered alphabetically, but only those that were sold in January to the customer *A-2-Z Solutions* are displayed.

Customer	Month	Product Type	Sales
Totals			\$104,852,674.81
A-2-Z Solutions	Jan	Baking Goods	\$248.83
A-2-Z Solutions	Jan	Beer and Wine	\$129.25
A-2-Z Solutions	Jan	Breakfast Foods	\$68.29
A-2-Z Solutions	Jan	Canned Soup	\$45.24
A-2-Z Solutions	Jan	Carbonated Beverages	\$187.42
A-2-Z Solutions	Jan	Dairy	\$8,262.54
A-2-Z Solutions	Jan	Specialty	\$686.59
A-2-Z Solutions	Feb	Beer and Wine	\$24.60
A-2-Z Solutions	Feb	Breakfast Foods	\$270.72
A-2-Z Solutions	Feb	Canned Soup	\$91.80

By changing the sorting order, so that secondary sorting is by *Product Type*, followed by *Month*, all *Product Type* items sold to the customer *A-2-Z Solutions* are presented in alphabetical order, whereas only the months when they were sold are displayed under *Month*.

Customer	Product Type	Month	Sales
Totals			\$104,852,674.81
A-2-Z Solutions	Baking Goods	Jan	\$248.83
A-2-Z Solutions	Baking Goods	Jul	\$1,318.04
A-2-Z Solutions	Baking Goods	Nov	\$396.00
A-2-Z Solutions	Beer and Wine	Jan	\$129.25
A-2-Z Solutions	Beer and Wine	Feb	\$24.60
A-2-Z Solutions	Beer and Wine	Apr	\$129.25
A-2-Z Solutions	Beer and Wine	Jun	\$60.10
A-2-Z Solutions	Beer and Wine	Jul	\$129.25
A-2-Z Solutions	Beer and Wine	Oct	\$400.65
A-2-Z Solutions	Beer and Wine	Nov	\$10.09
A-2-Z Solutions	Beer and Wine	Dec	\$63.07
A-2-Z Solutions	Bread	Jul	\$158.56
A-2-Z Solutions	Bread	Oct	\$74.73

Internal sorting

Each dimension and measure has a default (**Auto**) internal sorting order, which can be changed. Under **Sorting**, click the item you want to change and click the button to switch to **Custom** sorting. Changes made to the internal sorting of an item may not have any effect if the sorting is in conflict with an item with higher priority.

Interactive sorting

During analysis, you can set which column to sort on by clicking the column header. The first click sorts the table according to the default sorting of the selected item. A second click reverses the sorting order.

Interactive sorting is session based and is not saved. If you want your changes to the sorting to be persistent, you need to make the changes in the properties panel.

Users with **Can edit** rights can sort interactively using chart exploration, click **Edit sheet**, and save those changes to the original table

Working with add-ons

Straight tables have the following options under **Add-ons** in the properties panel:

Data handling:

- **Include zero values:** When unselected, measures that have the value ‘0’ are not included in the presentation. If there is more than one measure value, all the measure values must have the value ‘0’ to be excluded from the presentation.
- **Calculation condition:** Specify an expression in this text field to set a condition that needs to be fulfilled (true) for the object to be displayed. The value may be entered as a calculated formula. For example: `count(distinct Team)<3`. If the condition is not fulfilled, the message or expression entered in **Displayed message** is displayed.

A calculation condition is useful when a chart or table is slow to respond due to a large amount of data. You can use the calculation condition to hide an object until the user has filtered the data to a more manageable level by applying selections. Use the **Displayed message** property to guide the user to filter the data.

[Using Calculation Conditions](#)

Styling the straight table

Straight tables have the following options under **Appearance** in the properties panel:

General

- **Show titles:** **On** by default in all visualizations except filter panes, KPIs, and text & image visualizations.
Enter **Title**, **Subtitle**, and **Footnote**. By default, the string is interpreted as a text string. However, you can also use the text field for an expression, or a combination of text and expression. An equals sign (=), at the beginning of a string shows that it contains an expression.
Click  if you want to create an expression by using the expression editor.



Titles are displayed on a single line. If you inject line breaks they will be ignored.

- **Show hover menu:** Turn this toggle on if you want the hover menu to display in the visualization.
- **Show details:** Set to **Show** if you want to allow users to be able to choose to view details, such as descriptions, measures, and dimensions.
- **Show expressions:** Set to **Show** if you want to allow users to be able to choose to view expressions.

Alternate states

State: Set the state to apply to the visualization. You can select:

- Any alternate state defined in **Master items**.
- **<inherited>**, in which case the state defined for the sheet is used.
- **<default state>**, which represents the state where no alternate state is applied.

Presentation

- **Styling:** You can change the styling of the table by clicking on  **Styling**. You can reset your styles by clicking .

- **General:** Style your visualization with the following settings:
 - **Title:** Set the font, emphasis style, font size, and color for the title in this visualization.
 - **Subtitle:** Set the font, emphasis style, font size, and color for the subtitle in this visualization.
 - **Footnote:** Set the font, emphasis style, font size, and color for the footnote in this visualization.
- **Chart:** Customize the styling of the table, overriding the app theme. You can add custom header, content font sizes, row height (in lines), and colors. You can set rows to be highlighted when hovered over and set colors for the row and font.
- **Totals:**
 - **Auto:** The totals (the result of the expression), are automatically included at the top of the table.
 - **Custom:** Select whether to display the totals and where to display them, at the top or bottom.
- **Totals label:** Set the label for the totals row. You can also use an expression as a label.
- **Use pagination:** Instead of displaying all rows at once, only a certain amount of rows are displayed. Users can use the arrow buttons at the bottom of the table to navigate.

Using chart exploration

Chart exploration allows app consumers and other users that do not have edit rights to customize the original straight table when they are in analysis mode. It is available in the  menu under  **Chart exploration**.

App consumers and viewers can use chart exploration to add or remove columns from a table, re-sort columns, change column width, and apply selections. You cannot change the size or layout of the entire table on the sheet in chart exploration mode.

Chart exploration mode is a great way to quickly remove or add data, and then share it, download it, or bookmark the new table state. It is very helpful in apps that have many viewers with different needs. The chart exploration panel is not shown in the resulting table that you have shared or downloaded.

If you customize a table using chart exploration mode, other users cannot see your changes, unless you save them as a public bookmark. This means that several users can alter the same table, at the same time. Your changes will remain visible to you if you refresh your browser page, but will be lost if you log out or your session times out. If this occurs, the table will return to its default state, as set by the person who created the straight table. If you want to save your table layout, create a bookmark.

Users with **Can edit** rights can make changes to the table using chart exploration, click **Edit sheet**, and save those changes to the original table.

App developers can turn on **Chart exploration** in the properties panel:

- **Enable chart exploration:** Toggle this on to allow chart exploration.
- **Visibility option:**
 - **Auto:** Chart exploration panel is visible when users open the sheet.
 - **Minimized:** Chart exploration is turned on, but not visible when users open the sheet. Users can open it in the hover menu by clicking  and then  **Chart exploration**.

For a table item to be available in chart exploration mode, the table creator (or a user with **Can edit** rights) must have added those fields, master items, or expressions to the table as columns or alternative columns. For more information, see *Working with table items* (page 388).

Chart exploration of a straight table. The table has three columns: Customer, Region, City.

The screenshot shows the Qlik Sense Chart exploration interface. On the left, there is a sidebar with a search bar and a tree view of available fields. The tree view includes categories like Customer, Region, Sales Rep Name, Date, Sales, Cost, Profit margin, Manager, Product Group, Gross sales, Customer Number, Invoice Date, City Code, Desc, Item Desc, and Manager Number. Some fields like Customer, Region, and City are checked. The main area displays a table titled "Total sales and cost" with three columns: Customer, Region, and City. The table contains 685 rows of data, with the first few rows being: A Superior System (USA, New York), A-2-Z Solutions (Spain, Madrid), A-ARVIN Laser Resources (UK, London), A&B (Spain, Barcelona), A&G (USA, Los Angeles), A&R Partners (UK, Birmingham), A1 Datacom Supply (Spain, Valencia), a2i (UK, Leeds), A2Z Solutions (Japan, Tokyo), AA-Wizard (Nordic, Stockholm), Aadast (USA, Chicago), Aaron D. Meyer & Associates (Japan, Yokohama), Aaron Products (Japan, Osaka), Abacus Niagara (Nordic, Gothenburg), Abbotsbury (Japan, Nagoya), Abbott (UK, Glasgow), Aberdeen (USA, Houston), ABI TruTrac (USA, Philadelphia), AboveNet (USA, Phoenix), Abplus (USA, San Antonio), ABSolute (USA, San Diego), Absolute Magic (USA, Dallas), Abstract (USA, San Jose). At the bottom of the table, there are pagination controls for "Rows per page" (set to 100), "Select page" (set to 1), and "1 - 100 of 685".

Total sales and cost			
Customer ↑=	Region	City	...
A Superior System	USA	New York	
A-2-Z Solutions	Spain	Madrid	
A-ARVIN Laser Resources	UK	London	
A&B	Spain	Barcelona	
A&G	USA	Los Angeles	
A&R Partners	UK	Birmingham	
A1 Datacom Supply	Spain	Valencia	
a2i	UK	Leeds	
A2Z Solutions	Japan	Tokyo	
AA-Wizard	Nordic	Stockholm	
Aadast	USA	Chicago	
Aaron D. Meyer & Associates	Japan	Yokohama	
Aaron Products	Japan	Osaka	
Abacus Niagara	Nordic	Gothenburg	
Abbotsbury	Japan	Nagoya	
Abbott	UK	Glasgow	
Aberdeen	USA	Houston	
ABI TruTrac	USA	Philadelphia	
AboveNet	USA	Phoenix	
Abplus	USA	San Antonio	
ABSolute	USA	San Diego	
Absolute Magic	USA	Dallas	
Abstract	USA	San Jose	

Rows per page: 100 | Select page: 1 | 1 - 100 of 685 | < < > >|

Chart exploration of the same table as above, but with two measures added to the table: Sales and Cost. The background colors are the result of an expression.

The screenshot shows a 'Chart exploration' interface with a sidebar on the left containing a search bar and a list of dimensions and measures. The main area displays a table titled 'Total sales and cost' with the following data:

Customer	Region	City	...	Sales	Cost
Totals				\$104,852,674.81	\$61,571,564.69
A Superior System	USA	New York		\$103,728.12	\$61,464.03
A-2-Z Solutions	Spain	Madrid		\$196,298.49	\$120,886.20
A-ARVIN Laser Resources	UK	London		\$4,053.05	\$2,515.87
A&B	Spain	Barcelona		\$92,120.60	\$53,402.92
A&G	USA	Los Angeles		\$12,502.61	\$6,616.37
A&R Partners	UK	Birmingham		\$30,392.45	\$20,028.79
A1 Datacom Supply	Spain	Valencia		\$259,599.52	\$155,091.57
a2i	UK	Leeds		\$451.64	\$181.39
A2Z Solutions	Japan	Tokyo		\$69,977.36	\$41,139.03
AA-Wizard	Nordic	Stockholm		\$94,209.44	\$50,301.75
Aadast	USA	Chicago		\$351,243.31	\$221,027.86
Aaron D. Meyer & Associates	Japan	Yokohama		\$90,017.11	\$50,372.25
Aaron Products	Japan	Osaka		\$4,901.96	\$3,152.51
Abacus Niagara	Nordic	Gothenburg		\$48,161.07	\$26,484.39
Abbotsbury	Japan	Nagoya		\$4,556.70	\$2,409.89
Abbott	UK	Glasgow		\$15,036.77	\$9,265.99
Aberdeen	USA	Houston		\$319,388.90	\$184,554.70
ABI TruTrac	USA	Philadelphia		\$14,082.35	\$7,691.37
AboveNet	USA	Phoenix		\$1,395.72	\$1,089.46
Abplus	USA	San Antonio		\$8,848.56	\$4,582.28
ABSolute	USA	San Diego		\$4,319.23	\$2,349.73
Absolute Magic	USA	Dallas		\$73,982.46	\$41,200.92
				\$9,649.19	\$4,939.97

At the bottom, there are pagination controls: 'Rows per page: 100', 'Select page: 1', '1 - 100 of 685', and navigation arrows.

Limitations

Number of rows displayed

If pagination is turned on, you can only display 100 rows at a time. If pagination is turned off, you can display up to 250 000 rows at a time. If your table has more than 250 000 rows, pagination will be applied.

Because huge tables are impractical and hard to manage, the limit for what is practical is far less than the theoretical maximum. In most cases, it is desirable to see all the columns without scrolling horizontally.

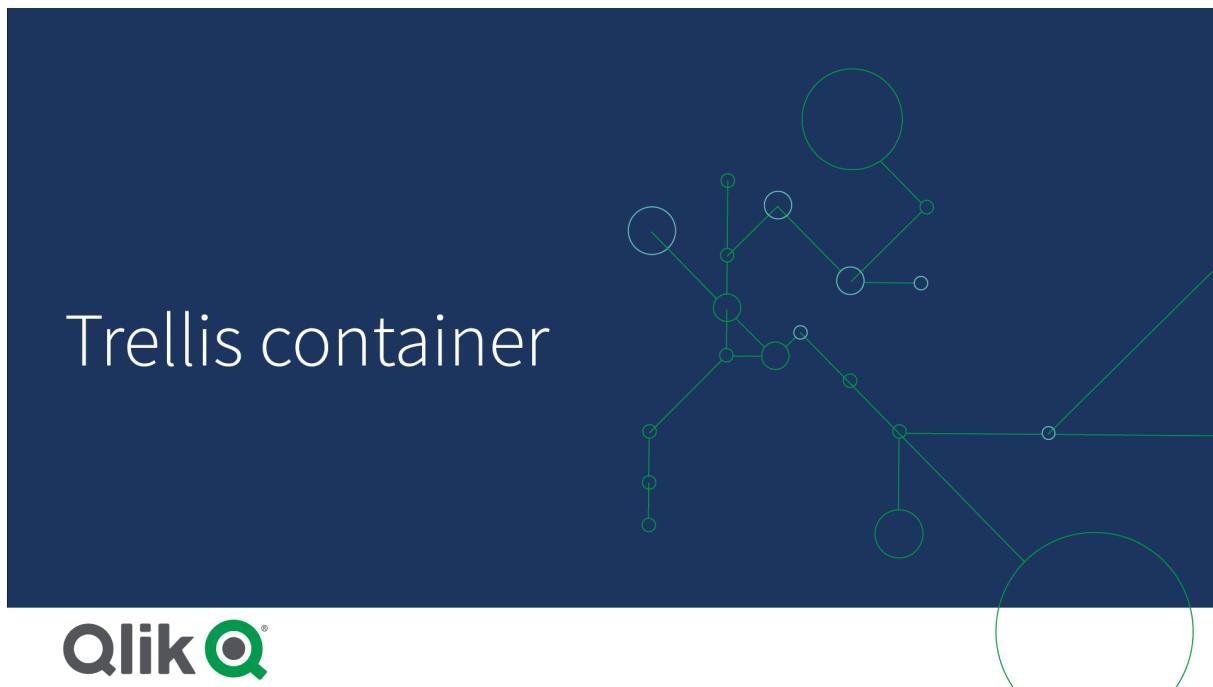
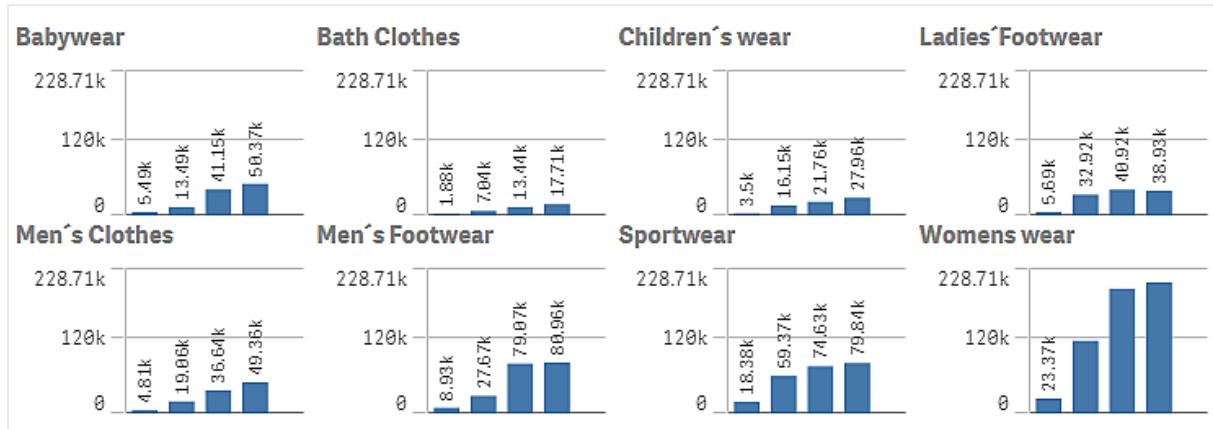
Accessibility

The straight table is only fully accessible if pagination is turned on.

Trellis container

You can use the Trellis container to show a trellis chart based on a master visualization. The trellis chart shows a grid of the same chart for different dimension values. You can use the trellis chart to compare metrics for different groups of data. The Trellis container is included in the Visualization bundle.

Trellis chart based on a bar chart of sales data, shown for different product categories



Creating a trellis chart

You can create a trellis chart on the sheet you are editing. The chart that you want to repeat for different dimension values must be a master visualization.

Do the following:

1. Create the chart that you want to repeat for different dimension values and save it as a master visualization.
2. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Trellis container** object to the sheet.
3. Select which master visualization to show in **Appearance > Trellis options > Base visualization** in the property panel.

4. Select the dimension that you want to use as the first grid dimension in **Data > Dimensions**.
5. Optionally, add a second grid dimension to create a grid with one dimension in each axis.

You will now see a grid of charts, each based on the grid dimensions you selected.

Troubleshooting

I do not see a grid of charts, and receive this message: **Too many dimension values!**

Possible cause

There are more dimension values than the maximum number of charts allowed in the grid.

Proposed action

You can increase the number of charts allowed in **Appearance > Trellis options > Maximum number of charts**.

Changing the appearance of the chart

You can customize your chart with one or more features.

Setting the number of columns

You can set the number of columns in the chart grid with **Appearance > Trellis options > Number of columns**.

Setting the y-axis range of charts

You can select which y-axis range to show for the charts in the trellis chart with **Appearance > Trellis options > Auto range**.

- **On** will show the same range for all charts. This is the best option if you want to compare values between different charts.
- **Off** will show an optimized range for each chart.

Show border

You can show a border for the trellis chart with **Appearance > Trellis options > Border**. You can adjust width, color and style. It is also possible to define a custom border.

Slide mode

You can view the charts in slide mode instead of a grid by setting **Appearance > Trellis options > Slide mode** to **On**. In slide mode, you view one chart at a time. You can scroll between the charts.

Best practices for creating the master visualization

Here are some tips for creating a master visualization that will work well in a trellis chart:

- Show the title in the master visualization. In the trellis chart, the title is replaced with the grid dimension value of each chart.
- If you use expressions: create a label to hide the set expression.
- If you are using a combo chart: use only one axis, and set the minimum and maximum of the y-axis.
- If you are using a box plot: set the minimum and maximum of the y-axis.

- If you are using a scatter plot: set the x-axis.
- If you are using a map: use advanced mode, and set the layer color and title.

Using advanced mode

You can use advanced mode to specify where to insert set analysis and dimension values in the master visualization. Activate it by setting **Appearance > Trellis options > Advanced mode** to **On**.

You can use the following placeholders in formulas in the master visualization. They will be replaced in the trellis chart by the corresponding values:

Advanced mode value replacements in a trellis chart

Placeholder	Replaced by value
<code>\$(vDim)</code>	Dimension Name
<code>\$(vDimValue)</code>	Dimension Value
<code>\$(vDimSet)</code>	<code>,[Dimension Name]={'Dimension Value'}</code>
<code>\$(vDimSetFull)</code>	<code>{<[Dimension Name]={'Dimension Value'}>}</code>

Limitations

For information about general limitations, see *Limitations (page 333)*.

Unsupported visualizations

You cannot use the following visualizations in a trellis chart:

- Filter pane
- Histogram

Multi KPI visualizations with embedded master visualizations

It is not possible to use a Multi KPI chart that contains an embedded master visualization.

Selecting by dimension

If you make a selection on the dimension that is used as grid dimension in the trellis chart, the selection is not reflected in the trellis chart.

Example:

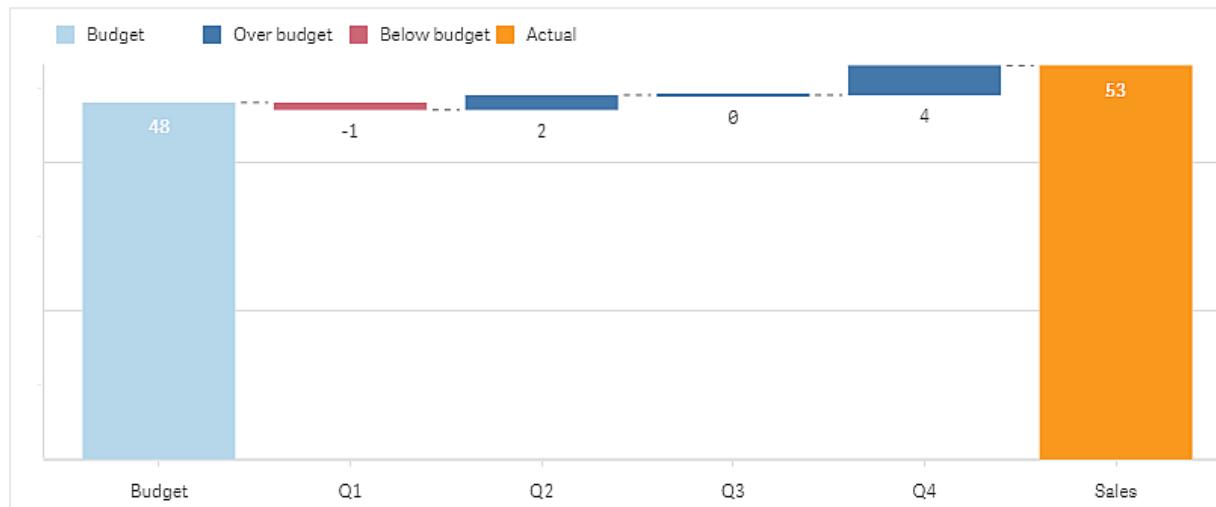
You have a filter pane with countries, and a trellis chart with country as the first grid dimension. When you select a country in the filter pane, the trellis chart will still show a chart for each country and ignore the selection.

Variance waterfall chart

You can use the variance waterfall chart (**Variance waterfall**) to show the variance between two measures over the different values of a dimension. The variance waterfall chart is included in the Visualization bundle.

You need to use two measures, start value and end value, and one bridge dimension.

Variance waterfall chart showing spending over fiscal quarters



Creating a variance waterfall chart

You can create a variance waterfall chart on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Variance waterfall** object to the sheet.
2. Click the **Add dimension** button to select the bridge dimension.
3. Click the first **Add measure** button to select the measure to use as start value.
4. Click the second **Add measure** button to select the measure to use as end value.

The variance waterfall chart is now displayed with one bar for the start value measure and one bar for the end value measure. Between the measure bars you will see the variance for each value of the bridge dimension.

Changing the appearance of the chart

You can customize the appearance of your chart.

Labels

You can turn off value labels by setting **Appearance > Presentation > Value labels** to **Off**.

Legend

You can customize the legend labels by setting **Appearance > Presentation > Labels** to **Custom**. You can set a custom text for the following legend labels:

- Start value (**Start value**)
- End value (**End value**)
- Positive variance (**Positive label**)
- Negative variance (**Negative label**)

You can also hide the legend by setting **Appearance > Colors and legend > Show legend** to **Off** or change the position of the legend with **Appearance > Colors and legend > Legend position**.

Example of a variance waterfall chart

In this simple example we will show how quarterly sales numbers contribute to sales compared to budget.

Dataset

The dataset we use contains sales numbers and budgeted sales for each quarter. You can paste it into a text file and load it in Qlik Sense.

```
Quarter,Sales,Budget
Q1,9,10
Q2,14,12
Q3,12,12
Q4,18,14
```

Visualization

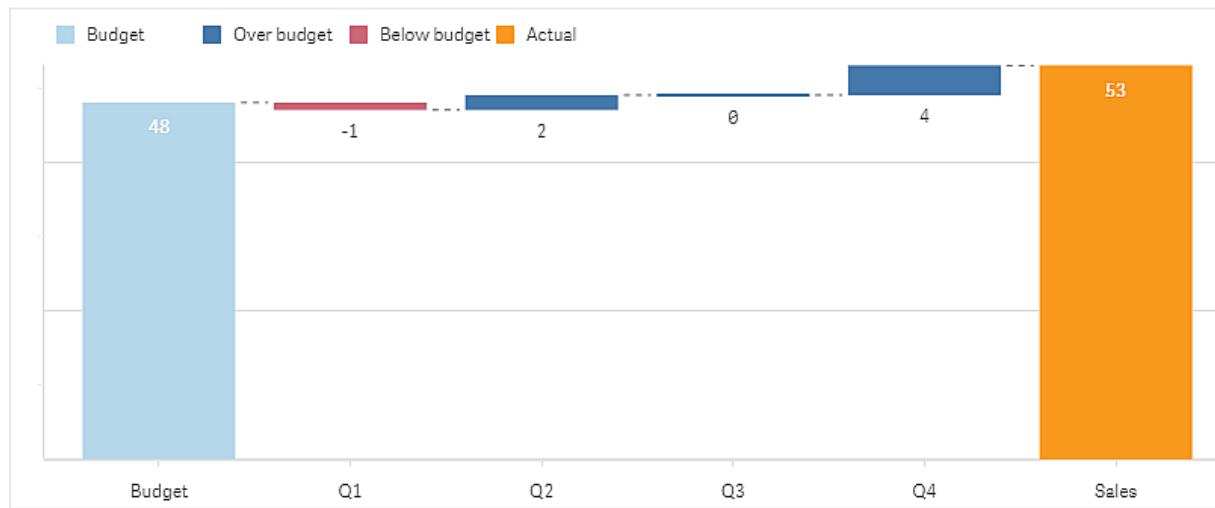
You can now create a variance waterfall chart.

- Add Quarter as dimension.
- Add Sum(Budget) as the first measure.
- Add sum(Sales) as the second measure.

The variance waterfall chart is created. We have adjusted the labels and colors in the example.

You can clearly see that Q1 had sales below budget, but the year ended with sales over budget, and that Q4 was the largest contribution.

Variance waterfall chart showing spending over fiscal quarters



Limitations

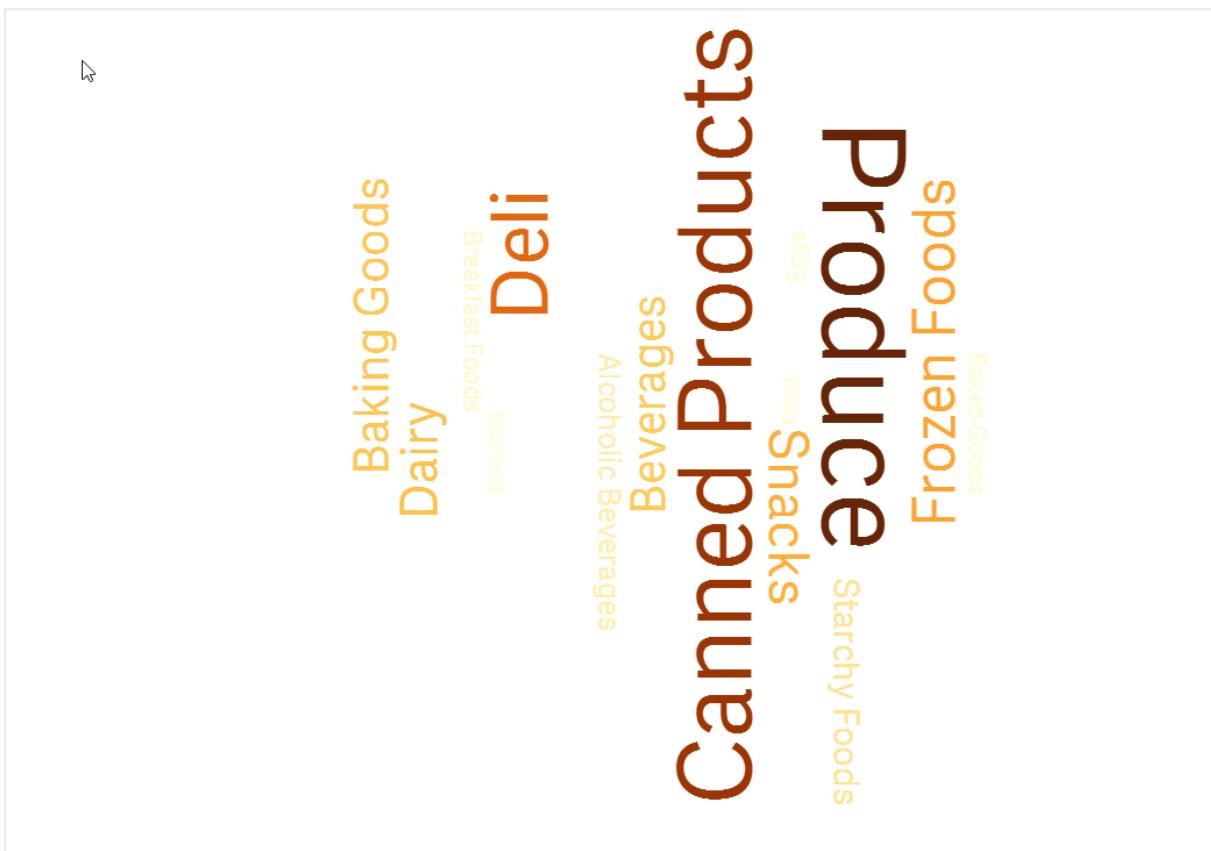
For information about general limitations, see *Limitations (page 333)*.

- It is not possible to use a variance waterfall chart in a Trellis container.
- You need to use the same number format for both measures to get correct number format for the dimension variance bars.

Word cloud chart

The Word cloud chart (**Word cloud chart**) lets you visualize text data. Text values are displayed with their size based on a measure value. The measure can be anything you want to measure against, for example: times used, alphabetically, by importance, or by context. You can customize your chart with different shapes, fonts, layouts, and color schemes. It is included in the Visualization bundle.

A word cloud chart displaying food items in different sizes and colors.





Word Cloud Chart



Requirements

Word cloud charts must have one dimension and one measure.

When to use it

A word cloud chart lets you visualize and identify the importance of a value against a measure. The more important the value is against the measure, the larger it displays in the cloud.

Creating a word cloud chart

You can create a word cloud on the sheet you are editing.

Do the following:

1. In the assets panel, open **Custom objects > Visualization bundle** and drag a **Word cloud chart** object to the sheet.
2. Click the **Add dimension** button and select the dimension.
3. Click the **Add measure** button to select the measure of the chart.

Once a dimension and a measure have been selected, the word cloud chart displays automatically.

Changing the appearance of the word cloud

You can customize your word cloud with one or more features.

Changing the orientation

You can set the number of orientations with **Appearance > Design > Orientations** in the properties panel.

You can set an integer from 1 to 10.

- 1 will display all words in the same direction, set with **Appearance > Design > Start angle**.
- 2 will display words in two orientations, **Appearance > Design > Start angle** and **Appearance > Design > End angle**.
- 3-10 will display words in the same number of orientations between **Appearance > Design > Start angle** and **Appearance > Design > End angle**.

Example:



A word cloud chart displaying food items in different orientations.

Adjusting the start and end angles

You can adjust the starting point (angle) parameter of the word cloud under **Appearance > Design > Start angle**, and the end point under **Appearance > Design > End angle** in the properties panel. The angles can have positive or negative numbers.

Changing font size

You can set maximum word font sizes under **Appearance > Design > Font max size** and the minimum under **Appearance > Design > Font min size** in the properties panel.

If you set a large font size, this can result in the larger words not being displayed in the chart as they do not fit.

Changing scale

The word cloud chart scale can be either in linear or in log scale. Select scale **Linear** or **Log** under **Appearance > Design > Scale** in the properties panel. Only positive values can be used for the log scale. Zero or negative values return nothing.

Setting custom ranges

You can also specify a range of colors or select from a predefined color scheme.

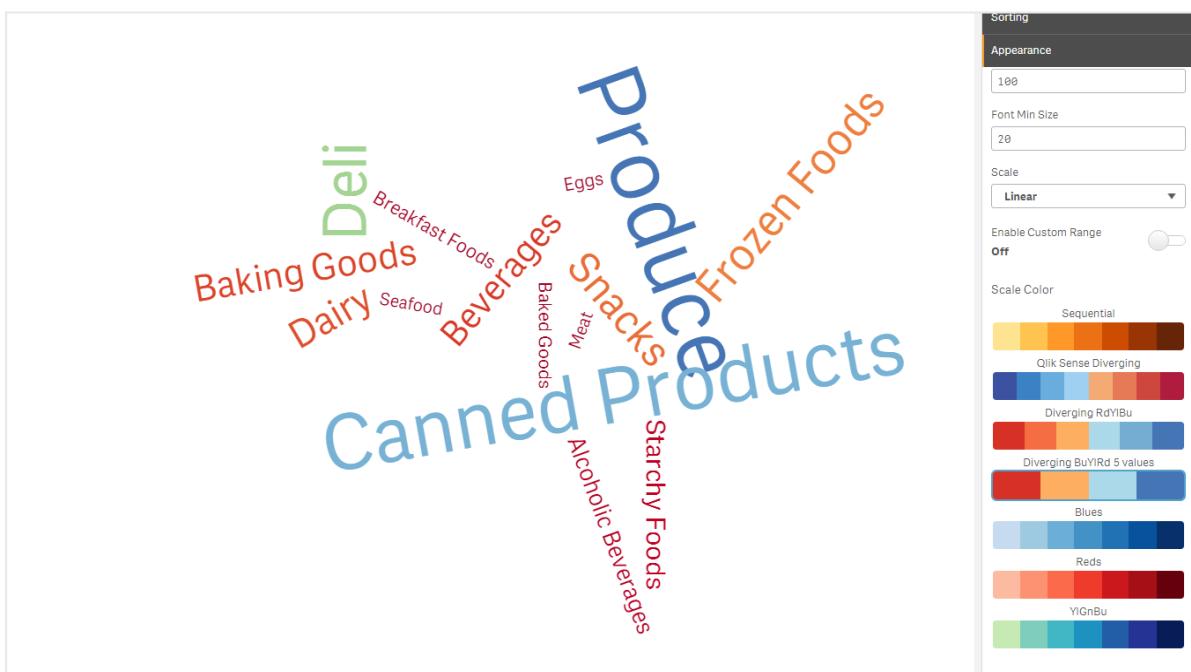
Do the following:

1. Click **Appearance > Design** in the properties panel.
2. Ensure that the **Enable color range** is set to On (default).
3. Click the color palette beside **From**, and choose a color.
4. Click the color palette beside **To**, and choose a color.

You can set your own color range by clicking the easel symbol in the color palette, and selecting a color. You can also enter a color code string in the field next to the easel symbol. The colors should be valid CSS colors.

Do the following:

1. Click **Appearance > Design** in the properties panel.
2. Move the **Enable custom range** slide button to the left to turn the option off.
3. Select a color scheme under **Scale color**.



A word cloud chart displaying food items in different sizes and colors.

Formatting numbers

It is possible to format the measure value. Different formatting can be applied to the same value, for example as money, date, duration. The chart updates to reflect the changed number type.

Do the following:

1. Click **Data > Measures** in the properties panel and click a measure.
2. Select applicable number formatting from the **Number formatting** menu.
3. Enter details in the panel fields. These display when choosing an option other than Auto when further configuring the chart.

Limitations

For information about general limitations, see *Limitations (page 333)*.

- The word cloud chart can handle maximum of 100 words per entry.
- Words that take up more space than the chart are not displayed. You can adjust the font size to show more values, but we do not recommend that you use fields with long text values.
- Word cloud charts cannot be used in Qlik NPrinting reports.

3.7 Creating and editing visualizations

You create visualizations from predefined charts, fields, or custom objects. Once added to your sheet, you can edit and refine your visualizations.

Your visualization design is guided by the data in your fields and the unlimited potential for exploring associations and correlations through measures and dimensions.

Get started here: *Creating visualizations (page 407)*

Insight Advisor offers assisted visualization creation. With Insight Advisor, you can

- Select types of analysis you want to see and generate visualizations of that type.
Creating analyses by selecting analysis types (page 413)
- Ask Insight Advisor what you want to see and let it generate visualizations.
Creating analyses using search (page 415)
- Select fields and master items and let Insight Advisor generate visualizations.
Creating analyses by asset selection (page 416)

For more information about creating visualizations with Insight Advisor, see *Creating visualizations with Insight Advisor (page 412)*.

Insight Advisor also can offer suggestions based on fields of interest when you make your own visualizations. For more information, see *Creating visualizations using Insight Advisor chart suggestions (page 434)*.

After creating a visualization, you may want to make adjustments to improve how it conveys information to users. For example, you can change the data used, or adjust the appearance of the visualization. You can add more dimensions or measures for further depth of information or remove some to improve clarity. Learn more about editing your visualizations here: *Editing visualizations (page 408)*

Adherence to design principals and clear communication of information is key to creation of apps that engage and invite data discovery. All skill levels will benefit from reviewing best practices for designing visualizations. See *Best practices for designing visualizations (page 409)*

Creating visualizations

You create visualizations by dragging the chosen type of visualization onto the sheet from the assets panel and configuring its properties settings. For instructions on creating specific types of visualizations, see that visualization type in *Visualizations (page 139)*.

Do the following:

1. Drag the visualization from the assets panel onto the sheet, or double-click the visualization.
2. Add dimensions and measures to the visualization.
You can add dimensions and measures using the buttons on the visualization. Or you can drag a field from the **Fields** tab of the assets panel, and then select it to use it as a dimension or measure. The number of dimensions and measures that are required depends on which visualization you select.
3. Adjust the presentation: for example: sorting, coloring, or labeling.
For more information, see *Changing the appearance of a visualization (page 454)*.



You can also add a visualization by copying visualizations. This is useful if you want to use existing visualization settings in another visualization type. For more information, see [Copying a visualization from an existing visualization \(page 447\)](#)

Custom objects are added in a similar manner. You start creating a visualization by dragging a visualization extension onto the sheet. For more information, see *Creating a visualization using a custom object (page 446)*.

You can use containers to save space on your dashboard, by quickly switching between tabs with different visualizations.

The types of data you have in your tables and fields impacts whether they can be used as dimensions or measures.

- Dimensions determine how the data in a visualization is grouped. For example: total sales per country or number of products per supplier. For more information, see *Dimensions (page 74)*.
- Measures are calculations used in visualizations, typically represented on the y-axis of a bar chart or a column in a table. Measures are created from an expression composed of aggregation functions, such as **Sum** or **Max**, combined with one or several fields. For more information, see *Measures (page 77)*.

Creating visualizations with assistance

Insight Advisor offers several methods for assisted visualization creation:

- The autochart automatically generates charts for you based on the fields you add to it. As you add fields, the autochart changes to the best visualization type for presenting the data added.
- You can use Insight Advisor Search to generate visualizations based on your searches or selections. You can then choose to add these visualizations to your sheets.

- You can use Insight Advisor Analysis Types to generate visualizations by picking the type of analysis you want to see and then selecting the fields. You can then choose to add these visualizations to your sheets.
- If you are using advanced options when creating a sheet, you can create visualizations using Insight Advisor chart suggestions by dragging a field onto the sheet from the assets panel and then dragging additional fields that you want in the visualization onto the first field. Qlik Sense then creates a suggested visualization based on the fields selected for the visualization. For more information, see *Creating visualizations using Insight Advisor chart suggestions* (page 434).

Insight Advisor offers several methods for assisted visualization creation.

- You can use Insight Advisor Search to generate visualizations based on your searches or selections. You can then choose to add these visualizations to your sheets.
- You can use Insight Advisor Analysis Types to generate visualizations by picking the type of analysis you want to see and then selecting the fields. You can then choose to add these visualizations to your sheets.
- You can create visualizations using Insight Advisor chart suggestions by dragging a field onto the sheet from the assets panel and then dragging additional fields that you want in the visualization onto the first field. Qlik Sense then creates a suggested visualization based on the fields selected for the visualization. For more information, see *Creating visualizations using Insight Advisor chart suggestions* (page 434).

Editing visualizations

After creating a visualization, you may want to make adjustments to improve how it conveys information to users. For example, you can change the data used, or adjust the appearance of the visualization. You can add more dimensions or measures for further depth of information, or remove some to improve clarity, and streamline a visualization.

The data in a visualization can be changed. For example, you might correct an invalid dimension or measure, or unlink a measure from a master measure so you can modify it without changing the master measure. For more information, see *Changing the data of a visualization* (page 449).

The appearance of a visualization can be edited to improve design and enhance understanding. There are a number of different ways you can adjust the appearance of your visualizations:

- Colors: Coloring is one of the best ways to highlight values in your visualizations. Qlik Sense provides a range of different coloring options.
For example, you can assign specific colors to the distinct values in a master dimension to ensure that those values use the same colors across all your visualizations.
For more information, see *Coloring a visualization* (page 462).
- Sorting: The sorting of your dimensions and measures helps ensure that content is presented in a logical and understandable manner.
For more information, see *Change the sorting of a visualization* (page 459).
- Titles and labels: Titles and labels can be changed for clarity and detail.
For example, in a pie chart showing sales by region, you could add an expression providing the total sales sum.

For more information, see *Changing the appearance of a visualization (page 454)*.

- Presentation: Different visualizations have different options that can be adjusted to enhance the display of data.

For example, you can set bars in a bar chart to display as grouped or stacked, as well as vertically or horizontally.

For more information, see *Changing the appearance of a visualization (page 454)*.

You can convert a visualization into another visualization type, and preserve your settings. For more information, see *Converting a visualization to another kind of visualization (page 482)*.

You edit visualization properties in the properties panel.

Do the following:

1. Click  **Edit sheet** in the toolbar.
The properties panel for the sheet opens to the right.
If it is hidden, click **Show properties**  in the lower right-hand corner.
2. Click the visualization that you want to edit.
The properties panel now shows the properties of the visualization.
3. Make your changes under **Properties**.
4. Make your changes in the properties panel.
5. Click  **Done editing** in the toolbar.

Best practices for designing visualizations

Decluttering your apps

Too much information in an app makes it difficult to see what is important. Today's modern user interface style is a cleaner, simpler, flatter style of design. A simplified design subtly guides the reader and allows them to stay focused.

Less is more

Users often try to include too much information in one app. Line charts with several measures can be confusing and difficult to interpret. Try creating several smaller visualizations to spread this information out onto the page. It also allows the reader to efficiently compare and contrast visualizations that are side-by-side. You can also use alternative dimensions and measures to allow the reader to quickly switch between measures without overcrowding a visualization. For more information, see *Changing the data of a visualization (page 449)*.

There are number of different ways you can improve the aesthetics and functionality of your app. Depending on your audience and what data you want to highlight, the way you design your visualization may have a serious impact on the reader's interpretation of the data.

Know your limits

Consumers of your visualizations may be working with limited screen space or resolution. Qlik Sense uses responsive design to address these limitations. However, if screen space or resolution is too limited, certain design accommodations are necessary. These can include:

- A subset of data being displayed in bar charts and line charts. When the number of dimension values exceeds the width of the visualization, a mini chart with a scroll bar is displayed.
- Collapsed menus for selecting data filters. When the number of dimension values exceeds the space available for the filter pane, the menu is collapsed. App consumers have to click on the dimension name in the title of the filter pane to open a new pane. They can then make their selections in the new pane. Additionally, when there is limited space, dimension names may appear as ellipses in filter pane titles. App consumers have to click on the ellipses to view the name of the filter pane.
- Truncated names in chart legends. Names that are too long for the space available for dimensions and measures are truncated. An ellipsis is added to the end of truncated name. App consumers can hover their mouse pointer over the truncated name to view the complete name.
- Missing legends, labels and titles.

App consumers can expand visualizations to address some of these issues. However, we recommend that you test your apps on devices where the apps may be consumed. You can also use various tools to test, such as responsive design mode in Firefox (Ctrl + Shift + M). If necessary, you can move visualizations to new sheets, reduce the amount of data shown in visualizations, and so on.

Color accessibility

The spectrum of colors is narrower for people who have color-based visual impairment. They may interpret your visualization differently than you intended.

For example, some people see the colors red and green more as yellow or brown. This form of red-green color vision deficiency is the most common. This is worth noting since red often carries a negative connotation in data visualizations, especially in finance.

A red or green KPI status can be confusing. You can use shapes with colors as performance indicators to make your designs more accessible. For example, use a red empty circle to denote bad, a green full circle for good, and a triangle as a warning symbol that only appears when a KPI status is at an unacceptable level.

Lines, bars, and pie slices can be difficult to distinguish when the colors are distorted.

For more information, see *Changing the appearance of a visualization (page 454)*.

Filter and icon placement

Filters and icons are an essential part of data visualization, but it can be difficult to know where to place them or how to sort them. You can often anticipate where the user will begin to read based on a few well-established design principles.

Left placement

Several popular websites use left-side navigation tiles and filters. This is due to the fact that many languages read from left to right. As a result, the left-side of the screen is where these readers look most frequently. Users who are scanning for content tend to gravitate toward the left side of the screen. The farther to the right objects are, the less users will look at them. If all your filters and icons are stacked vertically on the left, it gives them equal weight.

Alternately, with languages where text is written right to left, the opposite of this is true. This should be kept in mind if your apps are translated into these languages.

Top placement

Another common placement option for icons and filters is along the top of an app. By not placing filters or icons on the left it gives more space for larger visualizations with distracting menus. When filters and icons are placed above visualizations they are also seen as separate from the content below. This can help show the reader that you are prioritizing the filters or icon. If all your filters and icons are side-by-side at the top, the one furthest to the left carries more weight and is prioritized by the reader.

For more information, see *Structuring an app using sheets (page 9)*.

Information hierarchy

Sometimes you want your reader to prioritize certain visualizations over others. You can show hierarchy of information by using a few key design best practices. For example, you can use different sizes to emphasize some visualizations. Larger information is seen as more important: by increasing the font or chart size, you can show the reader where to look first.

Page placement also plays a part in information hierarchy. The information at the top of a page is perceived as more important than information at the bottom of the page because it is read first. Information on the first page is perceived as more important than information on the last page.

Adding context to KPIs

KPIs are a great way to communicate some of the big ideas inside your app. However, KPI values do not provide any context to the numbers and calculations that are happening behind the scenes. A green light next to a KPI does not tell the reader if the goal was barely achieved, or if you greatly surpassed it.

To help bring context to your KPIs, include supporting information next to the value in smaller text. For example, you can compare the current KPI value with the value from the previous year. You can also add a small bar chart without axes or values to provide information about the current trend.

Avoid the pitfalls of data visualization

To experience the benefits of data visualizations you must avoid the pitfalls. Here are some common ones:

Color abuse

Do not overdo colors. Be aware that the wrong color in the wrong place might cause confusion rather than clarity. Also, the same color may mean different things in different parts of the world.

Misuse of pie charts

Avoid having pie charts side by side to compare. Try not to squeeze too much information into them.

Visual clutter

Too much information defeats the purpose of clarity. Use a maximum of nine KPIs, and remove all visual clutter.

Style over substance

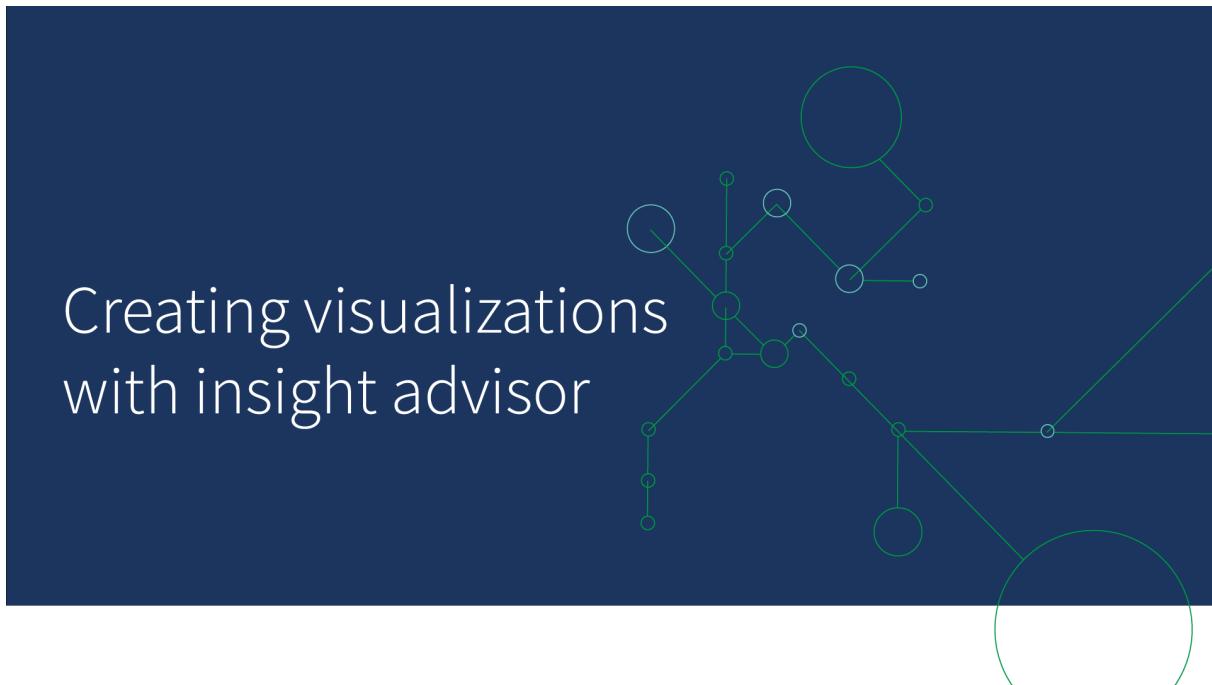
A beautiful visualization is not necessarily the most effective. Use design best practices at all times.

Bad data

Spot and correct issues with your data before you present it. Do not let your visualization take the blame for bad information.

Creating visualizations with Insight Advisor

Explore your data and create visualizations with Insight Advisor Analysis Types and Insight Advisor Search. Insight Advisor creates visualizations for you using the Qlik cognitive engine and your app's logical model. Click **Insight Advisor** in a sheet to use Insight Advisor Search and Insight Advisor Analysis Types.



Insight Advisor Analysis Types

Select the type of analysis you want to see, such as breakdown, trend over time, or mutual information. Then select the data you want used in the analysis. Insight Advisor generates charts using your choices. Insight Advisor Analysis Types is the fastest way to create charts when you know what you want to see and the data sources to use, but not how to build the analysis yourself.

Creating analyses by selecting analysis types (page 413)

Navigating Insight Advisor (page 423)

Insight Advisor Search

Enter a natural language question or statement. Insight Advisor then uses the Qlik cognitive engine to interpret your question and create visualizations to answer your questions from your data model. Insight Advisor Search is the fastest way to create charts when you have a specific question to answer.

You can also select fields and master items from the assets panel. Insight Advisor generates results based on your selections.

Creating analyses using search (page 415)

Creating analyses by asset selection (page 416)

Navigating Insight Advisor (page 423)

Users can also access Insight Advisor in the hub with Insight Advisor Chat.

Insight Advisor and the logical model

Insight Advisor creates visualizations using the Qlik cognitive engine and a logical model of your data. The logical model defines how fields in your data model should be used as dimensions or measures, as well as the relationships between fields. By default, Insight Advisor builds a logical model by learning from user interactions. If you add charts to sheets or edit charts, Insight Advisor learns these preferences.

You can also create your own custom logical model for the app. Business logic lets you define the relationship and use of data in your logical model. When business logic is turned on, precedent-based learning is not available. If you define calendar periods and set them as default periods, period analysis and period performance analysis types are available.



As of August 2022, Insight Advisor, including business logic, is no longer supported with Qlik Sense Desktop. In November 2022, Insight Advisor will be upgraded to a new experience. This will only be available on Qlik Sense Enterprise on Windows. Users who want to continue using Insight Advisor and business logic on Qlik Sense Desktop should not upgrade to August 2022.

Creating visualizations with Insight Advisor Analysis Types and Insight Advisor Search

You can generate charts using Insight Advisor in three ways:

- Select the type of analysis you want to see and the fields or master items that should be used with it. Insight Advisor Analysis Types then generates matching analyses .
- Ask Insight Advisor a question and let Insight Advisor Search generate charts using natural language analytics.
- Select fields or master items and let Insight Advisor Search generate charts through auto-analysis of your data.

The data assets available to you in Insight Advisor depend on if the app uses business logic to define its logical model and your space permission. If you have **Can view** permission will only be able to see master items, unless the app defines its logical model with business logic.

Creating analyses by selecting analysis types

Pick an analysis type and then add fields to create analyses with Insight Advisor Analysis Types. This is a fastest method of creating visualizations when you know what analyses you want to see and the data you want to use, but not how to build the analyses yourself.

Click **Create an analysis** on the Insight Advisor start page. In **Analysis options**, all available analysis types display with descriptions of their use.

3 Visualizations

Analysis options

The screenshot shows the Qlik Sense interface with the 'Insight Advisor' tab selected. On the left, there's a sidebar for 'Fields' and 'Master items'. The main area has three sections: 'Explore your data' (with a 'Calculated measure (KPI)' card showing '+3.1M'), 'Ask a question' (with a 'Ranking' card), and 'Create an analysis' (with cards for 'Breakdown (geospatial)', 'Breakdown', 'Overview', 'Trend over time', 'Comparison', and 'Relative importance'). A search bar at the top says 'Ask a question'.

After selecting an analysis type, you then select fields or master items to use in the analysis. Insight Advisor offers suggestions of assets to use for each parameter. You can change analysis types by selecting a new analysis type from the dropdown or clicking **View all analyses** to view the analysis options. You can clear your selections by clicking **Reset analysis**.

Building an analysis

This screenshot shows the 'Breakdown' analysis type being configured. The 'Fields' sidebar lists various dimensions like City, Cost, Customer, etc. The main panel shows 'Add measures: 1' with 'Sales' selected, and 'Add dimensions: 2 to 3' with 'Product Line', 'Sales Rep Name1', 'Region', and 'Product Group' selected. A note at the bottom says 'The analysis will be shown once the minimum amount of measures and dimensions have been added above...'.

Do the following:

1. In a sheet, click **Insight Advisor**.
2. Click **Create an analysis**.
3. Under **Analysis options**, select your analysis type.
4. Add the required fields or master items for your analysis type.
When you have added the required fields, Insight Advisor generates your charts.

Creating analyses using search

Ask Insight Advisor Search natural language questions and it generates relevant charts using field names, field values, and master items. You can search Insight Advisor in the following ways:

- Enter a natural language question or statement, such as *what are the top products for sales* or *Show me Product by Revenue in Sweden*. Natural language questions work best when they refer to field or master items directly.
- For guidelines on using natural language in your searches, see *Using natural language with Insight Advisor (page 426)*.
- For information about language support in Insight Advisor Search, see *Supported languages (page 420)*.
- If your browser's voice-to-text capability is supported in Insight Advisor, you can click the microphone icon to ask Insight Advisor your question.

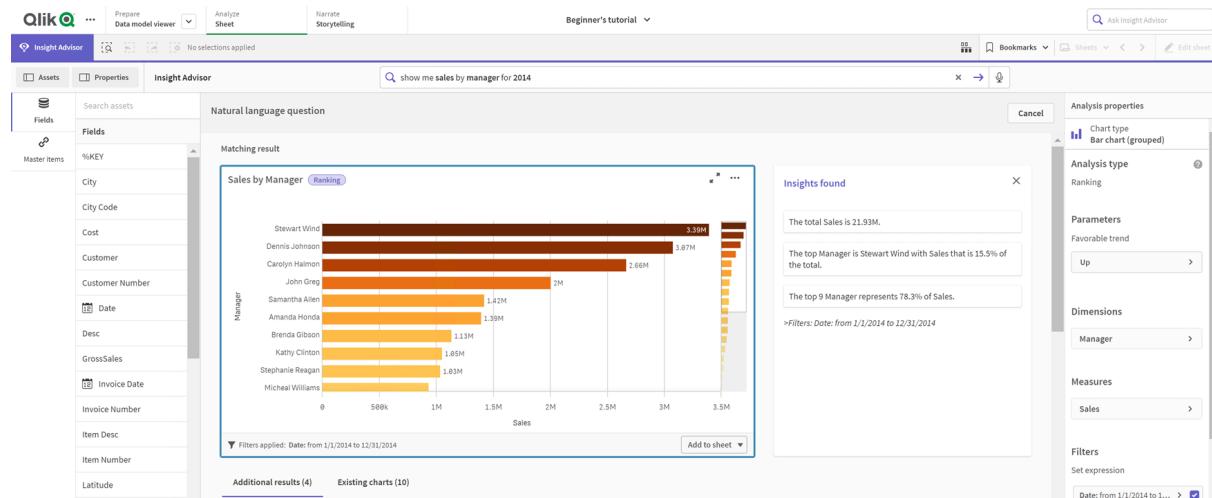


Qlik Sense Desktop does not support voice-to-text capabilities for questions.

Insight Advisor uses the fields or terms you specify. It may use additional fields in the generated visualizations. Insight Advisor uses any precedents you have given it for hiding charts, setting fields as dimensions or measures, or excluding fields from analysis. In Qlik Sense Enterprise, Insight Advisor can use precedents learned from other published apps if they use a similar or identical data model.

For natural language questions, Insight Advisor tries to find a matching Insight chart result. If one is found, it shows related results as well. You can click to view how Insight Advisor generated results from your question. .

Matching result with natural language insights



If you receive the following error message: "Unable to generate insights - Please try again later," you might not meet the CPU requirements for Insight Advisor.

Do the following:

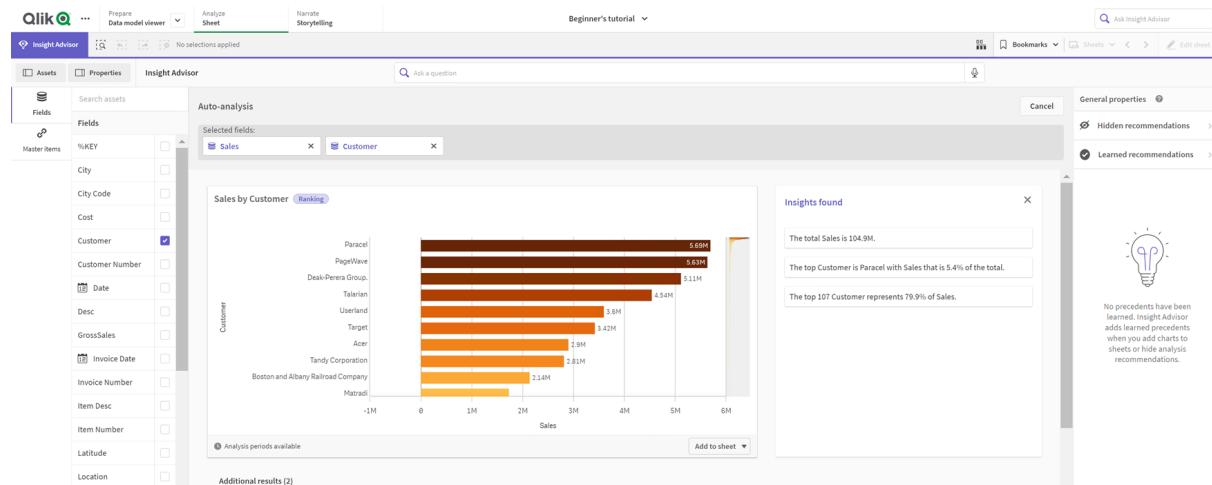
- In a sheet, enter a question or statement in the **Ask Insight Advisor** search bar and click →.
Alternatively, click **Insight Advisor**, enter a question in the Insight Advisor search bar and click →.

Creating analyses by asset selection

Select fields and master items to use in charts and let Insight Advisor Search auto-generate charts, select assets to use from the assets panel. You can also enter asset names in the Insight Advisor search bar. Insight Advisor then uses the fields you selected to generate charts. It may use additional fields in the generated visualizations.

Insight Advisor uses any precedents you have given it for hiding charts, setting fields as dimensions or measures, or excluding fields from analysis. In Qlik Sense Enterprise, Insight Advisor can use precedents learned from other published apps if they use a similar or identical data model.

Insight Advisor analyses generated by selection



Do the following:

1. In a sheet, click **Insight Advisor**.
2. In the assets panel, select fields and master items to create analyses.
Alternatively, enter field or master item names in the search bar.
As you select assets, Insight Advisor generates visualizations.

Voice-to-text data transfer considerations

In many internet browsers, Insight Advisor voice-to-text search functionality uses the [Web Speech API](#) for external processing of voice-to-text data. This is used by both Insight Advisor Search and Insight Advisor Chat. The user's internet browser may send the audio recorded by the browser API to an external source for transcription. Voice processing will be handled by the web service chosen by the user's web browser. As a result, the captured information could be sent to a third-party server. Qlik is not responsible for the data processing by this third-party.

If users want to restrict this information flow to their relevant third-party browser (turn off the voice-to-text feature), this might be achieved through specific controls, including settings within their browser of choice and other security configurations.

Using Insight Advisor analyses

You can expand an Insight Advisor analysis by selecting it or clicking . You can then make selections in the analysis.

Note the following regarding selections:

- Selections apply to all analyses.
- Selections made in charts from your sheets are kept when you open Insight Advisor.
- Selections do not affect the charts created by Insight Advisor.

If an analysis is new, you can add it to your sheets. If the chart exists in a sheet, you can click the sheet name at the bottom of the chart to go to that sheet.

Some analysis types, such as period over period dashboard, provide multiple related visualizations rather than a single chart. These show a preview of the kind of visualizations included in the analysis. Click **Open analysis** to view the analysis with data.

A period over period dashboard analysis



You can view additional options by clicking . The following options are available:

- **Full screen:** Expand a chart to make selection.
- **Edit analysis properties:** Edit the chart properties to change dimension, measures, and the chart type.
Editing Insight Advisor analysis properties (page 418)
- **Add to sheet...:** Add the chart to a new or existing sheet.
- **Add to new sheet:** Add the charts to a new sheet (period changes, period changes (detailed), period over period, period over period (selected) analysis types only).
- **Download:** Download the chart as an image, PDF, or data.
- **Hide:** Hide charts that you do not want Insight Advisor to use again. You can view the charts hidden from your current search in **General properties**.



Hide is not available in apps using business logic.

Editing Insight Advisor analysis properties

Edit Insight Advisor analysis by selecting a chart and clicking . The following properties are available in **Analysis properties:**

- **Chart type:** If available, select a different chart type. For an outline of the alternate chart types available for the different analyses, see *Insight Advisor analysis types (page 427)*.
- **Favorable trend:** (Rank analysis only) Indicate if the favorable trend for ranking is to increase or decrease.
- **Parameters:** Make adjustments to parameters specific to the analysis type of the chart.
- **Dimensions and Measures:** Change the fields used as **Dimensions** or as **Measures**. You can change the measure aggregation. Reorder measures or dimensions by dragging them.
Dimensions and measures from Insight Advisor analysis can be added to your master items. Click a dimension or measure and then click **Add new**.
If you specified filters in a natural language question, such as specific field values, you can modify them as well.
- **Details:** View information about why the chart was generated and the precedents Insight Advisor has learned from a chart. Select details by clicking them to change or reject them. You can set fields as dimensions or measures or exclude fields from future analysis. You can hide charts from future analysis by selecting **We suggested a chart for this combination of data** and clicking **Hide this chart**.
- **Analysis period:** Change which period from business logic calendar periods is applied to the chart. You can adjust the analysis period by selecting new values from **Period 1** or **Period 2**.
Select an analysis period for rank type charts to view period and period performance analysis types.



*Analysis period is only available when business logic is turned on and calendar periods have been created for the app. If there is no business logic, **Analysis period** is available if the app has date/time fields with autoCalendar derivations in the load script.*

Editing calendar period Insight Advisor analyses

If a default calendar period is assigned to a group, additional period analysis types are available for Insight Advisor analysis. Default calendar periods are created in business logic. These have different properties than other charts. The following types are available:

- **Period changes:** Shows the change of a measure from the current or latest period in the selected analysis period.
- **Period changes (detailed):** Compares a change in a measure from the current period to the previous period. The change is measured using a forecasted target value. You define the percentages for meeting the target, almost meeting the target, and missing the target.
- **Period over period:** Compares a change in a measure of the current period versus the previous period.
- **Period over period (detailed):** Compares changes in a measure over time between periods. It includes a filter pane for exploring dimension values period over period.

These analysis types have unique properties. **Period changes** and **Period over period (detailed)** have the following properties:

- **Breakdown:** Select the dimension to use with the measure to view the period changes.
- **Measures:** Select the measure for which you want to see period changes.
- **Analysis period:** Select the analysis period. You can adjust the analysis period by selecting new values from **Period 1** or **Period 2**.

Period over period has the following properties:

- **Measures:** Select the measure for which you want to see period changes.
- **Analysis period:** Select the analysis period. You can adjust the analysis period by selecting new values from **Period 1** or **Period 2**.

Period over period (detailed) has the following properties:

- **Parameters:** Set the percentage limits, expected change margin, and expected change rate.
- **Favorable trend:** Indicate if the favorable trend for the measure is to increase or decrease.
- **Breakdown:** Select the dimension to view the detailed period changes with the measure.
- **Measures:** Select the measure for which you want to see period changes.
- **Analysis period:** Select the analysis period. You can adjust the analysis period by selecting new values from **Period 1** or **Period 2**.

These analysis types have unique properties. **Period changes** and **Period over period (detailed)** have the following properties:

Insight Advisor learns your preferences for that chart if you add it to a sheet or click **Learn** after editing the chart. Precedents set by a user only apply to that user's instance of Insight Advisor.

Managing general properties

In **General properties**, you can view and edit your hidden and learned charts and the precedents you have set for generating charts.



General properties are not available when business logic is turned on.

The following properties are available:

- **Hidden recommendations** contains charts you have hidden from searches. You can click  to show the chart again in searches.
- **Learned recommendations** contains charts you selected **Learn** on after editing. You can delete learned charts by clicking .
- **Your defined preferences** shows all the preferences Insight Advisor has learned. You can delete preferences by clicking .

You can clear all hidden charts, learned charts, and preferences by clicking **Clear all**.

Supported languages

Supported questions by language

The natural language questions you can ask Insight Advisor are processed as combinations of measures and dimensions. You can also apply filters, such as a time period or location.

As shown above, Insight Advisor supports natural language questions in ten languages. Certain combinations of measures, dimensions, and filters might not be supported in all of these languages.

Examples: Questions that can be asked in all supported languages

The following are examples of the kinds of questions that can be asked in all languages that Insight Advisor and Insight Advisor Chat support. Punctuation, such as question marks, is not required.

Measures only

- Show me sales
- Show me average cost
- Show me sales and margin

Dimensions only

- Show me products
- What is sales quantity?
- What are regions?

Measures and dimensions

- What are sales by product group?
- Top 5 products by cost
- Lowest sales by product

Categorical filters

- Show me sales by manager in USA
- Show me sales not including Portugal
- What are sales of produce by region?

Time filters

- What are sales for 2022?
- Show me sales between January 2021 and March 2022

Condition or comparator filters

- Sales where quantity is < 10
- Sales where costs are between 20 and 30 dollars
- Sales where orders are less than 10 000 units

Questions without support in certain languages

The following types of questions are not supported in certain languages:

Example questions not supported in certain languages

Question variation	Example	Dutch	Polish	Russian	Swedish
Time filters	What were sales last year?	No	No	No	No
Time filters	Sales for 2021 over 2022	No	No	No	No
Contextual questions (only in Insight Advisor Chat)	(Initial question: What were sales in December?) How about for March?	No	No	No	No

Qlik Sense supports English natural language queries.

If your Qlik Sense deployment includes access to a Qlik Sense SaaS tenant, administrators can enable additional support for the following languages:

- German
- Spanish
- French
- Italian
- Dutch
- Polish
- Portuguese
- Russian
- Swedish

English is used by default for browsers not set to a supported language. The language used for queries can be changed by selecting a new language from the **Language** button. For more information, see [Enabling multi-language natural language queries in Qlik Sense Enterprise on Windows](#).

Insight Advisor Search has the following language-related limitations:

Limitations

Insight Advisor has the following limitations:

- Insight Advisor supports master items that use dollar-sign expansions in their expressions with the following limitations:
 - Expressions and captions for master items are expanded once and the results are cached. The dollar-sign expressions are not reevaluated until the expressions change or the cache expires (usually 30 days after the evaluation).
 - Dollar-sign expressions that are defined in a way that expands to values, such as =sum(sales) are not supported. These expressions cannot be combined with filters in Insight Advisor. Dollar-sign expressions must expand to expressions to be used in Insight Advisor.
- In published apps, only master items are available for use in Insight Advisor. Insight Advisor Chat can use fields from published apps when creating Insights charts. If you go to Insight Advisor using from **Explore this further**, your search may not be available.
- If a field name contains only numerical values, that field will be used when generating results from a natural language question instead of a superlative (such as top or bottom with the same numerical value).
For example, if you searched for *top 3 service providers* and one of your fields was named 3, 3 will be used in the results instead of the top 3 results for service providers.

Limitations for natural language insights

The following limitations apply to natural language insights in Insight Advisor:

- Natural language insights are not always included with supported analysis types, depending on how a natural language question is processed and the data being used.
- Natural language insights only provide a general statement when the question includes more than two filters or more than two data value filters on a dimension. The filters used are included in the response, however.

Limitations for natural language insights

The following limitations apply to natural language insights in Insight Advisor:

- Natural language insights are not always included with supported analysis types, depending on how a natural language question is processed and the data being used.
- Natural language insights only provide a general statement when the question includes more than two filters or more than two data value filters on a dimension. The filters used are included in the response, however.
- Natural language insights are not available for searches with multiple dimensions or three or more measures.

Natural language insights are available for the following analysis types:

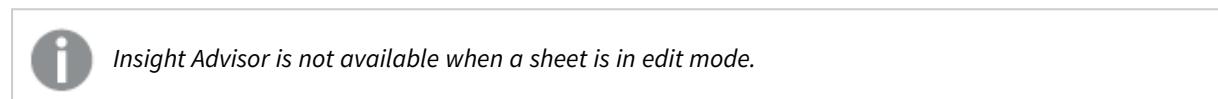
- Ranking
- Trend over time
- Comparison
- Breakdown (geospatial)
- Clustering

- Correlation
- Process control (mean)

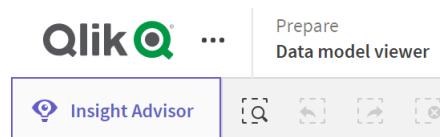
Navigating Insight Advisor

Click Insight Advisor in a open sheet to open Insight Advisor Search and Insight Advisor Analysis Types.

Insight Advisor is available from the sheet view of an app. Click the **Insight Advisor** button to open Insight Advisor.

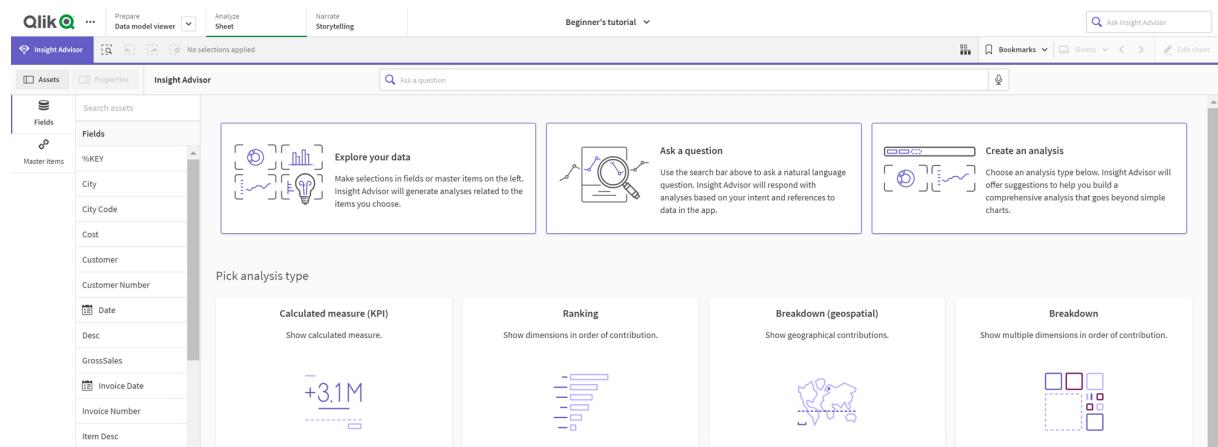


Insight Advisor button



From Insight Advisor, you can start generating analyses with Insight Advisor Search or Insight Advisor Analysis Types.

Insight Advisor



Enter a question or select fields from the assets panel to start creating analyses with Insight Advisor Search. Click **Create an analysis** to start building analyses with Insight Advisor Analysis Types. You can also select from the common analysis types to start with Insight Advisor Analysis.

Insight Advisor assets panel

In Insight Advisor Search, you can create queries by selecting fields and master items. In Insight Advisor Analysis Types, you can select fields and master items to use in your analysis.

If the app is published, only master items are available.

Insight Advisor assets panel

The screenshot shows the 'Assets' tab selected in the top navigation bar. Below it, a sidebar titled 'Master items' lists various fields and objects. The list includes: Fields, %KEY, City, City Code, Cost, Customer, Customer Number, Date, Desc, GrossSales, Invoice Date, Invoice Number, Item Desc, Item Number, Latitude, Location, Longitude, Manager, and Manager. A vertical scrollbar is visible on the right side of the list.

Insight Advisor search box

You can also access Insight Advisor Search by entering a natural language question in the search box.

Search box

The screenshot shows the Qlik Sense interface with the 'Edit sheet' mode selected. At the top, there is a search bar labeled 'Ask Insight Advisor'. Below the search bar, there are navigation buttons for 'Bookmarks', 'Sheets', and 'Edit sheet'.

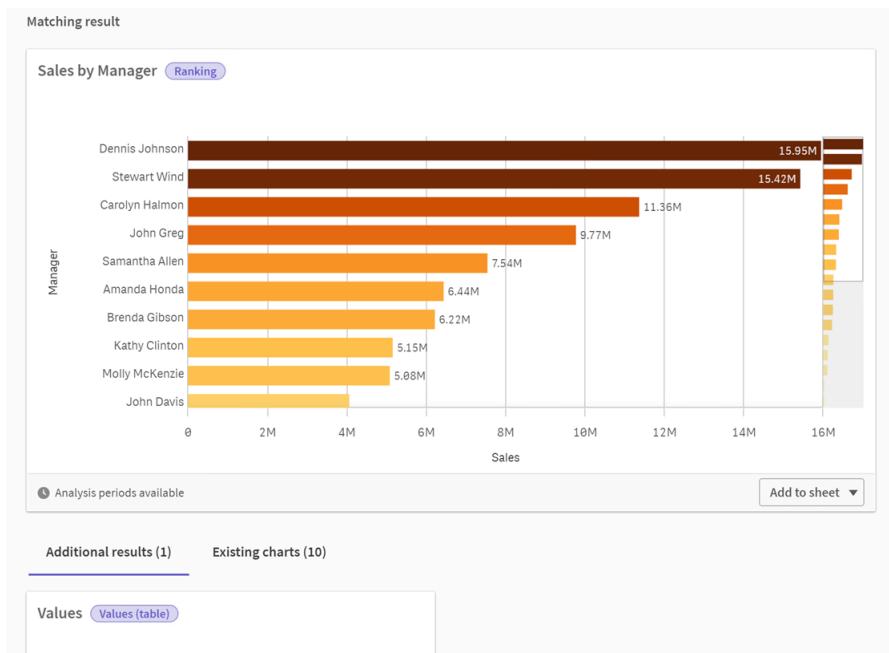
You can enter your search here, either by entering asset names or using natural language. If you used a natural language question, you can click **i** to view the filters generated from your question.

Insight Advisor analyses

These are the charts created by Insight Advisor. Insight Advisor Analysis Types shows the primary chart for your analysis type, with alternative chart types for your analysis beneath. Insight Advisor Search indicates the number of results and breaks them down as follows:

- How many results are found.
- How many charts already exist in your sheets.
- How many charts are newly generated by Insight Advisor.

Analyses generated by Insight Advisor



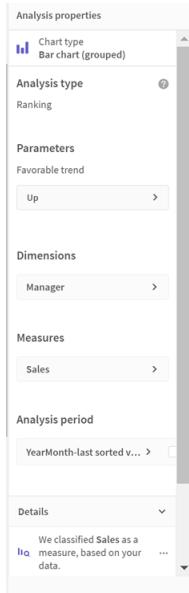
Insight Advisor charts can be added to existing or new sheets. Charts can also be downloaded as an image, PDF, or as an Excel spreadsheet containing the data used in that chart.

Using Insight Advisor analyses (page 417)

Properties panel

The properties panel contains options for editing analysis and changing the precedents you have set in Insight Advisor.

Properties panel in Insight Advisor



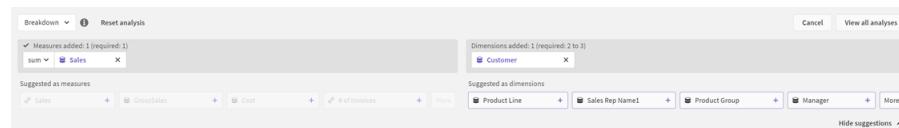
In **General properties**, you can view and edit your charts and the precedents you have set for generating analysis. For more information, see *Managing general properties (page 419)*.

In **Analysis properties**, you can edit an analysis and view details. You can see the **Analysis type** used to create the chart. Click  to learn more about the analysis type used. For more information, see *Editing Insight Advisor analysis properties (page 418)*.

Insight Advisor Analysis Types parameters

When using Insight Advisor Analysis Types, you select the parameters for your analysis. You can select from the assets panel or from the recommendations for each parameter type.

Parameters for Insight Advisor Analysis Types



Using natural language with Insight Advisor

Insight Advisor supports natural language questions, such as *Show me Product Inventory for Japan under 2500*.

Natural language questions support three kinds of filters:

- Time: A unit of time or date. For example, *Show me 2019 Sales*.
- Category: A value from one of the dimensions. For example, *Show me Sales by Product for Sweden*.
- Measure: A value or values from a measure. For example, *Show me Sales for Sweden by Product under 1K*.

You can search for facts, comparisons, and rankings. Facts are statements such as *What are my sales*, or *Show expenses over time for 2019*. You can ask for a comparison by adding *vs* or *compare* to your question. For example, *Compare sales to expenses over time*. You can ask for rankings by adding *top* to your question. For example, *Show me top 10 product by sales for 2020*.



Qlik Sense supports English natural language queries.

*English is used by default for browsers not set to a supported language. The language used for queries can be changed by selecting a new language from the **Language** button. For more information, see [Enabling multi-language natural language queries in Qlik Sense Enterprise on Windows](#).*

If your Qlik Sense deployment includes access to a Qlik Sense SaaS tenant, administrators can enable support for additional languages. For more information on supported languages in Insight Advisor Search, see [Supported languages](#).

Natural language searches need to reference field names and values in your data model. You can also tag master items with synonyms using tags on your master items. Use the format `alt:<term>` in synonym tags. If you wanted to use `cities` as a synonym, you would tag the master item `alt:cities`. For more information, see [Tagging master items \(page 117\)](#).

You can like a natural language question analysis by clicking . Liking helps improve Qlik Sense natural language responses over time.



When searching field values, natural language queries only search the first 100,000 values per field in the question.

For more information about natural language questions, see [Qlik Sense Natural Language Query features](#).

Insight Advisor analysis types

Insight Advisor provides results using a wide range of analysis types. These analysis types provide best practice visualizations for generated charts.

Different analysis types are used depending on the inputs of the search and the characteristics of your data. The Qlik cognitive engine determines the best analysis type for your search depending on the available data. The following table describes the analysis types. Not all conditions for each analysis type are listed. The table also lists charts potentially available as alternatives when editing an Insight Advisor analysis.

Analysis types

Analysis type	Description	Dimensions	Measures
Breakdown	View nested dimensions of data that provide a breakdown of relative contributions to a measure.	2-3	1
Breakdown (geospatial)	Group data by simple and hierarchical geographic divisions.	1-2	
Calculated measure (KPI)	Summarize performance in a given business segment or dimension using a key performance indicator (KPI).	0	1-2
Clustering (k-means)	Cluster data points aggregated by similarities from 2 measures over a dimension using a machine learning k-means algorithm.	1	2
Comparison	Compare two measures for a dimension.	1	2
Correlation	Identify complementary and inverse relationships between two data values.	0-2	2

Analysis type	Description	Dimensions	Measures
Mutual information	<p>Create a measure of certainty between pairs of values using a machine learning algorithm that applies random data distributions.</p> <p>The dependency indicator ranges between 0 percent (no dependency) and 100 percent(strong dependency).</p> <p>Mutual information selects one field (measure or dimension) as the target and then selects 1 to 10 dimensions or measures as drivers.</p> <p>Results for this analysis type for the same fields or selections might vary due to the random data selection.</p>	variable	variable
Overview	Describe how data ranges relate to each other in terms of an absolute measure.	1-2	1
Period changes	<p>Build a sheet with measures, rank, and comparison analysis for dimensions across different time periods.</p> <p>Requires a default calendar period set for the group containing the measure in the logical model.</p>	1-2	1
Period changes (detailed)	<p>Build a sheet with measures, rank, and comparison analysis for a hierarchy of dimensions across different time periods.</p> <p>Requires a default calendar period set for the group containing the measure in the logical model.</p>	1	1
Period over period	<p>Compare dimensions across time periods.</p> <p>Requires a default calendar period set for the group containing the measure in the logical model.</p>	1	1
Period over period (selected)	<p>Compare dimensions across time periods. It includes a filter pane for selecting dimension values.</p> <p>Requires a temporal fields with autoCalendar-derived field selected as a part of the query.</p>	1-3	1

Analysis type	Description	Dimensions	Measures
Process control (mean)	Monitor data against expected statistical ranges based on mean values.	1 date/time dimension	1
Process control (rolling mean)	Monitor data against expected statistical ranges based on nearby values.	1 date/time dimension	1
Ranking	Rank dimension values by relative importance with a measure.	1-2	1
Ranking (grouped)	Rank hierarchical dimension values by relative importance with a measure.	1-2	1
Relative importance	Show the size of dimension values that contribute to the whole. Can also be used to perform Pareto or 80-20 contribution analysis.	1	1
Trend over time	Show data trends over time, optionally broken down by a dimension with low cardinality.	1 date/time dimension and optionally 1 other dimension	1-3
Values (table)	Show data organized in rows and columns that show measures and dimensions.	0-10	0-10
Year to date	Compare dimensions over the same period in a previous year.	1	1

Editing visualizations created by Insight Advisor

Insight Advisor selects and generates charts driven by analysis types. Analysis types and chart functions are chosen based on the inputs of the search and characteristics of the data. Business analysts design the business logic components of apps and can also edit properties and extend the functions underlying the charts further. For more information, see [Business logic](#). For more information on analysis types, see .

The following examples show ways that Insight Advisor charts can be edited through properties and the expression editor to enhance visualizations that best capture and frame your data. These examples are based on charts that can be generated and then extended from the Insight Advisor business logic tutorial app. .



Screenshots for these examples are from Qlik Sense SaaS and may differ in Qlik Sense Enterprise on Windows. Screenshots in the examples may differ depending on the date on which you loaded the business logic tutorial app.

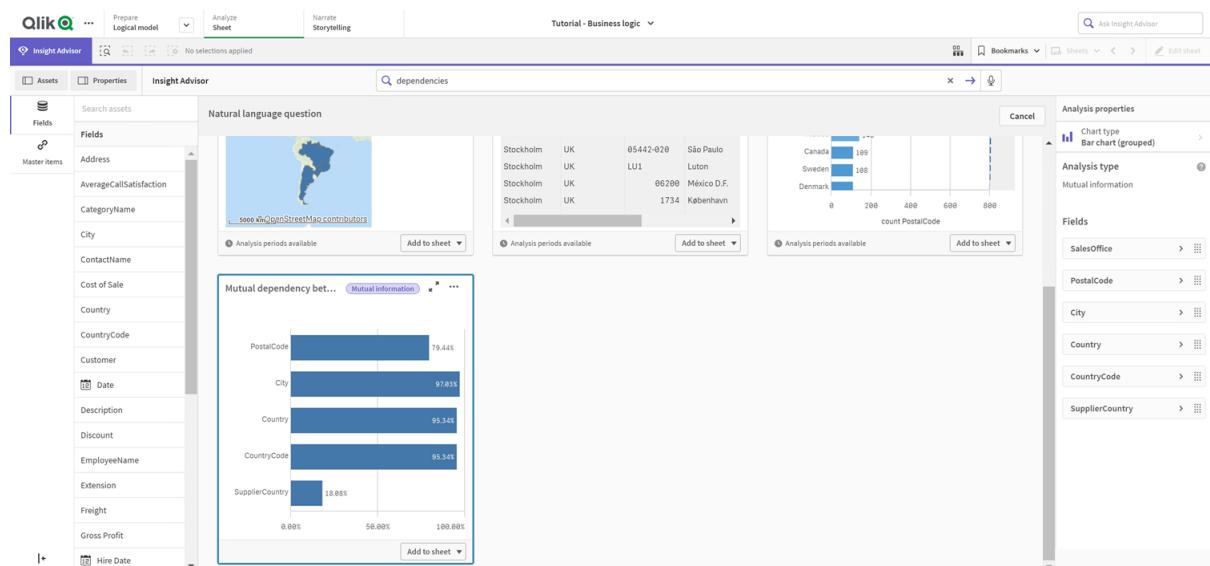
Mutual Information: Changing measures for improved charts

Statistical dependencies between a specified value (measure) and other selected items is presented in Insight Advisor by the **Mutual Information** chart. The dependency ranges between 0% (no dependency) and 100% (strong dependency). This example shows how you can replace measures to present more interesting and meaningful relationships in the chart.

Do the following:

1. In the business logic tutorial app, enter the search *dependencies* in the search box. Look for a result displaying **Mutual dependency between SalesOffice and selected items**.

Default results for Mutual Information search



2. Select **Add to sheet** in the chart, then **Create new sheet**.
 3. From the app overview, click **Sheets** to view the sheets. Select **My new sheet** and give the sheet a **Title** and **Description**. For this example, name the sheet *Mutual Information Sales* and an optional description *Mutual information between sales fields*.
 4. Click **Edit sheet** in the toolbar and select the mutual dependencies chart.
 5. Click anywhere in the chart title to change the title from **Mutual dependency between SalesOffice and selected items** to **Mutual information around sales**.
 6. Modify the fields analyzed in this chart. Insight Advisor selected four measures to include in the chart that are related to geography. This was a reasonable interpretation of what dependencies the field **SalesOffice** might have because **SalesOffice** was classified as a **city** in the logical model of the app. Modify the measures in the properties panel and change them to fields that will better reflect how **Sales** is impacted by variables related to sales quantity, cost of sales, and sales support.
 - a. Select **Data** in the properties panel and in the **Measures** section; expand the measure **SupplierCountry**. Select in the **Expression** field to open the **Edit expression** dialog. Click **Apply**.
- Edit the expression by replacing *SalesOffice* with *Quantity* and replace *SupplierCountry* with

Discount.

Select the **Label** field. Replace *SupplierCountry* with *Quantity and Sales*.

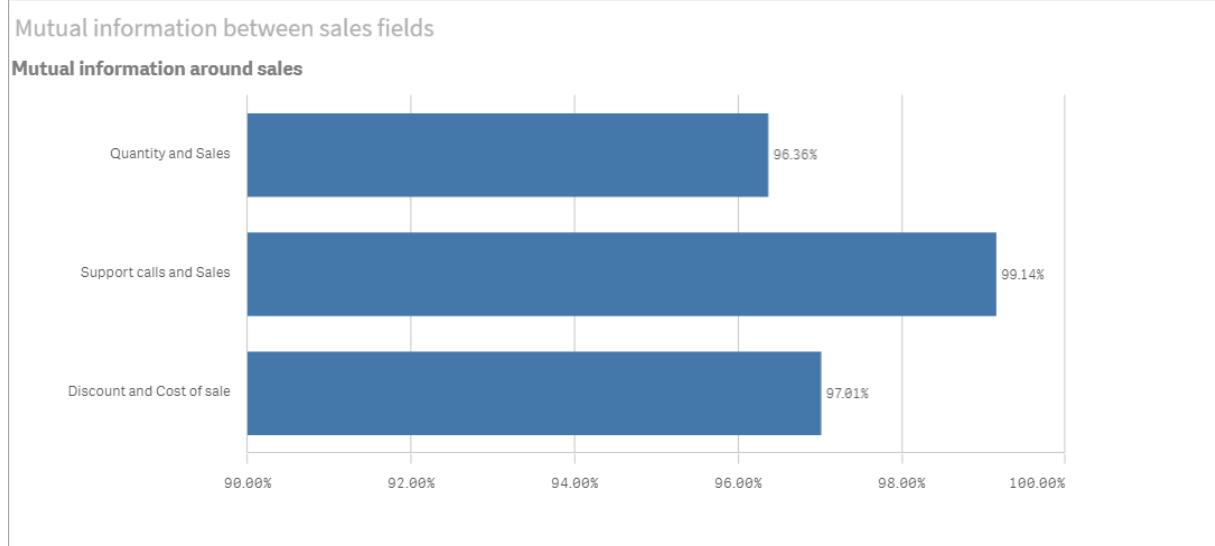
Edit field parameters

Edit expression

```
1 =MutualInfo([Quantity],[Discount], 'dd', Null(), 10000)
```

- b. Expand the measure *PostalCode*. Select **fx** in the **Expression** field to open the **Edit expression** dialog.
Edit the expression by replacing *SalesOffice* with *Sales* and replace *PostalCode* with *SupportCalls*.
Select the **Label** field . Replace *PostalCode* with *Support calls and Sales*.
 - c. Expand the measure *City*. Select **fx** in the **Expression** field to open the **Edit expression** dialog.
Edit the expression by replacing *SalesOffice* with *Discount*, and replace *City* with *Cost of Sale*.
Select the **Label** field. Replace *City* with *Discount and Cost of sale*.
 - d. Delete the measure *Country* by right-clicking on the measure and choosing **Delete**.
7. Edit the X-axis range. These three measure comparisons all represent high levels of dependency. To emphasize the differences in values and make the chart more interesting, modify the range of the bar chart:
- a. Expand the **Appearance** section and then **X-axis**.
 - b. Switch **Range** from **Auto** to **Custom**.
 - c. Select **Min/Max** for values to set. Set **Min** to 0.9 and **Max** to 1.
8. Select ✓ **Done editing**.

Mutual Information chart after editing



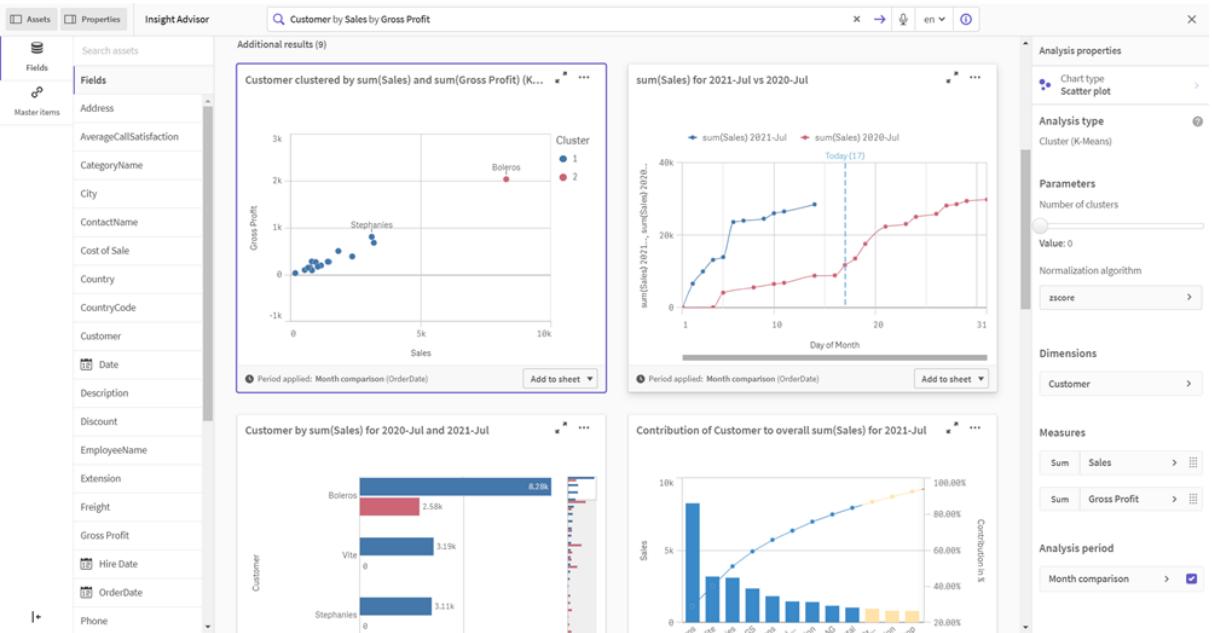
K-Means clustering: Editing the number of clusters and selections

Insight Advisor generates a **scatter plot** chart driven by k-means functions to group similar items into clusters. The following example shows how you can specify the number of clusters generated by the chart and remove outlying datapoints. For more information, see [and](#) .

Do the following:

1. In the business logic tutorial app, enter the search *Customer by Sales by Gross Profit* in the search box. Look for the result titled **Customer clustered by sum(Sales) and sum(Gross Profit) (K-Means)**.

Default results for kmeans search



2. Select **Add to sheet** in the lower right of this chart, then **Create new sheet**.
3. From the app overview, click **Sheets** to view the sheets. Select **My new sheet** and give the sheet a **Title** and **Description**. For this example, name the sheet *Customers clustered by sales* and add the description *Kmeans applied to customer data by Sales and Gross Profit*.

4. Select **Edit sheet** in the toolbar and select the k-means cluster chart.

5. Modify the scatter plot chart:
 - a. Enlarge chart: Make the chart larger to take up the canvas by dragging the lower right corner.
 - b. Edit the number of clusters argument. Expand the **Appearance** section and then **Colors and legend**. Insight Advisor has colored the clusters by dimension. Select **fx** under the **Select dimension** field to open the expression editor. Modify the **num_clusters** argument by changing the value 0 to 6 (Insight Advisor utilized auto-clustering where if 0 is entered for the number of clusters, an optimal number of clusters for that dataset is automatically determined). Select **Apply**.

Edit the first parameter in the expression (num_clusters)

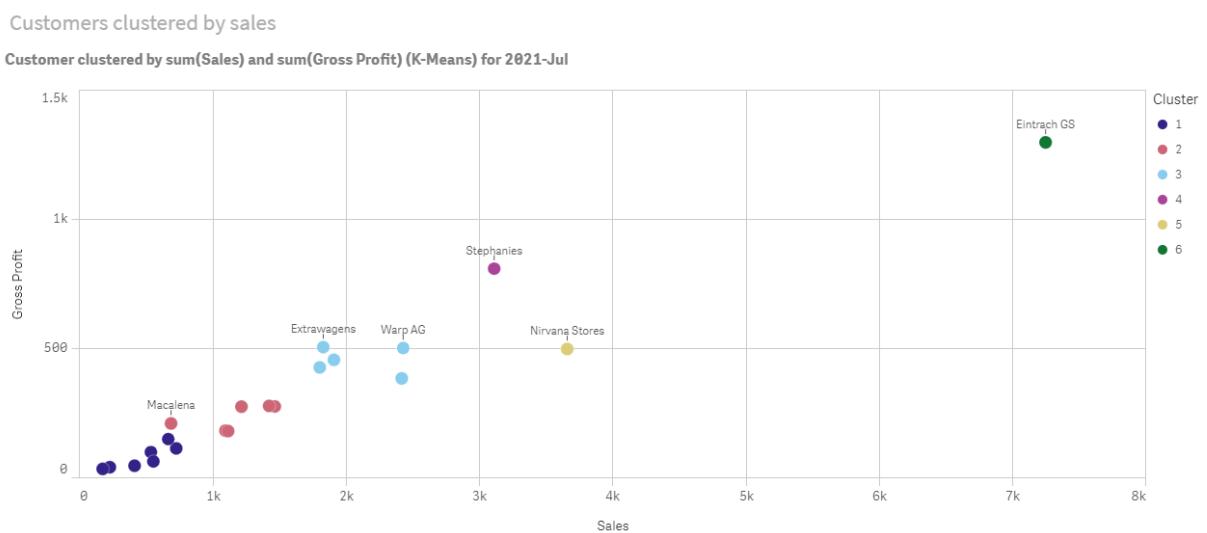
```
Edit expression
1 =aggr(KMeans2D(6,sum({<[OrderDate.autoCalendar.MonthsAgo]={1}>} [Sales]),sum({<[OrderDate.autoCalendar.MonthsAgo]={1}>} [Gross_Profit]),'zscore')+1, [Customer])
```

- c. Edit axis ranges: The default chart that is generated shows range that is less than 0. Negative numbers are not meaningful in this context and this range takes up space in the chart. In the

Appearance section, expand **X-axis: Sales** and change **Range** from **Auto** to **Custom**. Note that **Min** automatically resets to 0. Expand **Y-axis: Gross Profit** and change **Range** from **Auto** to **Custom** and note that **Min** automatically resets to 0 for the Y-axis as well.

6. Select **✓ Done editing**.
7. Deselect data: Data can be excluded from the chart by deselecting items that are not of interest. Exit edit mode by selecting **✓ Done editing** and open the **Selections tool**  . Search for **Customer** and select all customers except the following customers: **Big Foot Shoes, Boleros, Bond Ltd, El Carnevale, Fritid AB, Las Corbatas, The Fashion**, and **Vite**.

KMeans scatter plot chart after edits



Period over period analysis: Changing the analysis period

Insight Advisor supports creation of behaviors to use a preferred default calendar period with a measure group. The following example demonstrates how to change the calendar period to view results for a different period. You can change the calendar period property to generate a **Period over period** chart that displays sales over the last year rather than over the last month.

Do the following:

1. From the business logic tutorial app, enter the search *Show me sales* in the search box. Locate the **Period over period** result.

Period over period result for sales search



2. Select the **Period over period** chart to display options to the right.
3. Expand **Analysis period** in the properties panel to display calendar period options.
4. Select **Quarter comparison**.
5. Change the value in **Period 1** to **2020-Q1**.
6. Change the value in **Period 2** to **2021-Q1**.
7. Sales now displays the results for the first quarter of 2021 compared to the first quarter of 2020.

Period analysis chart after changing the Analysis period



Creating visualizations using Insight Advisor chart suggestions

Qlik Sense offers a wide range of visualizations to use with your data. Deciding on the correct chart type can be challenging when creating your first Qlik Sense app.

Insight Advisor chart suggestions enable you to select data fields and let Insight Advisor choose the dimensions, measures, and visualization types. As you add or remove fields, the suggested visualization adjusts itself based on your changes. You can customize a suggested visualization with a focused set of properties.

If you started a visualization, enabling **Chart suggestions** will change your visualization to a suggested visualization.



*If you enable **Chart suggestions** and then disable them, you will lose changes made to your visualization. You can restore your old visualization by undoing the changes. However, if you navigate away from Sheet view or make changes in the assets panel, you will not be able to undo and restore your visualization.*

Creating a new visualization using chart suggestions

Do the following:

1. Click **Edit sheet** in the toolbar.
The assets panel opens on the left-hand side.
 2. From **Fields**, drag and drop a field into your sheet.
Fields suggested as dimensions are added as tables or histograms. Fields suggested as measures are added as KPIs.
To add fields as filter panes hold the shift key when you drag and drop the field.
 3. Add additional fields by doing one of the following:
 - Drag and drop a field onto the visualization created from the first field, or on **Suggest** on the right-hand side.
Qlik Sense determines if the field should be used as a dimension or as a measure, and which measure aggregation to use.
 - Click **Add** on the properties panel and select a field.
 - Drag and drop a field on **Drop item here** under **Dimensions** or **Measures**.The visualization will change as fields are added to it.
4. Optionally, remove unwanted fields.
 5. Click **Done**.

Changing an existing visualization using chart suggestions

You change an existing visualization by adding fields or removing unwanted fields. Enabling **Chart suggestions** in the properties panel on the right-hand side changes the selected visualization to a chart suggestion based on the fields of the visualization. You can change the suggestions made by Qlik Sense. For example you can:

- Drag fields between **Dimensions** and **Measures** to change how a field is used. Moving a field to **Dimensions** removes its aggregation. Moving a field to **Measures** assigns it an aggregation.
- Choose a different aggregation for a field used as a measure. Your aggregation choice will be used

whenever you use this field as a measure, while **Chart suggestions** is enabled.

- Use **Change chart type** to choose a different type of chart from the one suggested.

Adjusting the settings when using chart suggestions

You can adjust the settings in the properties panel for **Data** and **Appearance**. The properties panel for visualizations created with chart suggestions contains a focused set of properties settings. Disabling **Chart suggestions** brings back access to all available properties. For descriptions of the available fields see the visualization properties topics in *Visualizations* (page 139).

Limitations when using chart suggestions

- You cannot enable **Chart suggestions** for a master visualization.
- You cannot enable **Chart suggestions** for filter panes, histograms or maps.
- Charts with **Chart suggestions** enabled are not supported in Qlik NPrinting reports. To include charts created using chart suggestions in Qlik NPrinting reports, disable **Chart suggestions**.
- You can only change the aggregation of a measure when its label is the default label.
- You cannot drag a master dimension to **Measures**. You cannot drag a master measure to **Dimensions**.
- You can only drag a field from **Measures** to **Dimensions** if its expression is simple. For more information on how to use expressions in visualizations see *Using expressions in visualizations* (page 122).

Guidelines for visualizations, fields, and naming

There are certain conventions and limitations you need to be aware of when working with Qlik Sense. For example: the maximum number of characters to use in names, descriptions, and expressions, as well as characters reserved for use by Qlik Sense only.

Max number of visualizations

The maximum number of visualizations there can be on a sheet is limited to the number of cells on a sheet: 288 (24x12). The maximum practical number will be less than this because of the limited use for visualizations that are made up of only 1 cell.

Upper limits on name lengths

The following limits apply to the number of characters that can be used in various situations in Qlik Sense:

Upper limits on name lengths

Situation	Upper limit
Names (title, dimension, footnote...)	Max 255 characters.
Descriptions	Max 512 characters.
Expressions	Max 64,000 characters.
Tags	Max 31 characters per tag and max 30 tags per master item.
Text & image chart:	Max 12,000 characters.

Naming conventions

You can refer to a number of entities by their names in Qlik Sense, such as:

- Fields
- Dimensions
- Measures
- Variables
- Bookmarks

Some characters are reserved for system purposes in Qlik Sense. To avoid potential errors, avoid using the following characters in names:

- :
- =
- [
-]
- {
- }
- (
-)
- \$
- ,
- `
- '

If you use long names for your dimensions and measures they will be displayed truncated. “...” will be shown to denote that part of the name is hidden.

Max number of characters in expressions

The maximum number of characters that can be written in a visualization expression is 64,000. If you attempt to build an expression with more than this number, the expression will be truncated.

Conventions for number and time formats

In many interpretation and formatting functions it is possible to set the format for numbers and dates by using a format code. This topic describes the formats for the number, date, time, and timestamp functions. These formats apply both to script and chart functions.

Number formats

To denote a specific number of digits, use the symbol "0" for each digit.

To denote a possible digit to the left of the decimal point, use the symbol "#".

To mark the position of the thousands separator or the decimal separator, use the applicable thousands separator and the decimal separator.

The format code is used for defining the positions of the separators. It is not possible to set the separator in the format code. Use the **DecimalSep** and **ThousandSep** variables for this in the script.

It is possible to use the thousand separator to group digits by any number of positions, for example, a format string of "0000-0000-0000" (thousand separator="-") could be used to display a twelve-digit part number as "0012-4567-8912".

Examples:

Example of number formats

Number format	Description
# ##0	describes the number as an integer with a thousands separator. In this example " " is used as a thousands separator.
###0	describes the number as an integer without a thousands separator.
0000	describes the number as an integer with at least four digits. For example, the number 123 will be shown as 0123.
0.000	describes the number with three decimals. In this example "." is used as a decimal separator.

Special number formats

Qlik Sense can interpret and format numbers in any radix between 2 and 36 including binary, octal and hexadecimal. It can also handle roman formats.

Special number formats

Format	Description
Binary format	To indicate binary format the format code should start with (bin) or (BIN).
Octal format	To indicate octal format the format code should start with (oct) or (OCT).
Hexadecimal format	To indicate hexadecimal format the format code should start with (hex) or (HEX). If the capitalized version is used A-F will be used for formatting (for example 14FA). The non-capitalized version will result in formatting with a-f (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code.
Decimal format	The use of (dec) or (DEC) to indicate decimal format is permitted but unnecessary.
Custom radix format	To indicate a format in any radix between 2 and 36 the format code should start with (rxx) or (Rxx) where xx is the two-digit number denoting the radix to be used. If the capitalized R is used letters in radices above 10 will be capitalized when Qlik Sense is formatting (for example 14FA). The non-capitalized r will result in formatting with non-capital letters (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code. Note that (r02) is the equivalent of (bin), (R16) is the equivalent of (HEX), and so on.

Format	Description
Roman format	To indicate roman numbers the format code should start with (rom) or (ROM). If the capitalized version is used capital letters will be used for formatting (for example MMXVI). The non-capitalized version will result in formatting with lower cap letters (mmxvi). Interpretation will work for both variants regardless of the capitalization of the format code. Roman numbers are generalized with minus sign for negative numbers and 0 for zero. Decimals are ignored with roman formatting.

Examples:

Examples of special number formats

Example	Result
num(199, '(bin)')	returns 11000111
num(199, '(oct)')	returns 307
num(199, '(hex)')	returns c7
num(199, '(HEX)')	returns C7
num(199, '(r02)')	returns 11000111
num(199, '(r16)')	returns c7
num(199, '(R16)')	returns C7
num(199, '(R36)')	returns 5J
num(199, '(rom)')	returns cxcix
num(199, '(ROM)')	returns CXCIIX

Dates

You can use the following symbols to format a date. Arbitrary separators can be used.

Symbols to format a date

Symbols	Description
D	To describe the day, use the symbol "D" for each digit.
M	To describe the month number, use the symbol "M". Use "M" or "MM" for one or two digits. "MMM" denotes short month name in letters as defined by the operating system or by the override system variable MonthNames in the script. "MMMM" denotes long month name in letters as defined by the operating system or by the override system variable LongMonthNames in the script.
Y	To describe the year, use the symbol "Y" for each digit.

Symbols	Description
W	<p>To describe the weekday, use the symbol "W".</p> <p>"W" will return the number of the day (for example 0 for Monday) as a single digit.</p> <p>"WW" will return the number with two digits (e.g. 02 for Wednesday).</p> <p>"WWW" will show the short version of the weekday name (for example Mon) as defined by the operating system or by the override system variable DayNames in the script.</p> <p>"WWWW" will show the long version of the weekday name (for example Monday) as defined by the operating system or by the override system variable LongDayNames in the script.</p>

Examples: (with 31st March 2013 as example date)

Examples of date formats

Example	Result
YY-MM-DD	describes the date as 13-03-31.
YYYY-MM-DD	describes the date as 2013-03-31.
YYYY-MMM-DD	describes the date as 2013-Mar-31.
DD MMMM YYYY	describes the date as 31 March 2013.
M/D/YY	describes the date as 3/31/13.
W YY-MM-DD	describes the date as 6 13-03-31.
WWW YY-MM-DD	describes the date as Sat 13-03-31.
WWWW YY-MM-DD	describes the date as Saturday 13-03-31.

Times

You can use the following symbols to format a time. Arbitrary separators can be used.

Symbols to format a time

Symbols	Description
h	To describe the hours, use the symbol "h" for each digit.
m	To describe the minutes, use the symbol "m" for each digit.
s	To describe the seconds, use the symbol "s" for each digit.
f	To describe the fractions of a second, use the symbol "f" for each digit.
tt	To describe the time in AM/PM format, use the symbol "tt" after the time.

Examples: (with 18.30 as example time):

Examples of time formats

Example	Result
hh:mm	describes the time as 18:30
hh.mm.ss.ff	describes the time as 18.30.00.00
hh:mm:tt	describes the time as 06:30:pm

Time stamps

The same notation as that of dates and times above is used in time stamps.

Examples: (with 31th March 2013 18.30 as example time stamp):

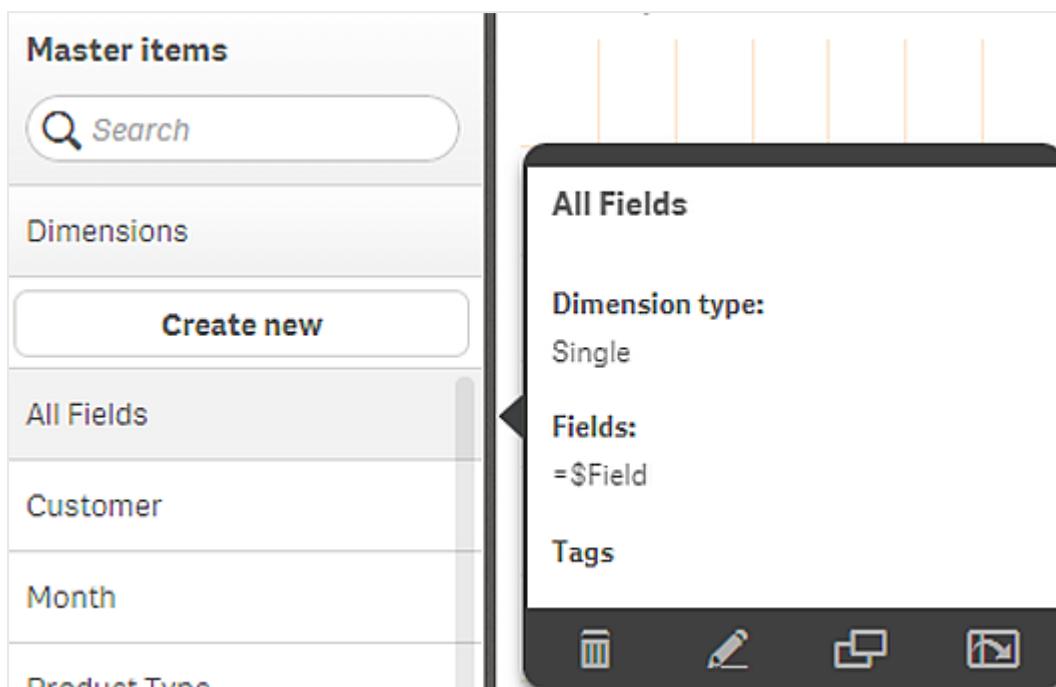
Examples of time stamp formats

Example	Result
YY-MM-DD hh:mm	describes the time stamp as 13-03-31 18:30.
M/D/Y hh.mm.ss.ffff	describes the time stamp as 3/31/13 18.30.00.0000.

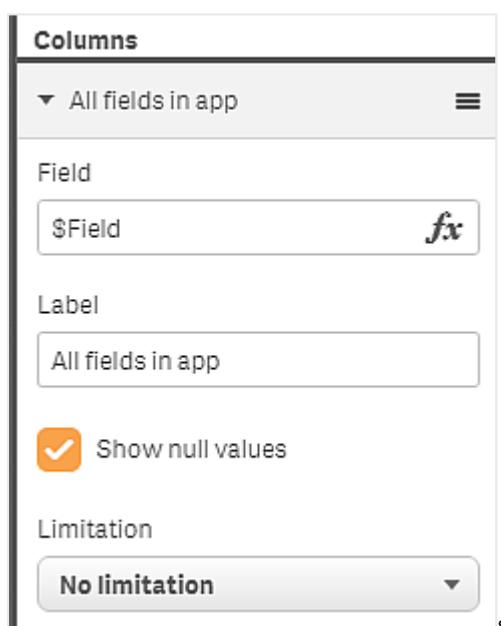
Using system fields in a visualization

You can use system fields in a visualization. System fields are created by Qlik Sense when the data load script is generated, and include information about the fields and tables in the loaded data. A system field begins with "\$", and you need to reference it by typing the field name including the "\$" manually. You can use a system field to create a dimension either as a master item or from the properties panel.

Preview of a dimension based on a system field .



A system field added as dimension in the properties panel.

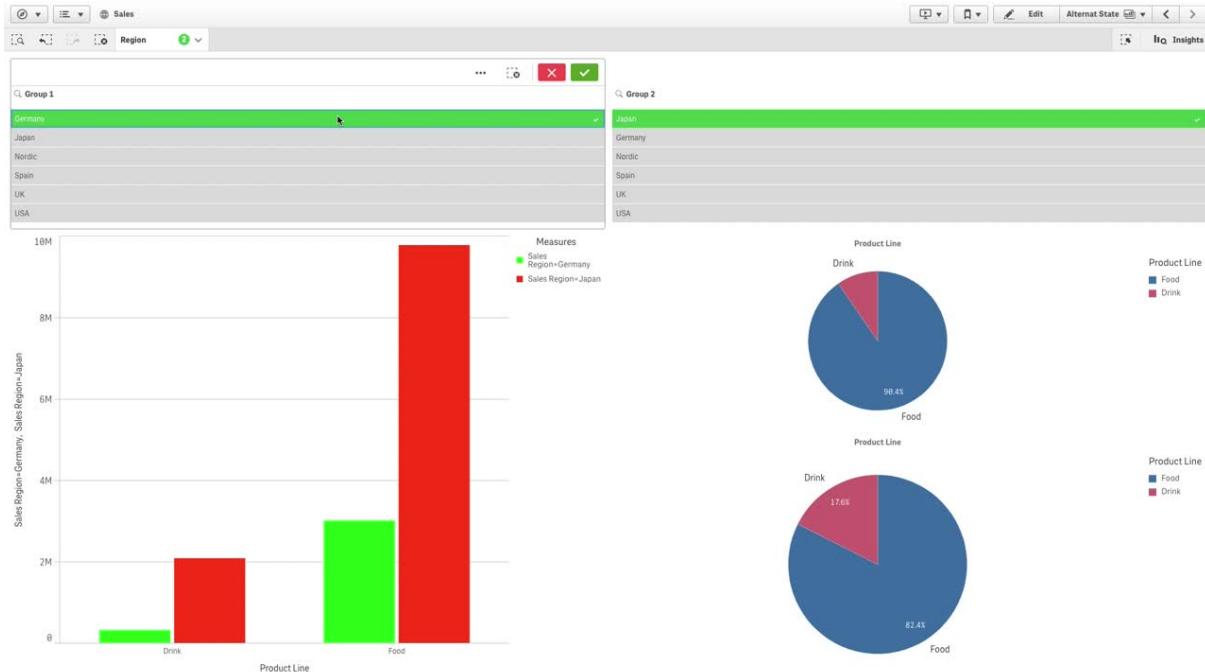


Using alternate states for comparative analysis

If you want to perform comparative analysis you can use alternate states in Qlik Sense. Alternate states allow you to make different selections on the same dimension, and compare the selections in a single visualization or in two or more visualizations side by side.

You can use alternate states in visualizations in two different ways:

- Applying a state to the visualization. This connects the selection of the visualization to the selection of the state.
- Using states in the set expression of a measure. This is useful to compare measure values of different states.



Creating alternate states

You can create a new alternate state under **Alternate states in Master items**:

1. Select **Create new**.
2. Provide a name for the new state. There are some naming limitations:
 - Do not use \$, 0, or 1 as a state name.
 - Do not use a state name starting with \$ or \$_ followed by a number, for example, \$3.
 - Do not use a state name already used as a bookmark name.

You have now created a new alternate state that you can use to perform comparative analysis. Typically you would want to create another new state to compare the two states.

Applying alternate states to sheets and visualizations

You can apply a state to a visualization, or a sheet. If you apply a state to a sheet, all visualizations on the sheet inherit the state unless you apply another state to a specific visualization. When you apply a state to a visualization, it reflects the selections made in the state. Any selections you make will be applied to the state and be reflected in other visualizations that have the same state applied.

The easiest way to apply an alternate state is dropping it on a sheet or a visualization, and then selecting **Apply state**.

You can also apply an alternate state with the **State** setting in the property panel:

- For a sheet, you find the setting under **Alternate states**.
- For a visualization, you find the setting under **Appearance > Alternate states**.

You can select:

- Any alternate state defined in **Master items**.
- **<inherited>**, in which case the state defined for the sheet is used.
- **<default state>**, which represents the state where no alternate state is applied.

Using alternate states in visualizations

Besides applying a state to a visualization you can also use alternate states in the set expression of a measure. This is useful when you want to compare measure values for different dimension selections side by side.

You can set the alternate state as an identifier in the set expression. If you want to use the measure `sum({Group1}Sales)` in your visualization, for example a bar chart, with a state called *Group1* you use the following expression as measure:

`Sum({Group1}Sales)`

To compare with a different state called *Group2*, you can create another measure with the expression `sum({Group2}Sales)`.

The bar chart will now show sales for the selection in *Group1* side by side with sales for *Group2*.

Getting information about the selection of an alternate state

You can see the selections of different states in the selections bar.

You may also want to be able to use information about which, and how many selections are made in a alternate state in labels or titles of visualizations. You can use the following chart functions with the **state_name** parameter to return selections associated with the specified state name:

- **GetCurrentSelections()** to return all current selections.
- **GetFieldSelections()** to return current selections of a field.
- **GetSelectedCount()** to return the number of selected values in a field.

Expanding variables in alternate states

You can specify which state to use when expanding a variable. Changes in a specific state do not affect variable values that are expanded in another state. If you do not specify a state, the variable is expanded in the default state.

Example:

If you have a state named `Mystate`, and a variable named `vMyvar`:

- `$(vMyVar)` expands the variable in the default state.
- `$(Mystate) vMyvar` expands the variable in the `Mystate` state.

Limitations

It is not possible to add visualizations to master items if you have set the state of the visualization to any other value than <inherited>.

Comparative analysis example

In this example we want to be able to compare the sales numbers of product lines for different selections of sales regions. We want to select the regions we compare dynamically, either as single regions or a combination of regions.

Dataset and app

If you want to follow this example, you need to download the Qlik Sense Tutorial - Building an App to get the dataset. If you have completed the tutorial you can use the app you created. Otherwise you need to create an app, add all six data files in the *Tutorials* source folder, and associate them using automatic recommendations in the data manager.

Create alternate states

For this example we need two alternate states. In **Master items > Alternate states:**

1. Create a new state called *Group 1*.
2. Create a new state called *Group 2*.

You have now created the two alternate states we need in this example.

Create filter panes for selection

Do the following:

1. Add a filter pane with the field *Region*.
2. Edit the label of the filter pane to say `=StateName()`. This is to make it easier to tell them apart, as the state is not indicated in the filter pane. The **StateName()** function returns the state that is applied to the function.
3. Drop the state *Group 1* on the filter pane and select **Apply state**.
4. Add another filter pane with the field *Region*.
5. Edit the label of the second filter pane to say `=StateName()`.
6. Drop the state *Group 2* on the second filter pane and select **Apply state**.

You have now created the two filter panes that are used to control the selections of each of the two states. When you make a selection in the *Group 1* filter pane, the same selection is applied to the state *Group 1* which is reflected in all visualizations connected to that state.

Create a bar chart for analysis

Do the following:

1. Create a master item measure with name *Group1Sales*.
Set **Expression** to `Sum({[Group 1]}[Sales])`.

This expression sums the sales for all selections in the *Group 1* state.

Set **Label expression** to 'Sales '&GetCurrentSelections(chr(13)&chr(10), '=' , ' ', 9, 'Group 1').

We use a label expression to be able to show the current selection of the state as a label in the chart, instead of the default label.

2. Create another master item measure with name *Group2Sales*.

Set **Expression** to Sum({[Group 2]}[Sales]).

Set **Label expression** to 'Sales '&GetCurrentSelections(chr(13)&chr(10), '=' , ' ', 9, 'Group 2').

3. Add a bar chart to the sheet.
4. Set the *Product Line* field as dimension.
5. Add the measures *Group1Sales* and *Group2Sales*.

You have now created a bar chart that shows the sales by product line for the two groups of regions selected in the filter panes. When you make a new selection in one of the filter panes, the corresponding measure value changes according to the new selection.

Discovery

You can now make selections in *Group 1* and *Group 2*, and see the results of the selected combinations of regions in the bar chart.

Creating a visualization using a custom object

You can enhance your apps with custom-objects.

Custom objects that are available are:

- visualization extensions
- object bundles supplied by Qlik:
 - *Dashboard bundle* (page 313)
 - *Visualization bundle* (page 331)

You can find custom objects in the assets panel under  **Custom objects** when you are editing a sheet.

You can build your own visualization extensions in the Dev Hub.

 For the Dev Hub, see [Dev Hub](#).

Adding a custom object to the sheet

You start creating a visualization by dragging a visualization extension onto a sheet.

Do the following:

1. Click  **Edit sheet** in the toolbar.
2. Click  in the panel on the left-hand side to expand custom objects.
3. Drag a visualization extension onto the sheet.

You can drop it in an empty location on the sheet, split the area of an existing visualization into two, or replace an existing visualization.



If you double-click a custom object, it is added to the sheet immediately.

4. Change the required settings for the custom object in the properties panel. The required settings are defined by the extension developer, this can be dimensions, measures, or other settings.

You now have a complete visualization that you can start using while exploring the data in the app.

Copying a visualization from an existing visualization

You can copy a visualization:

- Within the same sheet
- Between sheets in the same app
- Between sheets belonging to different apps.



For a copied visualization to work in a different app, the same dimensions and measures have to be a part of the target app as well.

Do the following:

1. While editing a sheet, click on the item you want to copy.
The item is highlighted.
2. On the edit bar, click .
3. To insert the item on another sheet, navigate to the sheet via the sheet navigator.
4. Click  to paste the item.

The copied item is added to the sheet.



Depending on what situation you are in, different things will happen when you paste the copied visualization on a sheet:

- *If a visualization is selected, then the selected visualization will be replaced.*
- *If no visualization is selected, then the pasted visualization will be placed in the largest empty space.*
- *If there is no empty space, then the largest visualization on the sheet will be split in half to make space for the pasted visualization.*

Creating time-aware charts



Time-aware charts are visualizations that use a continuous scale to provide a complete and accurate view of time-based data. That is, when you enable continuous scaling on the x-axis in a chart with date fields, data points are separated from each other by a distance relative to their associated time. As well, the axis labels are evenly separated whether or not there is data for that point and the chart view is compressed to avoid scrolling.



Continuous scale is supported for line charts, bar charts, and combo charts.

A continuous scale is most commonly used with date fields such as:

- Second
- Minute
- Hour
- Week
- Month
- YearMonth
- Quarter
- YearQuarter
- Year
- Date
- Timestamp

Adding a continuous scale

Do the following:

1. In sheet view, click  **Edit sheet** in the toolbar.
2. Click the line chart that you want to edit.
3. In the properties panel, click the **Appearance** tab.
4. In the **X-axis** section, check **Use continuous scale**.

The chart is compressed and the data points and labels are readjusted.

Making selections in a time-aware chart

When navigating a time-aware chart, you can zoom into a smaller time span to take snapshots of the data displayed and select data values. Selections made on a time axis using range selection select all data values (even those that are not visible). Selections made on the measure axis or using lasso selections only select visible values.

Changing the data of a visualization

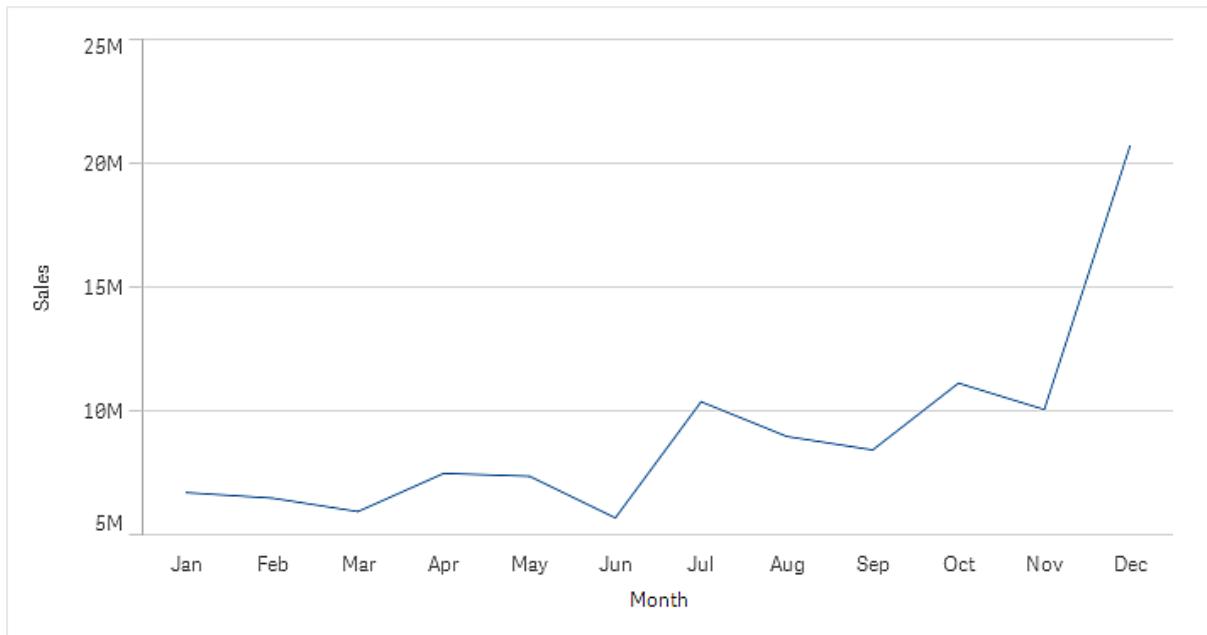
The data displayed within a visualization can be changed after the visualization is created.

For example, you may want to add additional dimensions or measures to a chart to add depth to the existing data. Alternatively, you may need to correct an invalid dimension or measure.

Adding dimensions and measures to a visualization

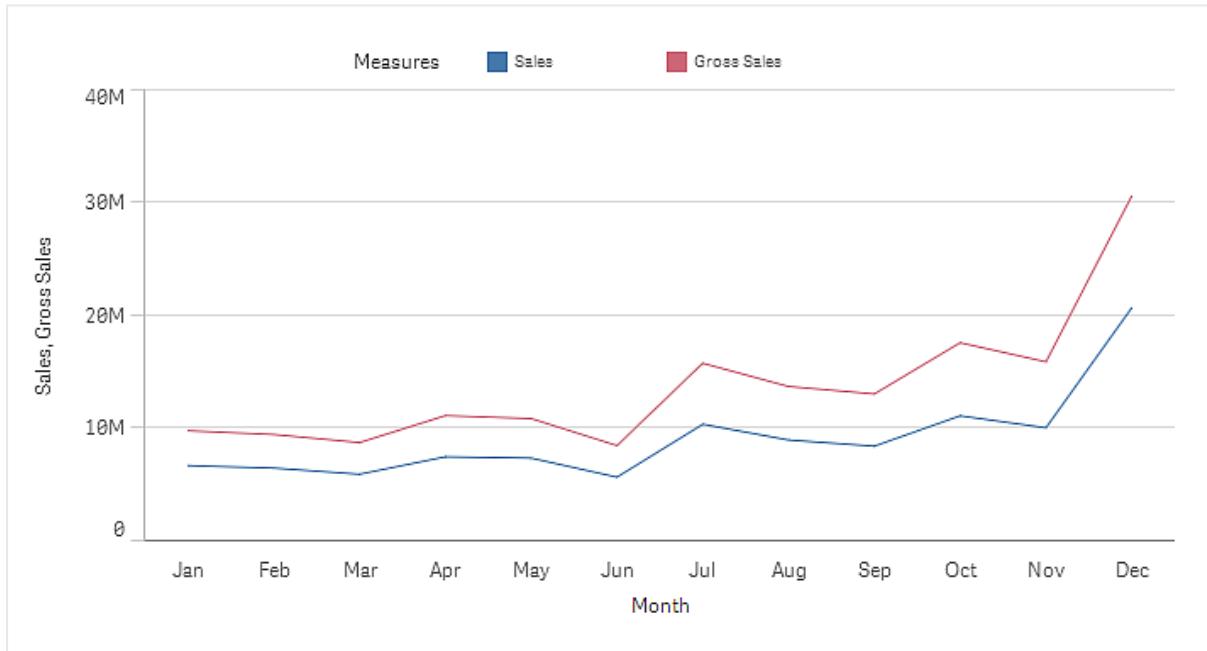
You can add multiple dimensions and measures to a visualization to add depth to the data displayed. Adding dimensions and measures enabled you to view more information in a single visualization. For example, the image below contains a visualization with the measure *Sales* and a dimension *Month*.

Line chart with the measure Sales.



You can add a second measure, *Gross Sales*. The example line chart now gives use more context by enabling a comparison of *Gross Sales* to *Sales*.

Line chart with the measures Sales and Gross Sales.



Do the following:

1. In the properties panel, click the **Data** tab.
The **Data** tab is expanded.
2. In the **Dimensions or Measures** section, click **Add** to add a dimension or measure.

A dialog with a text box opens. Below the text box all available dimensions or measures are listed, grouped into **Dimensions** or **Measures** (that is master items) and **Fields**.

3. Start typing in the text box.

A list of matching fields and dimensions or measures is displayed.



*You can also create a dimension by entering an expression directly in the text box, or by clicking **fx** to create a dimension in the expression editor.*



*If no measure is displayed, you need to create one. You can enter an expression directly in the text box, or you can click **fx** to create a measure in the expression editor.*

4. Select the dimension or measure that you want to use.

The dimension or measure is added to the visualization. The new dimension or measure settings are shown in the properties panel.

Adding alternative dimensions and measures to a visualization

Alternative dimensions and measures are dimensions and measures that are added to a visualization, but are not displayed until a user chooses to switch which dimensions and measures are being displayed during visual exploration. You can toggle which of your dimensions and measure are displayed using the exploration menu of a visualization.

[Adding alternative dimensions and measures to a visualization](#)



You can add alternative dimensions and measure to all chart types but changing alternative dimension and measures in the visualization can only be done in bar, line and combo charts. Only alternative dimensions can be changed in the pie charts visualization and only alternative measures in scatter plots.

Alternative dimensions and measures enable you conserve space on your sheets. Rather than make multiple similar visualizations of the same type, you can use alternative dimensions and measure with a single visualization. You can then switch between the displayed dimensions and measures. For example, if you wanted to see total sales by product category and total sales by product subcategory, you could add product category as a dimension and product subcategory as an alternate dimension in the same visualization.

Alternative dimensions and measure enable you to have more dimensions and measures associated to a visualization that the display limit. Many visualizations have limitations on how many dimensions and measures that can be displayed at the same time. A line chart with two or more measures can only have one dimension, and a line chart with two dimensions can only have one measure. Alternative dimensions and measures enable you to exceed that limit.

Alternative dimensions and measures can be added like normal dimensions and measures in the properties pane. You can also drag dimensions and measures in the **Data** section of the properties panel into the **Alternative dimensions** or **Alternative measures** section.

Do the following:

1. In the properties panel, click the **Data** tab.

The **Data** tab is expanded.

2. In the **Dimensions or Measures** section, click **Add alternative**.

A dialog with a text box opens. Below the text box all available dimensions or measures are listed, grouped into **Dimensions or Measures** (that is master items) and **Fields**.

3. Start typing in the text box.

A list of matching fields and dimensions or measures is displayed.



*You can also create a dimension by entering an expression directly in the text box, or by clicking **fx** to create a dimension in the expression editor.*



*If no measure is displayed, you need to create one. You can enter an expression directly in the text box, or you can click **fx** to create a measure in the expression editor.*

4. Select the dimension or measure that you want to use.

The alternate dimension or measure is added to the visualization.

Editing data in a visualization

You can edit and adjust the data in a visualization. You may need to edit an existing dimension or measure to adjust it to create a stronger visualization or correct an invalid dimension or measure. You may also want to edit a master item in a visualization to change the master item across all visualizations.

Invalid dimensions and measures

Dimensions and measures are invalid when the associated expression cannot be interpreted by Qlik Sense.

If you create an invalid dimension or edit an existing one so that it becomes invalid, the dimension is presented in the properties panel as dimmed with a red hue and the text **Invalid dimension** to indicate that the dimension is invalid. If you use an invalid dimension in a visualization, the visualization cannot be displayed.

If you create an invalid measure or edit an existing one so that it becomes invalid, the **Expression** text box under **Measures** in the properties panel is presented with a red border to indicate that the measure is invalid.

Editing a dimension

You can edit dimensions, including master dimensions, in the properties panel. Select the dimension that you want to edit. Dimensions have the following properties:

- **Field:** Start typing the field name to display a list of matching fields to choose from. You can also click **fx** to open the expression editor, where you can create a calculated dimension.
- **Label:** Enter a name for the dimension.

- **Include null values:** When this is checked, the visualization will include the null values of the dimension, presented as a gap or a dash, depending on the type of visualization. For example, if you have sales figures but do not have any information about what company the figures belong to, the figures will be added to the measure value for the null value dimension.
- **Limitation:** You can limit the number of dimension values that are displayed.
- **Show others:** When you have set a limitation for the number of dimension values displayed, you have an option to summarize the measure values for the remaining dimensions by selecting **Show others**.
- **Master item:** Edit a master dimension to update all instances where the dimension is used, or create a new master dimension by adding a dimension to the master items.

Editing a measure

You can edit measures, including master dimensions, in the properties panel. Select the measure that you want to edit. Measures have the following properties:

- **Expression:** Enter the expression. You can also click  to open and use the expression editor.
- **Label:** Enter a name for the measure. The label is not automatically updated when you make changes in **Expression**.
- **Number formatting:** Set the number formatting for the measure values. The options **Number** and **Date** offer custom formatting options for defining your own format pattern.
- **Master item:** Edit a master measure to update all instances where the measure is used, or create a new master measure by adding a measure to the master items.

Editing master items

Dimensions and measures that are linked to a master item are displayed with a  in the properties panel. You can edit the master item to update all instances that use the dimension or measure, or you can unlink the item from its master to edit only the current instance of the dimension or measure.

A visualization that is linked to a master item is indicated by  **Linked visualization** on the sheet. You can edit a master visualization to update all instances that use the visualization, or you can unlink a visualization from its master to edit only the current instance of the visualization. Unlinking a visualization does not unlink any master dimensions or master measures used in the visualization.

Deleting dimensions and measures

In the properties panel, you can delete a dimension or measure. Long-touch or right-click the dimension or measure and select **Delete** in the dialog. Alternatively, click the dimension or measure and click . If you delete an instance of a master item, the master item is still available in the assets panel.



You can undo a deletion by clicking . Changing which sheet you are viewing will prevent you from undoing your deletion.

Changing the appearance of a visualization

The **Appearance** section in the properties panel offers several options to set and modify the appearance of a visualization.

Many of the settings have **Auto** options that support an optimal presentation of the visualization, taking into account the number of dimensions and measures and the type of data used. Normally, you do not need to change these settings, unless you have a special reason for doing so, for example, when space is very limited.

Appearance can be affected by the sorting of the dimensions and measures.

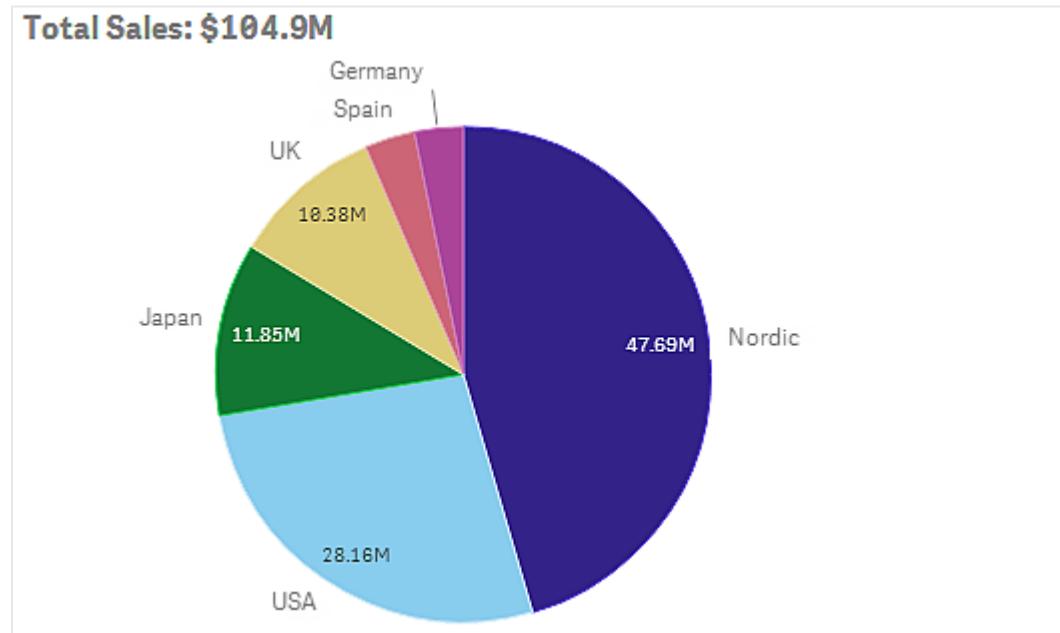
General

Show titles: **On** by default in all visualizations except filter panes and text & image visualizations. Filter panes have the name of each dimension, and in most cases do not need any additional title. The text & image visualization includes an editing toolbar with many options to format the text, and therefore the title field can be used for other purposes.

Title, Subtitle, and Footnote: Apart from the obvious use of title, subtitle, and footnote as text fields, you can use these fields to also display an expression, which provides additional information that complements the measure in the visualization. You could, for example, show the totals in the title field, so that the totals of the selected values are always available.

Example:

In the following image, the total sales are calculated and used in the title. When a selection is made, the total sales are updated accordingly.



The following string was used to add the *Total Sales* expression to the field **Title**:

```
='Total Sales: $'& Round(Sum(Sales)/1000000, 0.1) & 'M'.
```

Because the title field is primarily a text field, it is necessary to start the string with an equals sign (=), to signify that the strings contains an expression.

Because 'Total Sales: \$' is a text string when it is used in an expression, the string must be surrounded by single quotation marks.

& is used to concatenate the string and the expression.

Round(Sum(Sales)/1000000, 0.1) is the expression. The aggregation *Sum(Sales)* is divided by 1000000 and the function *Round(x,0.1)* reduces the number of decimals to one.

& 'M', finally, concatenates the expression with the unit M for million.

For the title there are three options for adding an expression:

- Directly in the title field of a visualization. Start the string with an equals sign (=).
- Directly in the box **Title** under **Appearance** in the properties panel. Start the string with an equals sign (=).
- Through the expression editor in the box **Title**. Click **fx** to open the expression editor. No equals sign is needed.

For the subtitle and footnote only the last two options are available.

Show details: Hide by default. When set to **Show**, users can click **i** when analyzing to view details such as descriptions measures and dimensions.

Presentation

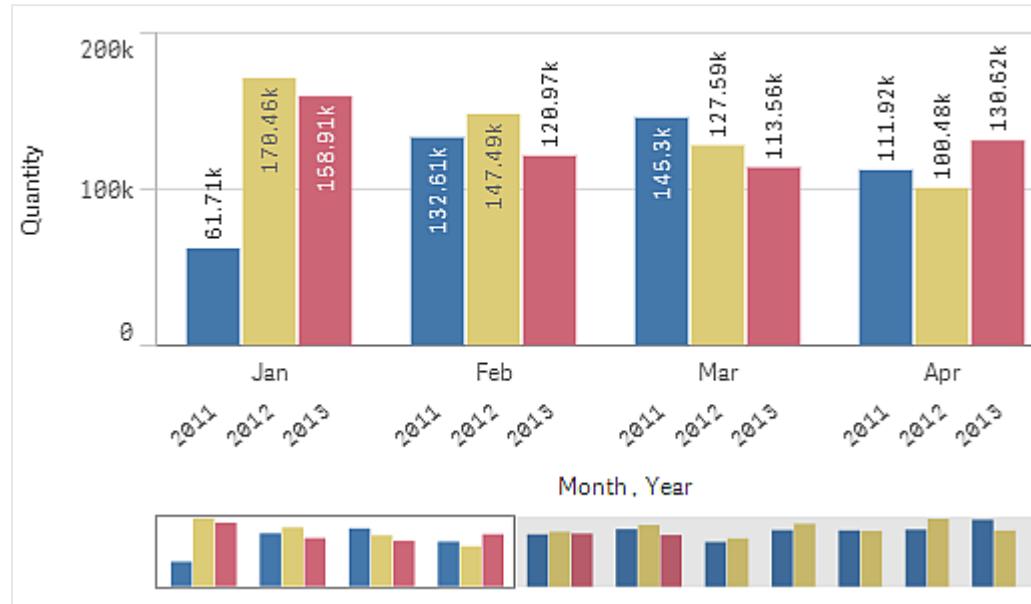
Many of the presentation settings only apply to a certain visualization.

Presentation settings in visualizations

Visualization	Description
Bar chart	Set the bars to be displayed grouped or stacked, vertically or horizontally.
Box plot	Select to show whisker ticks and grid lines. You can show the visualization vertically or horizontally.
Distribution plot	Select to show point, background or both. You can show the visualization vertically or horizontally.
Gauge	Set the gauge to be displayed as a radial or a bar. You can set range limits and use segments with limits.
Histogram	Select to show grid lines.
Line chart	Set the line chart to be displayed as a line or an area.
Pie chart	Set the pie chart to be displayed as a pie or a donut.
Scatter plot	Turn on/off navigation. Set the size of the bubbles in a scatter plot. Set the compression resolution for large data sets in scatter plot.
Table	Set the totals to be displayed at the top, bottom, or not at all.
Treemap	Set the headers, labels, overlay labels, and leaf values. Select to show the data values.

Example:

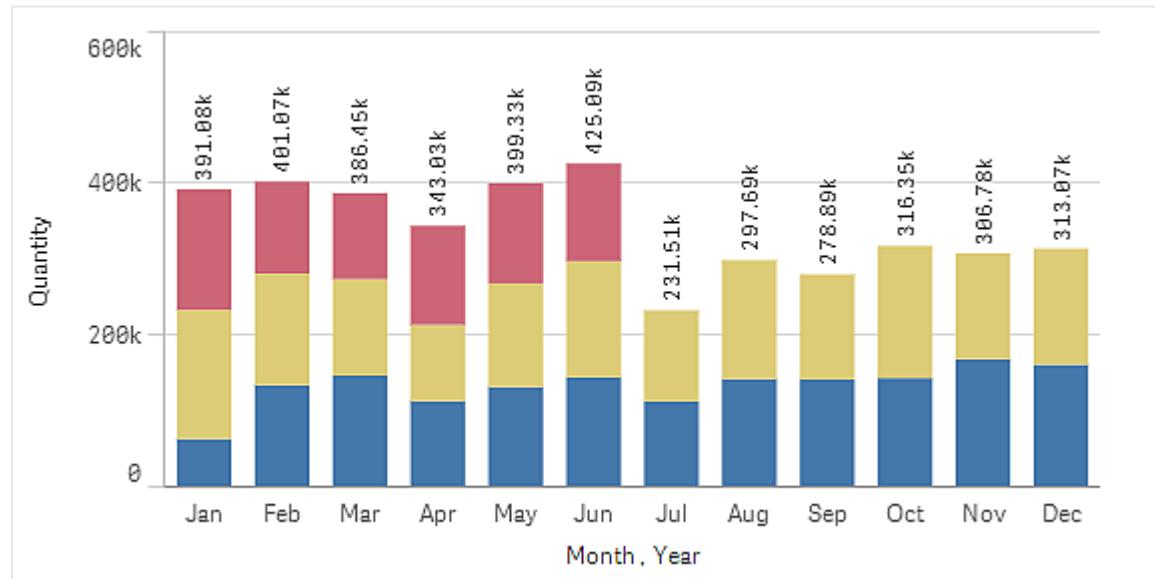
A bar chart with two dimensions is by default presented with the bars grouped.



Let us assume that you want to compare the total monthly quantity for these years. Then it would be a good idea to switch to a stacked bar chart.

In the properties panel, under **Appearance > Presentation** there is an option **Stacked**.

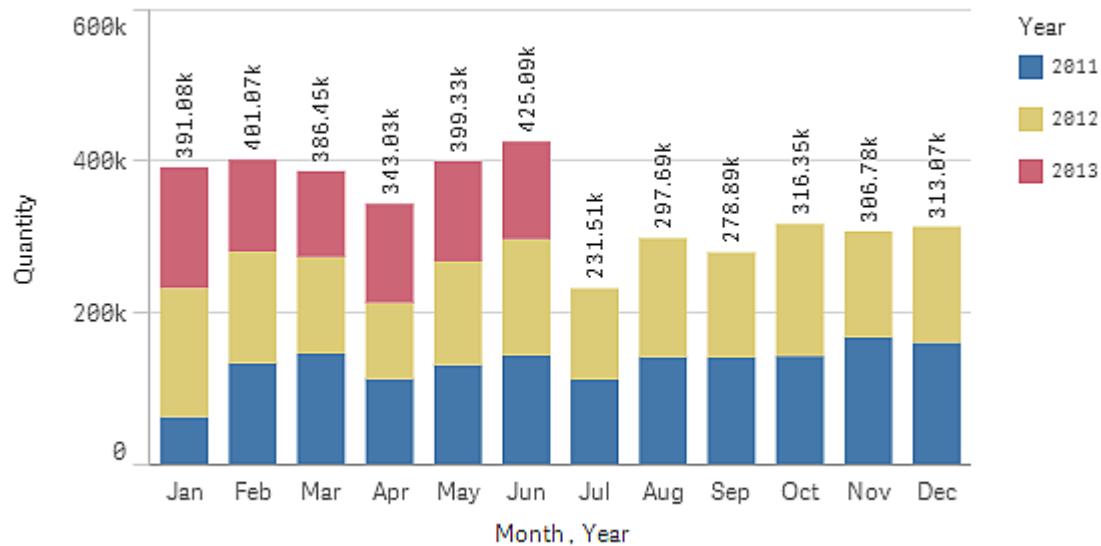
With stacked bars it is easier to compare the quantity between different months.



Now it is quite easy to compare the quantities per month. There is a legend to display the years.

Under **Colors and legend, Show legend** is set to **Auto**, which means that the legend is displayed when there is enough space. In the properties panel, you can also set where to display the legend and whether to display the legend title.

There is only data for the first half of 2013 (red bars).



Colors and legend

The **Colors and legend** section of the properties panel sets your color and legend options. Qlik Sense automatically colors visualizations as they are added to your sheets. As a best practice, it is recommended to add or change colors only when it serves as purpose in the visualization. Too many colors or indistinct color choices can make visualizations less clear.

You can manually set the colors and legends by deselecting the **Auto** option and selecting your color preferences. Qlik Sense enables you to color your visualizations by:

- Single color
- Multiple colors
- Dimension
- Measure
- Master items
- Expression

For more information on different visualization coloring options, see *Coloring a visualization* (page 462). For examples of each of these methods of coloring visualizations and the settings used can be found, see *Example 1: Coloring by a dimension in the visualization* (page 478).

X-axis and Y-axis

For both the y-axis and the y-axis, you have options for deciding what combination of labels and title to display, as well as their orientation and position. Sometimes it may feel unnecessary to have labels and/or a title, because the visualization is self-explanatory and then it would be good to be able to hide them. Furthermore, when you create a visualization that is very small, for example, three by three squares, the labels are automatically hidden.

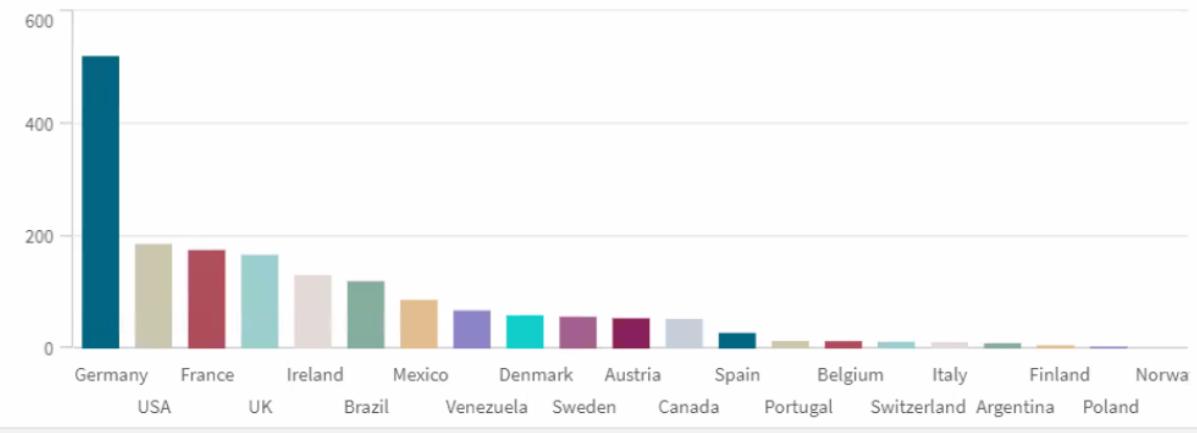
Range: The measure axis (usually the y-axis) has an option to set the range of the axis. By default, the range is adjusted according to the highest positive or the lowest negative measure value, but if, for example, a single measure value is much larger than all the other values, you may want to set a range that is suitable for the lower values. In the properties panel, under **Appearance > Y-axis <measure name>**, there is a button for **Range**, which is set to **Auto**. Click the button to switch to **Custom**. Now you can set the range for **Max**, **Min**, or both. In a bar chart, the bars that are out of the range are cut diagonally to indicate that they are out range. In a line chart, only the parts that are within the range are visible.

Label orientation: You can change the orientation of data labels on the dimensions axis (usually the x-axis). In the properties panel, under **Appearance > X-axis <dimension name>**, there is a drop-down menu for label orientation. By default, this is set to **Auto**. If there is not enough room for a label to fully appear on the chart, it will be truncated with an ellipsis. The following options are available:

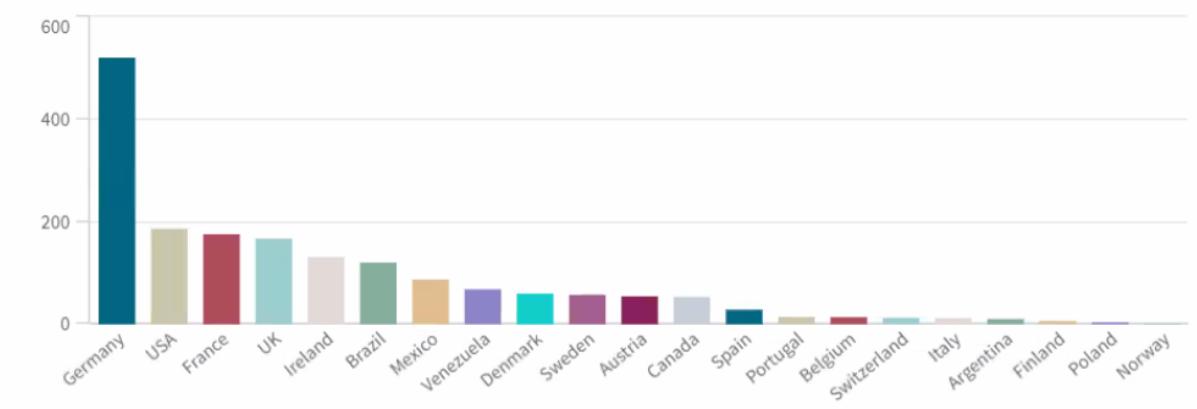
- **Auto:** Automatically selects one of the other options depending on the space available on the chart.
- **Horizontal:** Labels are arranged in a single horizontal line.
- **Tilted:** Labels are stacked horizontally at an angle.
- **Layered:** Labels are staggered across two horizontal lines.

Examples of layered and tilted labels

Layered Labels



Tilted Labels



Change the sorting of a visualization

You can change the sorting order of dimensions and measures so that your data is presented in the way you intend.

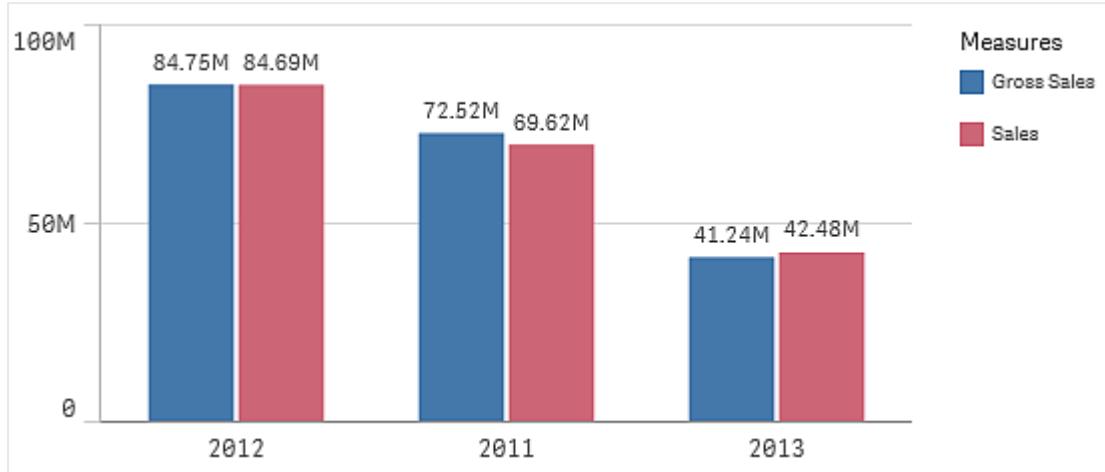
Most visualizations have a **Sorting** section in the properties panel where you can put the cursor on the drag bars and drag the dimension or measure to rearrange the sorting order. In visualizations without a sorting section, you can still adjust the sorting to some extent.

In the following screenshot, the primary sorting is on the measure *Gross Sales*.

The screenshot shows the 'Sorting' section of the properties panel. It contains three items, each with a number and a right-pointing arrow:

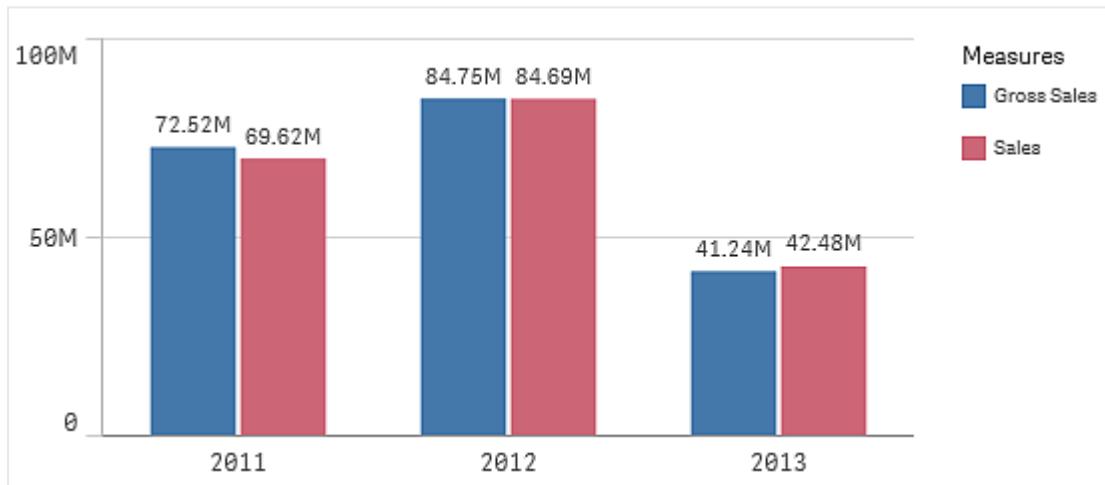
- 1 ▶ Gross Sales
- 2 ▶ Year
- 3 ▶ Sales

A bar chart with that sorting order will look as follows.



Since there are two measures, *Gross Sales* and *Sales*, the bars are by default grouped. *Gross Sales* is presented first, because it has sorting priority 1. If you were to drag *Sales* to the top of **Sorting**, the first bar would be *Sales* and the second bar *Gross Sales*.

To sort by *Year*, instead, you need to drag *Year* to the top in **Sorting**. The bar chart is then updated and sorted by *Year*.



*In bar charts with multiple dimensions, sorting is locked to the first dimension. This dimension is what groups and stacks are based on, and sorting on a different dimension or a measure would break up these groups in an undesirable way. If you still want to sort by the measure value, try using the option **Sort by expression** on the first dimension under **Sorting**.*

Sorting in the dimensions and measures sections

Although it is primarily under **Sorting** that you set the sorting order, you can also adjust the order in the properties panel section **Data** under **Dimensions** and **Measures**. In **Dimensions** you can change the priority order between the different dimensions by dragging them, and, likewise, in **Measures**, you can drag the measures to change the sorting order. Put the cursor on the drag bars and drag the dimension or measure to rearrange the order. Changes are reflected in the visualization.

Internal sorting

Apart from setting the sorting order between dimensions and measures, you can also set the internal sorting order, under **Sorting**.

Click the dimension or measure name to open the settings and click the sorting button to switch to **Custom** sorting. The following table shows the internal sorting priority order and sorting options. The sorting is either **Ascending** or **Descending**.

Sorting options:

- Sort by expression (Enter an expression to sort by. Only available for dimensions.)
- Sort numerically
- Sort alphabetically

Additionally, you can sort by load order by switching to **Custom** and leaving all sorting options unselected.

If you have set a custom order for a field, that custom order will override any selected internal sort order in **Sorting**.

Default sorting

By default, the dimensions and measures are sorted in the order they were added, with the most recently added item last. Each dimension is sorted internally in the most common way for that type of data. Numbers are sorted numerically, ascending. Text is sorted alphabetically, ascending.

Default sorting in visualizations

Visualization	Description
Bar chart	By default, a bar chart with one measure and one dimension is presented with vertical bars sorted descending on the measure. When a dimension has less than 10 values, the sorting is by dimension, alphabetically.
Box plot	By default, a box plot is sorted by the center line. You can also sort by first whisker, box start, box end or last whisker.
Combo chart	Sorted by the first item added, either the dimension or the measure.
Distribution plot	By default, a distribution plot is sorted by the outer dimension.
Filter pane	By default, data in filter panes is presented ascending.
Gauge	A gauge only uses a single measure value, the first one under Measures .
Histogram	The histogram does not have any sorting section.
KPI	By default, the first added measure becomes the main value.
Line chart	By default, a line chart is sorted by the dimension.

Visualization	Description
Map	A map layer can only have one dimension. Sorting is used to determine the order points or areas are added to the map.
Pie chart	A pie chart uses one measure and one dimension. By default, a pie chart is sorted by the measure in descending order.
Scatter plot	The scatter plot does not have any sorting section, but the order of the measures decides where they are used. The first measure is used on the x-axis, the second measure is used on the y-axis, and the third (optional) measure is used for the bubble size (it is used to set the color on large data sets) in the scatter plot. You can only have one dimension in a scatter plot.
Table	<p>By default, the column presents the dimensions and measures in the order they were added.</p> <p>Sorting order of rows: By default, the table is sorted in ascending order by the first dimension or measure under Sorting. You can temporarily change the sorting by clicking the header of the row you want to sort on. One click - ascending order, two clicks - descending order.</p>
Text & image	The text & image visualization does not have any sorting section, but you can drag the measure tokens in the visualization to change the order.
Treemap	The treemap does not have any sorting section. The sorting is automatically by measure size.

Coloring a visualization

Qlik Sense automatically colors visualizations as they are added to your sheets. You can manually set the colors in your visualizations to meet your requirements or preferences.

Color options for most visualizations are set in the properties panel, in **Appearance > Colors and legend**. By selecting **Custom**, you can manually apply colors to your visualizations using the following methods:

- Color by single color
- Color by multiple colors
- Color by dimension
Dimension fields can also be dragged and dropped from the assets panel onto a visualization to color the visualization by dimension (if supported by the visualization type).
- Color by measure
Measure fields can also be dragged and dropped from the assets panel onto a visualization to color the visualization by measure (if supported by the visualization type).
- Color by expression

Tables and pivot tables can only be colored by expression. Options for coloring tables and pivot tables are found in the properties panel in **Data**.

If you want to keep colors consistent for dimensions or measures between different visualizations, you can assign specific colors to master items in your library. Most visualizations will use any colors assigned to master items automatically. In cases where a visualization uses both a master dimension and a master measure with assigned colors, the color assigned to the master dimension is used by default. You can select which master item color to use or disable master item colors entirely.

For more information, see [Assigning colors to master items \(page 110\)](#).

You can also assign colors to individual master dimension values to ensure the colors of different values are consistent across visualizations.

For more information, see [Assigning colors to master dimension values \(page 112\)](#).

To keep visualizations clear when you manually set colors, you should select colors for accessibility and only use different colors when they serve a purpose.

Color by single color

When you color by single color, one color is used for all objects in the chart. Coloring by a single color is best used for visualizations, such as bar or line charts, with a single dimension and measure.

If you have a master dimension or master measure that has a color assigned to it, you can color the visualization by that single color. In cases where a visualization uses both a master dimension and a master measure with assigned colors, the color assigned to the master dimension is used by default. You can select which master item color to use or disable master item colors entirely.

The following options are available when **Single color** is selected from **Colors in Appearance > Colors and legend:**

Options for color by single color (Advanced)

Option	Description
Use library colors	Select to use master item colors. In cases where a visualization has both a master dimension and a master measure that have colors assigned to them, you can select which to use in the visualization. This option is available when a master dimension or master measure used in the visualization has a color assigned to it.
Color	Select a color using the color picker. You can select a color from the default palette, enter a hex value for a color, or select a color from a color wheel.

Color by multiple colors

When you have multiple measures in a visualization, you can select **Multicolor** to color each measure with a different color. Qlik Sense offers a 12 color and a 100 color palette to apply to the visualization. By default, **12 colors** is selected as the color scheme dimensions.

If you are using master measures in your visualization, you can also choose to use them in your visualization. When a visualization is colored by master measures, master measures will use their assigned colors and any other measures are assigned colors from the **12 colors** scheme.

The following options are available when **Multicolored** is selected from **Color**:

Options for color by multiple colors

Option	Description
Use library colors	Select to use master item colors. In cases where a visualization has both a master dimension and a master measure that have colors assigned to them, you can select which to use in the visualization. This option is available when a master dimension or master measure used in the visualization has a color assigned to it.

The following options are available when **Multicolor** is selected from **Colors** in **Appearance > Colors and legend**:

Options for color by multiple colors (Advanced)

Option	Description
Use library colors	Select to use master item colors. In cases where a visualization has both a master dimension and a master measure that have colors assigned to them, you can select which to use in the visualization. This option is available when a master dimension or master measure used in the visualization has a color assigned to it.
Color scheme	Select the color scheme used in the visualization. The following schemes are available: 12 colors: The colors are reused when there are more than 12 values. The 12 colors in this color scheme can all be distinguished by people with a color vision deficiency. 100 colors: The colors are reused when there are more than 100 values. Not all of the 100 colors can be distinguished by people with a color vision deficiency.

The following options are available when **Multicolor** is selected from **Colors** in **Appearance > Colors and legend**:

Options for color by multiple colors

UI item	Description
Use library colors	Select to use master item colors. In cases where a visualization has both a master dimension and a master measure that have colors assigned to them, you can select which to use in the visualization. This option is available when a master dimension or master measure used in the visualization has a color assigned to it.

UI item	Description
Color scheme	Select the color scheme used in the visualization. The following schemes are available: 12 colors: The colors are reused when there are more than 12 values. The 12 colors in this color scheme can all be distinguished by people with a color vision deficiency. 100 colors: The colors are reused when there are more than 100 values. Not all of the 100 colors can be distinguished by people with a color vision deficiency.

Color by dimension

When you color a visualization by a dimension, all values in the visualization are colored by the corresponding values in the dimension field selected. By default, the visualization is colored by the primary dimension of the visualization, but you can select other dimensions. Qlik Sense offers a 12 color and a 100 color palette. By default, **12 colors** is set as the palette for color by dimensions.

If you are using a master dimension, you can color the visualization using the colors assigned to the distinct values of that dimension.

Coloring by dimension is useful when you want to keep track of related information in your visualizations, such as coloring multiple charts by the dimension of *Region* to clearly see the values related to each region in each chart.

The following options are available when **By dimension** is selected from **Colors** in **Appearance > Colors and legend**:

Options for color by dimension

UI item	Description
Select Dimension	Select the dimension used to color this visualization with this field. By default, if you have already selected a dimension for the visualization, it is set with that dimension. Click ▾ to select a different dimension. You can enter an expression by clicking fx to open the expression editor
Persistent colors	When selected, colors persist between selection states. If cleared, colors will be changed and reassigned for different dimension values as selections are made in the visualization.
Color scheme	Select the color scheme used in the visualization. The following schemes are available: 12 colors: The colors are reused when there are more than 12 values. The 12 colors in this color scheme can all be distinguished by people with color vision deficiency. 100 colors: The colors are reused when there are more than 100 values. Not all of the 100 colors can be distinguished by people with color vision deficiency.
Library colors	Select to use master dimension color values. This option is available when a master dimension is used in the visualization.

Color by measure

When you color a visualization by a measure, all values in the visualization are colored by a gradient or class based on the values in the selected measure. By default, the visualization is colored by the primary measure of the visualization, but you can select another measure. There are four available color schemes.

Coloring by measure is useful when you want to clearly see objects colored by their corresponding measure value.

The following options are available when **By measure** is selected from **Colors** in **Appearance > Colors and legend**:

Options for color by measure

UI item	Description
Select Measure	Select the measure used to color this visualization. By default, if a measure has been added to the visualization, that measure is selected. Click ▼ to select a measure. You can enter an expression by clicking fx to open the expression editor
Color scheme	Select the color scheme used in the visualization. The following schemes are available: Sequential gradient: The transition between the different color groups is made using different shades of colors. High measure values have darker hues. Sequential classes: The transition between the different color groups is made using distinctly different colors. Diverging gradient: Used when working with data that is ordered from low to high, for instance, to show the relationship between different areas on a map. Low and high values have dark colors, mid-range colors are light. Diverging classes: Can be seen as two sequential classes combined, with the mid-range shared. The two extremes, high and low, are emphasized with dark colors with contrasting hues, and the mid-range critical values are emphasized with light colors.
Reverse colors	Select this option to switch which colors are used for low values and which colors are used for high values in the selected color scheme.
Range	Set the measure value ranges used to color the visualization. When set to Auto , Qlik Sense creates ranges based on the detected minimum and maximum values. When set to Custom , Qlik Sense automatically creates ranges based on user-defined minimum and maximum values. You must enter values or expressions that calculate those values in the fields Min and Max . You can enter an expression by clicking fx to open the expression editor

Color by expression

Coloring by expression applies colors to a visualization based on a user-defined expression. This enables you to use expressions to define both the colors used and the values upon which the colors are applied in a visualization. You could, for example, use an expression to set conditional colors in a chart.

The following options are available when **By expression** is selected from **Colors** in **Appearance > Colors and legend**:

Options for color by expression

UI item	Description
Expression	<p>Enter an expression by clicking fx to open the expression editor.</p> <p>For more information, see <i>Examples</i> (page 469).</p>
The expression is a color code	<p>Selected by default. In most cases, it is best to keep this setting. When the selection is cleared, the expression evaluates to a number, which in turn is plotted against one of the chart gradients.</p>
Label	<p>Enter the label to appear for the legend.</p> <p>This expression is a color code must be cleared.</p>
Color scheme	<p>Color scheme sets the colors used in the visualization. The following color schemes are available:</p> <p>Sequential gradient: The transition between the different color groups is made using different shades of colors. High measure values have darker hues.</p> <p>Sequential classes: The transition between the different color groups is made using distinctly different colors.</p> <p>Diverging gradient: Used when working with data that is ordered from low to high, for instance, to show the relationship between different areas on a map. Low and high values have dark colors, mid-range colors are light.</p> <p>Diverging classes: Can be seen as two sequential classes combined, with the mid-range shared. The two extremes, high and low, are emphasized with dark colors with contrasting hues, and the mid-range critical values are emphasized with light colors.</p> <p>This expression is a color code must be cleared.</p>
Reverse colors	<p>When selected, the color scheme is reversed.</p> <p>This expression is a color code must be cleared.</p>
Range	<p>This setting sets the value ranges for coloring results in the visualization.</p> <p>Auto: Qlik Sense creates ranges based on the detected minimum and maximum values.</p> <p>Custom: Qlik Sense automatically creates ranges based on user-defined minimum and maximum values. You must enter values or expressions that calculate those values in the fields Min and Max. You can enter an expression by clicking fx to open the expression editor</p> <p>This expression is a color code must be cleared.</p>

Color by expression in table visualizations

Expressions can be used to color table and pivot table backgrounds and text. This enables you to use expressions to define both the colors used and the conditional values upon which the colors are applied in a visualization. You could, for example, use expressions to change text and background colors depending on the values within different table cells.

The following options are available in **Data** for coloring table and pivot table visualizations:

Options for coloring table and pivot table visualizations

UI item	Description
Background color expression	Enter an expression by clicking  to open the expression editor. The text color automatically changes to white when a dark background color is used. For more information, see <i>Examples (page 469)</i> .
Text color expression	Enter an expression by clicking  to open the expression editor. If you use the same expression as in the background color, the text will not be visible. For more information, see <i>Examples (page 469)</i> .

Color by expression

Coloring by expression sets colors using a user-defined expression. When coloring by expression, you can define both what colors to use and which values to use them with, enabling more control over how colors are used in the visualization.

For example, you might highlight values of particular interest, or differentiate between values within different value ranges. Coloring by expression can also be used to color a visualization by values not included within a visualization, such as coloring products and the sum of their monthly sales by the country of origin for the product.

When you select to color **By expression**, you can choose to either use the expression as a color code or to define how **By measure** color options are applied to the visualization using an expression.

The following visualizations support color by expression:

- Bar chart
- Combo chart
- KPI chart
- Line chart
- Map
- Pie chart
- Pivot table
- Scatter plot
- Table
- Treemap

You can also use color by expression to set a background color on a sheet. For more information, see [Structuring an app using sheets \(page 9\)](#).



Legend selection is not available in a visualization when coloring by expression. Visualizations that are colored by expression with a color code do not support legends.

Coloring by expression as a color code

By default, if you choose to color by expression, **The expression is a color code** is enabled. If you have this option selected, your expression must include a color code in a supported expression format to define the colors to use. Using this method provides you with manual control over visualization colors as well as the conditions for the colors being used in a visualization. With tables and pivot tables, you can use expressions to define the background color and the text color of columns.



When coloring by expression, objects in visualizations are colored gray if the expression contains errors or if objects in the visualization have multiple colors they could be assigned in the expression.

Examples

Here are a few examples to show what you can do with expressions by color.

Example: Coloring by random color range

```
argb(255,rand()*255,rand()*255,rand()*255)
```

This example uses ARGB color. It starts with alpha value that sets full opacity, and then uses the rand() function to generate random values for the red, green, and blue colors, creating a random color.

Example: Coloring by single measure value

```
if(sum([Budget Amount]) > 1000000, 'cornflowerblue', magenta())
```

In this example, there is a condition. If sum([Budget Amount]) is greater than 1 million, the corresponding measure values will be colored 'cornflowerblue', otherwise they will be colored magenta.

'cornflowerblue' is the color keyword for the color `rgb(100, 149, 227)`.

`magenta()` is the Qlik Sense color function that generates a magenta color.

Example: Coloring by single measure value using an aggregated expression

```
if(avg(Value) > avg(Total aggr(avg(Value), Name)), Blue(), Brown())
```

In this example, there is a condition. If the `avg(Value)` value is greater than the aggregated `avg(Value)` value of the entire table, then the corresponding measure value is colored blue. If the `avg(Value)` value is less than the aggregated `avg(Value)` value of the entire table, then the corresponding measure value is colored brown.

Example: Coloring by multiple measure values

```
if(Sum(Sales) > 3000000, 'green', if(Sum(Sales) > 2000000, 'yellow', if(Sum(Sales) > 1000000, 'orange', red())))
```

In this example, there are multiple conditions. If Sum(Sales) is greater than 3,000,000, then corresponding measure values will be colored green. If Sum(Sales) is between 2,000,000 and 3,000,000, then the corresponding measure values will be colored yellow. If Sum(Sales) is between 1,000,000 and 2,000,000, the corresponding measure values will be colored orange. All other measure values will be colored red.

Example: Coloring by multiple dimensions

```
if([CompanyName]= 'A Corp', rgb(100, 149, 227), if([CompanyName]= 'B Corp', rgb(100, 149, 200), if([CompanyName]= 'C Corp', rgb(100, 149, 175), if([CompanyName]= 'D Corp', rgb(100, 149, 150), 'grey'))))
```

In this example, the expression is used to define a RGB color for each specific dimension value in the field CompanyName.

Example: Coloring table object font and background by measure value.

```
if(Sum([Sales]) < 10000, 'red', green())
```

```
if(Sum([Sales]) > 200000, 'gold', )
```

In this example, two expressions are used to color the background and text of the *Sales* column. Measure values in *Sales* that are lower than \$10000 have a red background color, all other values have a green background. In addition, the values that are higher than \$200000 have the text color 'gold'.

Customer KPIs				
Customer	Sales	Quantity	Margin (%)	# of Invoices
Totals	\$104,852,674.81	1,816,372	4127.8%	38314
A-2-Z Solutions	\$196,298.49	1,418	3841.7%	58
A-ARVIN Laser Resources	\$4,053.05	25	3792.6%	13
A Superior System	\$103,728.12	868	4074.5%	167
A&B	\$92,120.60	891	4202.9%	18
A&G	\$12,502.61	133	4708.0%	12
A&R Partners	\$30,392.45	156	3409.9%	6
A1 Datacom Supply	\$259,599.52	5,830	4025.7%	111
a2i	\$451.64	14	5983.7%	9
A2Z Solutions	\$69,977.36	454	4121.1%	94
AA-Wizard	\$94,209.44	917	4660.6%	41
Aadast	\$351,243.31	881	3707.3%	35
Aaron D. Meyer & Associates	\$90,017.11	1,869	4404.1%	58

Coloring by expression without a color code

You can color by expression without a color by disabling **The expression is a color code** when you enable coloring by expression. In this method of coloring, the expression is used to evaluate to a numeric value that is plotted against a **By measure** chart gradient, treating the expression like a measure when coloring by measure.



When coloring by expression, objects in visualizations are colored gray if the expression contains errors or if objects in the visualization have multiple colors they could be assigned in the expression.

Example:

`100*Sum([Sales Margin Amount])/Sum([Sales Amount])`

In this example, charts would have **By measure** color gradients applied to them based on the profit margin percentile calculated in the expression.

Supported expression formats

The following formats are supported when you create your expressions.

RGB

With RGB colors you enter an integer value between 0 and 255 (or an expression that evaluates to such a value) for each of the colors red, green, and blue. The resulting color is produced by adding the three colors together.

Example:

`rgb(0,0,255)`

This example generates the color blue. Many of the RGB colors have a corresponding keyword in plain text that can be used instead of the RGB code. If you use '*blue*' as expression, you would get exactly the same color. Hexadecimal numbers are also supported, and the color blue has the string '#0000ff'.

ARGB

The ARGB color model has the same support as the RGB color model, but extends it with an additional alpha value to set the opacity of a color.

Example:

`argb(125,0,0,255)`

The first value (125), sets the alpha value. The value 0 generates full transparency and the value 255 full opacity.

HSL

In HSL, the color is defined by a hue value, a saturation value, and a luminosity value. You use values between 0 and 1. Hue is represented as an angle of the color circle (that is, the rainbow represented in a circle).

Saturation is full with the value 1 and a shade of gray with the value 0. Lightness with the value 1 is white, and black with the value 0. The value 0.5 is commonly used.

Example:

`hsl(0,0.5,0.5)`

This example generates a red color with medium saturation and lightness.

Color keywords

Qlik Sense supports W3C recommended color keywords. With color keywords, specific colors are defined by a name which corresponds to a RGB hex value. Enter the color name in the expression to use the color.

Use the following links to find out more about W3C color keywords:

- ➡ <http://www.w3.org/TR/CSS21/syndata.html#value-def-color>
- ➡ https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Example:

`'cornflowerblue'`

This example generates a blue color with the hex value of #6495ed and a RGB value of (100, 149, 237).

Qlik Sense color functions

The following color functions can be used in expressions when coloring by expression.

- black()
- darkgray()
- lightgray()
- white()
- blue()
- lightblue()
- green()
- lightgreen()
- cyan()
- lightcyan()
- red()
- lightred()
- magenta()
- lightmagenta()
- brown()
- yellow()

Creating an expression

You create expressions for colors in the properties panel.

Do the following:

1. In the properties panel, open **Appearance > Colors and legend**.
2. Click the **Colors** button to switch to **Custom**.
3. In the drop-down list, select the option **By expression**.
An expression text box is opened.
4. Enter your expression in the text box, or click  to open the expression editor.

If the expression is valid, the visualization is updated.

Visualization support for coloring methods

Not all Qlik Sense visualizations support the same coloring options. Additionally, some visualization types have specific behaviors or limitations when using certain coloring methods.

Color methods supported by visualizations and their limitations are determined primarily by the kinds of data the visualizations displays. For example, visualizations that only support displaying measures cannot be colored by dimension or through using master dimensions.

Color method support by visualization

The following table outlines color method support by visualization type.

Color methods that are supported in the visualization

Visualizations	Single	Multicolor	Master measure	Master dimension	Dimension	Measure	Expression
Bar chart	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Box plot	Yes	-	-	-	-	-	-
Combo chart	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Distribution plot	Yes	-	-	-	-	-	-
Filter pane	-	-	-	-	-	-	-
Gauge	Yes	Yes	Yes	-	-	-	-
Histogram	Yes	-	-	-	-	-	-
KPI	Yes	Yes	-	-	-	-	-
Line chart	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Map	Yes	-	-	Yes	Yes	Yes	Yes
Pie chart	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Pivot table	-	-	-	-	-	-	Yes
Scatter plot	Yes	-	-	Yes	Yes	Yes	Yes
Table	-	-	-	-	-	-	Yes
Text & image	-	-	-	-	-	-	-
Treemap	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Visualization coloring limitations and behaviors

Different visualizations have different behaviors with the methods of setting color in visualizations. This section outlines specific considerations when coloring different visualizations.

Line chart

Line charts do not support coloring by measure if they have two or more dimensions.

Map

Colors assigned to master dimensions that contain geopoint data or area data (polygons of geopoints) cannot be used to color a map.

Pie chart

Pie charts do not use master item colors when **Auto** is selected under **Colors and legend**.

Examples of visualization color settings

You can use a number of different methods to control the use of colors in your visualizations.

3 Visualizations

You can manually apply colors to your visualizations using the following methods:

- Color by single color
- Color by multiple colors
- Color by dimension
- Color by measure
- Color by expression

In the example dashboard below, each method of setting colors has a corresponding visualization. This section outlines each example as well as the specific settings used in the properties panel.



Color by single color

Visualizations can be colored with a single user-defined color. Colors can be selected from a palette or color wheel or by entering a hex color code.

In this example visualization, a single color has been applied to the line chart.

1. Color by single color



Properties panel settings

For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

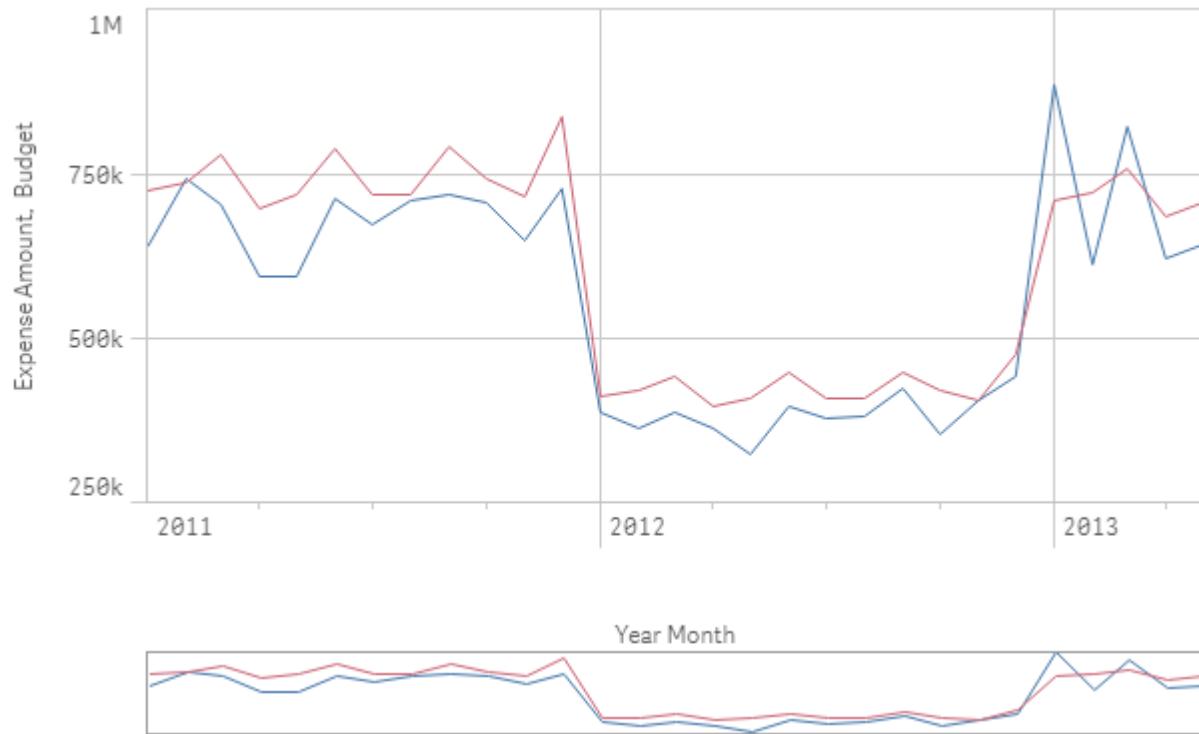
- **Colors:** Set to **Custom** and **Single color**. Color set as hex value `4477aa`.

Color by multiple colors

Visualizations with multiple measures can have different colors applied to each measure. When a visualization is colored using **Multicolor**, colors are automatically applied from a default color scheme of 12 colors or 100 colors.

In this example visualization, multiple colors have been applied to the measures of *Expense Amount* and *Budget* in the line chart.

2. Color by multiple colors



Properties panel settings

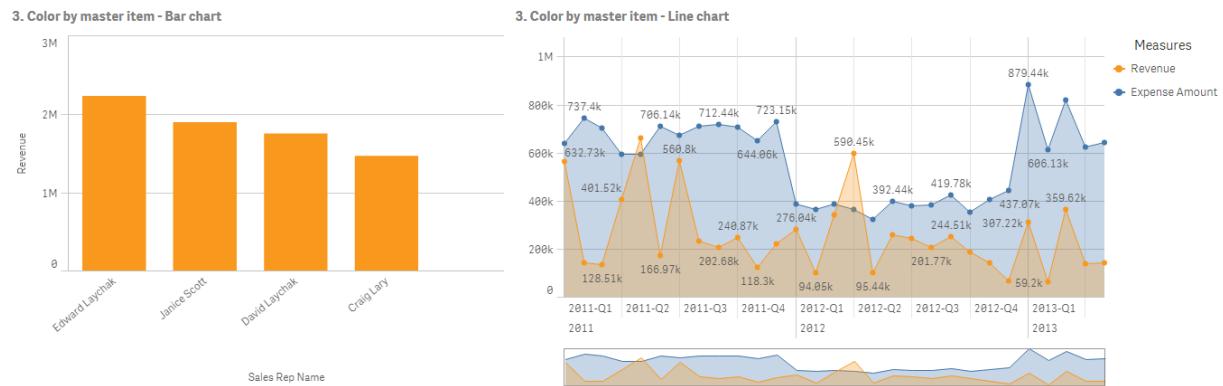
For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **Multicolored**.
- **Color scheme:** Set to **12 colors**.

Color by master item

Colors can be kept consistent across visualizations for dimensions or measures through setting colors in master items. When set to use master item colors, visualizations will use all colors associated to the master items in the visualization. Master item colors can be used when **Color** is set to **Single color** or **Multicolor**.

In these example visualizations, both the bar chart and line chart share a master measure, *Revenue*, that is colored orange. In each visualization, the same assigned color is used for each instance of *Revenue*. The line chart is colored by a second master measure, *Expense Amount*, which is colored blue.



Master measure settings

For this visualization, the following settings were applied to the master measures in **Edit measure**:

- **Color:** Hex color set as *f8981d* for *Revenue* and *4477aa* for *Expense Amount*.

For more information, see *Assigning colors to master items* (page 110).

Properties panel settings

For the bar chart, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **Single color**.
- **Use library colors:** Set to enabled.

For the line chart, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

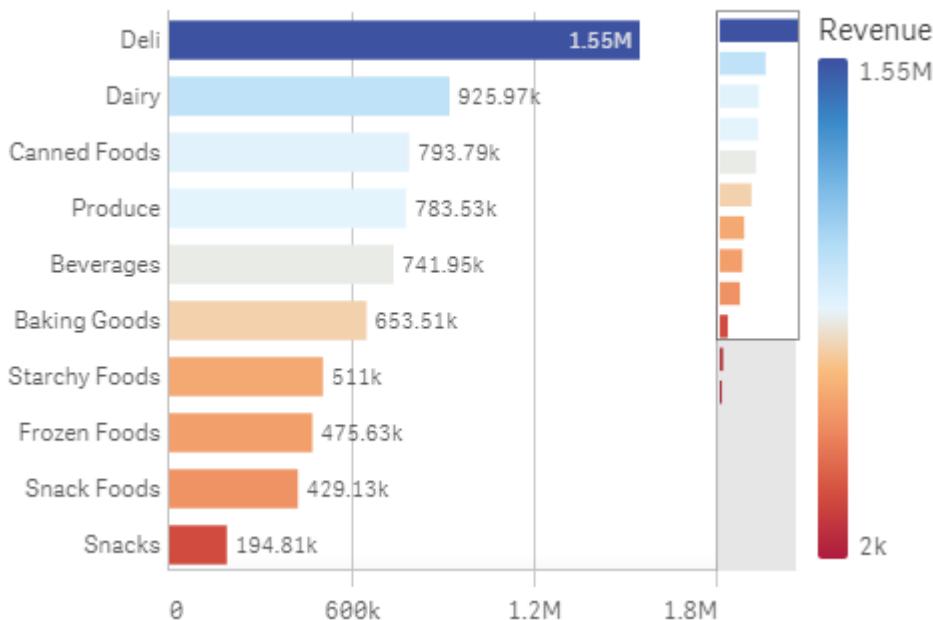
- **Colors:** Set to **Custom** and **Multicolor**.
- **Use library colors:** Set to enabled.

Color by measure

When a visualization colored by measure, sequential or diverging gradients or classes are applied to values in the chart based on the values of the selected measure. Visualizations can be colored by measures within the visualization or they can be colored with measures associated to values in the visualization.

In this example, this bar chart is colored by the measure used in the visualization, *Revenue*. A diverging gradient has been applied to the values in the chart based on the *Revenue* value for each dimension value.

4. Color by measure



Properties panel settings

For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **By measure**. The measure selected is *Revenue*.
- **Color scheme:** Set to **Diverging gradient**.
- **Reverse colors:** Set to enabled.
- **Range:** Set to **Auto**.

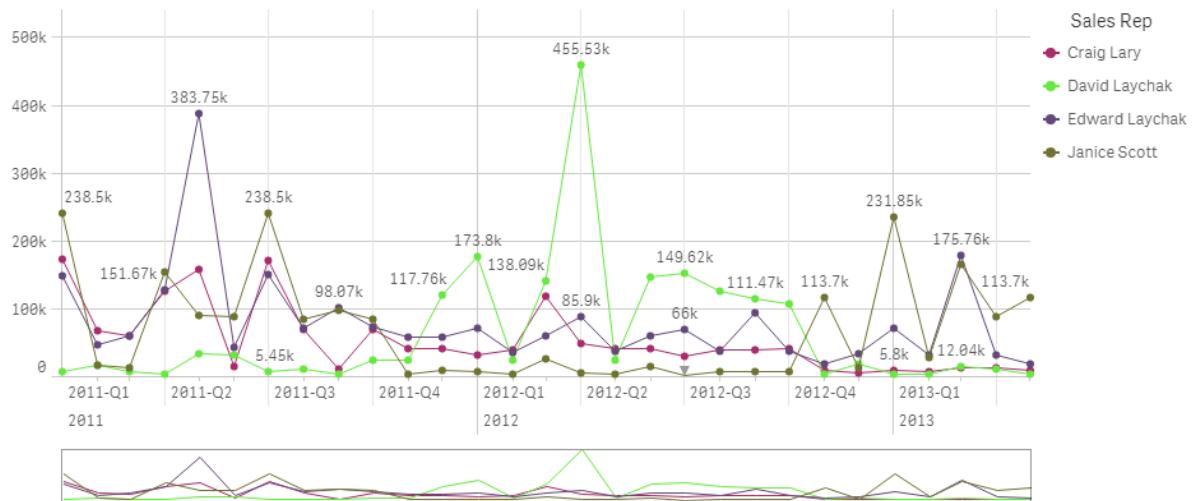
Color by dimension

When a visualization is colored by dimension, each value within the visualization is assigned a color based on an associated value from the coloring dimension. When colored by dimension, colors are automatically applied from a default palette set of 12 or 100 colors.

Example 1: Coloring by a dimension in the visualization

In this example, the line chart is colored by the dimension of the different sales representatives, using the **100 colors** scheme. Each sales representative has their own distinct color in the visualization.

5. Color by dimension



Line chart colored by dimension

Properties panel settings

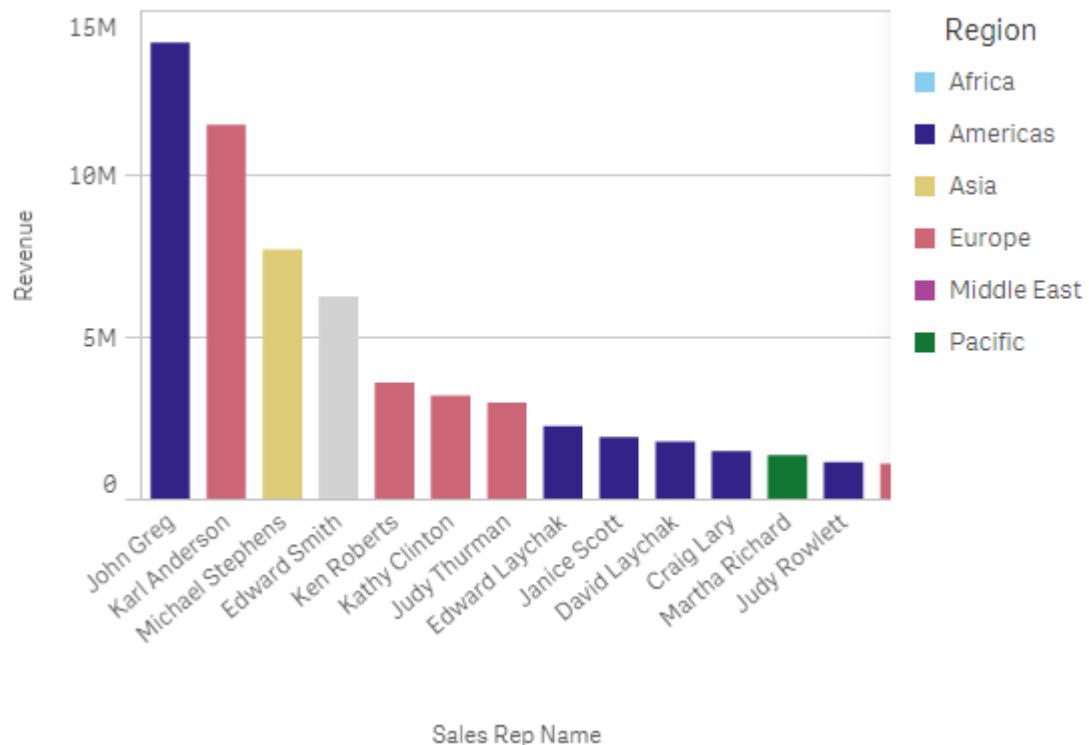
For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **By dimension**. The dimension *Sales Rep Name* is selected.
- **Persistent colors:** Set to enabled.
- **Color scheme:** Set to **100 colors**.

Example 2: Coloring by a dimension not included in the visualization

In this example, the bar chart is colored by the dimension of **Region**, using the **12 colors** scheme. The bar for each sales representative is colored by the region in which they work.

5. Color by dimension



Properties panel settings

For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **By dimension**. The dimension *Region* is selected.
- **Persistent colors:** Set to enabled.
- **Color scheme:** Set to **12 colors**.

Color by expression

You can use expressions to set specific colors to appear with specific values, enabling conditional coloring of values in your visualizations. When a visualization is colored by expression, you define the colors and how the colors are applied to values within the expression.

Example 1: Color by expression in a table

In this example, the table visualization uses two expressions, one for the background color and one for the text. These expressions apply conditional colors to the background and text based on which rows contain the top 10 and bottom 10 values for *Revenue*.

6. Color by expression

Customer	Revenue
Homebound	\$1,263,085.68
Icon Site Builders	\$9,420.32
Kari & Associates	\$7,364.12
Livermore Laboratories (LSLI)	\$50,151.75
MATRIX	\$512,901.49
Pacific Matics	\$24,625.51
Ra Co Amo	\$1,203,542.53
Ready-to-Run	\$98,191.57
Remedy	\$226,538.83
Reuters Usability Group	\$45,384.54
RFI Corporation	\$1,772,832.86
Satronix	\$126,630.22

Properties panel settings

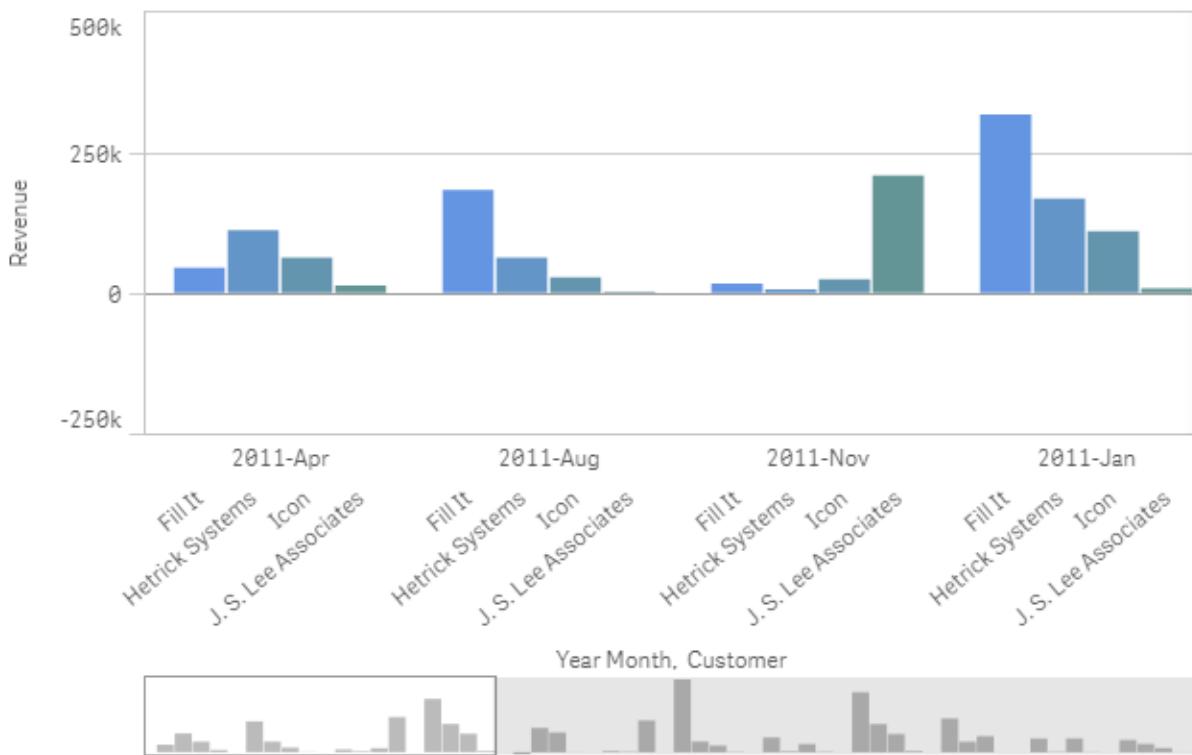
For this visualization, the following properties were set in the properties panel under **Data > Columns**:

- **Background color expression:** `if(Rank(Sum([Sales Quantity]*[Sales Price])) <= 10, 'honeydew', if(Rank(-Sum([Sales Quantity]*[Sales Price])) <= 10, 'mistyrose',))`
- **Text color expression:** `if(Rank(Sum([Sales Quantity]*[Sales Price])) <= 10, 'green', if(Rank(-Sum([Sales Quantity]*[Sales Price])) <= 10, 'red',))`

Example 2: Color by expression in a chart

In this example, the bar chart uses an expression to assign specific colors to different values in the *Customer* field.

6. Color by expression



Properties panel settings

For this visualization, the following properties were set in the properties panel under **Appearance > Colors and Legends**:

- **Colors:** Set to **Custom** and **By expression**.
- **Expression:** Set to `if([Customer]= 'Fill It', rgb(100, 149, 227), if([Customer]= 'Hetrick Systems', rgb(100, 149, 200), if([Customer]= 'Icon', rgb(100, 149, 175), if([Customer]= 'J. S. Lee Associates', rgb(100, 149, 150), 'grey'))))`.
- **This expression is a color code:** Set to enabled.

Converting a visualization to another kind of visualization

You can convert from one visualization type to another by dragging a chart from the assets panel on the left-hand side onto the visualization that you want to convert.

All properties that the original visualization has are transferred to the new type. The new visualization uses the dimensions, measures, and settings that are applicable to that visualization type. If a visualization requires an additional primary dimension or measure, the first listed alternative dimension or measure is used by default. If no alternative dimensions or measures exist and one is required the new visualization, you will be prompted to add one.

All properties from the original visualization are saved, even if they are not available or visible in the new visualization. This means that properties can become available again if you decide to convert to yet another visualization type where those properties are used.

Do the following:

1. While editing a sheet, drag a new chart from the assets panel onto the visualization that you want to convert.
The shortcut menu opens.
2. Select the conversion option.

The new visualization is displayed, using the data from the original visualization.



When you convert to a new visualization type, some of the settings from the original visualization may not be optimal for the new visualization, for example, the sorting order. Therefore, you may need to make some changes in the properties panel, so that the new visualization is displayed as you want.



You cannot convert to or from a map or a text & image visualization, nor can you convert a master visualization.

Embedding a visualization or a sheet in a web page

You can integrate Qlik Sense visualizations or sheets into an `iframe` element on a web page with the Single Integration API. This could for example be a web page on your intranet. You select which visualization or sheet to embed, and make some settings to customize selections, interaction and appearance. Then you can copy the `iframe` code containing the URL of the object and add it to your web page.

The embedded object will be subject to the same access rules as the original object and app. This means that anyone who wants to view the embedded object must have:

- Access to Qlik Sense.
- Access to the app. If the app is unpublished (in **My work**), you are the only one with access.
- Access to the sheet. If the sheet is unpublished (in **My sheets**) in a published app, you are the only one with access.
- Access to any bookmark used for selection. Private bookmarks will be applied for you, but not for anyone else.
- Same access to the data used in the selection as you, if section access is used to restrict data access.

Embedding a sheet

You can embed a sheet in your web page.

Do the following:

1. Click **Embed sheet** in the global menu.
2. Customize selections, appearance and interaction options for the embedded sheet.

3. Click **Open preview in new tab** to see a preview of the embedded sheet.
4. Click **Copy**.

You now have the `iframe` code of the sheet in your clipboard, ready to add to your web page.

Embedding a visualization

You can embed a visualization in your web page.

Do the following:

1. Right-click on the visualization or click the hover menu .
2. Select **Share**. Then choose **Embed**.
3. Customize selections, appearance and interaction options for the embedded visualization.
4. Click **Open preview in new tab** to see a preview of the embedded visualization.
5. Click **Copy**.

You now have the `iframe` code of the visualization in your clipboard, ready to add to your web page.

Setting appearance and interaction

You can customize how you can interact with the embedded object.

- **Allow interaction**

You can select if you want the user to be able to interact with the embedded object.

- **Enable context menu**

You can select if you want the context menu to be available when the embedded object is right-clicked.

- **Language**

You can select which language to use in menus for the embedded object.

- **Theme**

You can select which theme to use for the embedded object.

Selections in the embedded object

You can choose if the user can make selections in the embedded object, and which selection state to show in the embedded object.

- **Use current selections**

You can select to use the current selections in the app.

- **Show selections bar**

You can select to show the selections bar above the sheet.

- **Clear app selections on refresh**

You can select to clear all selections made in the app when the object is rendered.

- **Apply bookmark**

You can select to apply a bookmark and use the selections defined in the bookmark.

When the page containing the embedded object is rendered, the initial selection state is defined by the order of execution and your settings.

1. **Clear app selections on refresh**
2. **Apply bookmark**
3. **Use current selections**

When you make selections in the embedded object, they will be replicated in the app. If you select **Clear app selections on refresh**, for example, and use the embedded object, selections are cleared in the app.



You can use **Open preview in new tab** to preview the embedded object, and interact without affecting selections in the app.

Limitations of embedded objects

- **Embed sheet** and **Embed chart** features are not available on devices with a small screen.
- The sheet title is not included when embedding sheets.
- The URL is limited to 2083 characters due to browser limitations. If your URL is too long, this is most likely due to a large number of individual selections.



You can create a bookmark with the same set of selections, and apply the bookmark. This will create a shorter URL.

3.8 Troubleshooting - Creating visualizations

This section describes problems that can occur when creating visualizations in Qlik Sense.

I cannot find the fields in the assets panel

I can find **Charts** and **Master items** in the assets panel, but not **Fields**.

Possible cause

You are working with a published app. Some content is not available in the assets panel in a published app.

My chart is not sorted correctly

I set my chart to sort automatically on the dimension, but the results are not sorted correctly.

Possible cause

The dimension is an expression with a result that has a different data type than the data fields used in the expression.

Proposed action

Change sorting of the dimension to **Custom**, and select a sorting option that matches the result of the expression. For example, if your expression concatenates two fields to a string, like `Month(salesdate)& '/'&Day(salesdate)`, select to sort alphabetically.

My calendar measures display incorrect aggregations in visualizations

When I use my calendar measures in visualizations, I see incorrect aggregation results. For example, calendar measures created from identical fields and aggregations but different time ranges may display identical totals.

Possible cause

The table containing the aggregated field is not associated to the table containing the date field, preventing accurate aggregation of the field by the selected time ranges.

Proposed action

Create an association between the table containing the aggregated field and the table containing the date field. If there is no association possible, in **Data manager**, add a table that includes a date field that has an association to the table containing the aggregated field.

There are no time ranges to select in **Create calendar measures**

When I have a date field selected in the **Create calendar measures** dialog try to create calendar measures from a field, there are no time ranges to select in the **Create calendar measures** dialog.

Possible cause

The selected date field does not have the correct time flags to work with calendar measures. If you have no valid date fields, you cannot create calendar measures. If you have at least one valid date field, all date fields will be available in **Date field**. However, only those with the correct time flags set in autoCalendar enable the selection of time ranges from the Time Range drop-down list.

Proposed action

Select a date field that uses autoCalendar. If you are unsure which calendar is associated to your date field, date fields in the **Field** section of the **Assets** panel display which calendar it uses when clicked.

My date field selected for calendar measures does not use the correct calendar

I have two calendars to which I have manually added time flags. My time flags have the same names as those in autoCalendar, making both qualified for use with calendar measures. However, only one of my calendars has the same definition for the time flags as autoCalendar. I have a date field associated to both calendars . When I try to create calendar measures using that date field, the calendar with the correct names but different definitions than autoCalendar is used.

Possible cause

In cases where a date field is associated to multiple calendars and each calendar has the correctly named time flags set in it, calendar measures uses the first qualified calendars defined in your data load script.

Proposed action

Move the script section containing the calendar you want to use with calendar measures before other qualified calendars in your data load script.

I cannot edit a variable value

I cannot edit a variable that is listed in the variables dialog.

Possible cause

The variable is defined in the script.

Proposed action

Edit the variable in the script, using the data load editor, or delete the variable from the script, using the data load editor, to make the variable editable from the variables dialog.

For more information, see *Editing a variable* (page 130)



You cannot rename a variable.

The map is placing the locations in my location field incorrectly

When I add a location field to a layer in my map, the locations do not display in the correct places.

Possible cause

The map does not have enough context to locate the locations in your field. This can happen when a location in the field shares a name with a number of other possible locations on the map.

Proposed action

Set **Scope for Location** in your layer to **Custom** and enter additional information. Alternatively, qualify your location field using an expression that contains additional fields with relevant geographic information. If your location field contained cities and you had *Country* and *State* fields, you could use `[city]&',[&[State]&',[&[Country]`.

No map is displayed

I am using Qlik Geoanalytics server, the background map is not displayed.

Possible cause

Your browser cannot access `qlikcloud.com` or your firewall settings or proxies prevent the use of JavaScript from `qlikcloud.com`..

Proposed action

Visit the [status page](#) from your browser.

If your browser cannot access `qlikcloud.com` contact your system administrator for assistance.

If the page prints "ok" then your browser can access `qlikcloud.com`. Make sure that JavaScript from `qlikcloud.com` is allowed.

Do the following:

- Add https://*.qlikcloud.com as a trusted site. (recommended)
- Enable JavaScript from untrusted sites. (not recommended)

Error message: The data contains invalid geometries that could not be shown on the map. Review your data for errors and try again.

I loaded geoshapes from a KML file to Qlik Sense. When I try to add the field to my map, an error message tells me that my data contains invalid geometries that could not be shown on the map.

Possible cause

There is an error in your geometries that is preventing Qlik Sense from displaying them on the map or your geometry data may be in an invalid format.

Proposed action

Review your geometry data for errors and try again.

Error message: The following locations could not be found: <locations>.

Review the values in your data and try again.

I added my location field to my map and I received an error messaging telling me locations in my location field could not be found.

Possible cause

Qlik Sense could not find the location. There may be a spelling error in the name or the location is not available in the Qlik Sense location database.

Proposed action

Review the values in your data for errors and try adding the field again. If a location cannot be found, you can alternatively manually add it using coordinates for points or geoshapes for areas.

Error message: The following locations could not be located: <locations>.

Review the values in your data and try again.

I added a location field to my layer and I received an error message telling me that certain locations in my location field could not be located.

Possible cause

Qlik Sense could not find the location. There may be a spelling error in the name or the location is not available in the Qlik Sense location database.

Proposed action

Review the values in your data for errors and try adding the field again. If a location cannot be found, you can alternatively manually add it using coordinates for points or geoshapes for areas.

Error message: The following locations had more than one result: <locations>. Set a custom scope to clarify which locations to display.

I added a location field to my layer and I received an error message telling me that certain locations in my field had more than one possible result on the map.

Possible cause

One or more locations in your location field are ambiguous, with multiple possible locations on your map.

Proposed action

Set **Scope for Location** in your layer to **Custom** and enter additional information. Alternatively, qualify your location field using an expression that contains additional fields with relevant geographic information. If your location field contained cities, you , such as: [city]&' , '&[County]&' , '&[State].

Error message: Some lines could not be shown because of invalid data in the width expression. Review your data for errors and try again.

I entered a field or expression to control the width of the lines on my map and I received an error message telling me that certain lines have invalid data.

Possible cause

There is an error in your width expression that is preventing Qlik Sense from displaying specific lines on the map.

Proposed action

Your expression may contain non-numeric values. Review your data for errors and try again.

Error message: Some density points could no be shown because of invalid data in the weight expression. Review your data for errors and try again.

I entered a field or expression to control the density of the points on my map and I received an error message telling me that certain densities have invalid data.

Possible cause

There is an error in your weight expression that is preventing Qlik Sense from displaying specific densities on the map.

Proposed action

Your expression may contain non-numeric values. Review your data for errors and try again.

I added an image background layer and cannot see my image

I added an image background layer to my map visualization, but I cannot see it on the map.

Possible cause

Depending on the projection of your map and the scale of the image, the map may be too zoomed out to see the image.

Proposed action

Do one of the following:

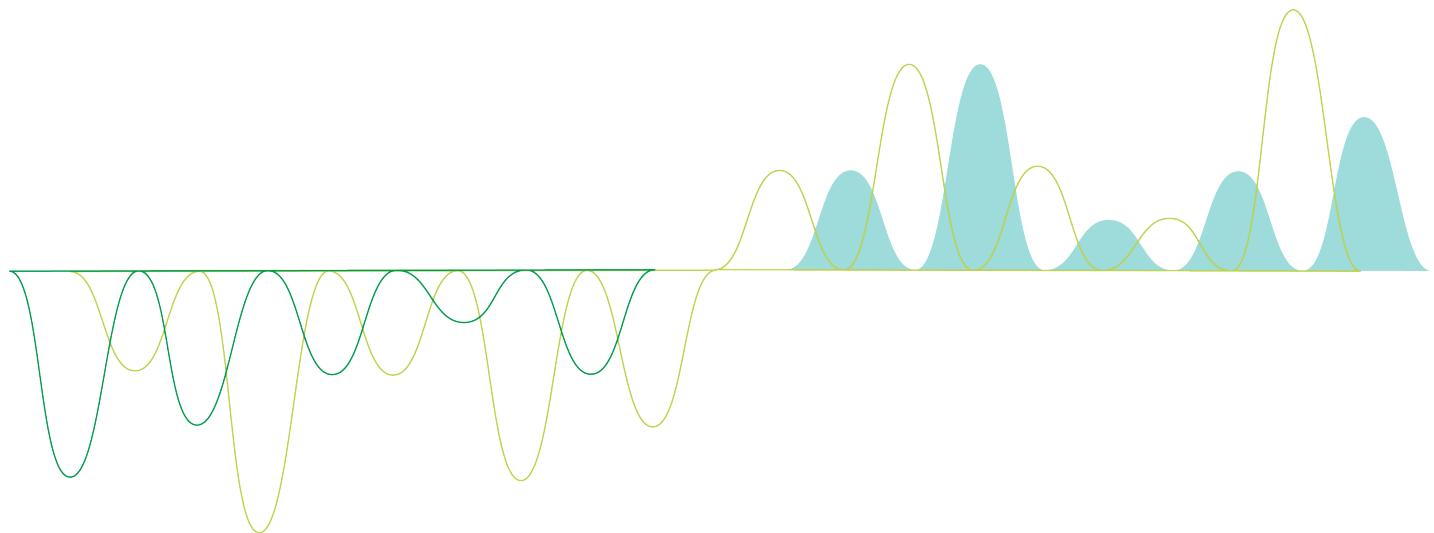
- In **Map settings**, enable **Auto-zoom**. Add a layer containing location data that would put it over the same area as your image background layer.
- In **Presentation**, enable **Show debug info**. The map now includes coordinates for the center of the displayed area.

Manage data

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 About this document	10
2 Managing data	11
3 Loading and managing data with Data Manager	12
3.1 Previewing a data table	12
3.2 Adding a new data table	12
3.3 Editing a data table	13
3.4 Deleting a data table	13
3.5 Managing data table associations	13
3.6 Applying changes and reloading data	14
3.7 Undo and Redo actions in Data manager	15
3.8 Viewing table transformation details in Data manager	15
3.9 Interaction between Data manager and the data load script	15
3.10 Concatenating tables in Data manager	16
3.11 Adding data to the app	16
In-App	17
File locations	17
Data connections	17
Data content	17
Attach files to this app	18
Connect to a new data source	18
Add data	18
Which data sources are available to me?	18
Adding data from an existing data source	18
Adding data from a new data source	20
Attaching data files and adding the data to the app	22
Adding data manually	25
Selecting data fields	26
Filtering data from files	31
Filtering data from data connectors	33
3.12 Editing a table	34
Renaming a table	34
Renaming a field	35
Managing associations to other tables	35
Changing field type and display format	35
Hiding fields from analysis	36
Assessing table field data before loading data	36
Replacing field values in a table	36
Setting field values as null in a table	36
Setting a custom order for field values	36
Splitting a field in a table	37
Grouping measure data into ranges	37
Viewing field transformation details	37
Unpivoting crosstab data	37
Updating a table from the data source	37
Adding a calculated field	38
Sorting a table	38
Undo and redo actions	38

Contents

Associating data in the table editor	38
Using calculated fields	40
Changing field types	54
Hiding fields from analysis	55
Assessing table field data before loading data	56
Replacing field values in a table	58
Setting field values as null in a table	60
Customizing the order of dimension values	61
Splitting a field in a table	62
Grouping measure data into ranges	64
Unpivoting crosstab data in the data manager	67
3.13 Concatenating tables in Data manager	69
Automatically concatenating tables	70
Forcing concatenation between tables	70
Splitting concatenated tables	72
3.14 Joining tables in Data manager	73
Join operators	74
Joining tables	76
Splitting joined tables	78
3.15 Viewing table and field transformation details in Data manager	79
Viewing table details	79
Viewing field details	79
3.16 Step-by-step - Combining tables using forced concatenation	80
Concatenation at a glance	80
Walkthrough - Forced concatenation	80
A step further - adding a new table and concatenating the data fields	87
3.17 Synchronizing scripted tables in Data manager	89
Synchronizing scripted tables	90
Removing managed scripted tables	90
3.18 Managing data associations	90
Associating tables using the Recommended associations panel	91
Associating tables manually	93
Breaking associations	94
Editing associations	94
Previewing data	95
Synthetic keys	95
Limitations	95
Applying changes and reloading data	96
4 Loading and transforming data with scripting	97
4.1 Interaction between Data manager and the data load script	97
4.2 Using the data load editor	98
Toolbar	98
Data connections	98
Text editor	98
Sections	98
Output	99
Connect to data sources in the data load editor	99

Contents

Select data in the data load editor	101
Edit the data load script	108
Organizing the script code	111
Debug the data load script	112
Saving the load script	115
Run the script to load data	115
Keyboard shortcuts in the Data load editor	115
4.3 Understanding script syntax and data structures	117
Extract, transform, and load	117
Data loading statements	118
Execution of the script	118
Fields	119
Logical tables	123
Data types	134
Dollar-sign expansions	137
Using quotation marks in the script	149
Wild cards in the data	153
NULL value handling	154
4.4 Guidelines for data and fields	157
Guidelines for amount of loaded data	157
Upper limits for data tables and fields	157
Recommended limit for load script sections	157
Descriptions for number and time formats	157
4.5 Working with QVD files	161
Purpose of QVD files	162
Creating QVD files	162
Reading data from QVD files	162
QVD format	163
4.6 Configuring analytic connections in Qlik Sense Desktop	163
Qlik open source SSE repositories	164
Description of the elements	164
5 Managing data security with Section Access	165
5.1 Sections in the load script	165
Section Access system fields	165
5.2 Managing user access to an app	167
5.3 Managing user access to specific data in an app	168
Managing access to row-level data	168
Managing access to column-level data	169
Managing access to user groups	170
5.4 Using impersonation to reload data	171
5.5 Managing user access in a multi-cloud environment	171
5.6 Using Section Access and Insight Advisor Chat	172
5.7 Guidelines and tips for using Section Access	173
6 Managing big data with on-demand apps	174
6.1 On-demand app components	174
6.2 Constructing on-demand apps	175
6.3 Publishing on-demand apps	176

Contents

6.4 Advantages of on-demand apps	176
6.5 Limitations	177
6.6 Creating an on-demand selection app	177
6.7 Creating an on-demand template app	178
Structure of a template app	178
Single Sign-On (SSO)	179
Reload nodes for template apps	180
Binding expressions in on-demand template apps	180
6.8 Building an on-demand app	187
7 Managing data with dynamic views	190
7.1 Dynamic views overview	190
Dynamic views	191
Dynamic view template apps	192
Dynamic charts	192
7.2 Dynamic views limitations	192
7.3 Streams and dynamics views	193
7.4 Creating dynamic views and charts	193
Creating dynamic views	193
Adding dynamic charts to sheets	194
Editing dynamic views	194
7.5 Using dynamic views and charts	195
Selections in dynamic views	195
Viewing dynamic view details	197
Refreshing dynamic views	200
8 Connecting to data sources	201
8.1 Create a connection	201
8.2 Data connection types	201
Attached files	201
Database connectors	201
Essbase	202
Local or network files	202
ODBC connections through DSN	202
Qlik Web Connectors	202
REST	203
Salesforce	204
SAP	204
Web files	204
Web Storage Provider Connectors	204
Third-party connectors	204
8.3 Where is the data connection stored?	204
8.4 Loading data from files	205
File formats	205
Connection types	205
How do I load data from files?	205
Loading files from local and network file folders	206
Loading files from web resources	206

Contents

Loading data from Microsoft Excel spreadsheets	207
8.5 Loading data from databases	210
Loading data from an ODBC database	210
ODBC	211
Loading data from ODBC data sources	211
OLE DB	214
Logic in databases	216
8.6 Accessing large data sets with Direct Discovery	216
Differences between Direct Discovery and in-memory data	217
Direct Discovery field types	223
Data sources supported in Direct Discovery	224
Limitations when using Direct Discovery	225
Multi-table support in Direct Discovery	227
Using subqueries with Direct Discovery	229
Logging Direct Discovery access	231
9 Viewing and transforming the data model	232
9.1 Moving tables	232
9.2 Resizing tables	233
9.3 Data model performance	233
9.4 Previewing tables and fields in the data model viewer	236
Showing a preview of a table	237
Showing a preview of a field	237
9.5 Creating a master dimension from the data model viewer	238
9.6 Creating a master measure from the data model viewer	238
10 Best practices for data modeling	239
10.1 Turning data columns into rows	239
10.2 Turning data rows into fields	239
10.3 Loading data that is organized in hierarchical levels, for example an organization scheme	240
10.4 Loading only new or updated records from a large database	241
10.5 Combining data from two tables with a common field	241
10.6 Matching a discrete value to an interval	241
10.7 Handling inconsistent field values	242
10.8 Handling inconsistent field value capitalization	243
10.9 Loading geospatial data to visualize data with a map	244
10.10 Loading new and updated records with incremental load	244
Append only	245
Insert only (no update or delete)	245
Insert and update (no delete)	245
Insert, update and delete	246
10.11 Combining tables with Join and Keep	247
Joins within a SQL SELECT statement	247
Join	248
Keep	248
Inner	248
Left	249
Right	251

Contents

10.12 Using mapping as an alternative to joining	252
10.13 Working with crosstables in the data load script	253
Unpivoting a crosstab with one qualifying column	254
Unpivoting a crosstab with two qualifying columns	255
10.14 Generic databases	256
10.15 Matching intervals to discrete data	258
Intervalmatch example	258
Using the extended intervalmatch syntax to resolve slowly changing dimension problems	259
10.16 Creating a date interval from a single date	261
10.17 Loading hierarchy data	264
10.18 Loading your own map data	265
Supported name data for fields in a map visualization	265
Loading point and area data from a KML file	266
Loading map data with data profiling	266
Loading and formatting point data	267
10.19 Data cleansing	269
Mapping tables	269
Using a mapping table	270
11 Customizing logical models for Insight Advisor	272
11.1 Building logical models for Insight Advisor with Business logic	272
Understanding logical models	273
Customizing logical models	274
Enabling custom business logic	274
Resetting business logic	274
Disabling business logic	275
Defining fields and groups	275
Setting logical model scope with packages	279
Creating drill-down analysis with hierarchies	281
Applying behaviors to logical models	282
Defining analysis periods with calendar periods	284
11.2 Creating vocabularies for Insight Advisor	292
Limitations	292
Adding synonyms to Insight Advisor	293
Adding custom analyses to Insight Advisor	294
Adding example questions to Insight Advisor	299
11.3 Tutorial – Customizing how Insight Advisor interprets data	300
What you will learn	300
Who should complete this tutorial	300
What you need to do before you start	300
Lessons in this tutorial	300
Further reading and resources	301
What is Insight Advisor and business logic?	301
Enabling a custom logical model	304
Customizing fields and groups	305
Configuring your packages	311
Reviewing your hierarchies	312
Configuring your calendar periods	314

Contents

Configuring your behaviors	319
Creating vocabularies	321
12 Troubleshooting - Loading data	324
12.1 Attaching a file by dropping it in Add data does not work	324
12.2 Character set problems with non-ANSI encoded data files	324
12.3 Circular references warning when loading data	324
12.4 Columns are not lining up as expected when selecting data from a fixed record file	325
12.5 Connector is not working	325
The connector is not properly installed	325
The connector is not adapted for Qlik Sense	325
12.6 Data connection stops working after SQL Server is restarted	326
12.7 Data load editor does not display the script	326
12.8 Data load script is executed without error, but data is not loaded	327
A statement is not terminated with a semicolon	327
Single quote character inside a string	327
12.9 Data manager does not show tables in app that contains data	327
12.10 Data manager work flows are broken for all users creating apps on a server	328
12.11 Data selection problems with an OLE DB data source	328
12.12 Date fields are not recognized as date fields in sheet view	328
Data profiling was disabled when the table was added	328
Date format was not recognized	329
12.13 Error message "Invalid path" when attaching a file	329
12.14 Errors when loading an app converted from a QlikView document	329
Absolute file path references are used in the script	329
Unsupported functions or statements are used in the script	330
12.15 Microsoft Excel: Loading data from files in data manager or data load editor fails	330
12.16 Microsoft Excel: Problems connecting to and loading data from files through ODBC	330
12.17 Running out of disk space	330
12.18 Synthetic keys warning when loading data	331
12.19 Tables with common fields are not automatically associated by field name	331

1 About this document

This document describes how to add and manage data, how to build a data load script for more advanced data models, how to view the resulting data model in the data model viewer, and presents best practices for data modeling in Qlik Sense.



For detailed reference regarding script functions and chart functions, see the Script syntax and chart functions.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

You find the online help, additional guides and much more at help.qlik.com/sense.

2 Managing data

When you have created a Qlik Sense app, the first step is to add some data that you can explore and analyze. This section describes how to add and manage data, how to build a data load script for more advanced data models, how to view the resulting data model in the data model viewer, and presents best practices for data modeling in Qlik Sense.

There are two ways to add data to the app.

- **Data manager**

You can add data from your own data sources, or from other sources such as Qlik DataMarket, without learning a script language. Data selections can be edited, and you can get assistance with creating data associations in your data model.

- **Data load editor**

You can build a data model with ETL (Extract, Transform & Load) processes using the Qlik Sense data load script language. The script language is powerful and enables you to perform complex transformations and creating a scalable data model.



*You can convert a data model built in **Data manager** into a data load script, which can be developed further in **Data load editor**, but it is not possible to convert a data load script to a **Data manager** data model. The **Data manager** data model and data tables defined in the data load script can still co-exist, but this can make it harder to troubleshoot problems with the data model.*

3 Loading and managing data with Data Manager

Add and manage data from the **Data manager** so that you can use the data in your app.

There are two views in data manager:

-  **Associations**

You can create and edit association between tables.

-  **Tables**

You get an overview of all data tables in the app, whether you added them using **Add data**, or loaded them with the data load script. Each table is displayed with the table name, the number of data fields, and the name of the data source.

3.1 Previewing a data table

You can preview a table to see which columns it contains, and a sample set of the data.

Do the following:

- Select the data table you want to preview.

A preview of the table data set is displayed.

3.2 Adding a new data table

You can quickly add a data table to your app. Open the **Data manager** and then click  . You can also click **Add data** in the  . You are also prompted to add data when you create a new app.

You can add data from the following data sources:

Data sources

Data source	Description
In-App	Select from data sources that are available in your app. These can be files that you have attached to your app. You can also create a data source and manually add data to it using Manual entry .
File locations	Select from files on a network drive, for example a drive that has defined by your administrator.
Data connections	Select from existing data connections that have been defined by you or an administrator.

3.3 Editing a data table

You can edit all the data tables that you have added with **Add data**. You can rename the table and fields in the data table, and update the fields from the data source. It is also possible to add a calculated field and adjust date and time formats.

Do the following:

1. Click  on the data table you want to edit.
The data table editor opens, and you can perform the edits and transformations you want to do.
2. Click **Close** to return.

The table is now marked **Pending update**, and the changes will be applied to the app data the next time you reload data.



*You can only edit data tables added with **Add data**. If you click  on a table that was loaded using the load script, the data load editor opens. For more information, see Using the data load editor (page 98).*

3.4 Deleting a data table

You can only delete data tables added with **Add data**. Data tables that were loaded using the load script can only be removed by editing the script in the data load editor.

Do the following:

- Click  on the data table you want to delete.

The table is now marked **Pending delete** and will be removed the next time you reload data.

You can undo and redo your delete actions by clicking  and .



If you have used fields from the data table in a visualization, removing the data table will result in an error being shown in the app.

3.5 Managing data table associations

When you add several tables that need to be associated, the perfect situation is that the tables associate with key fields that have identical names in the different tables. If that is the case, you can add them to Qlik Sense with the data profiling disabled option of **Add data**, and the result will be a data structure with the tables associated correctly.

If you have less than ideal data sources, there are a number of possible association problems.

3 Loading and managing data with Data Manager

- If you have loaded two fields containing the same data but with a different field name from two different tables, it's probably a good idea to name the fields identically to relate the tables.
- If you have loaded two fields containing different data but with identical field names from two different tables, you need to rename at least one of the fields to load them as separate fields.
- If you have loaded two tables containing more than one common field.

If you want to associate your data, we recommend that you use the **Add data** option with data profiling enabled. This is the default option. You can verify this setting by clicking  beside the **Add data** button in the lower right corner of the Add Data page.

Qlik Sense performs data profiling of the data you want to load to assist you in fixing the table association. Existing bad associations and potential good associations are highlighted, and you get assistance with selecting fields to associate, based on analysis of the data.

You can manage table associations in two different ways:

- In the  **Associations** view of data manager.
You can create associations based on Insight Advisor recommendations, or create custom associations based on one or more fields.
- Using the **Associate** option in the table editor.
You can create custom associations and composite key associations based on several fields.



If you disable data profiling when adding data, Qlik Sense will associate tables based on common field names automatically.

3.6 Applying changes and reloading data

Changes that you have made in the **Data manager** will not be available in the app until you have reloaded data. When you reload data, the changes are applied and any new data that you have added is loaded from the external data sources. Data that you loaded previously is not reloaded.

You can reload all the data from the external data sources by using the  button in the **Data manager** footer.



The  button reloads all the data for the selected table. It does not reload all the data for all the tables in the app.

If the data in **Data manager** is out of sync with the app data, the **Load data** button is green. In the **Associations** view, all new or updated tables are indicated with *, and deleted tables are a lighter color of gray. In the **Tables** view, all new, updated, or deleted tables are highlighted in blue and display an icon that shows the status of the table:

3 Loading and managing data with Data Manager

- Tables marked with **Pending delete**  will be deleted.
- Tables marked with **Pending update**  will be updated with fields that have been added, renamed, or removed, or the table will be renamed.
- Tables marked with **Pending add**  will be added.

Do the following:

- Click **Load data** to load the changes in the app.

The app data is now updated with changes you made in **Data manager**.

To apply changes and reload all the data in the selected table from the external data sources:

Do the following:

- Click the  button in the **Data manager** footer.

3.7 Undo and Redo actions in Data manager

When you are editing in **Data manager**, you can undo or redo some actions by clicking  and 

The log of actions is cleared if you:

- Change view, for example, going from the table overview to **Associations**.
- Load data.
- Close **Data manager**.

3.8 Viewing table transformation details in Data manager

You can view the operations and transformations performed on a table in **Data manager** using the **Details** dialog. The **Details** dialog is available in the **Associations** and **Table** views.

Details displays the current operations and transformations made to the selected table. This shows you the source of a table, the current changes that have been made, and the sequence in which the changes have been applied. **Details** enables you to more easily understand how a table got into a current state. You can use **Details**, for example, to easily see the order in which tables were concatenated.

3.9 Interaction between Data manager and the data load script

When you add data tables in the **Data manager**, data load script code is generated. You can view the script code in the **Auto-generated section** of the data load editor. You can also choose to unlock and edit the generated script code, but if you do, the data tables will no longer be managed in the **Data manager**.

3 Loading and managing data with Data Manager

By default, data tables defined in the load script are not managed in **Data manager**. That is, you can see the tables in the data overview, but you cannot delete or edit the tables in **Data manager**, and association recommendations are not provided for tables loaded with the script. If you synchronize your scripted tables with **Data manager**, however, your scripted tables are added as managed scripted tables to **Data manager**.



*If you have synchronized tables, you should not make changes in the data load editor with **Data manager** open in another tab.*

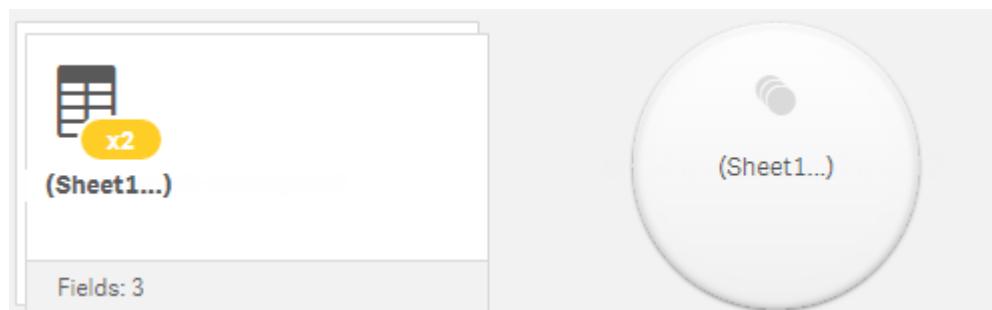
You can add script sections and develop code that enhances and interacts with the data model created in **Data manager**, but there are some areas where you need to be careful. The script code you write can interfere with the **Data manager** data model, and create problems in some cases, for example:

- Renaming or dropping tables added with **Data manager** in the script.
- Dropping fields from tables added with **Data manager**.
- Concatenation between tables added with **Data manager** and tables loaded in the script.
- Using the **Qualify** statement with fields in tables added with **Data manager**.
- Loading tables added with **Data manager** using **Resident** in the script.
- Adding script code after the generated code section. The resulting changes in the data model are not reflected in **Data manager**.

3.10 Concatenating tables in **Data manager**

Concatenation combines two tables into a single table with combined fields. It consolidates content, reducing the number of separate tables and fields that share content. Tables in **Data manager** can be automatically or forcibly concatenated.

Concatenated table in Tables view and Associations view.



3.11 Adding data to the app

You can add data to your app quickly. Open the **Data manager** and then click **+**. You can also click **Add data** in the **⋮**. You are also prompted to add data when you create a new app. When you are editing a sheet, you can also click **Add** in the **Fields** panel to add data.

3 Loading and managing data with Data Manager



The add data options and data sources that are available to you depend on your Qlik Sense platform and configuration.

Add data view

The screenshot shows the Qlik Sense Data Manager interface. On the left, there's a sidebar with navigation links: 'New', 'IN-APP' (Attached files, Manual entry), 'FILE LOCATIONS' (File locations, QVD Catalog), 'DATA CONTENT' (Qlik DataMarket), and 'DATA SOURCES'. The main area has two sections: 'Attach files to this app' (with a 'Drop a file here or click to select a file' button) and 'Connect to a new data source' (listing various data connectors like Amazon Redshift, Apache Drill, Apache Hive, etc.).

In-App

Attached files. Deployments: Qlik Sense Enterprise on Windows. Click to attach files to the app, or to view the files that have been attached to the app. You can load data from these files.

Manual entry. Click to create a table in-app and add the data to the app.

File locations

File locations. Deployments: Qlik Sense Enterprise on Windows. Provides access to folder locations that your administrator has defined.

My computer. Deployments: Qlik Sense Desktop.

Click to upload a data file, or to add data from a file that has already been uploaded.

Data connections

Deployments: All.

Displays connections that have been created to an external data source. The connection appears after it has been created under **Connect to a new data source**.

Click a connection to add data to the app.

Data content

Qlik Catalog Service. Platforms: Qlik Sense Enterprise on Windows.

3 Loading and managing data with Data Manager

Provides access to Qlik Catalog Service QVD sources. Only available if your administrator has created a connection to Qlik Catalog.

Click to add data to the app.

Attach files to this app

Click to attach a file to the app.

Connect to a new data source

Click to create a connection to a new data source.

Add data

Click to add data to an app. The button is enabled after you have created a connection and selected your data to load. You can add data with profiling enabled or disabled. Data profiling is recommended and enabled by default. Click *** to disable data profiling.

Which data sources are available to me?

What data source types are available to you depends on a number of factors:

- Access settings
Administrator settings determine which types of data sources you can connect to.
- Installed connectors
Qlik Sense contains built-in support for many data sources. Built-in connectors are installed automatically by Qlik Sense. To connect to additional data sources you may need to install connectors separately that are specifically for those data sources. Such separately installed connectors are supplied by Qlik or a third party.
- Local file availability
Local files on your desktop computer are only available in Qlik Sense Desktop. They are not available for use with a server installation of Qlik Sense.



If you have local files that you want to load on a server installation of Qlik Sense, you need to attach the files to the app, or transfer the files to a folder available to the Qlik Sense server, preferably a folder that is already defined as a folder data connection.

Adding data from an existing data source

You can add data to your app from connections that have already been defined by you or an administrator. These can be a database, a folder containing data files, or a connector to an external data source, such as Salesforce.com.



*Do not add a table in **Data manager** that has already been added as a scripted table with the same name and same columns in **Data load editor**.*

3 Loading and managing data with Data Manager

You can delete connections from **Add data** by right-clicking the connection and selecting .



If you delete a connection, you must delete any tables from **Data manager** that used that connection before you load data.

Do the following:

1. Open your app.
2. Open the **Data manager** and then click . You can also click **Add data** in the .
3. Under **Data connections**, select an existing connection.
Some connections go directly to their data sources, where you select tables and fields to load. For example, connections to Salesforce.com or a database go directly to the source for data selection.
4. Select the specific data source you want to add data from if the connection offers a selection. This differs depending on the type of data source.
 - File-based data sources: Select a file.
 - Databases: Set which database to use.
 - Web files: Enter the URL of the web file.
 - Other data sources: Specified by the connector.
5. Select the tables and fields to load.
6. Optionally, select to apply a data filter if you want to select a subset of the data contained in the fields you have selected.

If your data source is a file, select **Filters**. Beside the table to which you want to add a filter, click **Add filter**, select a field, select a condition, and then enter a value with which to filter.



Qlik Sense does not support filters on date fields from QVD files.

Note the following:

- You can apply multiple filters to the same field.
- You can remove filters in the **Associations** view of **Data manager** or from **Select data from source**. For the changes to take effect, reload data by clicking the **Load data** button.

For databases and connectors, when you select **Filter data**, a text box opens for the filter criteria.

Note the following:

- Filters are applied to field names from the database. If you rename a field in the **Data manager**, you have to apply the filter to the original field name from the database. For example, if a field is named EMP in your database, and you rename it to EMPLOYEE in the **Data manager**, you have to apply the filter `EMP = 'filter_value'`.
 - You can clear data filters in the **Associations** view of the **Data manager**. For the changes to take effect, reload data by clicking the **Load data** button. You have to split concatenated tables before clearing filters.
 - Filtering data is not currently available for all connectors, or for attached files.
7. Click **Add data** to open the data in the **Associations** view of the data manager. This allows you to continue to add data sources, transform the data, and associate the tables in **Data manager**.

3 Loading and managing data with Data Manager

Data profiling is enabled by default when you click **Add data**. Data profiling does the following:

- Recommends data associations.
- Auto-qualifies common fields between tables. This adds a unique prefix based on table name.
- Maps date and time fields to autoCalendar.

Tables are not associated on common field names automatically. You can associate tables in the **Associations** view.



If you want to load the data directly into your app, click and then disable data profiling. This will load the newly selected data from the external data source when you add data. Tables will be associated on common field names automatically. Date and time fields will not be created.

For more information, see *Managing data associations (page 90)*.

8. Click **Load data** when you are done preparing the data. If serious problems are detected, you need to resolve the problems in **Data manager** before you can load data into the app.

For more information, see *Troubleshooting - Loading data (page 324)*.

To reload all the data that you have selected from the external source, use the button in the **Data manager** footer. This ensures you get all the current data from the source for the selections you have made. Reloading all the data can take longer than loading only the new data. If the data you loaded previously has not been changed in the data source, it is not necessary to reload all the data.

Adding data from a new data source

You can add data to your app from a new data source. When you add data from a new data source, a connection to the data source is created in Data Connections, making it easier to add more data from the same data source.



*Do not add a table in **Data manager** that has already been added as a scripted table with the same name and same columns in **Data load editor**.*

You can delete connections from **Add data** by right-clicking the connection and selecting .



*If you delete a connection, you must delete any tables from **Data manager** that used that connection before you load data.*

Do the following:

1. Open an app.
2. Open the **Data manager** and then click . You can also click **Add data** in the .
3. Under **Connect to a new data source**, select a source.
4. Enter the connection parameters required by the data source.

3 Loading and managing data with Data Manager

For example:

- File based data sources require that you specify a path to the files and select a file type.
 - Databases such as Oracle and IBM DB2 require database properties and access credentials.
 - Web files require the URL of the web file.
 - ODBC connections require DSN credentials.
5. Select the tables and fields to load.
6. Optionally, select to apply a data filter if you want to select a subset of the data contained in the fields you have selected.

If your data source is a file, select **Filters**. Beside the table to which you want to add a filter, click **Add filter**, select a field, select a condition, and then enter a value with which to filter.



Qlik Sense does not support filters on date fields from QVD files.

Note the following:

- You can apply multiple filters to the same field.
- You can remove filters in the **Associations** view of **Data manager** or from **Select data from source**. For the changes to take effect, reload data by clicking the **Load data** button.

For databases and connectors, when you select **Filter data**, a text box opens for the filter criteria.

Note the following:

- Filters are applied to field names from the database. If you rename a field in the **Data manager**, you have to apply the filter to the original field name from the database. For example, if a field is named EMP in your database, and you rename it to EMPLOYEE in the **Data manager**, you have to apply the filter `EMP = 'filter_value'`.
 - You can clear data filters in the **Associations** view of the **Data manager**. For the changes to take effect, reload data by clicking the **Load data** button. You have to split concatenated tables before clearing filters.
 - Filtering data is not currently available for all connectors, or for attached files.
7. Click **Add data** to open the data in the **Associations** view of the data manager. This allows you to continue to add data sources, transform the data, and associate the tables in **Data manager**.

Data profiling is enabled by default when you click **Add data**. Data profiling does the following:

- Recommends data associations.
- Auto-qualifies common fields between tables. This adds a unique prefix based on table name.
- Maps date and time fields to autoCalendar.

Tables are not associated on common field names automatically. You can associate tables in the **Associations** view.



*If you want to load the data directly into your app, click *** and then disable data profiling. This will also reload all existing data from data sources when you add the data. Tables will be associated on common field names automatically. Date and time fields will not be created.*

For more information, see *Managing data associations (page 90)*.

8. Click **Load data** when you are done preparing the data. If serious problems are detected, you need to resolve the problems in **Data manager** before you can load data into the app.
For more information, see *Troubleshooting - Loading data (page 324)*.

Attaching data files and adding the data to the app

You can attach data files to your app, and then use the data in your app.

An attached file is only available in the app that it is attached to. There is no connection to your original data file, so if you update the original file you need to refresh the attached file.



To avoid exposing restricted data, remove all attached files with section access settings before publishing the app. Attached files are included when the app is published. If the published app is copied, the attached files are included in the copy. However, if section access restrictions have been applied to the attached data files, the section access settings are not retained when the files are copied, so users of the copied app will be able to see all the data in the attached files.

Limitations

- The maximum size of a file attached to the app is 50 MB.
- The maximum total size of files attached to the app, including image files uploaded to the media library, is 200 MB.
- It is not possible to attach files in Qlik Sense Desktop.

Attaching several data files quickly

The quickest and in most cases the easiest way to attach and add a set of data files to your app is to just drop the files in the app.

Do the following:

- Drop one or more data files in your app.
The files are uploaded and attached to the app, and added to the data model.

When you attach files this way Qlik Sense will try to select the optimal settings for loading the data, for example, recognizing embedded field names, field delimiters or character set. If a table is added with settings that are not optimal you can correct the settings by opening the table in the table editor, and clicking **Select data from source**.



It is not possible to drop files in the data load editor or in the data model viewer.

Attaching a single data file

You can attach data files one by one. This way you gain more control over file import settings, for example, embedded field names, field delimiters or character set used.

3 Loading and managing data with Data Manager



*Do not add a table in **Data manager** that has already been added as a scripted table with the same name and same columns in **Data load editor**.*

Do the following:

1. Open an app.
2. Open the **Data manager** and then click **+**. You can also click **Add data** in the **⋮**.
3. Drop a data file, or click and select a file from your computer to load.

If you try to attach a file with the same name as an already attached file, you get the option to replace the attached file with the new file.



Each attached file needs to have an unique file name.

4. Select the tables and fields to load.
5. Optionally, select to apply a data filter if you want to select a subset of the data contained in the fields you have selected.

If your data source is a file, select **Filters**. Beside the table to which you want to add a filter, click **Add filter**, select a field, select a condition, and then enter a value with which to filter.



Qlik Sense does not support filters on date fields from QVD files.

Note the following:

- You can apply multiple filters to the same field.
 - You can remove filters in the **Associations** view of **Data manager** or from **Select data from source**. For the changes to take effect, reload data by clicking the **Load data** button.
6. Click **Add data** to open the data in the **Associations** view of the data manager. This allows you to continue to add data sources, transform the data, and associate the tables in **Data manager**.

Data profiling is enabled by default when you click **Add data**. Data profiling does the following:

- Recommends data associations.
- Auto-qualifies common fields between tables. This adds a unique prefix based on table name.
- Maps date and time fields to autoCalendar.

Tables are not associated on common field names automatically. You can associate tables in the **Associations** view.



*If you want to load the data directly into your app, click ******* and then disable data profiling. This will also reload all existing data from data sources when you add the data. Tables will be associated on common field names automatically. Date and time fields will not be created.*

7. Click **Load data** when you are done preparing the data. If serious problems are detected, you need to resolve the problems in **Data manager** before you can load data into the app.

3 Loading and managing data with Data Manager

Deleting an attached file

When you delete a table based on an attached file in the data manager, the table is deleted from the data model, but the attached data file remains in the app. You can delete the data file from the app permanently.

Do the following:

1. Open an app.
2. Open the **Data manager** and then click .
3. Click  **Attached files**.
4. Delete the appropriate file.

The data file is now permanently deleted from the app.



*If you delete an attached file that is used in the app, you will not be able to reload the app until you have removed references to the file in **Data manager**, or in the load script. You edit load scripts in **Data load editor**.*

Reload data from an attached file

A file that you upload for an app is attached to the app. It is only available to that app.

There is no connection to your original data file. If you update the original file, you need to refresh the file that is attached to the app. You can then load the updated data into the app. After reloading the data in **Data manager**, click  (**Refresh data from source**) to see the updated data in the table view.



*Do not add a table in **Data manager** that has already been added as a scripted table with the same name and same columns in **Data load editor**.*

Do the following:

1. Open an app.
2. Open the **Data manager** and then click .
3. Click  **Attached files**.
4. Replace the existing file. The updated file needs to have the same name as the original file. The content of the data file is refreshed.
5. Click **Add data**. Ensure that data profiling is enabled by clicking .
6. In the **Associations** view or the **Tables** view, click the table.
7. Click  to update the data.
8. Click **Load data** to reload the data into the app.

3 Loading and managing data with Data Manager



If you have made changes to the field structure of the data file, that is, removed or renamed fields, this can affect the data model in your app, especially if this involves fields that are used to associate tables.

Adding data manually

Manual entry in **Add data** enables you to enter data into an editor in Qlik Sense, and then add it as a table in **Data manager**.

Manually entering data is useful if you want to use a limited set of data from another source. For example, if you only wanted a selection of rows from an Excel spreadsheet or from a table on a webpage to be loaded as a table into **Data manager**, you could copy and paste the selected data into **Manual entry** and then add it as a table in **Data manager**. It is also useful if you have a small amount of data that might be faster to add manually than importing from another data source.

To add data manually, you open **Add data**, select **Manual entry**, enter your data into the table, and then add the table to **Data manager**. The table editor starts with one row and two columns, but as you add data to the table, additional empty rows and columns are automatically added to the table.



Manual entry does not automatically save as data is entered. Data entered may be lost if the screen is refreshed, if the session times out, or if the connection is lost before the data is added to **Data manager**.

In addition to typing data, you can copy and paste it from other sources. **Manual entry** preserves the columns and rows of data copied from Excel tables.

There are a number of keyboard shortcuts you can use to work effectively and easily in **Manual entry**. Shortcuts behavior varies depending if you are selecting cells, rows, or columns, or if you are editing cells in the table. The following table contains the selecting shortcuts:

Keyboard shortcuts for selecting

Shortcut	Description
Arrow keys	Navigates between cell selection
Tab	Moves the cell selection right. If no cell exists to the right, it moves to the first cell in the next row.
Shift+Tab	Moves the cell selection left. If no cell exists to the left, it moves to the first cell in the previous row.
Enter	Toggles to editing mode
Delete	Deletes the current selection

The following table contains the editing shortcuts:

3 Loading and managing data with Data Manager

Keyboard shortcuts for editing

Shortcut	Description
Arrow keys	Moves the cursors in the cell.
Tab	Commits the edit and moves to the next cell to the right
Shift+Tab	Commits the edit and moves to the previous cell to the left
Enter	Commits the edit and moves to the next cell below
Shift+Enter	Commits the edit and moves to the previous cell above
Esc	Cancels the edit and toggles to selecting mode

Tables created using **Manual entry** can be edited later to add or remove content. For more information, see *Updating a table from the data source (page 37)*.

Adding data manually

Do the following:

1. Open an app.
2. Open the **Data manager** and then click . You can also click **Add data** in the *** menu.
3. Under **In-App**, click **Manual entry**.
4. Type a name for the table.
5. In the table editor, enter your data.
Double-click a cell to start entering data in the cell.
While editing a cell, clicking any other cell in the table commits your edit and selects the other cell.
6. When your data is complete, click **Add data**.

The table is added to **Data manager**.

Selecting data fields

You can select which tables and fields to use when you add data, or when you edit a table.

Some data sources, such as a CSV file, contain a single table, while other data sources, such as Microsoft Excel spreadsheets or databases, can contain several tables.

If a table contains a header row, field names are usually automatically detected, but you may need to change the **Field names** setting in some cases. You may also need to change other table options, such as **Header size** or **Character set**, to interpret the data correctly. Table options are different for different types of data sources.

Selecting data from a database

The steps for selecting data from a database depend on how you connect to the database. You can connect to an ODBC driver as a DSN source, or you can connect directly through a Qlik Database connector that is part of the Qlik ODBC Connector Package installed with Qlik Sense.

For more information, see *ODBC (page 211)*.

3 Loading and managing data with Data Manager

When you add data from a database, the data source can contain several tables.

Do the following:

1. Select a **Database** from the drop-down list.

Some selection dialogs do not have a **Database** drop-down list because the database name is entered when the connection is configured.

2. Select **Owner** of the database.

The list of **Tables** is populated with views and tables available in the selected database. Some databases do not require that owners be specified in the data selection process.

3. Select a table.

4. Select the fields you want to load by checking the box next to each field you want to load.

You can select all fields in the table by checking the box next to the table name.



You can edit the field name by clicking on the existing field name and typing a new name.

This may affect how the table is linked to other tables, as they are joined on common fields by default.

5. Select additional tables if you want to add data from them.

6. When you are done with your data selection, click **Add data** to continue with data profiling, and to see recommendations for table relationships.

If you want to load the data directly into your app, click *** beside **Add data** and then disable data profiling. This will load the selected data as it is, bypassing the data profiling step, and you can start creating visualizations. Tables will be linked using natural associations, that is, by commonly-named fields.

Selecting data from a Microsoft Excel spreadsheet

When you add data from a Microsoft Excel spreadsheet, the file can contain several sheets. Each sheet is loaded as a separate table. An exception is if the sheet has the same field/column structure as another sheet or loaded table, in which case the tables are concatenated.

Do the following:

1. Make sure you have the appropriate settings for the sheet:

Settings to assist you with interpreting the table data correctly

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names . Typically in an Excel spreadsheet, the first row contains the embedded field names. If you select No field names , fields will be named A,B,C...
Header size	Set to the number of rows to omit as table header, typically rows that contain general information that is not in a columnar format.

Example

My spreadsheet looks like this:

3 Loading and managing data with Data Manager

Spreadsheet example			
Machine:	AEJ12B	-	-
Date:	2015-10-05 09	-	-
Timestamp	Order	Operator	Yield
2015-10-05 09:22	00122344	A	52
2015-10-05 10:31	00153534	A	67
2015-10-05 13:46	00747899	B	86

In this case you probably want to ignore the two first lines, and load a table with the fields Timestamp, Order, Operator, and Yield. To achieve this, use these settings:

Settings to ignore the two first lines and load the fields

UI item	Description
Header size	2 This means that the first two lines are considered header data and ignored when loading the file. In this case, the two lines starting with Machine: and Date: are ignored, as they are not part of the table data.
Field names	Embedded field names. This means that the first line that is read is used as field names for the respective columns. In this case, the first line that will be read is the third line because the two first lines are header data.

2. Select the first sheet to select data from. You can select all fields in a sheet by checking the box next to the sheet name.
3. Select the fields you want to load by checking the box next to each field you want to load.



You can edit the field name by clicking on the existing field name and typing a new name. This may affect how the table is linked to other tables, as they are joined by common fields by default.

4. When you are done with your data selection, click **Add data** to continue with data profiling, and to see recommendations for table relationships.
If you want to load the data directly into your app, click ******* beside **Add data** and then disable data profiling. This will load the selected data as it is, bypassing the data profiling step, and you can start creating visualizations. Tables will be linked using natural associations, that is, by commonly-named fields.

Selecting data from a table file

You can add data from a large number of data files.

Do the following:

3 Loading and managing data with Data Manager

1. Make sure that the appropriate file type is selected in **File format**.
2. Make sure you have the appropriate settings for the file. File settings are different for different file types.
3. Select the fields you want to load by checking the box next to each field you want to load. You can also select all fields in a file by checking the box next to the sheet name.



You can edit the field name by clicking on the existing field name and typing a new name. This may affect how the table is linked to other tables, as they are joined by common fields by default.

When you are done with your data selection, click **Add data** to continue with data profiling, and to see recommendations for table relationships.



*If you want to load the data directly into your app, click *** beside **Add data** and then disable data profiling. This will load the selected data as it is, bypassing the data profiling step, and you can start creating visualizations. Tables will be linked using natural associations, that is, by commonly-named fields.*

4.

Choosing settings for file types

Delimited table files

These settings are validated for delimited table files, containing a single table where each record is separated by a line feed, and each field is separated with a delimited character, for example a CSV file.

File format settings

File format settings for delimited table files

UI item	Description
File format for delimited table files	Set to Delimited or Fixed record . When you make a selection, the select data dialog will adapt to the file format you selected.
Field names	Set to specify if the table contains Embedded field names or No field names .
Delimiter	Set the Delimiter character used in your table file.
Quoting	Set to specify how to handle quotes: None = quote characters are not accepted Standard = standard quoting (quotes can be used as first and last characters of a field value) MSQ = modern-style quoting (allowing multi-line content in fields)
Header size	Set the number of lines to omit as table header.

3 Loading and managing data with Data Manager

UI item	Description
Character set	Set character set used in the table file.
Comment	Data files can contain comments between records, denoted by starting a line with one or more special characters, for example //. Specify one or more characters to denote a comment line. Qlik Sense does not load lines starting with the character(s) specified here.
Ignore EOF	Select Ignore EOF if your data contains end-of-file characters as part of the field value.

Fixed record data files

Fixed record data files contain a single table in which each record (row of data) contains a number of columns with a fixed field size, usually padded with spaces or tab characters.

Setting field break positions

You can set the field break positions in two different ways:

- Manually, enter the field break positions separated by commas in **Field break positions**. Each position marks the start of a field.

Example: 1,12,24

- Enable **Field breaks** to edit field break positions interactively in the field data preview. **Field break positions** is updated with the selected positions. You can:
 - Click in the field data preview to insert a field break.
 - Click on a field break to delete it.
 - Drag a field break to move it.

File format settings

File format settings for fixed record data files

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names .
Header size	Set Header size to the number of lines to omit as table header.
Character set	Set to the character set used in the table file.
Tab size	Set to the number of spaces that one tab character represents in the table file.
Record line size	Set to the number of lines that one record spans in the table file. Default is 1.
Ignore EOF	Select Ignore EOF if your data contains end-of-file characters as part of the field value.

HTML files

HTML files can contain several tables. Qlik Sense interprets all elements with a <TABLE> tag as a table.

3 Loading and managing data with Data Manager

File format settings

File format settings for HTML files

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names .
Character set	Set the character set used in the table file.

XML files

You can load data that is stored in XML format.

There are no specific file format settings for XML files.

QVD files

You can load data that is stored in QVD format. QVD is a native Qlik format and can only be written to and read by Qlik Sense or QlikView. The file format is optimized for speed when reading data from a Qlik Sense script but it is still very compact.

There are no specific file format settings for QVD files.

QVX files

You can load data that is stored in Qlik data eXchange (QVX) format. QVX files are created by custom connectors developed with the Qlik QVX SDK.

There are no specific file format settings for QVX files.

KML files

You can load map files that are stored in KML format, to use in map visualizations.

There are no specific file format settings for KML files.

Returning to the previous step (**Add data**)

You can return to the previous step when adding data.

Do the following:

- Click the back arrow to return to the previous step of **Add data**.

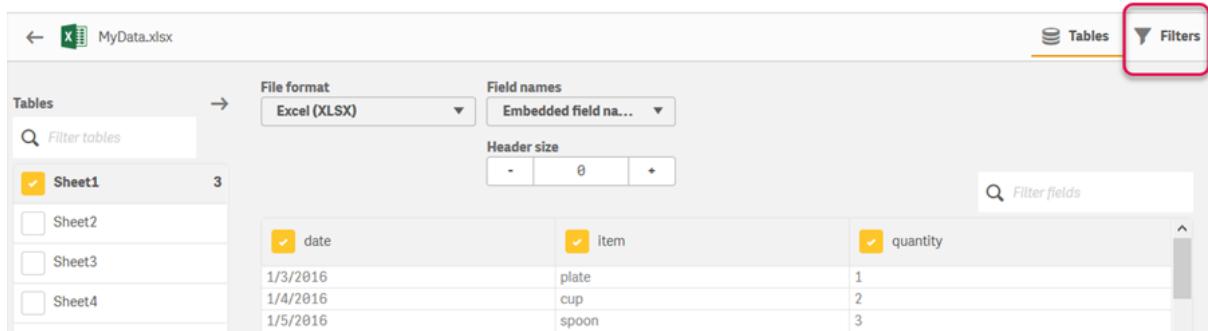
Filtering data from files

You can create filter conditions when you add data from files. This allows you to select a subset of data to load. This can be useful when you want to reduce the amount of data loaded, or only use specific data, such as only sales over \$40,000.

The first time that you add data from a file in the **Add data** step, you can apply filter conditions by clicking **Filters**.

Filters button

3 Loading and managing data with Data Manager



The screenshot shows the Qlik Sense Data Manager interface. On the left, there's a sidebar with 'Tables' and a list of sheets: Sheet1 (selected), Sheet2, Sheet3, and Sheet4. In the center, there's a preview area with three columns: 'date', 'item', and 'quantity'. Under 'date', there are three rows: 1/3/2016, 1/4/2016, and 1/5/2016. Under 'item', there are three rows: plate, cup, and spoon. Under 'quantity', there are three rows: 1, 2, and 3. At the top right, there are buttons for 'Tables' and 'Filters', with 'Filters' being the one highlighted by a red box.

Subsequently, you can change the conditions by clicking your table in the **Data manager**, and then clicking **Edit this table**. Click **Select data from source**, and then click **Filters**.

After you add the data to your app, and apply any filter conditions, you can then load the data into your app.



For more advanced data manipulation, use the **Data load editor**. See the *Scripting for beginners* and *Next steps in scripting* tutorials to learn more.

The available filter conditions are:

- =
- >
- <
- >=
- <=

Consider the following when filtering data. Examples are provided below.

- You can apply filter conditions to numbers, dates, or text.
- Wildcard characters are not supported.
- You can apply multiple conditions. However, conflicting conditions on the same field may result in no data being returned.
- Conditions are applied alphabetically to text data. Conditions are case sensitive.
- You can use more than one letter for text data. For example, `>ct` will return the word *cup*, as will `>=cu`. Note that `>c` will also return *cup*.
- When you use more than one `=` condition, all must evaluate to true to return values. However, when you use more than one `=` condition on the same field, all values that evaluate to true are returned.
- The `<` and `>` conditions, when combined, must all evaluate to true to return values. If these conditions are combined with `=`, all conditions must evaluate to true.
- The `<=` and `>=` conditions, when combined, must all evaluate to true to return values. If these conditions are combined with `=`, all conditions must evaluate to true.
- Filters on date fields from QVD files are not supported.

Examples

These examples use the following values from a single field (one column in a table): *cup*, *fork*, and *knife*.

3 Loading and managing data with Data Manager

- Conditions:
 - $=cup$
 - $=fork$
 - $=knife$
 - Returns: *cup, fork, knife*
 - The equals condition returns all values that are true.
- Conditions:
 - $>b$
 - $<d$
 - Returns: *cup*
 - The letter *c* is both greater than *b* and lesser than *d*.
- Conditions:
 - $<b$
 - $>d$
 - Returns: no values
 - There can be no values that are both lesser than *b* and greater than *d*.
- Conditions:
 - $=fork$
 - $>g$
 - Returns: no values
 - There can be no values that are both equal to *fork* and greater than *g*.

Filtering data from data connectors



Filter data is not available for all connectors.

You enter a data filter expression by selecting **Filter data** in the **Select data to load** step. Selecting **Filter data** opens a text box where you enter a filter expression. For example:

Sales >= 40000

Filter data selects from individual fields, such as *Sales*. It operates as an SQL WHERE clause. Most operators and keywords used in WHERE clauses can be used with **Filter data**. Valid operators include:

- =
- >
- >=
- <
- <=
- IN
- BETWEEN

- LIKE
- IS NULL
- IS NOT NULL

Qlik Sense builds a WHERE clause in the data load script from the expression entered in **Filter data**.

The AND operator can be used to combine operators, such as when you want to filter across more than one field. For example:

Sales <= 30000 AND RegionID = 45

The OR operator can be used to filter data that matches either condition. For example:

Name = 'Smith' OR Name = 'Jones'

You can get the same results from the IN operator. The IN operator is a shorthand method for using multiple OR conditions. For example:

Name IN ('Smith', 'Jones')

3.12 Editing a table

You can edit a table that was added to the app in the **Data manager** overview, to rename the table, associate the table to other tables, or make field transformations.

To edit a table, select the table in **Data manager** and click . The table editor is displayed, with a preview of the data in the table. Each field has a field menu with transformation options. You open the field menu by clicking . Selecting a field displays the data profiling card pane, which contains a summary of the field's data as well as additional transformation options.



If the data contains records with identical data in all fields that are loaded, they are represented by a single record in the preview table.

Renaming a table

When you add a table in **Data manager**, the table is assigned a default name, based on the name of the database table, data file, or Excel worksheet, for example. If the name is non-descriptive or unsuitable, you can rename it.

Do the following:

1. Click on the table name.
2. Edit the table name.
3. Press Enter or click outside the table name.

The table is now renamed.

Renaming a field

You can rename fields in a table to get a better name that is easier to understand.

Do the following:

1. Click on the field name that you want to rename, or select **Rename** from the field menu.
2. Type the new name.



Field names must be unique. If you have fields with the same name in several tables, Qlik Sense will qualify the field names when you add data, that is, add the table name as prefix.

3. Press the Enter key, or click outside the field.

The field is now renamed.

Managing associations to other tables

You can create custom associations to fields in other tables with **Associate** in the field menu.

Typically, these are the most common cases where you need to create a custom association instead of following the recommendations:

- You know which fields to associate the tables with, but the score for this table pair is too low to show in the list of recommendations.
Create an association based on a single field in each table.
- The tables contain more than one common field, and they need to be used to form the association.
Create a compound key.

In many cases it is easier to manage to your associations in the **Associations** view

Changing field type and display format

When data is added, Qlik Sense interprets the field type of each field. The following field types are currently supported:

- General
- Date
- Timestamp
- Geo data

If the data was not interpreted correctly, you can adjust the field type. You can also change the input and display format of a date or timestamp field.

Fields that contain geographical information in the form of names or codes, such as postal areas, cannot be used for mapping unless they are designated as **Geo data** fields.

Hiding fields from analysis

You can hide fields so that they are only available in **Data manager** and **Data load editor**. You might, for example, have fields that are only used for calculating another field. You can hide these fields so that they are not available in the sheets or Insight Advisor assets panel but remain available in **Data manager**.

For more information, see *Hiding fields from analysis (page 55)*.

Assessing table field data before loading data

You can examine the data in your table for potential quality issues such as null values and outlier values before you load it using the **Summary** data profiling card. The **Summary** card categorizes fields as dimensions, measures, or temporal fields, providing different data summaries for each and enabling different transformation options in other data profiling cards. Fields set as measures in the **Summary** card can be grouped using the **Bucket** card. Fields set as dimensions in the **Summary** card can have a custom order applied in the **Order** card. For fields that can be classified in multiple categories, you can switch between each possible category's summary for the field.

For more information, see *Assessing table field data before loading data (page 56)*.

Replacing field values in a table

You can replace values in a field using the **Replace** data profiling card. The **Replace** card enables you to select one or more values from a field and then replace them with another value. For example, in a data set that contains country names in both full and abbreviated formats, you could replace them with a single shared value.

For more information, see *Replacing field values in a table (page 58)*.

Setting field values as null in a table

You can set distinct values from a dimension field to be treated as null values using the **Set nulls** data profiling card. The **Set nulls** card enables you to select values from a table field and then manually set them as null. For example, if your data set represents nulls using a character such as X, you can use the **Set nulls** card to enable Qlik Sense to treat that value as null. The **Set nulls** card can also be used to clean your data set by setting unwanted values as nulls.

For more information, see *Setting field values as null in a table (page 60)*.

Setting a custom order for field values

Depending on your data, it may be more meaningful to display dimension values in an order other than alphabetical or numerical. Fields set as dimensions in the **Summary** data profiling card may have a custom order of data applied using the **Order** data profiling card, enabling you to set the default organization of field data in visualizations.

For more information, see *Customizing the order of dimension values (page 61)*.

Splitting a field in a table

You can extract information from an existing field into new fields using the **Split** data profiling card. The **Split** card enables you to split content from a field into multiple fields. You could, for example, split a field that contains an address to extract the zip or postal code. This enables you to quickly create new fields containing relevant sections of existing data.

For more information, see *Splitting a field in a table* (page 62).

Grouping measure data into ranges

You can group values in a table measure field into ranges using the **Bucket** data profiling card. The **Bucket** card enables you to group a field's value in user-defined buckets, creating a new field that is added to the table. You could, for example, group ages into age ranges to use as a dimensions in your visualizations.

Viewing field transformation details

You can view the current operations and transformations performed on a field and their sequence in the **Details** dialog. **Details** enables you to understand where a field came from, what changes have been made to it, and the sequence in which the transformations were applied.

For more information, see *Viewing table and field transformation details in Data manager* (page 79).

Unpivoting crosstab data

If you have loaded data in crosstab format, the best option is usually to unpivot the table, that is, transposing parts of the table into rows. This will make it easier to work with the data and create associations to your other data tables.

Year	Region	Sales
2007	Europe	234
2007	RoW	567
2008	Europe	345
2008	RoW	534

For more information, see *Unpivoting crosstab data in the data manager* (page 67).

Updating a table from the data source

You may want to change the selection of fields from the data source. For example, you may need to add a field that was left out, or the data source may have been updated with added fields. In this case, you can update the table from the data source. If the table was created with **Manual entry**, you can add, edit, or delete table data as well as add new rows and columns. For more information, see *Adding data manually* (page 25).

Do the following:

1. Click **Select data from source**.

The data selection wizard opens with your current selections.

2. Make the required changes in selection.

3. Click **Add data** with data profiling enabled.

The table is now updated with fields according to the selections you made.

Adding a calculated field

There are many cases where you need to adjust or transform the field data that is loaded. For example, you may need to concatenate a first name and a last name to a full name, extract part of a product number, convert the data format or multiply two numbers.

You can add calculated fields to manage many cases like this. A calculated field uses an expression to define the result of the field. You can use functions, fields and operators in the expression. You can only refer to fields in the table that you are editing.

Sorting a table

You can sort a table based on a specific field while you are editing the table, to get a better overview of the data. You can only sort on one field at a time.

Do the following:

- From the field menu, select **Sort**.

The table data is now sorted in ascending order according to this field. If you want to sort in descending order, select **Sort** again.



The sort order is not maintained in the loaded app data.

Undo and redo actions

You can undo and redo your table edit actions by clicking ↗ and ↘ .

The undo/redo history is cleared when you close the table editor.

Associating data in the table editor

You can create custom associations to fields in other tables with **Associate** in the field menu of the **Data manager** table editor.

In many cases it is easier to manage to your associations in the **Associations** view.

Typically, these are the most common cases where you need to create a custom association instead of following the recommendations:

- You know which fields to associate the tables with, but the score for this table pair is too low to show in the list of recommendations.

3 Loading and managing data with Data Manager

Create an association based on a single field in each table.

- The tables contain more than one common field, and they need to be used to form the association.
Create a compound key.

Creating an association using a single field

If the two tables contain related data, but the association does not show up as recommended, you can define a custom association in the table editor. This creates a key field to associate the tables.

Do the following:

1. From the data manager overview, click  on one of the tables you want to associate.
The table editor opens.
2. Select **Associate** from the field menu of the field you want to use in the key field.
The **Associate tables** editor opens, with a preview of the field you selected in the left table. Now you need to select which field to associate this with in the right hand table.
3. Click **Select table** and select the table to associate with.
4. Click  and select the field to associate with.
The right hand table will show preview data of the field you selected. Now you can compare the left table with the right to check that they contain matching data. You can search in the tables with  to compare them more easily.
5. Enter a name for the key field that will be created in **Name**.
It's not possible to use the same name as an existing field in either of the tables.
6. Click **Associate**.

The tables are now associated by the two fields you selected, using a key field. This is indicated with . Click  to display options to edit or break the association.

Creating a compound key

If two tables contain more than one common field that would create an association, Qlik Sense creates a synthetic key to handle the association. The recommended way of fixing this is to create a compound key. This can be achieved by creating a custom association containing all fields that should be associated.

Do the following:

1. From the data manager overview, click  on one of the tables you want to associate.
The table editor opens.
2. Select **Associate** from the field menu of one of the fields you want to include in the compound key field.
The **Associate tables** editor opens, with a preview of the field you selected in the left table.
3. Click  to add the other fields you want to include in the compound key field.
The preview is updated with the compound key data.
Now you need to select which fields to associate this with in the right hand table.
4. Click **Select table** and select the fields you want to include in the compound key field.

3 Loading and managing data with Data Manager

5. Click  and select the field to associate with. You need to select them in the same order as in the left hand table.

To make it easier to interpret the data in the key you can also add delimiter characters.

The right hand table will show preview data of the field you selected.

Now you can compare the left table with the right to check that they contain matching data. You can search in the tables with  to compare them more easily.

6. Enter a name for the key field that will be created in **Name**.
7. Click **Associate**.

The tables are now associated by the fields you selected, using a compound key field.

Limitations

There are some limitations to the use of compound keys.

- It is not possible to create a compound key in a concatenated table.
- If you use a calculated field in a compound key, the calculated field expression is expanded in the compound key expression. There is no reference to the calculated field, that is, if you edit the calculated field, the compound key is not updated.

Editing an association

You can edit an association to rename it, or change the associated fields.

Do the following:

1. Click  to show the association menu.
2. Click  to edit the association.

The **Associate tables** editor opens, and you can rename the association or change the associated fields.

Breaking an association

If you have created an association between two tables that is not needed, you can break it.

Do the following:

1. Click  to show the association menu.
2. Click  to break the association.

The tables are now not associated anymore.

Using calculated fields

There are many cases where you need to adjust or transform the field data that is loaded. For example, you may need to concatenate a first name and a last name to a full name, extract part of a product number, convert the data format or multiply two numbers.

3 Loading and managing data with Data Manager

You can add calculated fields to manage many cases like this. A calculated field uses an expression to define the result of the field. You can use functions, fields and operators in the expression. You can only refer to fields in the table that you are editing. You can reference another calculated field in your calculated field.

You add and edit calculated fields in the table editor of the data manager.

Adding a calculated field

Do the following:

1. Click **Add field** and select **Calculated field**.
The editor for **Add calculated field** opens.
 2. Type the name of the calculated field in **Name**.
 3. Define the expression of the calculated field in **Expression**. There are two different ways to do this.
 - Use the **fx (Functions)**, **(Fields)** and **(Operators)** lists to select and insert items into the expression.
The item you select is inserted at the cursor position in **Expression**.
 - Type the expression for the calculated field in **Expression**.
As you type, you get assistance with suggested functions and fields, as well as function syntax.
- Example results of the calculated field is displayed in **Preview**.
4. Click **Create** to create the calculated field and close the calculated field editor.

Editing a calculated field

You can change the name or edit the expression of a calculated field.

Do the following:

1. Select **Edit** from the drop-down menu next to the field name.
The editor for **Update calculated field** opens.
2. Edit the name of the calculated field in **Name** if you want to change it.
3. Edit the expression of the calculated field.
4. Click **Update** to update the calculated field and close the calculated field editor.

Which functions can I use in a calculated field expression?

You can use the functions listed here when you create a calculated field expression. This is a subset of the expressions available in the data load script. The expression cannot result in any aggregation of data from several records, or use inter-record functions to refer to data in other records.

String functions that can be used in a calculated field expression

These functions can be used to modify or extract data in text string format.

String functions

Function	Description
Capitalize	Capitalize() returns the string with all words in initial uppercase letters.

3 Loading and managing data with Data Manager

Function	Description
Chr	Chr() returns the Unicode character corresponding to the input integer.
FindOneOf	FindOneOf() searches a string to find the position of the occurrence of any character from a set of provided characters. The position of the first occurrence of any character from the search set is returned unless a third argument (with a value greater than 1) is supplied. If no match is found, 0 is returned.
Index	Index() searches a string to find the starting position of the nth occurrence of a provided substring. An optional third argument provides the value of n, which is 1 if omitted. A negative value searches from the end of the string. The positions in the string are numbered from 1 and up.
KeepChar	KeepChar() returns a string consisting of the first string , 'text', less any of the characters NOT contained in the second string, "keep_chars".
Left	Left() returns a string consisting of the first (leftmost) characters of the input string, where the number of characters is determined by the second argument.
Len	Len() returns the length of the input string.
Lower	Lower() converts all the characters in the input string to lower case.
LTrim	LTrim() returns the input string trimmed of any leading spaces.
Mid	Mid() returns the part of the input string starting at the position of the character defined by the second argument, 'start', and returning the number of characters defined by the third argument, 'count'. If 'count' is omitted, the rest of the input string is returned. The first character in the input string is numbered 1.
Ord	Ord() returns the Unicode code point number of the first character of the input string.
PurgeChar	PurgeChar() returns a string consisting of the characters contained in the input string ('text'), excluding any that appear in the second argument ('remove_chars').
Repeat	Repeat() forms a string consisting of the input string repeated the number of times defined by the second argument.
Replace	Replace() returns a string after replacing all occurrences of a given substring within the input string with another substring. The function is non-recursive and works from left to right.
Right	Right() returns a string consisting of the last (rightmost) characters of the input string, where the number of characters is determined by the second argument.
RTrim	RTrim() returns the input string trimmed of any trailing spaces.
SubStringCount	SubStringCount() returns the number of occurrences of the specified substring in the input string text. If there is no match, 0 is returned.
TextBetween	TextBetween() returns the text in the input string that occurs between the characters specified as delimiters.

3 Loading and managing data with Data Manager

Function	Description
Trim	Trim() returns the input string trimmed of any leading and trailing spaces.
Upper	Upper() converts all the characters in the input string to upper case for all text characters in the expression. Numbers and symbols are ignored.

Date functions that can be used in a calculated field expression

Qlik Sense date and time functions are used to transform and convert date and time values.

Functions are based on a date-time serial number that equals the number of days since December 30, 1899. The integer value represents the day and the fractional value represents the time of the day.

Qlik Sense uses the numerical value of the argument, so a number is valid as a argument also when it is not formatted as a date or a time. If the argument does not correspond to a numerical value, for example, because it is a string, then Qlik Sense attempts to interpret the string according to the date and time environment variables.

If the date format used in the argument does not correspond to the one set in the **DateFormat** system variable, Qlik Sense will not be able to interpret the date correctly. To resolve this, either change the settings or use an interpretation function.

Date functions

Function	Description
addmonths	This function returns the date occurring n months after startdate or, if n is negative, the date occurring n months before startdate .
addyears	This function returns the date occurring n years after startdate or, if n is negative, the date occurring n years before startdate .
age	The age function returns the age at the time of timestamp (in completed years) of somebody born on date_of_birth .
converttolocaltime	Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.
day	This function returns an integer representing the day when the fraction of the expression is interpreted as a date according to the standard number interpretation.
dayend	This function returns a value corresponding to a timestamp of the final millisecond of the day contained in time . The default output format will be the TimestampFormat set in the script.
daylightsaving	Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.
dayname	This function returns a value showing the date with an underlying numeric value corresponding to a timestamp of the first millisecond of the day containing time .

3 Loading and managing data with Data Manager

Function	Description
daynumberofquarter	Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.
daynumberofyear	This function calculates the day number of the year in which a timestamp falls. The calculation is made from the first millisecond of the first day of the year, but the first month can be offset.
daystart	This function returns a value corresponding to a timestamp with the first millisecond of the day contained in the time argument. The default output format will be the TimestampFormat set in the script.
firstworkdate	The firstworkdate function returns the latest starting date to achieve no_of_workdays (Monday-Friday) ending no later than end_date taking into account any optionally listed holidays. end_date and holiday should be valid dates or timestamps.
GMT	This function returns the current Greenwich Mean Time, as derived from the regional settings.
hour	This function returns an integer representing the hour when the fraction of the expression is interpreted as a time according to the standard number interpretation.
inday	This function returns True if timestamp lies inside the day containing base_timestamp .
indaytotime	This function returns True if timestamp lies inside the part of day containing base_timestamp up until and including the exact millisecond of base_timestamp .
inlunarweek	This function determines if timestamp lies inside the lunar week containing base_date . Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week., Apart from the final week of the year, each week will contain exactly seven days.
inlunarweektodate	This function finds if timestamp lies inside the part of the lunar week up to and including the last millisecond of base_date . Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.
inmonth	This function returns True if timestamp lies inside the month containing base_date .
inmonths	This function finds if a timestamp falls within the same month, bi-month, quarter, four-month period, or half-year as a base date. It is also possible to find if the timestamp falls within a previous or following time period.

3 Loading and managing data with Data Manager

Function	Description
inmonthstodate	This function finds if a timestamp falls within the part a period of the month, bi-month, quarter, four-month period, or half-year up to and including the last millisecond of base_date . It is also possible to find if the timestamp falls within a previous or following time period.
inmonthtodate	Returns True if date lies inside the part of month containing basedate up until and including the last millisecond of basedate .
inquarter	This function returns True if timestamp lies inside the quarter containing base_date .
inquartertodate	This function returns True if timestamp lies inside the part of the quarter containing base_date up until and including the last millisecond of base_date .
inweek	This function returns True if timestamp lies inside the week containing base_date .
inweektodate	This function returns True if timestamp lies inside the part of week containing base_date up until and including the last millisecond of base_date .
inyear	This function returns True if timestamp lies inside the year containing base_date .
inyeartodate	This function returns True if timestamp lies inside the part of year containing base_date up until and including the last millisecond of base_date .
lastworkdate	The lastworkdate function returns the earliest ending date to achieve no_of_workdays (Monday-Friday) if starting at start_date taking into account any optionally listed holiday . start_date and holiday should be valid dates or timestamps.
localtime	This function returns a timestamp of the current time for a specified time zone.
lunarweekend	This function returns a value corresponding to a timestamp of the last millisecond of the last day of the lunar week containing date . Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.
lunarweekname	This function returns a display value showing the year and lunar week number corresponding to a timestamp of the first millisecond of the first day of the lunar week containing date . Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.
lunarweekstart	This function returns a value corresponding to a timestamp of the first millisecond of the first day of the lunar week containing date . Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.
makedate	This function returns a date calculated from the year YYYY , the month MM and the day DD .

3 Loading and managing data with Data Manager

Function	Description
maketime	This function returns a time calculated from the hour hh , the minute mm , and the second ss .
makeweekdate	This function returns a date calculated from the year, the week number, and the day of week . This function returns a date calculated from the year YYYY , the week WW and the day of week D .
minute	This function returns an integer representing the minute when the fraction of the expression is interpreted as a time according to the standard number interpretation.
month	This function returns a dual value: a month name as defined in the environment variable MonthNames and an integer between 1-12. The month is calculated from the date interpretation of the expression, according to the standard number interpretation.
monthend	This function returns a value corresponding to a timestamp of the last millisecond of the last day of the month containing date. The default output format will be the DateFormat set in the script.
monthname	This function returns a display value showing the month (formatted according to the MonthNames script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the month.
monthsend	This function returns a value corresponding to a timestamp of the last millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for a previous or following time period.
monthsname	This function returns a display value representing the range of the months of the period (formatted according to the MonthNames script variable) as well as the year. The underlying numeric value corresponds to a timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date.
monthsstart	This function returns a value corresponding to the timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for a previous or following time period.The default output format is the DateFormat set in the script.
monthstart	This function returns a value corresponding to a timestamp of the first millisecond of the first day of the month containing date . The default output format will be the DateFormat set in the script.
networkdays	The networkdays function returns the number of working days (Monday-Friday) between and including start_date and end_date taking into account any optionally listed holiday .

3 Loading and managing data with Data Manager

Function	Description
now	This function returns a timestamp of the current time. The function returns values in the TimeStamp system variable format. The default timer_mode value is 1.
quarterend	This function returns a value corresponding to a timestamp of the last millisecond of the quarter containing date . The default output format will be the DateFormat set in the script.
quartername	This function returns a display value showing the months of the quarter (formatted according to the MonthNames script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the quarter.
quarterstart	This function returns a value corresponding to a timestamp of the first millisecond of the quarter containing date . The default output format will be the DateFormat set in the script.
second	This function returns an integer representing the second when the fraction of the expression is interpreted as a time according to the standard number interpretation.
timezone	This function returns the time zone, as defined on the computer where the Qlik engine is running.
today	This function returns the current date. The function returns values in the DateFormat system variable format.
UTC	Returns the current Coordinated Universal Time.
week	This function returns an integer representing the week number according to ISO 8601. The week number is calculated from the date interpretation of the expression, according to the standard number interpretation.
weekday	This function returns a dual value with: A day name as defined in the environment variable DayNames . An integer between 0-6 corresponding to the nominal day of the week (0-6).
weekend	This function returns a value corresponding to a timestamp of the last millisecond of the last day of the calendar week containing date . The default output format will be the DateFormat set in the script. This function returns a value corresponding to a timestamp of the last millisecond of the last day (Sunday) of the calendar week containing date . The default output format will be the DateFormat set in the script.
weekname	This function returns a value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the week containing date .

3 Loading and managing data with Data Manager

Function	Description
weekstart	This function returns a value corresponding to a timestamp of the first millisecond of the first day of the calendar week containing date . The default output format is the DateFormat set in the script.
weekyear	This function returns the year to which the week number belongs according to the environment variables. The week number ranges between 1 and approximately 52. This function returns the year to which the week number belongs according to ISO 8601. The week number ranges between 1 and approximately 52.
year	This function returns an integer representing the year when the expression is interpreted as a date according to the standard number interpretation.
yearend	This function returns a value corresponding to a timestamp of the last millisecond of the last day of the year containing date . The default output format will be the DateFormat set in the script.
yearname	This function returns a four-digit year as display value with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the year containing date .
yearstart	This function returns a timestamp corresponding to the start of the first day of the year containing date . The default output format will be the DateFormat set in the script.
yeartodate	This function finds if the input timestamp falls within the year of the date the script was last loaded, and returns True if it does, False if it does not.

Formatting and interpretation functions that can be used in a calculated field expression

The formatting functions use the numeric value of the input expression, and convert this to a text value. In contrast, the interpretation functions do the opposite: they take string expressions and evaluate them as numbers, specifying the format of the resulting number. In both cases the output value is dual, with a text value and a numeric value.

For example, consider the differences in output between the **Date** and the **Date#** functions.

Date and the Date# functions

Function	Output (text)	Output (numeric)
Date#('20140831', 'YYYYMMDD')	20140831	41882
Date(41882, 'YYYY.MM.DD')	2014.08.31	41882

These functions are useful when your data contains date fields that are not interpreted as dates as the format does not correspond to the date format setting in Qlik Sense. In this case, it can be useful to nest the functions:

```
Date(Date#(DateInput, 'YYYYMMDD'), 'YYYY.MM.DD')
```

3 Loading and managing data with Data Manager

This will interpret the DateInput field according to the input format, YYYYMMDD, and return it in the format you want to use, YYYY.MM.DD.

Formatting and interpretation functions

Function	Description
Date	Date() formats an expression as a date using the format set in the system variables in the data load script, or the operating system, or a format string, if supplied.
Date#	Date# evaluates an expression as a date in the format specified in the second argument, if supplied.
Dual	Dual() combines a number and a string into a single record, such that the number representation of the record can be used for sorting and calculation purposes, while the string value can be used for display purposes.
Interval	Interval() formats a number as a time interval using the format in the system variables in the data load script, or the operating system, or a format string, if supplied.
Interval#	Interval#() evaluates a text expression as a time interval in the format set in the operating system, by default, or in the format specified in the second argument, if supplied.
Money	Money() formats an expression numerically as a money value, in the format set in the system variables set in the data load script, or in the operating system, unless a format string is supplied, and optional decimal and thousands separators.
Money#	Money#() converts a text string to a money value, in the format set in the load script or the operating system, unless a format string is supplied. Custom decimal and thousand separator symbols are optional parameters.
Num	Num() formats a number, that is it converts the numeric value of the input to display text using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.
Num#	Num#() interprets a text string as a numerical value, that is it converts the input string to a number using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.
Text	Text() forces the expression to be treated as text, even if a numeric interpretation is possible.
Time	Time() formats an expression as a time value, in the time format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.
Time#	Time#() evaluates an expression as a time value, in the time format set in the data load script or the operating system, unless a format string is supplied.

3 Loading and managing data with Data Manager

Function	Description
Timestamp	TimeStamp() formats an expression as a date and time value, in the timestamp format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.
Timestamp#	Timestamp#() evaluates an expression as a date and time value, in the timestamp format set in the data load script or the operating system, unless a format string is supplied.

Numeric functions that can be used in a calculated field expression

You can use these functions to round numeric values.

Numeric functions

Function	Description
ceil	Ceil() rounds up a number to the nearest multiple of the step shifted by the offset number.
div	Div() returns the integer part of the arithmetic division of the first argument by the second argument. Both parameters are interpreted as real numbers, that is, they do not have to be integers.
evens	Even() returns True (-1), if integer_number is an even integer or zero. It returns False (0), if integer_number is an odd integer, and NULL if integer_number is not an integer.
fabs	Fabs() returns the absolute value of x . The result is a positive number.
fact	Fact() returns the factorial of a positive integer x .
floor	Floor() rounds down a number to the nearest multiple of the step shifted by the offset number.
fmod	fmod() is a generalized modulo function that returns the remainder part of the integer division of the first argument (the dividend) by the second argument (the divisor). The result is a real number. Both arguments are interpreted as real numbers, that is, they do not have to be integers.
frac	Frac() returns the fraction part of x .
mod	Mod() is a mathematical modulo function that returns the non-negative remainder of an integer division. The first argument is the dividend, the second argument is the divisor, Both arguments must be integer values.
odd	Odd() returns True (-1), if integer_number is an odd integer or zero. It returns False (0), if integer_number is an even integer, and NULL if integer_number is not an integer.
round	Round() returns the result of rounding a number up or down to the nearest multiple of step shifted by the offset number.
sign	Sign() returns 1, 0 or -1 depending on whether x is a positive number, 0, or a negative number.

3 Loading and managing data with Data Manager

Conditional functions that can be used in a calculated field expression

You can use these functions to evaluate a condition and then return different answers depending on the condition value.

Conditional functions

Function	Description
alt	The alt function returns the first of the parameters that has a valid number representation. If no such match is found, the last parameter will be returned. Any number of parameters can be used.
class	The class function assigns the first parameter to a class interval. The result is a dual value with $a \leq x < b$ as the textual value, where a and b are the upper and lower limits of the bin, and the lower bound as numeric value.
if	The if function returns a value depending on whether the condition provided with the function evaluates as True or False.
match	The match function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case sensitive.
mixmatch	The mixmatch function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case insensitive.
pick	The pick function returns the n :th expression in the list.
wildmatch	The wildmatch function compares the first parameter with all the following ones and returns the number of the expression that matches. It permits the use of wildcard characters (* and ?) in the comparison strings. * matches any sequence of characters. ? matches any single character. The comparison is case insensitive.

NULL functions that can be used in a calculated field expression

You can use these functions to return or detect null values.

NULL functions

Function	Description
Null	The Null function returns a NULL value.
IsNull	The IsNull function tests if the value of an expression is NULL and if so, returns -1 (True), otherwise 0 (False).

Mathematical functions that can be used in a calculated field expression

You can use these functions for mathematical calculations.

3 Loading and managing data with Data Manager

Mathematical functions

Function	Description
e	The function returns the base of the natural logarithms, e (2.71828...).
rand	The function returns a random number between 0 and 1. This can be used to create sample data.

Exponential and Logarithmic functions that can be used in a calculated field expression

You can use these functions for exponential and logarithmic calculations.

Exponential and Logarithmic functions

Function	Description
exp	The natural exponential function, e^x , using the natural logarithm e as base. The result is a positive number.
log	The natural logarithm of x . The function is only defined if $x > 0$. The result is a number.
log10	The common logarithm (base 10) of x . The function is only defined if $x > 0$. The result is a number.
pow	Returns x to the power of y . The result is a number.
sqr	x squared (x to the power of 2). The result is a number.
sqrt	Square root of x . The function is only defined if $x \geq 0$. The result is a positive number.

Distribution functions that can be used in a calculated field expression

You can use these functions for statistical distribution calculations.

Distribution functions

Function	Description
CHIDIST	<code>chidist()</code> returns the one-tailed probability of the chi ² distribution. The chi ² distribution is associated with a chi ² test.
CHIINV	<code>ChiInv()</code> returns the inverse of the one-tailed probability of the chi ² distribution.
FDIST	<code>FDist()</code> returns the accumulated probability of the F distribution.
FINV	<code>FInv()</code> returns the inverse of the accumulated probability of the F distribution.
NORMDIST	<code>NormDist()</code> returns the cumulative normal distribution for the specified mean and standard deviation. If mean = 0 and standard_dev = 1, the function returns the standard normal distribution.
NORMINV	<code>NormInv()</code> returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.
TDIST	<code>TDist()</code> returns the probability for the student's t distribution where a numeric value is a calculated value of t for which the probability is to be computed.
TINV	<code>TInv()</code> returns the t value of the student's t distribution as a function of the probability and the degrees of freedom.

3 Loading and managing data with Data Manager

Geospatial functions that can be used in a calculated field expression

You can use this function to handle geospatial data.

Geospatial functions

Function	Description
GeoMakePoint	GeoMakePoint() is used in scripts and chart expressions to create and tag a point with latitude and longitude.

Color functions that can be used in a calculated field expression

You can use these functions for setting and evaluating color properties.

Color functions

Function	Description
ARGB	ARGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component r , a green component g , and a blue component b , with an alpha factor (opacity) of alpha .
HSL	HSL() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by values of hue , saturation , and luminosity between 0 and 1.
RGB	RGB() returns an integer corresponding to the color code of the color defined by the three parameters: the red component r , the green component g , and the blue component b . These components must have integer values between 0 and 255. The function can be used in expressions to set or evaluate the color properties of a chart object.

Logical functions that can be used in a calculated field expression

You can use these functions for handling logical operations.

Logical functions

Function	Description
IsNum	Returns -1 (True) if the expression can be interpreted as a number, otherwise 0 (False).
IsText	Returns -1 (True) if the expression has a text representation, otherwise 0 (False).

System functions that can be used in a calculated field expression

You can use these functions for accessing system, device, and Qlik Sense app properties.

System functions

Function	Description
OSUser	This function returns a string containing the name of the user that is currently connected. It can be used in both the data load script and in a chart expression.
ReloadTime	This function returns a timestamp for when the last data load finished. It can be used in both the data load script and in a chart expression.

Changing field types

When data is added, Qlik Sense interprets the field type of each field. The following field types are currently supported:

-  **General**
-  **Date**
-  **Timestamp**
-  **Geo data**

If the data was not interpreted correctly, you can adjust the field type. You can also change the input and display format of a data or timestamp field.

To open the table editor, click  on the data table you want to edit.

It is not possible to change field type or display format of fields in some cases.

- The table is the result of concatenating two or more tables.
- The field is already recognized as a date or a timestamp.

Making sure a date or timestamp field is recognized correctly

If a date or timestamp field is not recognized as a date or a timestamp, that is, it is marked with  **General**, you can make sure it is interpreted correctly.

Do the following:

1. Click on  above the field heading.
The data format dialog opens.
2. Change **Field type** to **Date** or **Timestamp**.
3. Change the format string in **Input format** to interpret the date correctly. You can use a prepared format from the drop down list, or write your own format string.



It is not possible to use a single quote in the format string.

4. If you want to use a display format other than the default format in your app, write or select a format string in **Display format**.
If you leave it empty, the app default display format is used.

Changing the display format of a date or timestamp field

Each app has default display formats for date and timestamp fields. You can change the display format for an individual date or timestamp field.

Do the following:

3 Loading and managing data with Data Manager

1. Click on or above the field heading.
The data format dialog opens.
2. Change the format string in **Display format**. Either use a prepared format from the drop down list, or write your own format string..

Changing a field type to geographical data

If a field with values such as city and country names or ISO symbols is not recognized as geographical data, you can change the field type to **Geo data**.

Do the following:

1. Click on above the field heading.
The data format dialog opens.
2. Select **Geo data** from the **Field type** drop-down menu.
3. Select the type of geographical data from the **Geo data** drop-down menu.
The options are **City**, **Country**, **Country code ISO2**, and **Country code ISO3**. ISO2 and ISO3 are from ISO 3166, the International Standards Organization's codes for countries. ISO2 contains two-character codes, and ISO3 contains three-character codes. For example, the codes for Sweden are SE and SWE. When assigning an ISO code, be sure that the values in the field match the code you assign. If you assign ISO3 to a field with two-character code values, location coordinates will not be assigned correctly.
4. For **City** data, select the related field in the table that contains geographical data for countries.
There may be only one related country field, but it is possible to have multiple fields with geographical data for countries. For example, one field may have long names such as France and another field that designates countries by country codes such as ISO2. But the fields appear in the **Associated country** list only if they have been designated as **Geo data**.
If no field has been designated as a **Geo data** country field, then the **Associated country** list does not appear when you designate a field as **City**.

When a field is assigned the **Geo data** field type, either by the user or automatically by Qlik Sense, a field containing geographical coordinates, either point or polygon data, is associated with it. The associated fields containing the coordinates are visible in the **Data model viewer**. These coordinates are required for apps that use **Map** objects.

Fields that contain geographical information in the form of names or codes, such as postal areas, cannot be used for mapping unless they are designated as **Geo data** fields.

The fields assigned the **Geo data** type continue to hold string values, such as Mexico and MX, but when they are used in a **Map** object, the mapping coordinates come from the fields containing the point or polygon data.

Hiding fields from analysis

You can hide data fields that you do not want available when building visualizations in sheet view or in Insight Advisor.

3 Loading and managing data with Data Manager

You might, for example, have fields that are only used for calculating another field. You can hide these fields in **Data manager** so that they are not available in the assets panel of sheets or from Insight Advisor but remain available in **Data manager** and **Data load editor**. This can be used to remove information that may be redundant or unnecessary to your current analysis or insights, including only relevant information and making it easier to read and analyze.

When hiding a field, all existing relationships the field has, such as associations or use in calculations, will be maintained. If a field is currently in use, such as in a master item or in an existing chart, it will continue to be available there, but will not be available for use in new master items or visualizations until it is shown again.



*You can view all your hidden fields in **Data manager** by going into **Data load editor** and opening the auto-generated section. All hidden fields will be listed as TAG FIELD <field name> WITH '\$hidden';*

Hiding a field from analysis

Do the following:

1. Click on above the field heading.
2. Click **Hide in Analysis**.

The field is now hidden in sheet view and in Insight Advisor. Hidden fields have added above the field heading.

Showing a hidden field

Do the following:

1. Click on or above the field heading.
2. Click **Show in Analysis**.

The field is now available in sheet view and in Insight Advisor. The above the field heading will be removed.

Assessing table field data before loading data

To examine your data for potential quality issues such as null or outlier before you load it into Qlik Sense, you view a summary of the data using the **Summary** data profiling card.

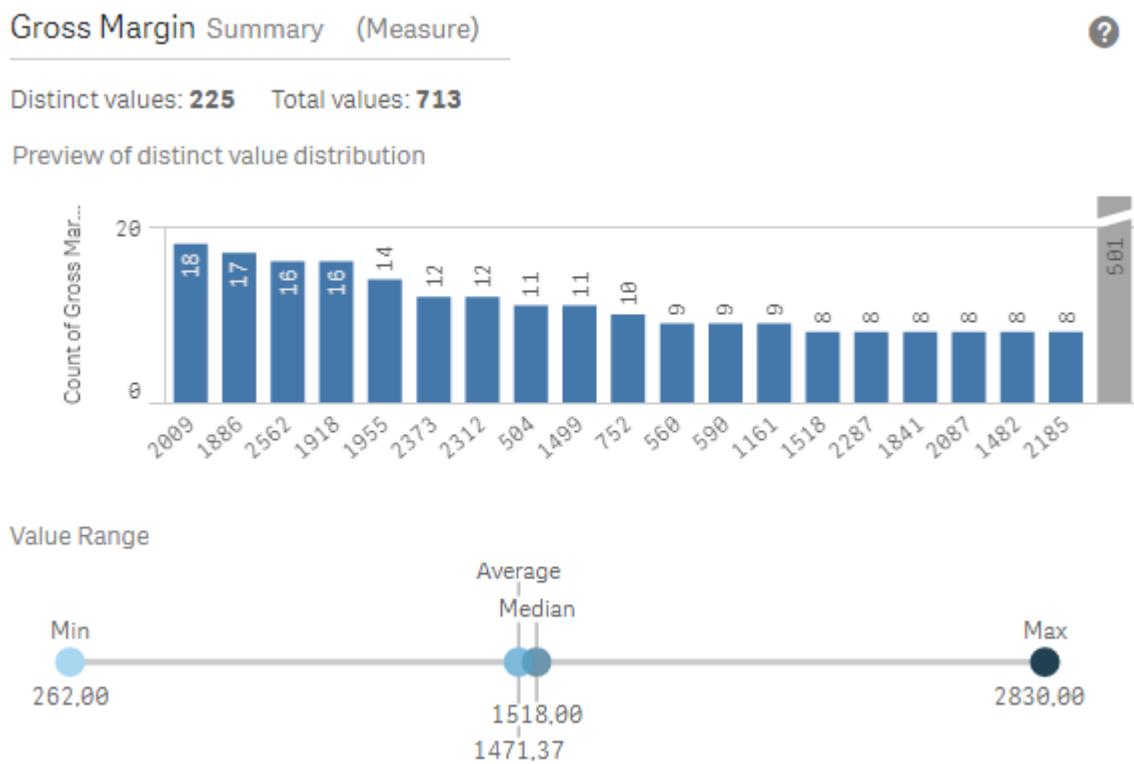
In addition, the **Summary** card enables you to view different possible data interpretations, such as viewing the field's data as a dimension or measure.

You access the **Summary** card by editing a table in **Data manager** and selecting a table field. When a field is selected in the table editor, Qlik Sense examines the data type, metadata, and values present. The field is then categorized as either a dimension, measure, or temporal field, and an analysis is presented in the **Summary** card. Fields whose data can be categorized as either a dimension or measure can switch preview to display

3 Loading and managing data with Data Manager

them as a dimension or measure. How a data field is categorized in the **Summary** card does not affect how you can use it in Qlik Sense visualizations, but it does determine what transformation options are available for the data field in other data profiling cards.

A summary of data using the Summary data profiling card.



The **Summary** card provides the following information:

- **Distinct values:** The number of distinct values in the field.
- **Total values:** The number of values in the field.
- **Preview of distinct value distribution:** In fields with more than 20 distinct values, only the 20 distinct values with the highest count display. All other values are grouped into a single value on the chart. If all values are distinct, no bar chart will display.
- **Value Range:** (Measure and Temporal only) For a measure field, the **Value Range** is a chart showing the Min, Median, Average, and Max values for the field. For a temporal field, the **Value Range** is the time period covered by the field's data.
- **Null values:** The number of null values in the data. This visualization only displays if there are null values in the field.
- **Mixed values:** The number of text value in a field that contains both text and numeric values. This visualization only displays if there are mixed values in the field.

Depending on how a field is categorized in the **Summary** card, it can be modified in other data profiling cards. Fields set as measures can have grouped values created from the field using the **Bucket** card. For more information, see *Grouping measure data into ranges* (page 64).

Fields set as dimensions can have:

- Distinct values replaced with other value using the **Replace** card.
For more information, see *Replacing field values in a table (page 58)*.
- Distinct values set as null values using the **Set nulls** card.
For more information, see *Setting field values as null in a table (page 60)*.
- A custom order applied to the values using the **Order** card.
For more information, see *Customizing the order of dimension values (page 61)*.
- Field data split into new table fields using the **Split** card.
For more information, see *Splitting a field in a table (page 62)*.

Accessing the **Summary** card

Do the following:

1. In **Data manager**, select a table and click .
2. Select a table field.

The Summary card appears.

Changing the data category of a field

Qlik Sense permits changing the data category of a field, provided the data in the field can be categorized in multiple categories.

Do the following:

- In the **Summary** card, click ▼ and select a different data category.

Replacing field values in a table

The **Replace** card enables you to replace different field values in your tables with other values.

In a data set, different terms might be used for the same object or concept. For example, the full name of a country and its abbreviation might both be found in the same data set. The **Replace** card enables you to ensure that these instances are treated as a single distinct value rather than different distinct values. For example, in a field that contains country data, you could replace *U.S*, *US*, and *U.S.A* with *USA*. You can also use the **Replace** card to change individual values, such as when a name in a data set needs to be changed.

You can set replace values in fields that contain up to a maximum of 5,000 distinct values.

In addition, calculated fields ignore replacement values and will use the original values instead.

The **Replace** card consists of two sections: **Distinct values** and **Replacement value**. **Distinct values** lists all distinct values and any replacement values. **Replacement value** contains a field to enter the replacement term and a list of values selected to be replaced. You replace field values by selecting distinct values, entering the replacement value, and applying the replacement. You can also select a replacement value in **Distinct values** and edit it to change the values being replaced or the value being used for the replacement. You can add or edit multiple replacement values before applying the replacements.

Replacing values

Do the following:

1. In **Data manager**, select a table and click .
2. Select a dimension field.
3. In the data profiling card, click the **Replace** tab.
4. Under **Distinct values**, select the distinct values you want to replace.
5. Under **Replacement value**, enter the new value for these distinct values.
6. Click **Replace**.

Editing a replacement value

You can add more distinct values to a replacement value, remove distinct values from a replacement value, or change the replacement value.

Adding distinct values to a replacement value

Do the following:

1. In the **Replace** card, under **Distinct values**, select a replacement value.
2. Under **Distinct values**, select the distinct values you want to add.
3. Click **Replace**.

Removing distinct values from a replacement value

Do the following:

1. In the **Replace** card, under **Distinct values**, select a replacement value.
2. After each distinct value you want to remove, click .
3. Click **Replace**.

Changing a replacement value

Do the following:

1. In the **Replace** card, under **Distinct values**, select a replacement value.
2. Under **Replacement**, enter a new value.
3. Click **Replace**.

Deleting a replacement value

Replacement values can be deleted by removing all associated distinct values.

Do the following:

1. In the **Replace** card, under **Distinct values**, select a replacement value.
2. Click **Remove All**.
3. Click **Replace**.

Setting field values as null in a table

In the **Set nulls** card, you select distinct values from a dimension field to be treated as null values in Qlik Sense.

For example, if your source data includes representations such as X for nulls, you can use the **Set nulls** card to set that value to be treated as a null value in Qlik Sense. If your table contains fields with empty spaces, you can set these as null values using the **Set nulls** card. You can also use the **Set nulls** card to clean your data of unwanted values by setting these values as null.

If you want to use a specific value as your null value, you can replace the default null value, - (**Null**), using the **Replace** card. For more information, see *Replacing field values in a table* (page 58).

You can set field values as null in fields that contain up to a maximum of 5,000 distinct values.

The **Set nulls** card consists of two sections, **Distinct values** and **Manual null values**. When you select values from **Distinct values**, they are added to **Manual null values**. When you apply the null values, all instances of the selected values are set to null in the field's data. You can restore individual or all values set as null.

Setting field values as null

Do the following:

1. In **Data manager**, select a table and click .
2. Select a field.
3. In the data profiling card, click the **Set nulls** card.
4. Under **Distinct values**, select the values you want set as null.
5. Click **Set null values**.

Restoring values manually set as null

Do the following:

1. In **Data manager**, select a table and click .
2. Select a field.
3. In the data profiling card, click the **Set nulls** card.
4. In the **Set nulls** card, under **Manual null values**, do one of the following:
 - Click  after the values you no longer want set as null.

- Click **Remove All** to restore all values set as null.
5. Click **Set null values**.

Customizing the order of dimension values

Custom ordering enables you to set the order of dimension values in visualizations.

While alphabetical or numerical order may work for many visualizations, in some instances, an alternative order is useful. For example, in a field containing cities, you could set a custom order so that in your charts cities are listed in order of east to west. You can define custom ordering for fields that meet the following requirements:

- Fields must be set as dimensions in the **Summary** card.
- Fields must contain up to a maximum of 5,000 distinct values.
It is recommended that your field have up to a maximum of 25 values, but you can choose to use the **Order** card with fields containing up to 5,000 distinct values.

The **Order** card consist of two sections: **Current Order** and **Preview of order**. **Current Order** displays all the distinct values from the dimension. By default, the distinct values are organized by load order. You set your custom order by dragging the values in **Current Order** into the desired order. **Preview of order** is a bar chart that displays a count of values in each distinct value, organized by the current order.

A custom order overrides all other sorting options available in Qlik Sense visualizations except for sorting by load order. If you require alphabetical or numeric ordering for this field, you must remove the custom order by resetting the order.

Changing the order of values in a dimension field

Do the following:

1. In **Data manager**, select a table and click .
2. Select a dimension field.
3. In the data profiling card, click the **Order** tab.
4. Click and drag the values into the new order.
5. If you want to cancel your current ordering, click **Cancel**.



Cancel is only available to new custom orders. To cancel your changes if you are changing the order of values in an existing custom order, select a different field in the table and then select this field again.

6. Click **Reorder**.

Resetting the order of values in a dimension field

Do the following:

1. In **Data manager**, select a table and click .
2. Select a reordered dimension field.

3. In the data profiling card, click the **Order** tab.
4. Click **Reset**.
5. Click **OK**.

Splitting a field in a table

The **Split** card enables you to create new fields using data from an existing field.

You could, for example, split a field that contains an address to create a new field that contains only a zip or postal code. This lets you quickly create new fields containing segments of existing data. The **Split** card can create new fields from table fields that meet the following requirements:

- Fields must be set as dimensions in the **Summary** card.



*Table fields containing date and time information are automatically split into date fields when their tables are prepared in **Data manager** and do not require the **Split** card.*

The **Split** card consists of an input field containing a template value and a preview of the new fields with their values. By default, the template value is the first value in numerical order from a field, but you can select other values from the source field to use as the template. You should select a value that is representative of all values in the table. Splitting using an outlier value as a template may impact the quality of the new fields.

Fields are split by inserting split markers into the template value where you want to split the field. Split markers are added by selecting a point in the sample field where you want to add a split marker, adjusting your selection, and then selecting to split by instance or position. The **Split** card may automatically add recommended split markers to your template value.

Instances are occurrences of a selected delimiter, such as the character @ or a space between words.

Positions of instance split markers are relative to either:

- The start of the value, such as the first instance of @ in a value.
- Their position to the right of another instance or position split marker, such as the first instance of . after @.

If you remove the instance to which another instance is relative, the other instance's position adjusts to the same position relative to the next instance of a different delimiter set as a split marker or the start of the value. You can split a field on up to 9 delimiters.



*The **Split** card splits values using the characters specified as the split markers. If the data has variances in how these characters are composed, such as accented characters, those variances will not be included in the split.*

Positions are locations in the field value, such as after the first four characters. Positions are relative to either:

3 Loading and managing data with Data Manager

- The start of the value.
- Their position to the right of an instance split marker, such as the second character after an instance split marker.

If you removed an instance that has a position to its right, the position moves to the same position relative to the start of the value or the first next instance split marker to the left or the start of the value.

The field preview updates as you add split markers , showing the new fields and their data. You can rename the new fields in the field preview. You can also select to include or exclude split fields from your table before you apply the split. When you apply a split, the fields you selected in the field preview are added to your table.

A field can be split multiple times.

To split a field into new fields, do the following:

1. Access the **Split** card.
2. Set split markers in the input field.
3. You can also perform these optional tasks:
 - Remove split markers.
 - Select which new fields you want to be added to the table.
 - Rename the new fields.
4. Create the new fields.

Accessing the Splitting card

Do the following:

1. In **Data manager**, select a table and click .
2. Select a dimension field.
3. In the data profiling card, click the **Split** tab.

Inserting split markers

You insert split markers by clicking positions in the sample value and selecting the kind of split you want to apply.



You can change the sample value displayed in the input field by clicking ▾ and selecting a different value.

Do the following:

1. In the **Split** card, click the position in the sample value where you want to add split markers.
Clicking selects all of the template value up to any other split markers.
Double-clicking selects the insert point of your cursor in the template value.
2. Adjust your selection by clicking and dragging the selection tabs or highlighting the section you want to select.

3 Loading and managing data with Data Manager

3. Click the button that corresponds to the kind of split you want applied:
 - **This instance:** The field splits by the selected instance of the delimiter.
 - **All instances:** The field splits by all instances of the delimiter.
 - **These positions:** The field splits on either side of the selection.
 - **This position:** The field splits at this position.

The split marker is inserted into the template value.

Removing split markers

Do the following:

- In the **Split** card, do one of the following:
 - To remove a single split marker, click  above the split marker you want to remove.
 - To remove all split marker, click **Reset**.
Recommended split markers will not be removed and any removed recommended split markers will be added back to the template. You must remove these individually.

Selecting the fields to add to your table

You can select which fields created by the **Split** card to include or exclude from your table. By default, all split fields are included.

Do the following:

- In the **Split** card, in the field preview, do one of the following:
 - To include a field, select the field column checkbox.
 - To exclude a field, clear the field column checkbox.

Renaming new fields

Do the following:

- In the **Split** card, in a field header in the field preview, enter a new field name.

Creating new fields

Do the following:

- In the **Split** card, click **Split**.

Grouping measure data into ranges

Measures can provide additional insight for analysis when grouped into ranges and used as dimensions in visualizations.

3 Loading and managing data with Data Manager

For example, if a field had customer ages, you could group values into age ranges. The **Bucket** data profiling card enables the grouping of field data into ranges, creating a new field with the specified groupings. You access the **Bucket** card by editing a table in **Data manager** and selecting a table field that is set as a measure in the **Summary** card.

Requirements

You can group measures into ranges with fields that meet the following requirements:

- Fields must be classified as a measure in the **Summary** card
- Fields must contain at least 10 distinct values.
The **Bucket** card is not recommended with fields containing a low range of values, but you can choose to use the **Bucket** card with non-recommended fields.
- Values in the field cannot contain more than 20% mixed types.
The **Bucket** card is not recommended with fields containing mixed types of values, but you can choose to use the **Bucket** card with non-recommended fields.



*Bucket fields created by the **Bucket** card are categorized as dimensions in the **Summary** card.*

*Bucket fields cannot have a custom order applied to them using the **Order** card, however. Bucket fields cannot be used in calculated fields.*

Overview

The **Bucket** card provides a suggested number of groupings, a preview of the groupings, and a slider bar containing your groupings that enables modifying each bucket's name and value range. You can modify the suggested number of groupings by entering a new number into the **Bucket** field. Qlik Sense supports a maximum of 20 buckets and a minimum of 2 buckets. New buckets are added to the right of your bucket range while buckets are removed from right to left. If you have not modified any individual buckets, each bucket's range is adjusted to fit evenly within the value range of the field. If you change the number of buckets after having modified an individual bucket, new buckets are added to the right of your bucket range and are given a range of values equal to the size of the second rightmost bucket.

The **Preview of data buckets** bar chart gives overview of the data in your buckets, with a count of the number of distinct values in each bucket. The chart is updated as you alter your buckets. If a bucket has no values in it, it will have no bar in the chart.

The **Bucket** slider bar enables you to edit your buckets. By clicking a bucket segment, you can set that bucket's range, change the bucket's name, or remove the bucket entirely. Hovering your cursor over the bucket brings up the bucket's name and range of values. By default, buckets are named with the bucket's value range expressed as an interval notation. Buckets ranges include values between the starting value and up to but excluding the ending value.

When you adjust a single bucket's value range, Qlik Sense shifts the values of all buckets, ensuring there are no gaps and overlaps while respecting the existing quantitative ranges of the other buckets as much as possible. The leftmost bucket always has no lower boundary and the rightmost bucket has no upper boundary. This allows them to capture any values that might fall outside the set ranges of all your buckets. Modifying the lower range of a bucket will alter the ranges of the buckets to the right, modifying the upper range of a bucket will modify the buckets to the left.

3 Loading and managing data with Data Manager

When you create buckets from a field, a new field is generated containing all the buckets assigned to the rows with their corresponding measure values from the source field. By default, it will be named <field> (Bucketed). This field can be renamed, associated, sorted, and deleted like other table fields. You can edit the bucketing in the generated bucket field by selecting the field in the table and modifying the bucketing options. You can create multiple bucket fields from the same source measure field.

Grouping measures

To group measure values into ranges, do the following:

1. Access the **Bucket** card.
2. Optionally, change the number of buckets.
3. Optionally, modify individual buckets.
4. Optionally, reset the manual adjustments to buckets to the default settings.
5. Create the bucketed field.

Accessing the Buckets card

Do the following:

1. In **Data manager**, select a table and click .
2. If you are creating new groups of value ranges, select a field.
3. If you are editing an existing buckets field, select a buckets field

Changing the number of buckets

Changing the number of buckets adds buckets to or removes buckets from the end of the range.

Do the following:

- In the **Bucket** card, enter a new number before **Buckets**.

Modifying a bucket

You can rename a bucket, adjust a bucket's value range, or delete a bucket.



If you are modifying buckets, it is recommended to modify buckets in order from the leftmost bucket segment to the rightmost bucket segment.

Renaming a bucket

Do the following:

1. In the **Bucket** card, click the bucket segment.
2. In the name field, enter a new name.
3. To apply your changes, click anywhere outside the bucket segment.

Adjusting a bucket's range of values



If you are trying to use decimal values, you must enter them manually in the **From** and **Up To** fields.

Do the following:

1. In the **Bucket** card, click the bucket segment.
2. Do one of the following:
 - After **From** and **Up To**, enter new values to set the range of the bucket.
A bucket includes the values between the value in **From** and up to but excluding the value in **Up To**.
 - Adjust the segment's sliders to set the range of the bucket.
3. To apply your changes ,click anywhere outside the bucket segment.

Deleting a bucket

Do the following:

1. In the **Bucket** card, click the bucket's segment.
2. Click .

Resetting a field's buckets

Resetting buckets enables you to restore the bucket card settings to their default state. If this is a measure field, the default state is the Qlik Sense recommended bucketing. If this is a bucket field, the default state is the field's state after the last time **Create buckets** was clicked.

Do the following:

- In the **Bucket** card, click **Reset to default**.

Creating a bucket field

Do the following:

1. In the **Bucket** card, click **Create buckets**.
2. Click **OK**.

The new field containing your data grouping is added to your table.

Unpivoting crosstab data in the data manager

A crosstab is a common type of table featuring a matrix of values between two orthogonal lists of header data.

It is usually not the optimal data format if you want to associate the data to other data tables. This topic describes how you can unpivot data loaded in crosstab format, that is, transpose parts of it into rows using the data manager.

Unpivot data loaded in crosstab format transpose parts of it into rows.

3 Loading and managing data with Data Manager

The diagram illustrates the transformation of a crosstab from a pivoted format to an unpivoted format. On the left, a pivoted table has columns for Year (2007, 2008), Region (Europe, RoW), and Sales (234, 345, 567, 534). A blue arrow points to the right, where the same data is shown in an unpivoted format with three columns: Year (2007, 2008), Region (Europe, RoW), and Sales (234, 567, 345, 534).

Year	Europe	RoW
2007	234	567
2008	345	534

Year	Region	Sales
2007	Europe	234
2007	RoW	567
2008	Europe	345
2008	RoW	534

What's a crosstab?

A crosstab contains a number of qualifying columns, which should be read in a straightforward way, and a matrix of values. In this case there is one qualifying column, Year, and a matrix of sales data per month.

Crosstab						
Year	Jan	Feb	Mar	Apr	May	Jun
2008	45	65	78	12	78	22
2009	11	23	22	22	45	85
2010	65	56	22	79	12	56
2011	45	24	32	78	55	15
2012	45	56	35	78	68	82

If this table is simply loaded into Qlik Sense, the result will be one field for Year and one field for each of the months. This is generally not what you would like to have. You would probably prefer to have three fields generated:

- The qualifying field, in this case Year, marked with green in the table above.
- The attribute field, in this case represented by the month names Jan - Jun marked with yellow. This field can suitably be named *Month*.
- The data field, marked with blue. In this case they represent sales data, so this can suitably be named *Sales*.

This can be achieved by using the Unpivot option in the data manager table editor, and selecting the fields Jan - Jun. This creates the following table:

Unpivoted table		
Year	Month	Sales
2008	Jan	45
2008	Feb	65
2008	Mar	78
2008	Apr	12
2008	May	78

3 Loading and managing data with Data Manager

Year	Month	Sales
2008	Jun	22
2009	Jan	11
2009	Feb	23
...

Unpivoting a crosstab table into a flat table

Do the following:

1. Add a data file in crosstab format to your app.
2. Click  on the table in the data manager to open the table editor.
3. Click **Unpivot**.
4. Select the fields you want to transpose into rows. You need to have at least one qualifying field that is not unpivoted. There are two ways to make the selections.
 - Click on the field headers of the fields you want to transpose. Do not select the fields you want to keep as qualifying fields.
 - Click on the field headers of the fields you want to keep as qualifying fields, and then select **Invert selections** from the field menu. This is the easiest way to do it if you have a large number of fields to transpose.
5. Click **Apply unpivoting**
The selected data is now transposed to rows with two fields, Tablename.**Attribute field** and Tablename.**Data field**.
6. Rename **Attribute field** to something meaningful, in the example above, *Month*.
7. Rename **Data field** to something meaningful, in the example above, *Sales*.

You have now unpivoted the crosstable to a flat format, which will make it easier when you want to associate it to other data in the app.

Reverting to the original crosstab table

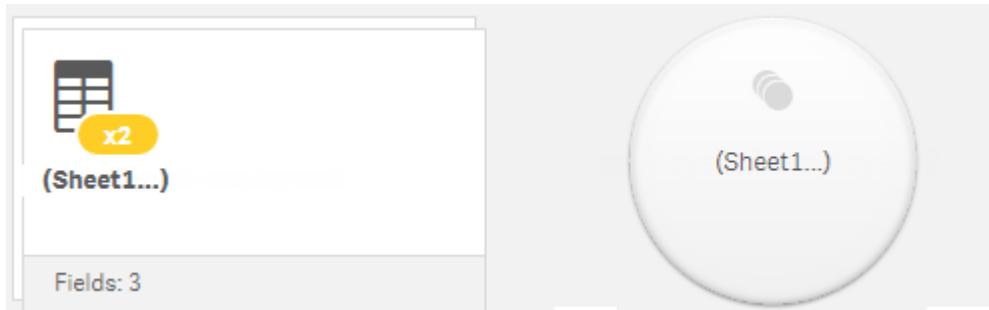
You can revert to the original crosstab format of your data source by clicking **Undo unpivot** in the table editor. If you created any associations to other data in the app, the associations will be deleted.

3.13 Concatenating tables in **Data manager**

Concatenation combines two tables into a single table with combined fields. It consolidates content, reducing the number of separate tables and fields that share content. Tables in **Data manager** can be automatically or forcibly concatenated.

If you need more granular control over which data is included in combined tables, see *Joining tables in Data manager* (page 73) to learn more about **Join** operations.

Concatenated table in Tables view and Associations view.



Automatically concatenating tables

Tables are automatically concatenated in **Data manager** when Qlik Sense detects that one or more added tables have both the same number of fields and identical field names as another table. When this happens, Qlik Sense automatically concatenates the tables into a single table. Automatically concatenated tables can be split if they were concatenated erroneously or if you do not want them concatenated. Automatically concatenated tables can be forcibly concatenated to other tables.

Automatically concatenated tables have the following restrictions:

- You cannot change field categories.
- You cannot unpivot an automatically concatenated table.
- You cannot add or remove data with **Select data from source**.

Forcing concatenation between tables

Concatenation can be forced between tables in **Data manager** using the **Concatenate or join** pane, even if they do not entirely share the same fields or data. Forced concatenation enables you to control mapping and exclude fields from the final concatenated table. Forced concatenation supports additional transformations. Using forced concatenation, you can:

- Concatenate a concatenated table with another table.
- Concatenate an unpivoted table with another table. Forcibly concatenated tables can be unpivoted.
- Concatenate tables with calculated fields. Calculated fields can be concatenated to other fields in a forced concatenation. Calculated fields can be added to forcibly concatenated tables.

Forcibly concatenated tables have the following restrictions:

- Forced concatenation requires at least one field from each table be included in the concatenated table, although they need not be mapped together.
- Date fields cannot be formatted after concatenation. Date fields must have the same format applied to them before concatenation. Concatenated date fields use the default time format set with **DateFormat** in the **Data load editor**.
- You cannot change field categories after concatenation.
- Calculated fields that refer to a field mapped to another field in a concatenated table will only contain data for the original field rather than the combined data in the concatenated field. Calculated fields created after two tables are concatenated that refer to a field in the concatenated table will use all data in that field.

3 Loading and managing data with Data Manager

- You cannot add or remove data from a concatenated table with **Select data from source**. You can, however, remove fields by clicking **Add data**, selecting the source table, and then excluding the fields. Null values are added for the removed field's data.

The **Concatenate or join** pane is accessed by clicking *** in **Data manager**, clicking **Concatenate or join**, and selecting two tables. When tables are selected in **Concatenate or join**, Qlik Sense analyzes the fields and automatically maps any fields together that match. If there are no clear matches, fields are left unmapped. When the concatenation is applied, mapped fields are combined in the concatenated table, while unmapped fields are included as individual fields with null values for the rows where there is no corresponding value.

The first table selected in **Concatenate or join** is set as the primary table, the table to which the other table is concatenated. The concatenated table uses the table and field names from the primary table unless these are manually renamed. You can change which table is the primary table with the ↪ button. **Concatenate or join** arranges fields in two rows, with the primary table fields in the top row and the secondary table fields on the bottom row. You can swap the primary and secondary tables with the ↪ button.

You can use **Edit mappings** to change the default mapping and select which fields to map, leave unmapped, or to exclude from the concatenated table. **Edit mappings** contains a drag and drop interface for editing mappings and the **Fields** pane, which lists all table fields. Fields can be mapped by dragging them beneath a primary table field. Fields can be added as a new unmapped field by  beside the field in the **Fields** pane or dragging them into the top row of fields. Unmapped fields are marked with  in the **Fields** pane. Fields removed from the concatenated table are not included in the table and are not available for use in Qlik Sense after concatenation is applied to the table.

Once mappings are applied and the tables are concatenated, you cannot edit them, but they can be removed from the tables by splitting the concatenated table, which restores the tables to their original state.

To forcibly concatenate tables in **Data manager**, do the following:

1. Select tables for concatenation.
2. Optionally, rename the concatenated table and the field names.
3. Optionally, edit the concatenation mappings.
4. Select the concatenation operator.
5. Concatenate the tables.

Selecting tables for concatenation

Do the following:

1. In **Data manager**, click *** in the bottom row.
2. Click **Concatenate or join**.
The **Concatenate or join** pane opens.
3. Select two tables.
The fields of both tables will be mapped or left unmapped in the **Concatenate or join** pane.

3 Loading and managing data with Data Manager

4. To preview a sample of unique values in each field, click .
5. To switch the primary and secondary tables, click .

Renaming the table and field names

Do the following:

1. In the **Concatenate or join** pane, in the table name field, enter a new table name.
2. In a field name field, enter a new field name.

Editing concatenation mappings

Do the following:

1. In the **Concatenate or join** pane, click **Edit mappings**.
2. To map two fields, click and drag a table field under a primary table field.
3. To add a new unmapped field, click and drag a table field into the upper row of fields.
4. To remove a field from the concatenated table, in the field click .
5. To return a removed field back to the table, click in the **Fields** pane, click  beside the field.
6. Click **Edit mappings** to close **Edit mappings**.

Selecting the concatenation operator

Do the following:

1. In the **Concatenate or join** pane, click **Select action**.
2. Select **Concatenate** from the list.

Concatenating tables

Do the following:

- In the **Concatenate or join** pane, click **Apply**.

The tables are now concatenated.

Splitting concatenated tables

In cases where concatenation is no longer needed, such as when Qlik Sense has performed an unwanted automatic concatenation, you can split the concatenated tables into their source tables.



*Splitting a concatenated table will remove any associations the concatenated table had as well as any associations the primary and secondary tables had with each other. If you want to preserve your associations while splitting concatenated tables, click  to undo the concatenation instead of splitting the table. You cannot use  to undo concatenation after you load data in **Data manager**.*

Splitting an automatically concatenated table

Do the following:

1. Select the concatenated table.
2. Click .
3. Select the tables to split from the concatenated table.
4. Click **Split**.

The table is now split into its source tables and all fields in the source tables are qualified. Qualified fields are renamed with the table name followed by the field name, separated by a period punctuation mark (the character “.”).

Example:

Table1 and Table2 both contain the fields Field1 and Field2. When you add them in **Data manager**, they are concatenated to a table called Table1-Table2 with the same fields, Field1 and Field2.

If you split Table1-Table2, the result is two tables:

- Table1 with fields Table1.Field1 and Table1.Field2
- Table2 with fields Table2.Field1 and Table2.Field2

Splitting a forcibly concatenated table

Do the following:

1. Select the concatenated table.
2. Click .

The table is now split into its source tables. All fields in the source tables and their fields have their pre-concatenation names. Splitting a concatenated table only splits one level of concatenation, so that any concatenated tables that were part of the split concatenated table have their own concatenation preserved.

3.14 Joining tables in **Data manager**

Join is an operation that can be used to manually combine two tables' data, producing varied results depending on the configuration you select.

This allows more granular control over combined tables than with concatenation.

The **Join** operation takes two tables and combines them into one, which will be a combination of the fields in both original tables, based on the overlap of a common value for one or several common fields. There are multiple operators that can be applied to **Join** operations: **Outer**, **Inner**, **Left**, and **Right**.



A joined table still occupies the amount of memory as the tables combined in it. Excessive use of joined tables may cause Qlik Sense to slow down. The information that is excluded by a join operation will not be accessible by Qlik Sense until the table is split.

Join operators

There are four join operators: **Outer join**, **Inner join**, **Left join**, and **Right join**. The selected operator determines which overlapping fields or values are included or excluded.



When the join operators refer to **Left** and **Right** tables, they are referring to the first and second tables respectively, in order of selection.

Outer join

The **Outer join** operator contains all possible combinations of values from the two tables, if the overlapping field values are represented in either one or both tables.

Example:

First table

A	B
1	aa
2	cc
3	ee

Second table

A	C
1	xx
4	yy

Joined table

A	B	C
1	aa	xx
2	cc	-
3	ee	-
4	-	yy

3 Loading and managing data with Data Manager

Inner join

The **Inner join** operator only contains combinations of values from the two tables, if the overlapping field values are represented in both tables.

Example:

First table		
A		B
1		aa
2		cc
3		ee

Second table		
A		C
1		xx
4		yy

Joined table		
A	B	C
1	aa	xx

Left join

The **Left join** operator contains combinations of values from the two tables, if the overlapping field values are represented in the first table.

Example:

First table		
A		B
1		aa
2		cc
3		ee

Second table		
A		C
1		xx
4		yy

3 Loading and managing data with Data Manager

Joined table

A	B	C
1	aa	xx
2	cc	-
3	ee	-

Right join

The **Right join** operator contains combinations of values from the two tables, if the overlapping field values are represented in the second table.

Example:

First table

A	B
1	aa
2	cc
3	ee

Second table

A	C
1	xx
4	yy

Joined table

A	B	C
1	aa	xx
4	-	yy

Joining tables

The **Concatenate or join** pane is accessed by clicking ******* in **Data manager**, clicking **Concatenate or join**, and selecting two tables. When tables are selected in **Concatenate or join tables**, Qlik Sense analyzes the fields and automatically maps any fields together that match. If there are no clear matches, fields are left unmapped. When the join is applied, mapped fields are combined in the joined table. Unmapped fields are either included as individual fields with null values for the rows where there is no corresponding value, or excluded entirely if there are no overlapping instances of the value.

3 Loading and managing data with Data Manager

The first table selected in **Concatenate or join tables** is set as the primary table, the table to which the other table is joined. The joined table uses the table and field names from the primary table unless these are manually renamed. You can change which table is the primary table with the  button. **Concatenate or join tables** arranges fields in two rows, with the primary table fields in the top row and the secondary table fields on the bottom row. You can swap the primary and secondary tables with the  button.

You can use **Edit mappings** to change the default mapping and select which fields to map, leave unmapped, or to exclude from the joined table. **Edit mappings** contains a drag and drop interface for editing mappings and the **Fields** pane, which lists all table fields. Fields can be mapped by dragging them beneath a primary table field. Fields can be added as a new unmapped field by  beside the field in the **Fields** pane or dragging them into the top row of fields. Unmapped fields are marked with  in the **Fields** pane. Fields removed from the joined table are not included in the table and are not available for use in Qlik Sense after joins are applied to the table.

Once mappings are applied and the tables are joined, you cannot edit the mapped fields, but they can be removed from the tables by splitting the joined table, which restores the tables to their original state.

To join tables in **Data manager**, do the following:

1. Select tables for joining.
2. Optionally, rename the joined table and the field names.
3. Optionally, edit mappings.
4. Select the join operator.
5. Join the tables.

Selecting tables for joining

Do the following:

1. In **Data manager**, click  in the bottom row.
2. Click **Concatenate or join**.
The **Concatenate or join** pane opens.
3. Select two tables.
The fields of both tables will be mapped or left unmapped in the **Concatenate or join tables** pane.
4. To preview a sample of unique values in each field, click .
5. To switch the primary and secondary tables, click .

Renaming the table and field names

Do the following:

1. In the **Concatenate or join** pane, in the table name field, enter a new table name.
2. In a field name field, enter a new field name.

Editing mappings

Do the following:

1. In the **Concatenate or join** pane, click **Edit mappings**.
2. To map two fields, click and drag a table field under a primary table field.
3. To add a new unmapped field, click and drag a table field into the upper row of fields.
4. To remove a field from the joined table, in the field click .
5. To return a removed field back to the table, click in the **Fields** pane, click  beside the field.
6. Click **Edit mappings** to close **Edit mappings**.

Selecting the join operator

Do the following:

1. In the **Concatenate or join** pane, click **Select action**.
2. Select an operator from the list: **Outer join**, **Inner join**, **Left join**, or **Right join**.

Joining tables

Do the following:

- In the **Concatenate or join** pane, click **Apply**.

The tables are now joined.

Splitting joined tables

In cases where joining is no longer needed, you can split the joined tables into their source tables.



*Splitting a joined table will remove any associations the joined table had as well as any associations the primary and secondary tables had with each other. If you want to preserve your associations while splitting joined tables, click  to undo the join instead of splitting the table. You cannot use  to undo joins after you load data in **Data manager**.*

Splitting a joined table

Do the following:

1. Select the joined table.
2. Click .

The table is now split into its source tables. All fields in the source tables and their fields have their pre-join names. Splitting a joined table only splits one level of joining, so that any joined tables that were part of the split joined table have their own join preserved.

3.15 Viewing table and field transformation details in **Data manager**

You can view the operations and transformations performed on tables and fields in **Data manager** using the **Details** dialog. The **Details** dialog is available in the **Associations** and **Table** views for tables and in the data table editor for fields.

Details displays the current operations and transformations made to the selected table or field, in the order they are applied in the generated data load script. This enables you to easily see the source of a table or field, the current changes that have been made, and the sequence in which the changes have been applied. You can use **Details**, for example, to easily see which tables were concatenated or if a field was reordered.

The information displayed in **Details** varies depending if you are viewing a table or field. Table **Details** displays:

- Source tables for the selected table.
- Transformations used on the table, such as unpivoting and concatenation.

Field **Details** displays:

- Source tables and fields for the selected field.
- Field type changes.
- Transformations used on the fields, such as from the data profiling card or from concatenation.

Viewing table details

Do the following:

- In **Data manager**, select a table, click  , and click **View details**.

The **Details** dialog opens.

Viewing field details

Do the following:

1. In **Data manager**, select a table and click .
2. Click  above a field heading and click **View details**.

The **Details** dialog opens.

3.16 Step-by-step - Combining tables using forced concatenation

This step-by-step walkthrough shows how you can use forced concatenation to combine two similar data tables.

Forced concatenation can be used to clean up your data before you use it for analysis in a sheet. You can concatenate two tables into one table. You can also add another table later, for example if you initially add a table from June, and then later want to add a second table from July.

Concatenation at a glance

- Tables are automatically concatenated in Data manager when Qlik Sense detects that one or more added tables have both the same number of fields and identical field names as another table. In this case, you can split the tables if needed.
- Two tables can be force concatenated when tables do not entirely share the same fields or data. Only two tables can be force concatenated. To concatenate three tables, for example, concatenate the first two tables into one table. Concatenate the third table to that created table.
- Tables that are not similar enough will not automatically be concatenated. You also will not be able to forcibly concatenate them. In this case, the fields in the table should instead be associated in the Data manager.

Walkthrough - Forced concatenation

These are the tasks required to complete the walkthrough:

1. Prepare the data tables
2. Add data tables to an app
3. Concatenate and load data tables into an app
4. A step further - adding a new table and concatenating the data fields

Prerequisites

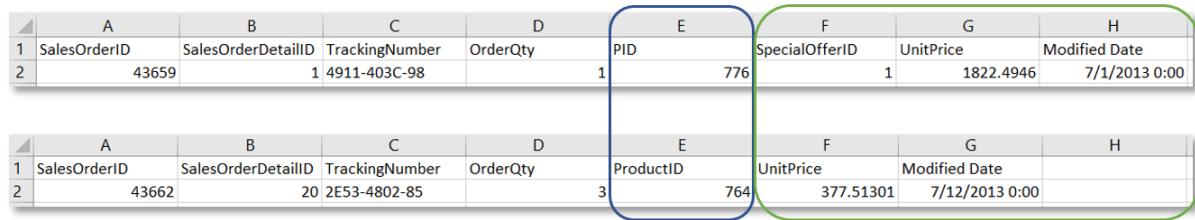
You should know how to create an app in Qlik Sense.

Prepare the data tables

We have supplied some sample data for you to use to demonstrate forced concatenation. You can also use your own data, provided the fields and data are mostly the same in your two tables.

For example, here is the header and first row of the data that we supplied below. It has been pasted into two Excel tables. Note the differences in the fields.

3 Loading and managing data with Data Manager



	A	B	C	D	E	F	G	H
1	SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	PID	SpecialOfferID	UnitPrice	Modified Date
2	43659	1	4911-403C-98		1	776	1	1822.4946

	A	B	C	D	E	F	G	H
1	SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	ProductID	UnitPrice	Modified Date	
2	43662	20	2E53-4802-85		3	764	377.51301	7/12/2013 0:00

If you want to use the sample data, copy the entire table, including the column headings, into an empty Excel file on your computer. For this walkthrough, we named the Excel tabs *Data Table 1* and *DataTable 2*. We named the Excel file *Concatenate_Data.xlsx*.

Data Table 1

Data Table 1

SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	PID	SpecialOfferID	UnitPrice	Modified Date
43659	1	4911-403C-98	1	776	1	1822.494	7/1/2016 3:00:00
43659	2	4911-403C-98	3	777	1	1822.494	7/2/2016 3:00:00
43659	3	4911-403C-98	1	778	1	1098.9	7/2/2016 3:00:00
43659	4	4911-403C-98	1	771	1	1835.994	7/2/2016 3:00:00
43659	5	4911-403C-98	1	772	1	1835.994	7/3/2016 3:00:00
43659	6	4911-403C-98	2	773	1	1835.994	7/3/2016 3:00:00
43659	7	4911-403C-98	1	774	1	1835.994	7/4/2016 3:00:00
43659	8	4911-403C-98	3	714	1	25.95636	7/4/2016 3:00:00
43659	9	4911-403C-98	1	716	1	25.95636	7/5/2016 3:00:00
43659	10	4911-403C-98	6	709	1	5.13	7/5/2016 3:00:00
43659	11	4911-403C-98	2	712	1	4.66785	7/6/2016 3:00:00
43659	12	4911-403C-98	4	711	1	18.16785	7/6/2016 3:00:00

3 Loading and managing data with Data Manager

SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice	ModifiedDate
43660	13	6431-4D57-83	1	762	1	377.51301	7/7/2013 0:00
43660	14	6431-4D57-83	1	758	1	787.3146	7/7/2013 0:00
43661	15	4E0A-4F89-AE	1	745	1	728.784	7/7/2013 0:00
43661	16	4E0A-4F89-AE	1	743	1	643.2338	7/8/2013 0:00
43661	17	4E0A-4F89-AE	2	747	1	643.2338	7/8/2013 0:00
43661	18	4E0A-4F89-AE	4	712	1	4.66785	7/8/2013 0:00
43661	19	4E0A-4F89-AE	4	715	1	25.95636	7/9/2013 0:00

DataTable 2

Data Table 2

SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	ProductID	UnitPrice	ModifiedDate
43662	20	2E53-4802-85	3	764	377.51301	7/12/2013 0:00
43662	21	2E53-4802-85	5	770	377.51301	7/12/2013 0:00
43662	22	2E53-4802-85	2	730	165.54438	7/13/2013 0:00
43662	23	2E53-4802-85	4	754	787.3146	7/14/2013 0:00
43662	24	2E53-4802-85	3	725	165.54438	7/14/2013 0:00
43662	25	2E53-4802-85	5	762	377.51301	7/14/2013 0:00
43662	26	2E53-4802-85	3	765	377.51301	7/14/2013 0:00
43662	27	2E53-4802-85	2	768	377.51301	7/15/2013 0:00

3 Loading and managing data with Data Manager

SalesOrderID	SalesOrderDetailID	TrackingNumber	OrderQty	ProductID	UnitPrice	ModifiedDate
43662	28	2E53-4802-85	1	753	1932.2658	7/15/2013 0:00
43663	29	2E53-4802-85	1	756	787.3146	7/16/2013 0:00
43663	30	2E53-4802-85	3	763	377.51301	7/17/2013 0:00
43664	31	2E53-4802-85	1	732	321.2082	7/18/2013 0:00
43664	32	2E53-4802-85	6	758	787.3146	7/19/2013 0:00
43665	33	2E53-4802-85	1	729	165.54438	7/19/2013 0:00

Add data tables to an app

Do the following:

1. Start Qlik Sense.
2. Click **Create new app** in your work area. The **Create new app** window opens.
3. Name your app, and then click **Create**. The app is created. We named our app *ConcatenateExample*.
4. Click **Open app**. The app opens and displays a dialog where you can add data.
5. Drag and drop your **Excel** file into the **Add data from files and other sources** dialogue. Your tables are displayed in the **Associations** view of the **Data manager**. Click a bubble to see the data for that table.



If you add data instead from **Data manager**, you will first be asked to select table fields before being taken to the **Associations** view of the **Data manager**. In this case, select all the

3 Loading and managing data with Data Manager

i fields for both tables.

The screenshot shows the Qlik Sense Data Manager interface. At the top, there are navigation icons for Home, Search, and a list icon with a '1' notification. Below the navigation is a large circular area containing two smaller circles labeled "Data Table 1*" and "Data Table 2*". To the left of these circles is a circle with a plus sign. A note at the bottom left states: "* This table has not been loaded or has changed since the last time it was loaded." On the right side of the main area, there is a table titled "Data Table 1 Concatenate_Data.xlsx" with 8 fields. The table contains 5 rows of data with columns for SalesOrderID, SalesOrderLineID, TrackingNumber, OrderQty, PID, SpecialOfferID, UnitPrice, and Modified Date. The table is currently in "Pending add" mode. At the bottom of the table are several small icons for file operations like Save and Print, followed by a "Hide data preview" button.

Data Table 1.SalesOr...	Data Table 1.SalesOr...	Data Table 1.Tracking...	Data Table 1.OrderQty	PID	SpecialOfferID	Data Table 1.UnitPrice	Data Table 1.Modified Date
43659	1	4911-403C-98	1	776	1	1822,4946	2013-07-01 00:00:00
43659	2	4911-403C-98	3	777	1	1822,4946	2013-07-02 00:00:00
43659	3	4911-403C-98	1	778	1	1098,9	2013-07-02 00:00:00
43659	4	4911-403C-98	1	771	1	1835,9946	2013-07-02 00:00:00
43659	5	4911-403C-98	1	772	1	1835,9946	2013-07-03 00:00:00

Concatenate tables and load data tables into an app

After the data tables have been added to the app, the tables can be concatenated.

Do the following:

1. In the **Associations** view of **Data manager**, select one table by clicking the bubble. Click ******* and then select **Concatenate or join**.

3 Loading and managing data with Data Manager

The screenshot shows the Qlik Sense Data Manager interface. At the top, there are tabs for 'Prepare Data manager' (selected), 'Sheet', 'Narrate Storytelling', and 'Concatenation'. Below the tabs, there are buttons for '+ Add data' and 'Concatenate or join'. On the right side, there are icons for associations, load data, and a search bar. The main area displays two circular nodes: 'Data Table 2*' at the top and 'Data Table 1*' below it. A context menu is open over 'Data Table 1*', containing three items: 'Concatenate or join', 'Synchronize scripted tables', and 'View details'. A note at the bottom left states: '* This table has not been loaded or has changed since the last time it was loaded.' The table preview for 'Data Table 1' shows five rows of sales data.

Data Table 1.SalesOr...	Data Table 1.SalesOr...	Data Table 1.Tracking...	Data Table 1.OrderQty	PID	SpecialOfferID	Data Table 1.UnitPrice	Data Table 1.Modified Date
43659	1	4911-403C-98		1	776	1	1822,4946
43659	2	4911-403C-98		3	Concatenate or join	1	1822,4946
43659	3	4911-403C-98		1	Synchronize scripted tables	1	1098,9
43659	4	4911-403C-98		1	View details	1	1835,9946
43659	5	4911-403C-98		1		1	1835,9946

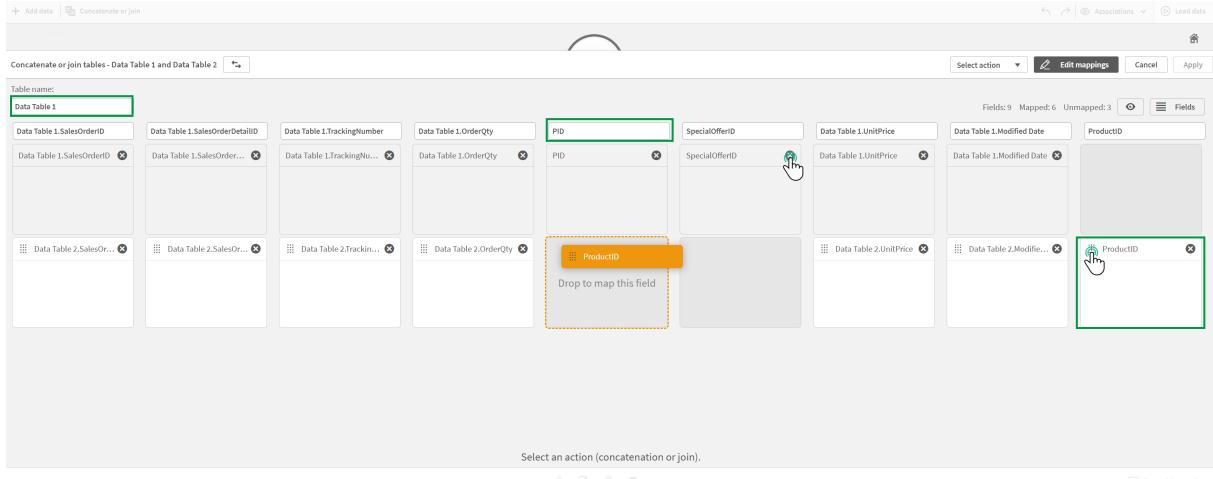
2. Click the bubble for the other table, and then click **Edit mappings**.

The screenshot shows the 'Concatenate or join tables' dialog. At the top, it says 'Concatenate or join tables - Data Table 1 and Data Table 2'. Below that is a 'Table name:' input field with 'Data Table 1' selected. To the right, there are buttons for 'Select action', 'Edit mappings' (which is highlighted with a cursor), 'Cancel', and 'Apply'. Underneath, there's a table mapping grid. The first row shows the source fields: 'Data Table 1.SalesOrderID', 'Data Table 1.SalesOrderDetailID', 'Data Table 1.TrackingNumber', 'Data Table 1.OrderQty', 'PID', 'SpecialOfferID', and 'Data Table 1.Modified Date'. The second row shows the target fields: 'Data Table 1.SalesOrderID', 'Data Table 1.SalesOrderDetailID', 'Data Table 1.TrackingNumber', 'Data Table 1.OrderQty', 'PID', 'SpecialOfferID', and 'Data Table 1.Modified Date'. The third row shows the target fields again: 'Data Table 2.SalesOrderID', 'Data Table 2.SalesOrderDetailID', 'Data Table 2.TrackingNumber', 'Data Table 2.OrderQty', 'PID', 'SpecialOfferID', and 'Data Table 2.Modified Date'. At the bottom, there's a note: 'Select an action (concatenation or join)' and a 'Show data preview' button.

3. You can now do the following as required:

3 Loading and managing data with Data Manager

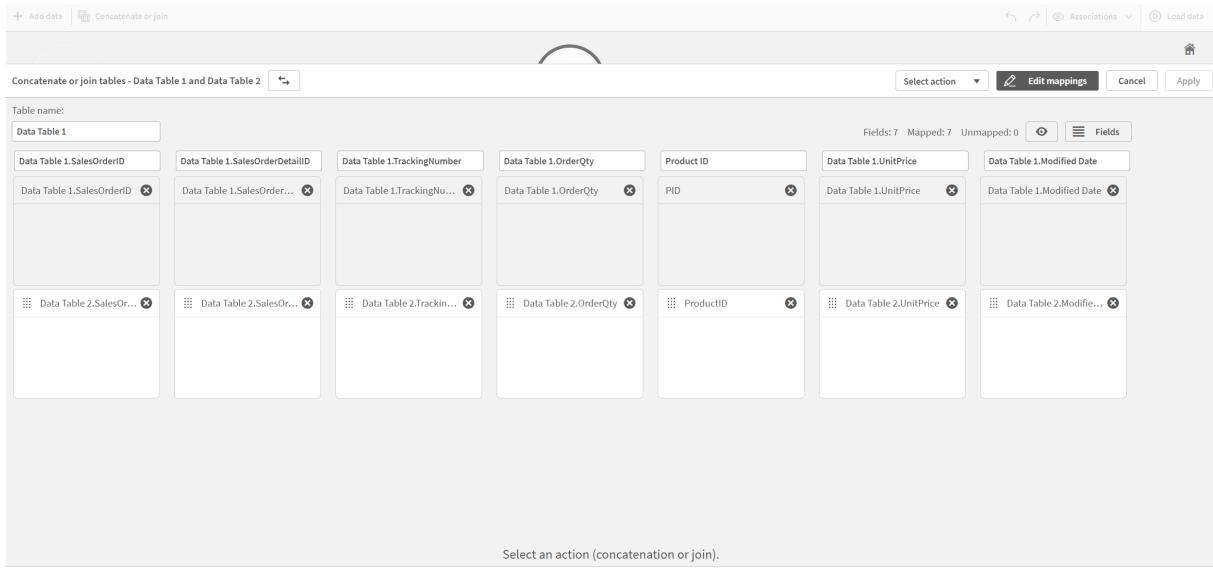
- i. In **Table name**, rename the table that will be created when you combine the tables.
- ii. Combine fields by dragging and dropping field labels.
- iii. Rename fields.
- iv. Delete fields by clicking  for the field.



In our example, we did the following:

- i. Renamed our table to Data Table.
- ii. Dragged the ProductID label and field to below the PID field, to combine the fields.
- iii. Renamed the PID field to Product ID.
- iv. Deleted the SpecialOfferID field.

Our table now looks like this:



4. In the **Selection action** drop-down, click **Concatenate**, and then click **Apply**. The tables are concatenated on the mapped fields. The * indicates that the data has not yet been loaded into the app.

3 Loading and managing data with Data Manager



5. Click **Load data**. A message is displayed indicating that the data was loaded successfully. Click **Edit sheet** to create visualizations using the data.

A step further - adding a new table and concatenating the data fields

The sample data provided above was pasted into two tabs in the same Excel file. However, the tables do not need to be in same file when you want to concatenate fields. The tables can be in separate files that are added to the app. Another table can also be added later, for example if you initially add a table from June, and then later want to add a second table from July.

In this example, we add another table with similar fields to the concatenated table we created above.

Here is the sample data. We named the tab that contains the table *DataTable_Newest*. We named the data file *Concatenate_Data2.xlsx*.

DataTable_Newest

DataTable_Newest

SalesOrderID	SalesOrderDetailID	TrackingNumber	ZIP	OrderQty	ID	UnitPrice	ModifiedDate
43666	34	568E-472E-9C	20012	3	764	377.51301	7/12/2013 0:00
43666	34	568E-472E-9C	23220	5	770	377.51301	7/12/2013 0:00
43667	35	AB6C-4FF9-9D	30004	2	730	165.54438	7/13/2013 0:00
43668	36	AB6C-4FF9-9D	11215	4	754	787.3146	7/14/2013 0:00
43668	36	AB6C-4FF9-9D	55401	3	725	165.54438	7/14/2013 0:00

3 Loading and managing data with Data Manager

SalesOrderID	SalesOrderDetailID	TrackingNumber	ZIP	OrderQty	ID	UnitPrice	ModifiedDate
43668	36	AB6C-4FF9-9D	20003	5	762	377.51301	7/14/2013 0:00
43668	36	AB6C-4FF9-9D	15213	3	765	377.51301	7/14/2013 0:00
43669	37	C618-4998-BE	33125	2	768	377.51301	7/15/2013 0:00
43669	37	C618-4998-BE	11215	1	753	1932.2658	7/15/2013 0:00

Do the following:

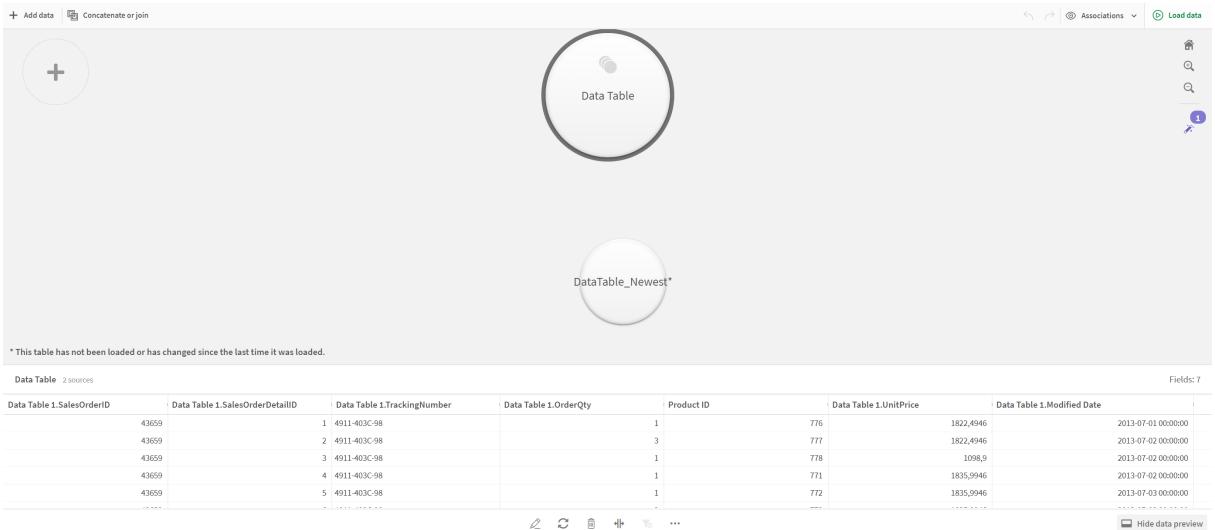
1. From the Qlik Sense hub, click the app that you created in the procedures above. The app opens.
2. Select **Data manager** from the drop-down list in the top toolbar. The **Data manager** opens and the table you created in the procedure above is shown.

The screenshot shows the Qlik Sense Data Manager interface. At the top, there are tabs for 'Add data' (selected), 'Concatenate or join', 'Associations', and 'Load data'. Below the tabs, there's a toolbar with icons for search, refresh, and other operations. On the left, there's a circular 'Add data' button with a plus sign. In the center, there's a circular icon containing a smaller circle with a gear-like pattern, labeled 'Data Table'. At the bottom, a data preview table is displayed with the following data:

Data Table 1.SalesOrderID	Data Table 1.SalesOrderDetailID	Data Table 1.TrackingNumber	Data Table 1.OrderQty	Product ID	Data Table 1.UnitPrice	Data Table 1.Modified Date
43659	1	4911-403C-98		1	776	1822.4946
43659	2	4911-403C-98		3	777	1822.4946
43659	3	4911-403C-98		1	778	1098.9
43659	4	4911-403C-98		1	771	1835.9946
43659	5	4911-403C-98		1	772	1835.9946

3. Click the **+** button to add data.
4. Add the new Excel file to the app by dragging it into the **Attach files to this app** dialogue. The **Add data** window opens.
5. Click **Add data** to add the data table to the app.
The new table is added to your app.

3 Loading and managing data with Data Manager



6. You can now concatenate the tables, edit the mappings, and then load the data.

3.17 Synchronizing scripted tables in Data manager

By default, scripted tables added in the data load editor cannot use the tools available in **Data manager**.

For example, you cannot associate scripted tables with other tables in **Data manager** or transform fields in scripted tables using the data profiling cards. If you synchronize your scripted tables in **Data manager**, you can replace your scripted tables in **Data manager** with managed scripted tables. These tables have access to all the same tools as tables added in **Data manager**, including:

- Editing tables, such as adding calculated fields.
- Transforming fields, such as changing the field types or transforming fields with data profiling cards.
- Transforming tables, such as unpivoting or concatenating tables.

Synchronization and managed scripted tables have the following limitations:

- Scripted tables must be located before the **Auto-generated section** in the data load script to be synchronized as managed scripted tables. Tables after the **Auto-generated section** in the data load script will not be synchronized.
- You cannot use **Select data from source** to change the selection of fields in a managed scripted table.



*Do not synchronize your scripted tables if your data load script contains an **Exit** statement or dynamic fields.*

To convert your scripted tables into managed scripted tables, synchronize your scripted tables in **Data manager**. Synchronization does the following:

- Replaces all synchronized scripted tables as managed scripted tables.
- Deletes any managed scripted tables whose scripted tables have been removed in the data load script.

3 Loading and managing data with Data Manager

- Updates any managed scripted tables whose source tables were changed in the data load script.



*If you have synchronized tables, you should not make changes in the data load editor with **Data manager** open in another tab.*



*Avoid changing the data load script for tables already synchronized in **Data manager**. If you remove or modify fields in data load editor, you must delete or redo any derived fields or associations in the synchronized table. Derived fields using a removed or modified field, such as a calculated field or fields created by the **Split** card, display null values.*

After synchronization, you can use the managed scripted tables in **Data manager** like any other table. **Data manager** prompts you to synchronize again if it detects differences between a managed scripted table and the source scripted table.

To change managed scripted tables back into scripted tables, delete them in **Data manager**. You must repeat the deletion if you synchronize again.

Synchronizing scripted tables

Do the following:

- In **Data manager**, click **...**.
Alternatively, select a scripted table.
- Click **Synchronize scripted tables**.

Managed scripted tables replace all the scripted tables in **Data manager**.

Removing managed scripted tables

Do the following:

- In **Data manager**, select **Tables** view.
- On the managed scripted table you want to remove, click .
- Click **Load data**.

The managed scripted table changes back to a scripted table.

3.18 Managing data associations

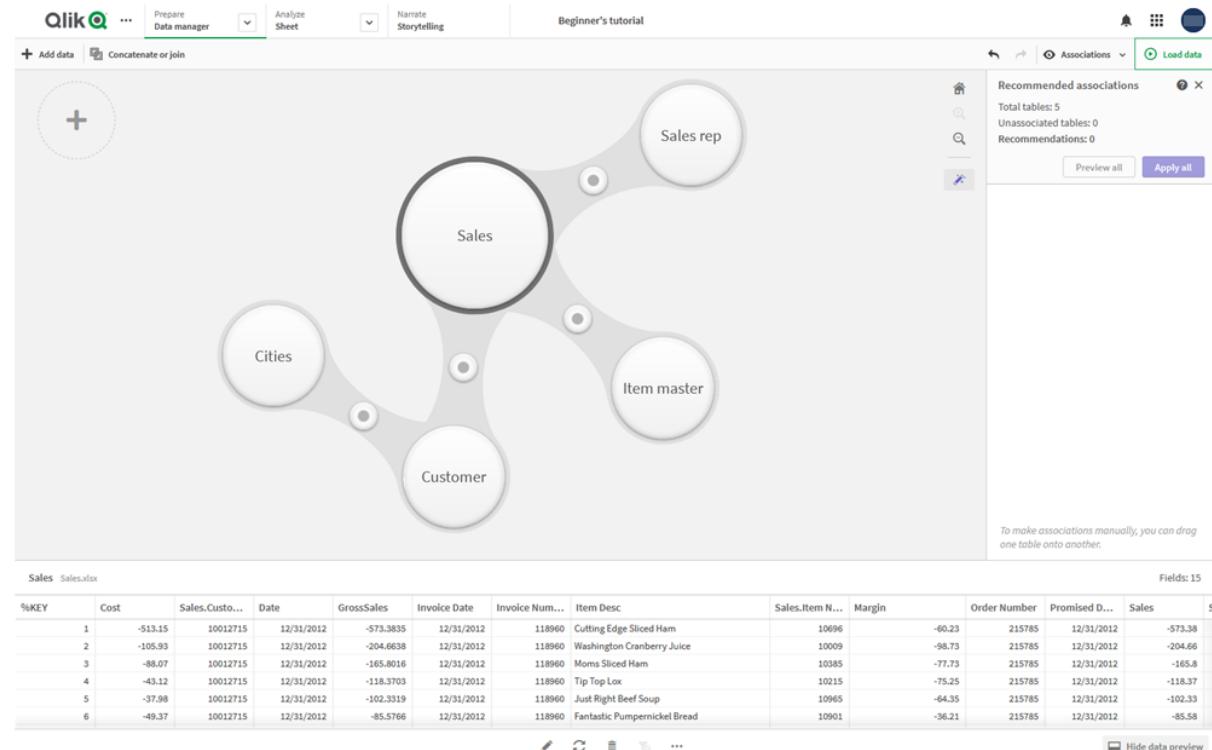
Qlik Sense can profile your data to help you create associations between tables. You can make associations by choosing from Qlik Sense Insight Advisor's analysis-based recommendations, or you can create your own.

3 Loading and managing data with Data Manager

If you want to associate your data, we recommend that you use the **Add data** option with data profiling enabled. This is the default option. You can verify this setting by clicking ******* beside the **Add data** button in the lower right corner of the Add Data page.

In the **Associations** view of the **Data manager**, your data is illustrated using bubbles, with each bubble representing a data table. The size of the bubble represents the amount of data in the table. The links between the bubbles represent the associations between tables. If there is an association between two tables, you can click the button in the link to view or edit the association.

Associations view in Data manager



*In most cases it is easier to edit table associations in the model view, but you can also edit a single table's associations using the **Associate** option in table edit view.*

For more information, see [Associating data in the table editor \(page 38\)](#).

Associating tables using the Recommended associations panel

In many cases, Qlik Sense Insight Advisor will recommend associations between data tables. The **Recommended associations** panel lets you view and apply these recommendations.

The **Recommended associations** panel will open by default if any tables are present. It can be closed by and re-opened by clicking on the .

3 Loading and managing data with Data Manager

If the panel is closed and recommendations exists, you will see a badge on top of the showing the number of recommendations.

Open recommended associations

* This table has not been loaded or has changed since the last time it was loaded.

Manager	Manager Nu...	Path	Sales Rep Name	Sales Rep Name1	Sales Rep Name2	Sales Rep Name3	Sales Rep ID
Amanda Honda		104 Amanda Honda	Amanda Honda	Amanda Honda			104
Amanda Honda		104 Amanda Honda-Amalia Craig	Amalia Craig	Amanda Honda	Amalia Craig		103
Amanda Honda		104 Amanda Honda-Cart Lynch	Cart Lynch	Amanda Honda	Cart Lynch		112
Amanda Honda		104 Amanda Honda-Molly McKenzie	Molly McKenzie	Amanda Honda	Molly McKenzie		159
Amanda Honda		104 Amanda Honda-Sheila Hein	Sheila Hein	Amanda Honda	Sheila Hein		176
Brenda Gibson		109 Brenda Gibson	Brenda Gibson	Brenda Gibson			109

Do the following:

1. Click in the upper right corner of the associations view, if the **Recommended associations** panel is closed.
The panel appears on the right.
2. You will see the following information:
 - **Total tables:** the total number of tables.
 - **Unassociated tables:** the total number of tables that have no associations.
 - **Recommendations:** the total number of recommended associations.
 - Recommended association details: showing the name of the recommended association, and then table and field names separated by colons
3. Click on a single recommendation to preview it in dark blue.
4. To accept only some of the recommendations, click the **Apply** button for the specific recommendation you need.
5. Click **Preview all** to see how all the recommended associations will affect your data tables.
Associations being previewed are highlighted.

3 Loading and managing data with Data Manager

Recommended associations panel

* This table has not been loaded or has changed since the last time it was loaded.

Manager	Manager Nu...	Path	Sales Rep Name	Sales Rep Name1	Sales Rep Name2	Sales Rep Name3	Sales Rep ID
Amanda Honda	104	Amanda Honda	Amanda Honda	Amanda Honda			104
Amanda Honda	104	Amanda Honda-Amalia Craig	Amalia Craig	Amanda Honda	Amalia Craig		103
Amanda Honda	104	Amanda Honda-Cart Lynch	Cart Lynch	Amanda Honda	Cart Lynch		112
Amanda Honda	104	Amanda Honda-Molly McKenzie	Molly McKenzie	Amanda Honda	Molly McKenzie		159
Amanda Honda	104	Amanda Honda-Sheila Hein	Sheila Hein	Amanda Honda	Sheila Hein		176
Brenda Gibson	109	Brenda Gibson	Brenda Gibson	Brenda Gibson			109

- Click **Apply all** to apply every recommended association. Associations that have been accepted are highlighted in light grey.

You can click at the bottom of the screen to see how your tables have changed.

You can now start making visualizations with your data.

Associating tables manually

You can associate tables manually, by dragging them together. When you drag table bubbles towards each other, they will be marked with a green, orange, or red stripe.

- Green: the **Data manager** is very confident about which fields to associate. For example, if two tables have fields labeled "Sales Region", the **Data manager** assumes they should be associated.
- Orange: the **Data manager** is fairly confident that these tables can be associated. For example, if two different fields have different labels, but contain single digit data, the **Data manager** will flag them as orange, because the data types are similar.
- Red: the **Data manager** does not know how to associate these tables. You will have to choose which tables and fields go together in the **Associate tables** editor.

3 Loading and managing data with Data Manager

I want to manually associate tables that are green or orange

Do the following:

1. Drag a table to one of the tables marked with green or orange.
2. The association is automatically applied.

The tables are now associated using the recommended fields.

I want to manually associate a tables that are red

You can associate the tables by creating a custom association.

Do the following:

1. Drag a table to one of the tables marked with red.
The **Associate tables** editor opens.
2. In the left table, select which fields to use in the association.
You choose a single field or multiple fields. You can also add delimiter characters to make it easier to interpret the data, or to match a field that already exists. You can see what the association data looks like in the preview.
3. In the right table, select which fields to use to match the selections you made in the left table.
4. Enter a name for the key field that will be created in **Name**.
This new field name cannot be the same as an existing field name in either table.
5. Click **Associate**.

The tables are now associated using your custom association.

Breaking associations

There are two ways of breaking associations that are not a good fit for your data model.

Do the following:

- Click one of the associated tables, and drag it away from the other table until the association breaks.
Or you can:
- Click on the link between the two bubbles, and then click the **Delete** button in the bottom panel.

The two tables are no longer associated.

Editing associations

You can edit an existing association between two tables if you need to adjust the data model.

Do the following:

1. Click the circle between the associated tables to open the data panel.
The panel opens with a preview of data in the associated fields.

2. Click .

You will see one or more buttons, each marked with green, orange, or red. Green means the **Data manager** is very confident in the association, orange means somewhat confident, and red means unsure. The current association is marked with grey.

3. Click one of the association buttons:

- Click a recommended association to select it.
- Click an existing custom association  to edit which fields to use in the association.
- Click **Custom association** to create a new association.

This button is only available if there is a recommended association for the table pair.
Custom associations can contain a single field or multiple fields.

You have now changed the association between the table pair.

Previewing data

You can preview tables in the associations view to get a better understanding of the data.

Do the following:

1. Select a table.
2. Click  at the bottom of the view.

The preview pane is displayed with the table data.

Synthetic keys

When two or more data tables have two or more fields in common, this suggests a composite key relationship. Qlik Sense handles this by creating synthetic keys automatically. These keys are anonymous fields that represent all occurring combinations of the composite key.

For more information, see *Synthetic keys (page 127)*.

If adding a table results in any of the following cases, you can only add data with profiling enabled:

- A synthetic key containing more than five fields is created.
- More than ten synthetic keys are created.
- Nested synthetic keys are created, that is, synthetic keys containing other synthetic keys.

These cases indicate that you need to adjust the data tables to resolve the issues.

Limitations

There are some cases where association recommendations are not provided, due to the structure of the loaded tables and the data in the tables. In these cases, you need to adjust the associations in the table editor.

- Many-to-many relationships.

3 Loading and managing data with Data Manager

- Field pairs with data does not match well in both directions. This may be the case when you have a small table with a few field values that match a field in a large table 100%, while the match in the other direction is significantly smaller.
- Compound key associations.

Additionally, the **Data manager** will only analyze tables that were added with **Add data**. Tables added using the data load script are not included in the association recommendations, unless they have been synchronized into **Data manager**.

For more information, see *Synchronizing scripted tables in Data manager (page 89)*.

Applying changes and reloading data

Changes that you have made in the **Data manager** will not be available in the app until you have reloaded data. When you reload data, the changes are applied and any new data that you have added is loaded from the external data sources. Data that you loaded previously is not reloaded.

You can reload all the data from the external data sources by using the  button in the **Data manager** footer.



The  button reloads all the data for the selected table. It does not reload all the data for all the tables in the app.

If the data in **Data manager** is out of sync with the app data, the **Load data** button is green. In the **Associations** view, all new or updated tables are indicated with *, and deleted tables are a lighter color of gray. In the **Tables** view, all new, updated, or deleted tables are highlighted in blue and display an icon that shows the status of the table:

- Tables marked with **Pending delete**  will be deleted.
- Tables marked with **Pending update**  will be updated with fields that have been added, renamed, or removed, or the table will be renamed.
- Tables marked with **Pending add**  will be added.

Applying changes

Do the following:

- Click **Load data** to load the changes in the app.

The app data is now updated with changes you made in **Data manager**.

4 Loading and transforming data with scripting

This introduction serves as a brief presentation of how you can load data into Qlik Sense using data load scripts.

Qlik Sense uses a data load script, which is managed in the data load editor, to connect to and retrieve data from various data sources. A data source can be a data file, for example an Excel file or a .csv file. A data source can also be a database, for example a Google BigQuery or Salesforce database.

You can also load data using the data manager, but when you want to create, edit and run a data load script you use the data load editor.

In the script, the fields and tables to load are specified. Scripting is often used to specify what data to load from your data sources. You can also manipulate the data structure by using script statements.

During the data load, Qlik Sense identifies common fields from different tables (key fields) to associate the data. The resulting data structure of the data in the app can be monitored in the data model viewer. Changes to the data structure can be achieved by renaming fields to obtain different associations between tables.

After the data has been loaded into Qlik Sense, it is stored in the app.

Analysis in Qlik Sense always happens while the app is not directly connected to its data sources. So, to refresh the data, you need to run the script to reload the data.

4.1 Interaction between Data manager and the data load script

When you add data tables in the **Data manager**, data load script code is generated. You can view the script code in the **Auto-generated section** of the data load editor. You can also choose to unlock and edit the generated script code, but if you do, the data tables will no longer be managed in the **Data manager**.

By default, data tables defined in the load script are not managed in **Data manager**. That is, you can see the tables in the data overview, but you cannot delete or edit the tables in **Data manager**, and association recommendations are not provided for tables loaded with the script. If you synchronize your scripted tables with **Data manager**, however, your scripted tables are added as managed scripted tables to **Data manager**.



*If you have synchronized tables, you should not make changes in the data load editor with **Data manager** open in another tab.*

You can add script sections and develop code that enhances and interacts with the data model created in **Data manager**, but there are some areas where you need to be careful. The script code you write can interfere with the **Data manager** data model, and create problems in some cases, for example:

- Renaming or dropping tables added with **Data manager** in the script.
- Dropping fields from tables added with **Data manager**.
- Concatenation between tables added with **Data manager** and tables loaded in the script.

4 Loading and transforming data with scripting

- Using the **Qualify** statement with fields in tables added with **Data manager**.
- Loading tables added with **Data manager** using **Resident** in the script.
- Adding script code after the generated code section. The resulting changes in the data model are not reflected in **Data manager**.

4.2 Using the data load editor

Create and run data load scripts from the data load editor, and create connections to data sources. When you have loaded the data it is available to the app for analysis.

Click **Data load editor** under the **Prepare** tab in the navigation bar to open the data load editor.

Data load editor



Toolbar

Toolbar with the most frequently used commands for the data load editor.

Data connections

Under **Data connections** you can save shortcuts to the data sources (databases or remote files) you commonly use. This is also where you initiate selection of which data to load.

Text editor

You can write and edit the script code in the text editor. Each script line is numbered and the script is color coded by syntax components. The text editor toolbar contains commands for **Search and replace**, **Help mode**, **Undo**, and **Redo**. The initial script contains some pre-defined regional variable settings, for example **SET ThousandSep=**, that you can optionally edit.

Sections

Divide your script into sections to make it easier to read and maintain. The sections are executed from top to bottom.

If you have added data using **Add data**, you will have a data load script section named **Auto-generated section** that contains the script code required to load the data.

Output

Output displays the autosave status and all messages that are generated during script execution.

Connect to data sources in the data load editor

Data connections in the data load editor let you save shortcuts to the data sources you commonly use: databases, local files, or remote files.

Data connections lists the connections you have saved in alphabetical order. You can use the search box to narrow the list down to connections with a certain name or type.



You can only see data connections that you own, or have been given rights to access. Please contact your Qlik Sense system administrator to acquire access if required.

Creating a new data connection

Do the following:

1. Click **Create new connection**.
2. Select the type of data source you want to create from the drop-down list.
The settings dialog, specific for the type of data source you selected, opens.
3. Enter the data source settings and click **Create** to create the data connection.
The connection name will be appended with your user name and domain to ensure that it is unique.

The data connection is now created with you as the default owner. If you want other users to be able to use the connection in a server installation, you need to edit the access rights of the connection in the Qlik Management Console.



The settings of the connection you created will not be automatically updated if the data source settings are changed. This means you need to be careful about storing user names and passwords, especially if you change settings between integrated Windows security and database logins in the DSN.



*If **Create new connection** is not displayed, it means you do not have access rights to add data connections. Please contact your Qlik Sense system administrator to acquire access if required.*

Deleting a data connection

Do the following:

1. Click on the data connection you want to delete.
2. Confirm that you want to delete the connection.

The data connection is now deleted.



If is not displayed, it means you do not have access rights to delete the data connection. Please contact your Qlik Sense system administrator to acquire access if required.

Editing a data connection

Do the following:

1. Under **Data connections**, select the space containing the data connection you want to edit.
2. Click on the data connection you want to edit.
3. Edit the data connection details. Connection details are specific to the type of connection. You may need to provide the connection's credentials.

The data connection is now updated.



If you edit the name of a data connection, you also need to edit all existing references (lib://) to the connection in the script, if you want to continue referring to the same connection.



If is not displayed, it means you do not have access rights to update the data connection. Please contact your Qlik Sense system administrator if required.

Inserting a connect string

Connect strings are required for most connections. Only folder and web file connections do not require connect strings.



This functionality is not available in Qlik Sense SaaS.

Do the following:

- Click on the connection for which you want to insert a connect string.

A connect string for the selected data connection is inserted at the current position in the data load editor.

Selecting data from a data connection

If you want to select data from a data connection to load in your app, do the following:

1. **Create new connection** linking to the data source (if the data connection does not already exist).
2. Select data from the connection.

Referring to a data connection in the script

You can use the data connection to refer to data sources in statements and functions in the script, typically where you want to refer to a file name with a path.

The syntax for referring to a file is 'lib://(connection_name)/(file_name_including_path)'

Example: Loading a file from a DataFiles connection

This example loads the file *orders.csv* from the location defined in the Folder data connection. This may be, for example, a folder that your administrator creates on the Qlik Sense server.

```
LOAD * FROM 'lib://datasource/orders.csv';
```

Example: Loading a file from a sub-folder

This example loads the file *Customers/cust.txt* from the DataSource data connection folder. Customers is a sub-folder in the location defined in data connection.

```
LOAD * FROM 'lib://DataSource/Customers/cust.txt';
```

Example: Loading from a web file

This example loads a table from the PublicData web file data connection, which contains the link to the actual URL.

```
LOAD * FROM 'lib://PublicData' (html, table is @1);
```

Example: Loading from a database

This example loads the table *Sales_data* from the DataSource database connection.

```
LIB CONNECT TO 'DataSource';
LOAD *;
SQL SELECT * FROM `sales_data`;
```

Where is the data connection stored?

Connections are stored using the Qlik Sense Repository Service. You can manage data connections with the Qlik Management Console in a Qlik Sense server deployment. The Qlik Management Console allows you to delete data connections, set access rights and perform other system administration tasks.



In Qlik Sense Desktop, all connections are saved in the app without encryption. This includes possible details about user name, password, and file path that you have entered when creating the connection. This means that all these details may be available in plain text if you share the app with another user. You need to consider this while designing an app for sharing.

Select data in the data load editor

You can select which fields to load from files or database tables and which views of the data source you want by using **Select data** in the data load editor.

As well as selecting fields, you can also rename fields in the dialog. When you have finished selecting fields, you can insert the generated script code into your script.

4 Loading and transforming data with scripting

Some data sources, such as a CSV file, contain a single table, while other data sources, such as Microsoft Excel spreadsheets or databases can contain several tables.



*Do not add a table in **Data load editor** that has already been added as a scripted table with the same name and same columns in **Data manager**.*

You open **Select data** by clicking  on a data connection in the data load editor.

Selecting data from a database

When selecting data from a database, the data source can contain several tables.

Do the following:

1. Open the **Data load editor**.
2. Under **Data connections** on the left, click  on a database connection.
The select data dialog is displayed.
3. Select a **Database** from the drop-down list.
Some selection dialogs do not have a **Database** drop-down list because the database name is entered when the connection is configured.
4. Select **Owner** of the database.
The list of **Tables** is populated with views and tables available in the selected database.
Some databases do not require that owners be specified in the data selection process.
5. Select a table.
6. Select the fields you want to load by checking the box next to each field you want to load.
You can select all fields in the table by checking the box next to the table name.



You can edit the field name by clicking on the existing field name and typing a new name. This may affect how the table is linked to other tables, as they are joined on common fields by default.

7. Select additional tables if you want to add data from them.



You cannot rename fields in the data selection wizard at the same time as you filter for fields by searching. You have to erase the search string in the text box first.



It is not possible to rename two fields in the same table so that they have identical names.

8. When you have finished your data selection, do the following:
 - Click **Insert script**.
The data selection window is closed, and the LOAD /SELECT statements are inserted in the script in accordance with your selections.

4 Loading and transforming data with scripting

Selecting data from a Microsoft Excel spreadsheet

When you select data from a Microsoft Excel spreadsheet, the file can contain several sheets. Each sheet is loaded as a separate table. An exception is if the sheet has the same field/column structure as another sheet or loaded table, in which case, the tables are concatenated.

Do the following:

1. Click  on the appropriate folder connection in the data load editor.
The select file dialog is displayed.
2. Select a file from the list of files accessible to this folder connection.
3. Select the first sheet to select data from. You can select all fields in a sheet by checking the box next to the sheet name.
4. Make sure you have the appropriate settings for the sheet:

Settings to assist you with interpreting the table data correctly

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names . Typically in an Excel spreadsheet, the first row contains the embedded field names. If you select No field names , fields will be named A,B,C...
Header size	Set to the number of rows to omit as table header, typically rows that contain general information that is not in a columnar format.

Example

My spreadsheet looks like this:

Spreadsheet example

Machine:	AEJ12B	-	-
Date:	2015-10-05 09	-	-
Timestamp	Order	Operator	Yield
2015-10-05 09:22	00122344	A	52
2015-10-05 10:31	00153534	A	67
2015-10-05 13:46	00747899	B	86

In this case you probably want to ignore the two first lines, and load a table with the fields Timestamp, Order, Operator, and Yield. To achieve this, use these settings:

4 Loading and transforming data with scripting

Settings to ignore the two first lines and load the fields

UI item	Description
Header size	2 This means that the first two lines are considered header data and ignored when loading the file. In this case, the two lines starting with Machine: and Date: are ignored, as they are not part of the table data.
Field names	Embedded field names. This means that the first line that is read is used as field names for the respective columns. In this case, the first line that will be read is the third line because the two first lines are header data.

5. Select the fields you want to load by checking the box next to each field you want to load.



You can edit the field name by clicking on the existing field name and typing a new name. This may affect how the table is linked to other tables, as they are joined by common fields by default.

6. When you are done with your data selection, do the following:

- Click **Insert script**.

The data selection window is closed, and the LOAD /SELECT statements are inserted in the script in accordance with your selections.



*You can also use a Microsoft Excel file as data source using the ODBC interface. In that case you need to use an **ODBC** data connection instead of a **All files** data connection.*

Selecting data from a table file

You can select data from a large number of data files:

- Text files, where data in fields is separated by delimiters such as commas, tabs or semicolons (comma-separated variable (CSV) files).
- HTML tables.
- XML files.
- KML files.
- Qlik native QVD and QVX files.
- Fixed record length files.
- DIF files (Data Interchange Format).

Do the following:

1. Click on the appropriate folder connection in the data load editor.
The select file dialog is displayed.
2. Select a file from the list of files accessible to this folder connection.

4 Loading and transforming data with scripting

3. Make sure that the appropriate file type is selected in **File format**.
4. Make sure you have the appropriate settings for the file. File settings are different for different file types.
5. Select the fields you want to load by checking the box next to each field you want to load. You can also select all fields in a file by checking the box next to the sheet name.



You can edit the field name by clicking on the existing field name and typing a new name. This may affect how the table is linked to other tables, as they are joined by common fields by default.

6. When you are done with your data selection, click **Insert script**.
7. The data selection window is closed, and the LOAD /SELECT statements are inserted in the script in accordance with your selections.



Users with edit permissions in a space can read, write, and load DataFiles in that space. Other users will not see the DataFiles.

Choosing settings for file types

Delimited table files

These settings are validated for delimited table files, containing a single table where each record is separated by a line feed, and each field is separated with a delimited character, for example a CSV file.

File format settings for delimited table files

UI item	Description
File format for delimited table files	Set to Delimited or Fixed record . When you make a selection, the select data dialog will adapt to the file format you selected.
Field names	Set to specify if the table contains Embedded field names or No field names .
Delimiter	Set the Delimiter character used in your table file.
Quoting	Set to specify how to handle quotes: None = quote characters are not accepted Standard = standard quoting (quotes can be used as first and last characters of a field value) MSQ = modern-style quoting (allowing multi-line content in fields)
Header size	Set the number of lines to omit as table header.
Character set	Set character set used in the table file.

4 Loading and transforming data with scripting

UI item	Description
Comment	Data files can contain comments between records, denoted by starting a line with one or more special characters, for example //. Specify one or more characters to denote a comment line. Qlik Sense does not load lines starting with the character(s) specified here.
Ignore EOF	Select Ignore EOF if your data contains end-of-file characters as part of the field value.

Fixed record data files

Fixed record data files contain a single table in which each record (row of data) contains a number of columns with a fixed field size, usually padded with spaces or tab characters.

You can set the field break positions in two different ways:

- Manually, enter the field break positions separated by commas in **Field break positions**. Each position marks the start of a field.

Example: 1,12,24

- Enable **Field breaks** to edit field break positions interactively in the field data preview. **Field break positions** is updated with the selected positions. You can:
 - Click in the field data preview to insert a field break.
 - Click on a field break to delete it.
 - Drag a field break to move it.

File format settings for fixed record data files

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names .
Header size	Set Header size to the number of lines to omit as table header.
Character set	Set to the character set used in the table file.
Tab size	Set to the number of spaces that one tab character represents in the table file.
Record line size	Set to the number of lines that one record spans in the table file. Default is 1.
Ignore EOF	Select Ignore EOF if your data contains end-of-file characters as part of the field value.

HTML files

HTML files can contain several tables. Qlik Sense interprets all elements with a <TABLE> tag as a table.

File format settings for HTML files

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names .
Character set	Set the character set used in the table file.

XML files

You can load data that is stored in XML format.

There are no specific file format settings for XML files.

QVD files

You can load data that is stored in QVD format. QVD is a native Qlik format and can only be written to and read by Qlik Sense or QlikView. The file format is optimized for speed when reading data from a Qlik Sense script but it is still very compact.

There are no specific file format settings for QVD files.

QVX files

You can load data that is stored in Qlik data eXchange (QVX) format. QVX files are created by custom connectors developed with the Qlik QVX SDK.

There are no specific file format settings for QVX files.

KML files

You can load map files that are stored in KML format, to use in map visualizations.

There are no specific file format settings for KML files.

Previewing scripts

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Including LOAD statements

If **Include LOAD statement** is selected, SELECT statements are generated with preceding LOAD statements using the SELECT statements as input.



If you rename fields in a table, a LOAD statement will be inserted automatically regardless of this setting.

Inline loads

You can load data in the data load editor using an inline load. A basic inline load creates a table, and inserts the data fields and records. For example:

```
MyTable:  
Load * Inline [  
Country, Year, Sales  
Argentina, 2014, 66295.03  
Argentina, 2015, 140037.89  
Austria, 2014, 54166.09  
Austria, 2015, 182739.87  
];
```

The following syntax is used for the above inline load:

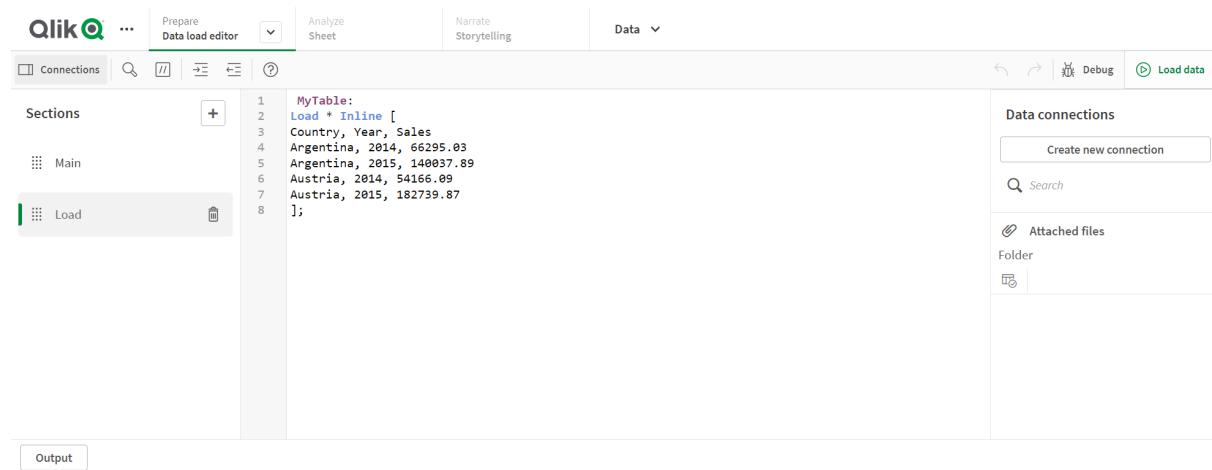
4 Loading and transforming data with scripting

- *MyTable*: creates a table for the data.
- *Load * Inline* indicates an inline (in the data load editor) data load.
- Square brackets enclose the data.
- The first line of the load statement defines the data fields.
- Commas separate data fields and records.
- A semi-colon closes the load statement.

The data is loaded using the **Load data** button.

The following image shows an inline load in a new script section called *Load*.

Inline load in data load editor



Edit the data load script

You write the script in the text editor of the **Data load editor**. Here you can make manual changes to the **LOAD** or **SELECT** statements you have generated when selecting data, and type new script.

The script, which must be written using the Qlik Sense script syntax, is color coded to make it easy to distinguish the different elements. Comments are highlighted in green, whereas Qlik Sense syntax keywords are highlighted in blue. Each script line is numbered.

There are a number of functions available in the editor to assist you in developing the load script, and they are described in this section.

Accessing syntax help for commands and functions

There are several ways to access syntax help for a Qlik Sense syntax keyword.

Accessing the help portal

You can access detailed help in the help portal in two different ways.

4 Loading and transforming data with scripting

- Click  in the toolbar to enter syntax help mode. In syntax help mode you can click on a syntax keyword (marked in blue and underlined) to access syntax help.
- Place the cursor inside or at the end of the keyword and press Ctrl+H.



You cannot edit the script in syntax help mode.

Using the auto-complete function

If you start to type a Qlik Sense script keyword, you get an auto-complete list of matching keywords to select from. The list is narrowed down as you continue to type, and you can select from templates with suggested syntax and parameters. A tooltip displays the syntax of the function, including parameters and additional statements, as well as a link to the help portal description of the statement or function.



You can also use the keyboard shortcut Ctrl+Space to show the keyword list, and Ctrl+Shift+Space to show a tooltip.

Inserting a prepared test script

You can insert a prepared test script that will load a set of inline data fields. You can use this to quickly create a data set for test purposes.

Do the following:

- Press Ctrl + 00.

The test script code is inserted into the script.

Indenting code

You can indent the code to increase readability.

Do the following:

1. Select one or several lines to change indentation.
2. Click  to indent the text (increase indentation) or, click  to outdent the text (decrease indentation).



You can also use keyboard shortcuts:

Tab (indent)

Shift+Tab (outdent)

Searching and replacing text

You can search and replace text throughout script sections.

Searching for text

Open the data load editor. Do the following:

1. Click  in the toolbar.
The search drop-down dialog is displayed.
2. In the search box, type the text you want to find.
The search results are highlighted in the current section of the script code. Also, the number of text instances found is indicated next to the section label.
3. You can browse through the results by clicking  and .
4. Click  in the toolbar to close the search dialog.



Also, you can select **Search in all sections** to search in all script sections. The number of text instances found is indicated next to each section label. You can select **Match case** to make case sensitive searches.

Replacing text

Do the following:

1. Click  in the toolbar.
The search drop-down dialog is displayed.
2. Type the text you want to find in the search box.
3. Type the replacement text in the replace box and click **Replace**.
4. Click  to find next instance of search text and do one of the following:
 - Click **Replace** to replace text.
 - Click  to find next.
5. Click  in the toolbar to close the search dialog.



You can also click **Replace all in section** to replace all instances of the search text in the current script section. The replace function is case sensitive, and replaced text will have the case given in the replace field. A message is shown with information about how many instances that were replaced.

Commenting in the script

You can insert comments in the script code, or deactivate parts of the script code by using comment marks. All text on a line that follows to the right of // (two forward slashes) will be considered a comment and will not be executed when the script is run.

The data load editor toolbar contains a shortcut for commenting or uncommenting code. The function works as a toggle. That is, if the selected code is commented out it will be commented, and vice versa.

Commenting

Do the following:

1. Select one or more lines of code that are not commented out, or place the cursor at the beginning of a line.

2. Click **//**, or press Ctrl + K.

The selected code is now commented out.

Uncommenting

Do the following:

1. Select one or more lines of code that are commented out, or place the cursor at the beginning of a commented line.
2. Click **//**, or press Ctrl + K.

The selected code will now be executed with the rest of the script.



There are more ways to insert comments in the script code:

- Using the **Rem** statement.
- Enclosing a section of code with /* and */.

Example:

```
Rem This is a comment ;  
  
/* This is a comment  
   that spans two lines */  
  
// This is a comment as well
```

Selecting all code

You can select all code in the current script section.

Do the following:

- Press Ctrl + A.

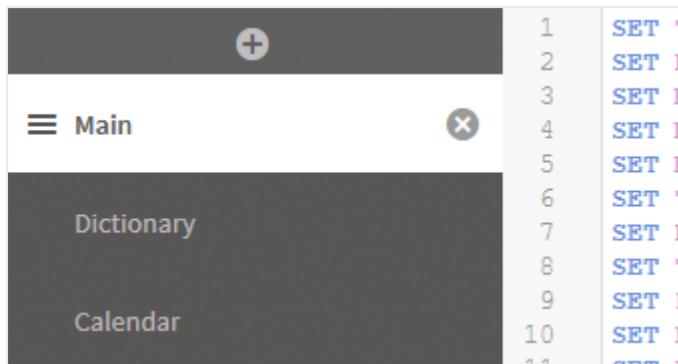
All script code in the current section is selected.

Organizing the script code

You can divide your script into sections to organize the structure. The script is executed in the order of the sections from top to bottom. A script must contain at least one section.

4 Loading and transforming data with scripting

Sections Main, Dictionary and Calendar in the Data load editor.



If you have added data using **Add data**, you will have a data load script section named **Auto-generated section** that contains the script code required to load the data.

Working with script sections

There are several ways to organize scripting sections:

- Click to insert new script sections to organize your code. The new section is inserted after the currently selected section.
- Click next to the section tab to delete it. You need to confirm the deletion.



Deleting a script section cannot be undone.

- Click on the section name and type to edit the name. Press Enter or click outside the section when you are finished.
- Put the cursor on and drag the section to rearrange the order.

Debug the data load script

You can use the debugging utilities in the Data load editor to step through the execution of your script by using breakpoints, which enable you to inspect variable values and output from the script execution.

You can select whether you want to view any or all of **Output**, **Variables** and **Breakpoints**.

To show the debug panel, do the following:

- Click **Debug** in the data load editor toolbar.
The debug panel is opened at the bottom of the Data load editor.



You can't create connections, edit connections, select data, save the script or load data while you are running in debug mode. Debug mode begins with debug execution and continues until the script is executed or execution has been ended.

Debug toolbar

The Data load editor debug panel contains a toolbar with the following options to control the debug execution:

Debug toolbar options

UI item	Description
Limited load	Enable this to limit how many rows of data to load from each data source. This is useful to reduce execution time if your data sources are large. Enter the number of rows you want to load.
	 <i>This only applies to physical data sources. Automatically generated and Inline loads will not be limited, for example.</i>
▶	Start or continue execution in debug mode until the next breakpoint is reached.
▶▶	Step to the next line of code.
■	End execution here. If you end before all code is executed, the resulting data model will only contain data up to the line of code where execution ended.

Output

Output displays all messages that are generated during debug execution. You can select to lock the output from scrolling when new messages are displayed by clicking .

Additionally, the output menu  contains the following options:

Output menu options

UI item	Description
Clear	Click this to delete all output messages.
Select all text	Click this to select all output messages.
Scroll to bottom	Click this to scroll to the last output message.

Variables

Variables lists all reserved variables, system variables and variables defined in the script, and displays the current values during script execution.

Setting a variable as favorite

If you want to inspect specific variables during execution, you can set them as favorites. Favorite variables are displayed at the top of the variable list, marked by a yellow star. To set a variable as favorite, do the following:

- Click on the ★ next to a variable.
The ★ is now yellow, and the variable moved to the top of the variable list.

Filtering variables

You can apply a filter to show only a selected type of variables by using the following options in the variables menu ☰:

Variables menu options

UI item	Description
Show all variables	Click this to show all types of variables.
Show system variables	Click this to show system variables. System variables are defined by Qlik Sense, but you can change the variable value in the script.
Show reserved variables	Click this to show reserved variables. Reserved variables are defined by Qlik Sense and the value cannot be changed.
Show user defined variables	Click this to show user defined variables. User defined variables are variables that you have defined in the script.

Breakpoints

You can add breakpoints to your script to be able to halt debug execution at certain lines of code and inspect variable values and output messages at this point. When you have reached a breakpoint, you can choose to stop execution, continue until the next breakpoint is reached, or step to the next line of code. All breakpoints in the scripts are listed, with a reference to section and line number.

Adding a breakpoint

To add a breakpoint at a line of code , do one of the following:

- In the script, click in the area directly to the right of the line number where you want to add a breakpoint.
A ━ next to the line number will indicate that there is a breakpoint at this line.



You can add breakpoints even when the debug panel is closed.

Deleting breakpoints

You can delete a breakpoint by doing either of the following:

- In the script, click on a ━ next to the line number.
- In the list of breakpoints, click X next to a breakpoint.

You can also click ☰ and select **Delete all** to delete all breakpoints from the script.

Enabling and disabling breakpoints

When you create a breakpoint it is enabled by default, which is indicated by next to the breakpoint in the list of breakpoints. You can enable and disable individual breakpoints by selecting and deselecting them in the list of breakpoints.

You also have the following options in the breakpoints menu :

- **Enable all**
- **Disable all**

Saving the load script

When you save a script the entire app is saved, but data is not automatically reloaded.

When the script is saved, the app will still contain old data from the previous reload, which is indicated in the toolbar. If you want to update the app with new data, click in the data load editor toolbar. The script is automatically saved to the app when data is loaded.

Data load editor saves your work automatically as you make changes to your load script. You can force a save by pressing CTRL+S.



The script is not saved automatically in Qlik Sense Desktop. You need to save the script manually.

When you save a script, it is automatically checked for syntax errors. Syntax errors are highlighted in the code, and all script sections containing syntax errors are indicated with next to the section label.

Run the script to load data

Click in the toolbar to run the script and reload data in the app. The app is automatically saved before loading the data.

The **Data load progress** dialog is displayed, and you can **Abort** the load. When the data load has completed, the dialog is updated with status (**Finished successfully** or **Data load failed**) and a summary of possible errors and warnings, such as for synthetic keys. The summary is also displayed in **Output**, if you want to view it after the dialog is closed.



*If you want the **Data load progress** dialog to always close automatically after a successful execution, select **Close when successfully finished**.*

Keyboard shortcuts in the Data load editor

There are a number of keyboard shortcuts you can use to work effectively and easily in the **Data load editor** environment.

Keyboard shortcuts



Keyboard shortcuts are expressed assuming that you are working in Windows. For macOS use Cmd instead of Ctrl.

Keyboard shortcuts in the Data load editor

Shortcut	Action
Ctrl+0,Ctrl+0	Generates sample data.
Alt+1	Shows the Output panel or hides it if it is visible.
Alt+2	Shows the Variables panel or hides it if it is visible, if the debug tool is on.
Alt+3	Shows the Breakpoints panel or hides it if it is visible, if the debug tool is on.
Alt+F5	Shows the debug tools or hides them if they are visible.
Alt+F6	Runs debugging if the debug tool is on.
Alt+F7	Proceeds to the next step in the debugger if the debug tool is on.
Alt+F8	Stops the debugger if the debug tool is on.
F9	Toggles insertion and removal of a debug breakpoint.
Alt+F10	Shows the right panel or hides it if it is visible.
Alt+F11	Expands the script editor to full screen.
Ctrl+C	Copies the selected item to the clipboard.
Ctrl+F	Shows the Find search entry field or hides it if is open.
Ctrl+H	Opens online help in the context of the current selected function, while in the data load editor or the expression editor.
Ctrl+K	Toggles insertion and removal of a code comment line.
Ctrl+P	Prints the current view or active sheet/story.
Ctrl+S	Saves changes.
Ctrl+V	Pastes the most-recently copied item from the clipboard.
Ctrl+X	Cuts the selected item and copies it to the clipboard. When using the Google Chrome browser: if the cursor is put in front of a row in the data load editor or in the expression editor, without selecting anything, the entire row is cut.
Ctrl+Z	Undo action. Repeat to undo earlier actions.
Alt+Insert	Inserts a new section in the script.
Alt+PgUp	Navigates to the previous section.
Alt+PgDn	Navigates to the next section.

Shortcut	Action
Ctrl+Shift+Enter/Return	Reloads data.
Ctrl+Shift+Space	Displays a tooltip. (Not supported in Qlik Sense Desktop)
Ctrl+Space	Automatically completes with the auto-text string.
Enter/Return in search field	Searches for the next instance of the search string.
Enter/Return in replace field	Replaces the selected instance of the search string.
Esc in search or replace field	Closes the Find search entry field.
Shift+Tab	Outdents a line in the script.
Tab	Indents a line in the script.

4.3 Understanding script syntax and data structures

Extract, transform, and load

In general, the way you load data into the app can be explained by the extract, transform and load process:

- Extract

The first step is to extract data from the data source system. In a script, you use **SELECT** or **LOAD** statements to define this. The differences between these statements are:

- **SELECT** is used to select data from an ODBC data source or OLE DB provider. The **SELECT SQL** statement is evaluated by the data provider, not by Qlik Sense.
- **LOAD** is used to load data from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically.

- Transform

The transformation stage involves manipulating the data using script functions and rules to derive the desired data model structure. Typical operations are:

- Calculating new values
- Translating coded values
- Renaming fields
- Joining tables
- Aggregating values
- Pivoting
- Data validation

- Load

In the final step, you run the script to load the data model you have defined into the app.

Your goal should be to create a data model that enables efficient handling of the data in Qlik Sense. Usually this means that you should aim for a reasonably normalized star schema or snowflake schema without any circular references, that is, a model where each entity is kept in a separate table. In other words a typical data model would look like this:

- a central fact table containing keys to the dimensions and the numbers used to calculate measures (such as number of units, sales amounts, and budget amounts).
- surrounding tables containing the dimensions with all their attributes (such as products, customers, categories, calendar, and suppliers).



In many cases it is possible to solve a task, for example aggregations, either by building a richer data model in the load script, or by performing the aggregations in the chart expressions. As a general rule, you will experience better performance if you keep data transformations in the load script.



It's good practice to sketch out your data model on paper. This will help you by providing structure to what data to extract, and which transformations to perform.

Data loading statements

Data is loaded by **LOAD** or **SELECT** statements. Each of these statements generates an internal table. A table can always be seen as a list of data, each record (row) then being a new instance of the object type and each field (column) being a specific attribute or property of the object.

The differences between these statements are:

- **SELECT** is used to select data from an ODBC data source or OLE DB provider. The **SELECT SQL** statement is evaluated by the data provider, not by Qlik Sense.
- **LOAD** is used to load data from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically.

Rules

The following rules apply when loading data into Qlik Sense:

- Qlik Sense does not make any difference between tables generated by a **LOAD** or a **SELECT** statement. This means that if several tables are loaded, it does not matter whether the tables are loaded by **LOAD** or **SELECT** statements or by a mix of the two.
- The order of the fields in the statement or in the original table in the database is arbitrary to the Qlik Sense logic.
- Field names are used in the further process to identify fields and making associations. These are case sensitive, which often makes it necessary to rename fields in the script.

Execution of the script

For a typical **LOAD** or **SELECT** statement the order of events is roughly as follows:

1. Evaluation of expressions
2. Renaming of fields by **as**
3. Renaming of fields by **alias**
4. Qualification of field names
5. Mapping of data if field name matches
6. Storing data in an internal table

Fields

Fields are the primary data-carrying entities in Qlik Sense. A field typically contains a number of values, called field values. In database terminology we say that the data processed by Qlik Sense comes from data files. A file is composed of several fields where each data entry is a record. The terms file, field and record are equivalent to table, column and row respectively. The Qlik Sense AQL logic works only on the fields and their field values.

Field data is retrieved by script via **LOAD**, **SELECT** or **Binary** statements. The only way of changing data in a field is by re-executing the script. The actual field values can not be manipulated by the user from the layout or by means of automation. Once read into Qlik Sense they can only be viewed and used for logical selections and calculations.

Field values consist of numeric or alphanumeric (text) data. Numeric values actually have dual values, the numeric value and its current, formatted text representation. Only the latter is displayed in sheet objects etc.

The content of a field can be represented in a filter pane.

Derived fields

If you have a group of fields that are related, or if fields carry information that can be broken up into smaller parts that are relevant when creating dimensions or measures, you can create field definitions that can be used to generate derived fields. One example is a date field, from which you can derive several attributes, such as year, month, week number, or day name. All these attributes can be calculated in a dimension expression using Qlik Sense date functions, but an alternative is to create a calendar definition that is common for all fields of date type. Field definitions are stored in the data load script.



*Default calendar field definitions for Qlik Sense are included in autoCalendar for date fields loaded using **Data manager**. For more information, see Adding data to the app (page 16).*

Declare the calendar field definitions

You use the **Declare** statement to create a definition of the derived fields. This is where you define the different attributes of the field, in this case date related attributes. Each field is described as `<expression> As field_name tagged tag`. Setting one or more tags is optional, but it can affect the sort order of the derived field. Use \$1 to reference the data field from which the derived fields should be generated.



Do not use autoCalendar as name for calendar field definitions, as this name is reserved for auto-generated calendar templates.

4 Loading and transforming data with scripting

Calendar:

```
DECLARE FIELD DEFINITION TAGGED '$date'  
Parameters  
    first_month_of_year = 1  
Fields  
    Year($1) As Year Tagged ('$numeric'),  
    Month($1) as Month Tagged ('$numeric'),  
    Date($1) as Date Tagged ('$date'),  
    Week($1) as Week Tagged ('$numeric'),  
    Weekday($1) as weekday Tagged ('$numeric'),  
    DayNumberofYear($1, first_month_of_year) as DayNumberofYear Tagged ('$numeric');
```

Map data fields to the calendar with Derive

The next step is to use the **Derive** statement to map existing data fields to the calendar. This will create the derived fields. You can do this in three alternative ways in the data load script:

- Map specific fields by field name.
`DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING Calendar;`
- Map all fields with one or more specific field tags.
`DERIVE FIELDS FROM EXPLICIT TAGS ('$date') USING Calendar;`
- Map all fields that are tagged with one of the tags of the field definition (\$date in the example above).
`DERIVE FIELDS FROM IMPLICIT TAG USING Calendar;`

In this case, you could use any of the three examples here.

Use the derived date fields in a visualization

Qlik Sense is prepared to recognize derived date fields if you have created a calendar definition and mapped the fields like in the example here. They are available in the **Date & time fields** section of the **Fields** asset panel. You will also find all derived fields in the expression editor and when you create or edit dimensions.

Field tags

Field tags provide the possibility of adding metadata to the fields in your data model. There are two different types of field tags:

- System field tags
System field tags are generated automatically when the script is executed and data is loaded. Some of the tags can be manipulated in the script. System tags are always preceded by a \$ sign.
- Custom field tags
You can add custom tags to fields in the data load script, using the **Tag** statement. Custom tags may not use the same name as any system tag.

System field tags

The following system field tags are generated automatically when data is loaded.

4 Loading and transforming data with scripting

System field tags

Tag	Description	Can be manipulated in the script
\$system	System field that is generated by Qlik Sense during script execution.	No
\$key	Key field providing a link between two or more tables.	No
\$keypart	The field is part of one or more synthetic keys.	No
\$syn	Synthetic key	No
\$hidden	Hidden field, that is, it is not displayed in any field selection list when creating visualizations, dimensions or measures. You can still use hidden fields in expressions, but you need to type the field name. You can use the HidePrefix and HideSuffix system variables to set which fields to hide.	Yes
\$numeric	All (non-NULL) values in the field are numeric.	Yes
\$integer	All (non-NULL) values in the field are integers.	Yes
\$text	No values in the field are numeric.	Yes
\$ascii	Field values contain only standard ASCII characters.	Yes
\$date	All (non-NULL) values in the field can be interpreted as dates (integers).	Yes
\$timestamp	All (non-NULL) values in the field can be interpreted as time stamps.	Yes
\$geoname	Field values contain names of geographical locations, related to a point field (\$geopoint) and/or an area field (\$geomultipolygon).	Yes
\$geopoint	Field values contain geometry point data, representing points on a map in the format [longitude, latitude].	Yes
\$geomultipolygon	Field values contain geometry polygon data, representing areas on a map.	Yes

Derived field tags

The following tags can be used when you declare derived fields to specify how to use and display the fields on a contiguous axis in a line chart. You can manipulate the tags in the data load script.

4 Loading and transforming data with scripting

Derived field tags

Tag	Description
\$axis	The \$axis tag is used to specify that the field should generate a tick on the contiguous axis of the chart.
\$qualified \$simplified	You can specify a qualified and a simplified version of an axis label by deriving two different fields. The qualified field is displayed as label when the axis is zoomed to a deeper level, to show full context. For example, you can generate two fields when showing data by the quarter: A simplified field, with the \$simplified tag, showing quarter, like 'Q1' and a qualified field, with the \$qualified tag, showing year and quarter, like '2016-Q1'. When the time axis is zoomed out, the axis shows labels in two levels, for year (2016) and quarter (Q1), using the simplified field. When you zoom in, the axis shows labels for quarter and month, and the qualified field (2016-Q1) is used to provide full year context for the quarter.
\$cyclic	The \$cyclic tag is used for cyclic fields, for example quarter or month, with a dual data representation.

System fields

In addition to the fields extracted from the data source, system fields are also produced by Qlik Sense. These all begin with "\$" and can be displayed like ordinary fields in a visualization, such as a filter pane or a table. System fields are created automatically when you load data, and are primarily used as an aid in app design.

Available system fields

The following system fields are available:

Available system fields

System field	Description
\$Table	Contains all tables that are loaded.
\$Field	Contains all fields in the tables that are loaded.
\$Fields	Contains the number of fields in each table.
\$FieldNo	Contains the position of the fields in the tables.
\$Rows	Contains the number of rows in the tables

None of the system fields can be manipulated in the script.

Using system fields in a visualization

System field data is associated. For example, if you add two filter panes, one with \$Table and one with \$Field, if you select a table, the \$Field filter pane will show the fields in the selected table as possible values.

4 Loading and transforming data with scripting

System fields are not included in field lists in the assets panel. They are included in the expression editor. If you want to use a system field in the assets panel, you need to reference it by typing it manually.

Example: In a dimension in the assets panel

```
=$Field
```

Renaming fields

Sometimes it is necessary to rename fields in order to obtain the desired associations. The three main reasons for renaming fields are:

- Two fields are named differently although they denote the same thing:
 - The field *ID* in the *Customers* table
 - The field *CustomerID* in the *Orders* table

The two fields denote a specific customer identification code and should both be named the same, for example *CustomerID*.

- Two fields are named the same but actually denote different things:
 - The field *Date* in the *Invoices* table
 - The field *Date* in the *Orders* table

The two fields should preferably be renamed, to for example *InvoiceDate* and *OrderDate*.

- There may be errors such as misspellings in the database or different conventions on upper- and lowercase letters.

Since fields can be renamed in the script, there is no need to change the original data. There are two different ways to rename fields as shown in the examples.

Example 1: Using the alias statement

The **LOAD** or **SELECT** statement can be preceded by an **alias** statement.

```
Alias ID as CustomerID;  
LOAD * from Customer.csv;
```

Example 2: Using the as specifier

The **LOAD** or **SELECT** statement can contain the **as** specifier.

```
LOAD ID as CustomerID, Name, Address, zip, City, State from Customer.csv;
```

Logical tables

Each **LOAD** or **SELECT** statement generates a table. Normally, Qlik Sense treats the result of each one of these as one logical table. However, there are a couple of exceptions from this rule:

- If two or more statements result in tables with identical field names, the tables are concatenated and treated as one logical table.

4 Loading and transforming data with scripting

- If a **LOAD** or **SELECT** statement is preceded by any of the following qualifiers, data is altered or treated differently.

Logical tables

Table	Description
concatenate	This table is concatenated with (added to) another named table or with the last previously created logical table.
crosstable	This table is unpivoted. That is, it is converted from crosstable format to column format.
generic	This table is split into several other logical tables.
info	This table is not loaded as a logical table, but as an information table containing links to external information such as files, sounds, URLs, etc.
intervalmatch	The table (which must contain exactly two columns) is interpreted as numeric intervals, which are associated with discrete numbers in a specified field.
join	This table is joined by Qlik Sense with another named table or with the last previously created logical table, over the fields in common.
keep	This table is reduced to the fields in common with another named table or with the last previously created logical table.
mapping	This table (which must contain exactly two columns) is read as a mapping table, which is never associated with other tables.
semantic	This table is not loaded as a logical table, but as a semantic table containing relationships that should not be joined, e.g. predecessor, successor and other references to other objects of the same type.

When the data has been loaded, the logical tables are associated.

Table names

Qlik Sense tables are named when they are stored in the Qlik Sense database. The table names can be used, for example, for **LOAD** statements with a **resident** clause or with expressions containing the **peek** function, and can be seen in the **\$Table** system field in the layout.

Tables are named in accordance with the following rules:

1. If a label immediately precedes a **LOAD** or **SELECT** statement the label is used as table name. The label must be followed by a colon.

Example:

```
Table1:  
LOAD a,b from c.csv;
```

2. If no label is given, the file name or table name immediately following the keyword **FROM** in the **LOAD** or **SELECT** statement is used. A maximum of 32 characters is used. The extension is skipped if the file name is used.
3. Tables loaded inline are named INLINExx, where xx is a number. The first inline table will be given the name *INLINE01*.
4. Automatically generated tables are named AUTOGENERATExx, where xx is a number. The first autogenerated table is given the name *AUTOGENERATE01*.
5. If a table name generated according to the rules above should be in conflict with a previous table name, the name is extended with -x , where x is a number. The number is increased until no conflict remains. For example, three tables could be named *Budget*, *Budget-1* and *Budget-2*.

There are three separate domains for table names: **section access**, **section application** and mapping tables. Table names generated in **section access** and **section application** are treated separately. If a table name referenced is not found within the section, Qlik Sense searches the other section as well. Mapping tables are treated separately and have no connection whatsoever to the other two domains of table names.

Table labels

A table can be labeled for later reference, for example by a **LOAD** statement with a **resident** clause or with expressions containing the **peek** function. The label, which can be an arbitrary string of numbers or characters, should precede the first **LOAD** or **SELECT** statement that creates the table. The label must be followed by a colon ":".

Labels containing blanks must be quoted using single or double quotation marks or square brackets.

Example 1:

```
Table1:  
LOAD a,b from c.csv;  
LOAD x,y from d.csv where x=peek('a',y,'Table1');
```

Example 2: Table label containing a blank

```
[All Transactions]:  
SELECT * from Transtable;  
LOAD Month, sum(Sales) resident [All Transactions] group by Month;
```

Associations between logical tables

A database can have many tables. Each table can be considered as a list of something; each record in the list represents an instance of an object of some type.

Example:

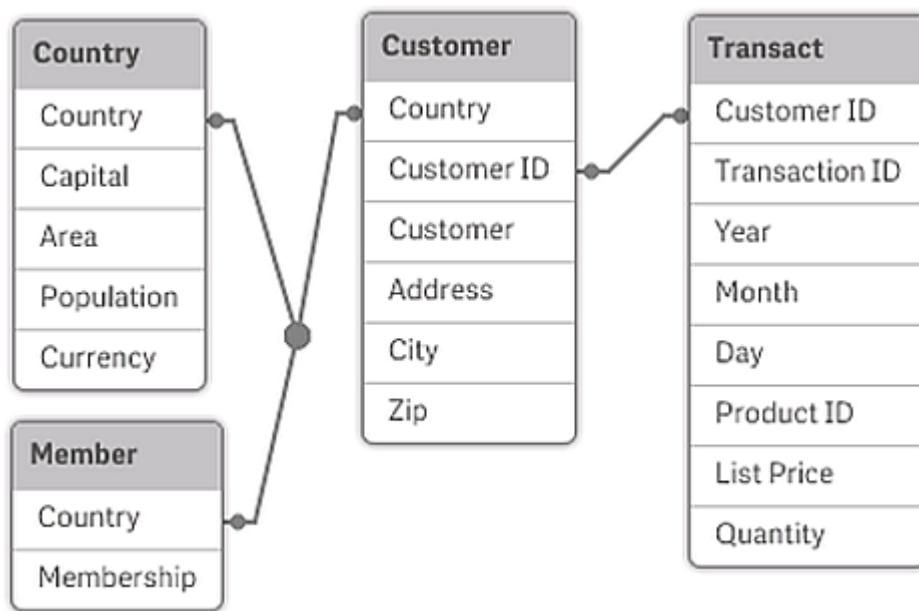
If two tables are lists of different things, for example if one is a list of customers and the other a list of invoices, and the two tables have a field such as the customer number in common, this is usually a sign that there is a relationship between the two tables. In standard SQL query tools the two tables should almost always be joined.

4 Loading and transforming data with scripting

The tables defined in the Qlik Sense script are called logical tables. Qlik Sense makes associations between the tables based on the field names, and performs the joins when a selection is made, for example selecting a field value in a filter pane.

This means that an association is almost the same thing as a join. The only difference is that the join is performed when the script is executed - the logical table is usually the result of the join. The association is made after the logical table is created - associations are always made between the logical tables.

Four tables: a list of countries, a list of customers, a list of transactions and a list of memberships, which are associated with each other through the fields Country and CustomerID.



Qlik Sense association compared to SQL natural outer join

A Qlik Sense association resembles a SQL natural outer join. The association is however more general: an outer join in SQL is usually a one-way projection of one table on another. An association always results in a full (bidirectional) natural outer join.

Frequency information in associating fields

There are some limitations in the use of most associating fields, that is, fields that are common between two or more tables. When a field occurs in more than one table, Qlik Sense has a problem knowing which of the tables it should use for calculating data frequencies.

Qlik Sense analyzes the data to see if there is a non-ambiguous way to identify a main table to count in (sometimes there is), but in most cases the program can only make a guess. Since an incorrect guess could be fatal (Qlik Sense would appear to make a calculation error) the program has been designed not to allow certain operations when the data interpretation is ambiguous for associating fields.

Limitations for associating fields

1. It is not possible to display frequency information in a filter pane showing the field.
2. Statistics boxes for the field show n/a for most statistical entities.

3. In charts, it is not possible to create expressions containing functions that depend on frequency information (such as Sum, Count functions, and Average) on the field, unless the **Distinct** modifier is activated. After each reload, Qlik Sense will scan all chart expressions to see if any ambiguities have occurred as a result of changes in data structures. If ambiguous expressions are found, a warning dialog will be shown and the expression will be disabled. It will not be possible to enable the expression until the problem has been corrected. If a log file is enabled, all ambiguous expressions will be listed in the log.

Workaround

There is a simple way to overcome these limitations. Load the field an extra time under a new name from the table where frequency counts should be made. Then use the new field for a filter pane with frequency, for a statistics box or for calculations in the charts.

Synthetic keys

Qlik Sense creates synthetic keys when two or more data tables have two or more fields in common. These keys are anonymous fields that represent all occurring combinations of the composite key.

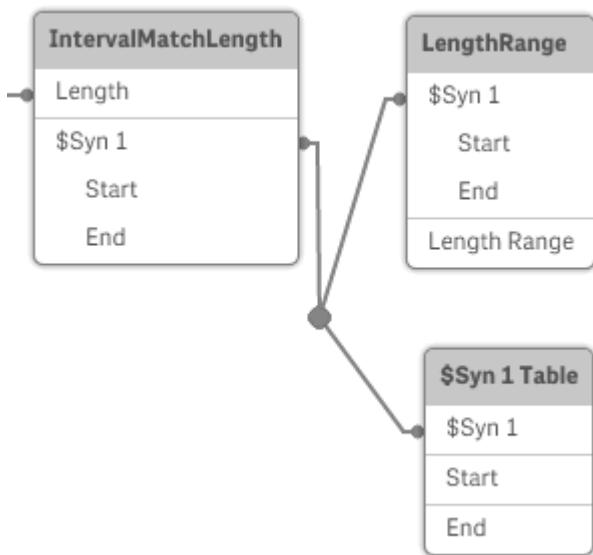
If you receive a warning about synthetic keys when loading data, you should review the data structure in the data model viewer. You should ask yourself whether the data model is correct or not. Sometimes it is, but most times the synthetic key is there due to an error in the script.

Multiple synthetic keys are often a symptom of an incorrect data model. However, a sure sign of an incorrect data model is if you have synthetic keys based on other synthetic keys.



When the number of synthetic keys increases, depending on data amounts, table structure and other factors, Qlik Sense may or may not handle them gracefully, and may end up using excessive amount of time and/or memory. In such a case you need to re-work your script by removing all synthetic keys.

Three tables associated with synthetic key \$Syn 1.



Handling synthetic keys

If you need to avoid synthetic keys, there are a number of ways of solving this in the data load script:

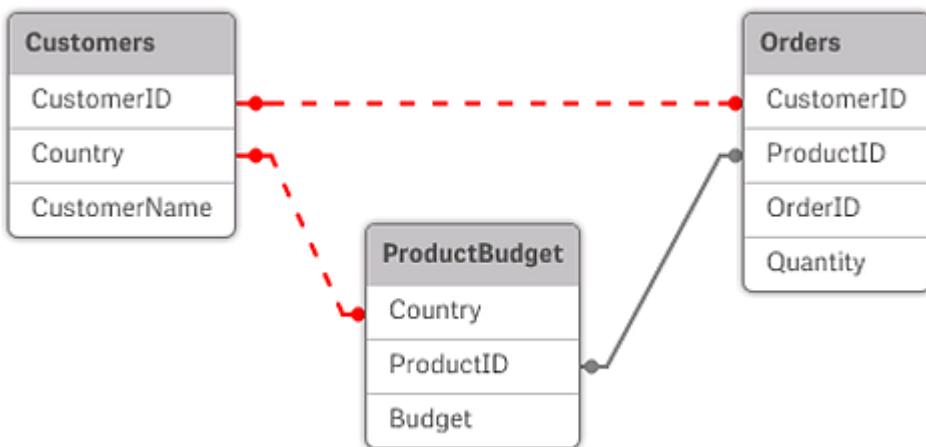
- Check that only fields that logically link two tables are used as keys.
 - Fields like “Comment”, “Remark” and “Description” may exist in several tables without being related, and should therefore not be used as keys.
 - Fields like “Date”, “Company” and “Name” may exist in several tables and have identical values, but still have different roles (Order Date/Shipping Date, Customer Company/Supplier Company). In such cases they should not be used as keys.
- Make sure that redundant fields aren’t used – that only the necessary fields connect. If for example a date is used as a key, make sure not to load year, month or day_of_month of the same date from more than one internal table.
- If necessary, form your own non-composite keys, typically using string concatenation inside an AutoNumber script function.

Understanding and solving circular references

If there are circular references ("loops") in a data structure, the tables are associated in such a way that there is more than one path of associations between two fields.

This type of data structure should be avoided as much as possible, since it might lead to ambiguities in the interpretation of data.

Three tables with a circular reference, since there is more than one path of associations between two fields.



Qlik Sense solves the problem of circular references by breaking the loop with a loosely coupled table. When Qlik Sense finds circular data structures while executing the load script, a warning dialog will be shown and one or more tables will be set as loosely coupled. Qlik Sense will typically attempt to loosen the longest table in the loop, as this is often a transaction table, which normally should be the one to loosen. In the data model viewer, loosely-coupled tables are indicated by the red dotted links to other tables.

Example:

Data is loaded from three tables that include the following:

- Names of some national soccer teams
- Soccer clubs in some cities
- Cities of some European countries

View of the source data tables in Excel.

NationalTeams		Clubs		Cities	
Country	Team	City	Team	Country	City
Germany	Die Mannschaft	Barcelona	Barcelona	Germany	Hamburg
Italy	Azzurri	Hamburg	Altona	Germany	Munich
Spain	La Roja	Madrid	Real Madrid	Italy	Milano
		Milano	Milan	Italy	Rome
		Munich	Bayern München	Italy	Turin
		Rome	Lazio	Spain	Barcelona
		Turin	Juventus	Spain	Madrid

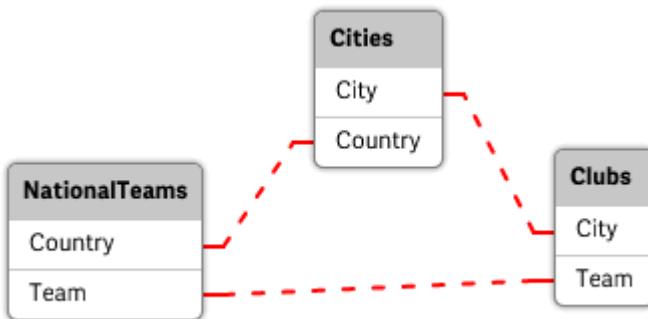
This data structure is not very good, since the field name *Team* is used for two different purposes: national teams and local clubs. The data in the tables creates an impossible logical situation.

When loading the tables into Qlik Sense, Qlik Sense determines which of the data connections that is least important, and loosens this table.

4 Loading and transforming data with scripting

Open the **Data model viewer** to see how Qlik Sense interprets the relevance of the data connections:

View of the circular references as noted by red dotted lines.



The table with cities and the countries they belong to is now loosely coupled to the table with national teams of different countries and to the table with local clubs of different cities.

Solving circular references

When circular references occur, you need to edit the data load script by assigning a unique name to one of the fields with identical names.

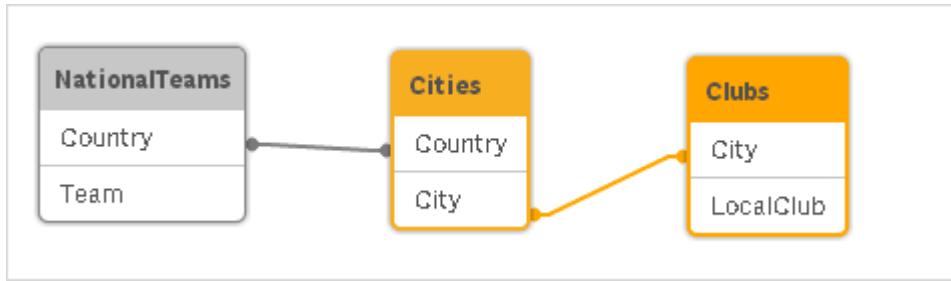
Do the following:

1. Open the data load editor.
2. Edit the **LOAD** statement for one of the duplicate field names.
In this example, the **LOAD** statement of the table that holds the local teams and their cities would include with a new name for *Team*, for example *LocalClub*. The updated **LOAD** statement reads:
`LOAD City, Team as LocalClub`
3. Click in the toolbar to reload data in the app.

You now have logic that works throughout all the tables. In this example, when *Germany* is selected, the national team, the German cities and the local clubs of each city are associated:

Country	Team	City	LocalClub
Germany	Die	Hamburg	Altona
Italy	Azzurri	Munich	Barcelona
Spain	La Roja	Barcelona	Madrid

When you open the **Data model viewer**, you see that the loosely coupled connections are replaced with regular connections:



Concatenating tables

Concatenation is an operation that combines two tables into one.

The two tables are merely added to each other. That is, data is not changed and the resulting table contains the same number of records as the two original tables together. Several concatenate operations can be performed sequentially, so that the resulting table is concatenated from more than two tables.

Automatic concatenation

If the field names and the number of fields of two or more loaded tables are exactly the same, Qlik Sense will automatically concatenate the content of the different statements into one table.

Example:

```
LOAD a, b, c from table1.csv;
LOAD a, c, b from table2.csv;
```

The resulting internal table has the fields a, b and c. The number of records is the sum of the numbers of records in table 1 and table 2.



The number and names of the fields must be exactly the same. The order of the two statements is arbitrary.

Forced concatenation

Even if two or more tables do not have exactly the same set of fields, it is still possible to force Qlik Sense to concatenate the two tables. This is done with the **concatenate** prefix in the script, which concatenates a table with another named table or with the last previously created table.

Example:

```
LOAD a, b, c from table1.csv;
concatenate LOAD a, c from table2.csv;
```

The resulting internal table has the fields a, b and c. The number of records in the resulting table is the sum of the numbers of records in table 1 and table 2. The value of field b in the records coming from table 2 is NULL.



*Unless a table name of a previously loaded table is specified in the **concatenate** statement the **concatenate** prefix uses the most recently created table. The order of the two statements is thus not arbitrary.*

Preventing concatenation

If the field names and the number of fields in two or more loaded tables are exactly the same, Qlik Sense will automatically concatenate the content of the different statements into one table. This is possible to prevent with a **noconcatenate** statement. The table loaded with the associated **LOAD** or **SELECT** statement will then not be concatenated with the existing table.

Example:

```
LOAD a, b, c from table1.csv;  
noconcatenate LOAD a, b, c from table2.csv;
```

Loading data from a previously loaded table

There are two ways to load and transform data from a table that already has been loaded.

- **Resident LOAD** - where you use the **Resident** predicate in a subsequent **LOAD** statement to load a new table.
- Preceding load - where you load from the preceding **LOAD** or **SELECT** statement without specifying a source.

Resident or preceding LOAD?

In most cases, the same result can be achieved by using either method. A preceding **LOAD** is generally the faster option, but there are some cases where you need to use a **Resident LOAD** instead:

- If you want to use the **Order_by** clause to sort the records before processing the **LOAD** statement.
- If you want to use any of the following prefixes, in which cases preceding **LOAD** is not supported:
 - **Crosstable**
 - **Join**
 - **Intervalmatch**

Resident LOAD

You can use the **Resident** predicate in a **LOAD** statement to load data from a previously loaded table. This is useful when you want to perform calculations on data loaded with a **SELECT** statement where you do not have the option to use Qlik Sense functions, such as date or numeric value handling.

Example:

In this example, the date interpretation is performed in the **Resident** load as it can't be done in the initial **Crosstable LOAD**.

```
PreBudget:  
Crosstable (Month, Amount, 1)  
LOAD Account,  
    Jan,
```

4 Loading and transforming data with scripting

```
Feb,  
Mar,  
...  
From Budget;  
  
Budget:  
Noconcatenate  
LOAD  
    Account,  
    Month(Date#(Month, 'MMM')) as Month,  
    Amount  
Resident PreBudget;  
  
Drop Table PreBudget;
```



A common case for using **Resident** is where you want to use a temporary table for calculations or filtering. Once you have achieved the purpose of the temporary table, it should be dropped using the **Drop table** statement.

Preceding load

The preceding load feature allows you to load a table in one pass, but still define several successive transformations. Basically, it is a **LOAD** statement that loads from the **LOAD** or **SELECT** statement below, without specifying a source qualifier such as **From** or **Resident** as you would normally do. You can stack any number of **LOAD** statements this way. The statement at the bottom will be evaluated first, then the statement above, and so on until the top statement has been evaluated.

You can achieve the same result using **Resident**, but in most cases a preceding **LOAD** will be faster.

Another advantage of preceding load is that you can keep a calculation in one place, and reuse it in **LOAD** statements placed above.

Example 1: Transforming data loaded by a **SELECT** statement

If you load data from a database using a **SELECT** statement, you cannot use Qlik Sense functions to interpret data in the **SELECT** statement. The solution is to add a **LOAD** statement, where you perform data transformation, above the **SELECT** statement.

In this example we interpret a date stored as a string using the Qlik Sense function **Date#** in a **LOAD** statement, using the previous **SELECT** statement as source.

```
LOAD Date#(OrderDate, 'YYYYMMDD') as OrderDate;  
SQL SELECT OrderDate FROM ... ;
```

Example 2: Simplifying your script by reusing calculations

In this example we use a calculation more than once in the script:

```
LOAD Age(FromDate + IterNo() - 1, BirthDate ) as Age,  
    Date(FromDate + IterNo() - 1 ) as ReferenceDate  
Resident Policies  
    while IterNo() <=ToDate - FromDate + 1 ;
```

By introducing the calculation in a first pass, we can reuse it in the **Age** function in a preceding **LOAD**:

```
LOAD ReferenceDate,  
    Age(ReferenceDate, BirthDate ) as Age;  
LOAD *,  
    Date(FromDate + IterNo() - 1 ) as ReferenceDate  
    Resident Policies  
        while IterNo() <=ToDate - FromDate + 1 ;
```

Limitations of preceding loads

- The following prefixes cannot be used in conjunction with preceding **LOAD: Join, Crosstable** and **Intervalmatch**.
- If you are using **distinct** to load unique records, you need to place **distinct** in the first load statement, as **distinct** only affects the destination table.

Data types

Qlik Sense can handle text strings, numbers, dates, times, timestamps, and currencies correctly. They can be sorted, displayed in a number of different formats and they can be used in calculations. This means, for example, that dates, times, and timestamps can be added to or subtracted from each other.

Data representation

In order to understand data interpretation and number formatting in Qlik Sense, it is necessary to know how data is stored internally by the program. All of the data loaded into Qlik Sense is available in two representations: as a string and as a number.

1. The string representation is always available and is what is shown in the list boxes and the other sheet objects. Formatting of data in list boxes (number format) only affects the string representation.
2. The number representation is only available when the data can be interpreted as a valid number. The number representation is used for all numeric calculations and for numeric sorting.

If several data items read into one field have the same number representation, they will all be treated as the same value and will all share the first string representation encountered. Example: The numbers 1.0, 1 and 1.000 read in that order will all have the number representation 1 and the initial string representation 1.0.

Number interpretation

When you load data containing numbers, currency, or dates, it will be interpreted differently depending on whether the data type is defined or not. This section describes how data is interpreted in the two different cases.

Data with type information

Fields containing numbers with a defined data type in a database loaded using ODBC will be handled by Qlik Sense according to their respective formats. Their string representation will be the number with an appropriate formatting applied.

Qlik Sense will remember the original number format of the field even if the number format is changed for a measure under **Number formatting** in the properties panel.

The default formats for the different data types are:

- integer, floating point numbers: the default setting for number
- currency: the default setting for currency
- time, date, timestamp: ISO standard formatting

The default settings for number and currency are defined using the script number interpretation variables or the operating system settings (**Control Panel**).

Data without type information

For data without specific formatting information from the source (for example, data from text files or ODBC data with a general format) the situation becomes more complicated. The final result will depend on at least six different factors:

- The way data is written in the source database
- The operating system settings for number, time, date and so on. (**Control Panel**)
- The use of optional number-interpreting variables in the script
- The use of optional interpretation functions in the script
- The use of optional formatting functions in the script
- The number formatting controls in the document

Qlik Sense tries to interpret input data as a number, date, time, and so on. As long as the system default settings are used in the data, the interpretation and the display formatting is done automatically by Qlik Sense, and the user does not need to alter the script or any setting in Qlik Sense.

By default, the following scheme is used until a complete match is found. (The default format is the format such as the decimal separator, the order between year, month and day, and so on, specified in the operating system, that is, in the **Control Panel**, or in some cases from the special number interpretation variables in the script.

Qlik Sense will interpret the data as:

1. A number in accordance with the default format for numbers.
2. A date according to the default format for date.
3. A timestamp according to the default format for time and date.
4. A time according to the default format for time.
5. A date according to the following format: yyyy-MM-dd.
6. A time-stamp according to the following format: YYYY-MM-DD hh:mm[:ss[.fff]].
7. A time according to the following format: hh:mm[:ss[.fff]].
8. Money according to the default format for currency.
9. A number with '.' as decimal separator and ',' as thousands separator, provided that neither the decimal separator nor the thousands separator are set to ','.
10. A number with ',' as decimal separator and '.' as thousands separator, provided that neither the decimal separator nor the thousands separator are set to '!'.
11. A text string. This last test never fails: if it is possible to read the data, it is always possible to interpret it as a string.

4 Loading and transforming data with scripting

When loading numbers from text files, some interpretation problems may occur, for example, an incorrect thousands separator or decimal separator may cause Qlik Sense to interpret the number incorrectly. The first thing to do is to check that the number-interpretation variables in the script are correctly defined and that the system settings in the **Control Panel** are correct.

When Qlik Sense has interpreted data as a date or time, it is possible to change to another date or time format in the properties panel of the visualization.

Since there is no predefined format for the data, different records may, of course, contain differently formatted data in the same field. It is possible for example, to find valid dates, integers, and text in one field. The data will therefore, not be formatted, but shown in its original form.

Date and time interpretation

Qlik Sense stores each date, time, and timestamp found in data as a date serial number. The date serial number is used for dates, times and timestamps and in arithmetic calculations based on date and time entities. Dates and times can thus be added and subtracted, intervals can be compared, and so on.

The date serial number is the (real valued) number of days passed since December 30, 1899, that is, the Qlik Sense format is identical to the 1900 date system used by Microsoft Excel and other programs, in the range between March 1, 1900 and February 28, 2100. For example, 33857 corresponds to September 10, 1992. Outside this range, Qlik Sense uses the same date system extended to the Gregorian calendar.



*If the field contains dates before January 1, 1980, the field will not contain the **\$date** or **\$timestamp** system tags. The field should still be recognized as a date field by Qlik Sense, but if you need the tags you can add them manually in the data load script with the **Tag** statement.*

The serial number for times is a number between 0 and 1. The serial number 0.00000 corresponds to 00:00:00, whereas 0.99999 corresponds to 23:59:59. Mixed numbers indicate the date and time: the serial number 2.5 represents January 1, 1900 at 12:00 noon.

The data is, however, displayed according to the format of the string. By default, the settings made in the **Control Panel** are used. It is also possible to set the format of the data by using the number interpretation variables in the script or with the help of a formatting function. Lastly, it is also possible to reformat the data in the properties sheet of the sheet object.

Example 1:

- 1997-08-06 is stored as 35648
- 09:00 is stored as 0.375
- 1997-08-06 09:00 is stored as 35648.375

and the other way around:

- 35648 with number format 'D/M/YY' is shown as 6/8/97
- 0.375 with number format 'hh.mm' is shown as 09.00

Qlik Sense follows a set of rules to try to interpret dates, times, and other data types. The final result, however, will be affected by a number of factors as described here.

4 Loading and transforming data with scripting

Example 2:

These examples assume the following default settings:

- Number decimal separator:
- Short date format: YY-MM-DD
- Time format: hh:mm

The following table shows the different representations when data is read into Qlik Sense without the special interpretation function in the script:

Table when data is read without the special interpretation function in the script

Source data	Qlik Sense default interpretation	'YYYY-MM-DD' date format	'MM/DD/YYYY' date format	'hh:mm' time format	'# ##0.00' number format
0.375	0.375	1899-12-30	12/30/1899	09:00	0.38
33857	33857	1992-09-10	09/10/1992	00:00	33 857.00
97-08-06	97-08-06	1997-08-06	08/06/1997	00:00	35 648.00
970806	970806	4557-12-21	12/21/4557	00:00	970 806.00
8/6/97	8/6/97	8/6/97	8/6/97	8/6/97	8/6/97

The following table shows the different representations when data is read into Qlik Sense using the date#(A, 'M/D/YY') interpretation function in the script:

Table when using the date#(A, 'M/D/YY') interpretation function in the script

Source data	Qlik Sense default interpretation	'YYYY-MM-DD' date format	'MM/DD/YYYY' date format	'hh:mm' time format	'# ##0.00' number format
0.375	0.375	0.375	0.375	0.375	0.375
33857	33857	33857	33857	33857	33857
97-08-06	97-08-06	97-08-06	97-08-06	97-08-06	97-08-06
970806	970806	970806	970806	970806	970806
8/6/97	8/6/97	1997-08-06	08/06/1997	00:00	35 648.00

Dollar-sign expansions

Dollar-sign expansions are definitions of text replacements used in the script or in expressions. This process is known as expansion - even if the new text is shorter. The replacement is made just before the script statement or the expression is evaluated. Technically it is a macro expansion.

The expansion always begins with '\$(' and ends with ')'. For example: \$(variablename). The content between brackets defines how the text replacement will be done.



A dollar-sign expansion is limited in how many expansions it can calculate. Any expansion with over 1000 levels of nested expansions will not be calculated.

Dollar-sign expansion using a variable

In the load script or a chart expression, use a variable in a dollar-sign expansion to:

- Reference text
- Reference a numeric value

Text variable

When using a variable for text replacement in the script or in an expression, the following syntax is used:

`$(variablename)`

`$(variablename)` expands to the value in the variable. If `variablename` does not exist, the expansion will result in an empty string.

Examples: Text variable load scripts

Example 1: Load script

Load script

Load the following data as an inline load in the data load editor:

```
Set x = 'red';           // Assign the value "red" to variable x
Set y = 'blue';          // Assign the value "blue" to variable y
Set z = '$(x) $(y)';    // Expands x and y, returns "red blue" in variable z

// Expand x and y, return "red green blue" in variable MyString

Let MyString='$(x)''&' green '&''$(y)';

// Create table MyTable, load variable values for x, y, z into fields x, Y, Z
// Concatenate with variable MyString into field NewString

MyTable:
Load '$(x)' as x, '$(y)' as Y, '$(z)' as z, '$(MyString)' as NewString autogenerate 1;
```

Explanation

This example demonstrates:

- How to expand a variable in variable assignments.
- How to expand variables combined with a textual operations.

This is a useful setup for creating dynamic labels and general text strings that combine a variable content with static strings.

4 Loading and transforming data with scripting

Output

Create the following table in Qlik Sense:

Table - Output from load script

X	Y	Z	NewString
red	blue	red blue	red blue green

Example 2: Load script

Load script

Load the following data as an inline load in the data load editor:

```
Set vFunction = 'upper'; // Assign the string "upper" to variable vFunction
Set vField = 'String'; // Assign the string "String" to variable vField
Let vEvaluate = '$(vFunction)''&(''&'$(vField)'&')';

// The variable vEvaluate returns the value "upper(string)"

MyTable: // Create table called MyTable
Load *, $(vEvaluate) as Upper; // vEvaluate expanded as a dynamic expression
Load *, '$(vEvaluate)' as Expression; // vEvaluate expanded as string
Load * inline [
ID, String
1, abc
2, def
3, ghi
4, jkl ];
```

Explanation

The Set and Let statements are used to assign values to variables in the load script. The difference between the two is that the Set statement assigns a string to the variable, while the Let statement evaluates the content of the string before assigning the resultant value to the variable. The load inline table in this example is supplemented with two preceding load statements that are used to visualize different evaluations of the variable vEvaluate both as a text string and as the corresponding expression.

Output

Create the following table in Qlik Sense:

Table - Output from load script

ID	String	Expression	Upper
1	abc	upper(String)	ABC
2	def	upper(String)	DEF

4 Loading and transforming data with scripting

ID	String	Expression	Upper
3	ghi	upper(String)	HIJ
4	JKL	upper(String)	JKL

Example: Text variable chart expression

Example: Load script

Load script

Load the following data as an inline load in the data load editor:

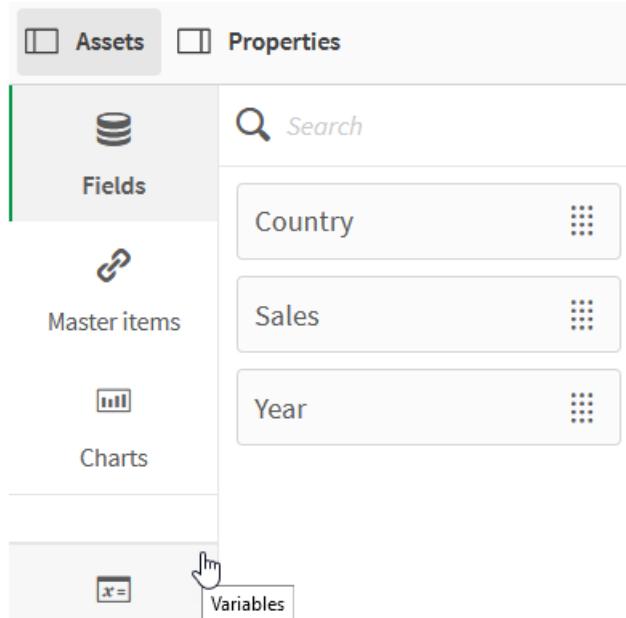
```
// Create table SalesByCountry
SalesByCountry:
Load * Inline [
Country, Year, Sales
Argentina, 2014, 66295.03
Argentina, 2015, 140037.89
Austria, 2014, 54166.09
Austria, 2015, 182739.87
Belgium, 2014, 182766.87
Belgium, 2015, 178042.33
Brazil, 2014, 174492.67
Brazil, 2015, 2104.22
Canada, 2014, 101801.33
Canada, 2015, 40288.25
Denmark, 2014, 45273.25
Denmark, 2015, 106938.41
Finland, 2014, 107565.55
Finland, 2015, 30583.44
France, 2014, 115644.26
France, 2015, 30696.98
Germany, 2014, 8775.18
Germany, 2015, 77185.68
];
```

Variables

In a sheet in edit mode, open the **Variables** dialog from the **Assets** panel.

4 Loading and transforming data with scripting

Open variables dialog



Create the following variables:

Name	Variable to create
Definition	
vSales	Sum(Sales)
vSales2014	Sum({<Year={2014}>}Sales)
vSales2015	Sum({<Year={2015}>} Sales)
vSalesAllYears	\$(vSales2014) +\$(vSales2015)
vSalesDifference	\$(vSales2015)/\$(vSales2014) - 1

4 Loading and transforming data with scripting

Variables

Variables

Create new **Delete**

	Name ▼	Definition	Value	Tags ▼	
<input type="checkbox"/>	vSales	Sum(Sales)	Sum(Sales)		...
<input type="checkbox"/>	vSales2014	Sum({<Year=[2014]>}S...)	Sum({<Year=[...}		...
<input type="checkbox"/>	vSales2015	Sum({<Year=[2015]>} ...)	Sum({<Year=[...}		...
<input type="checkbox"/>	vSalesAllYears	\$(vSales2014) +\$(vSal...	\$(vSales2014...		...
<input type="checkbox"/>	vSalesDifference	\$(vSales2015)/\$(vSale...	\$(vSales2015...		...

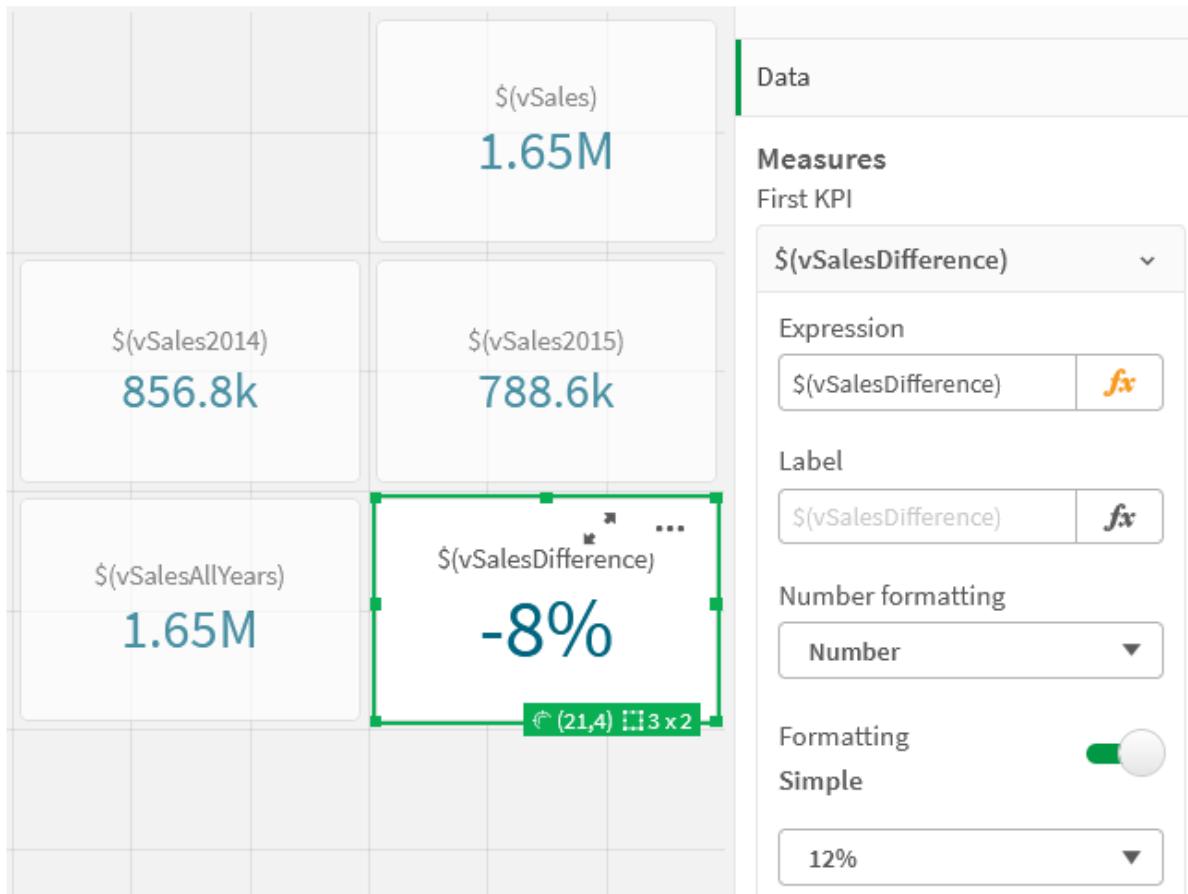
Showing: 1-5 of 5. |< < > >|

Done

Create KPI charts to see the expansions.

4 Loading and transforming data with scripting

KPIs using dollar sign expansions



Numeric variable expansion

For numeric variable expansions, the following syntax is used:

(#variablename)

The expansion always yields a valid decimal-point representation of the numeric value of the variable, possibly with exponential notation (for very large or small numbers). If variablename does not exist or does not contain a numeric value, it will be expanded to 0 instead of NULL.

Examples: Numeric variable load scripts

Example 1: Load script

Load script

Load the following data as an inline load in the data load editor:

```
Set DecimalSep = ','; // Set decimal comma as separator for this example.  
Let x = 7/2;           // Assign the expression 7/2 to variable x.
```

```
MyTable: // Create an inline table labeled "MyTable"  
Load 1 as ID, * inline [  
DecimalComma DecimalPoint
```

4 Loading and transforming data with scripting

```
$(x)  $(#x) ]
(delimiter is '\t');
```

Explanation

The #variable expansion always yields a valid decimal-point representation of the numeric value of the variable. This is useful when comma, rather than point is used as decimal separator and there is a risk of a conflict with comma separated lists.

The main reason for expanding these variables in a load-inline table is that no additional quoting of \$(x) is required.

Output

Create the following table in Qlik Sense:

Table - Output from load script

DecimalComma	DecimalPoint
3,5	3.5

Example 2: Load script

Load script

Load the following data as an inline load in the data load editor:

```
// The three Set statements below are required to mimic and initialize
// Format specifiers that are relevant to this particular example

Set ThousandSep=' ';           // Set example thousand separator
Set DecimalSep=',';           // Set example decimal separator
Set TimestampFormat='YYYY-MM-DD hh:mm:ss'; // Set example date format
Let vRaw = today()-1/1440;      // Timestamp minus one minute
Let vFormat = timestamp($(#vRaw)); // Formatted as timestamp

// Create MyTable as an inline table to expand variables as field values

MyTable:
Load * inline [
DecimalComma  DecimalPoint    FormattedNumber
$(vRaw)        $(#vRaw)        $(vFormat) ] (delimiter is '\t');
```

Explanation

The #variable expansion always yields a valid decimal-point representation of the numeric value of the variable. This is useful when comma, rather than point is used as decimal separator and there is a risk of a conflict with comma separated lists. It is also important to note that the numerical precision will be affected due to the decimal part being truncated when variables are expanded without correct decimal separator.

4 Loading and transforming data with scripting

The main reason for expanding these variables in a load-inline table is that no additional quoting of \$(x) is required.

Output

Create the following table in Qlik Sense:

Table - Output from load script

DecimalComma	DecimalPoint	FormattedNumber
44 470,00	44469.999305556	2021-09-18 23:59:00

Example 3: Load script

Load script

Load the following data as an inline load in the data load editor:

```
// The three Set statements below are required to mimic and initialize
// format specifiers that are relevant to this particular example

Set ThousandSep=' ';           // Set example thousand separator
Set DecimalSep=',';           // Set example decimal separator
Set TimestampFormat='YYYY-MM-DD hh:mm:ss';    // Set example date format

// Assign a numerical value and a valid format specifier to vStart

Let vStart = timestamp#('2021-03-23 12:34:56','$(TimestampFormat)');

// Calculate timestamp (vStart + 3 hours) with valid decimal separator: "."

Let vStop = timestamp($(#vStart)+1/8,'YYYY-MM-DD hh:mm:ss');

// Create MyTable as an inline table to expand variables as field values

MyTable:
Load * inline [
StartTime      StopTime
$(vstart)      $(vstop) ] (delimiter is '\t');

// This is a tab delimited inline table
// Tab delimited tables are useful for avoiding conflicting list separators
```

Explanation

The #variable expansion always yields a valid decimal-point representation of the numeric value of the variable. This is useful when comma, rather than point is used as decimal separator and there is a risk of a conflict with comma separated lists. It is also important to note that the numerical precision will be affected due to the decimal part being truncated when variables are expanded without correct decimal separator.

4 Loading and transforming data with scripting

The main reason for expanding these variables in a load-inline table is that no additional quoting of \$(x) is required.

Output

Create the following table in Qlik Sense:

Table - Output from load script

StartTime	StopTime	
2021-03-23 12:34:56	2021-03-23 15:34:56	

Expanding variables referencing alternate states

The variable has only one value, and this is used in all alternate states. When you expand a variable the value is also the same, independent of where it is made, and the state of the object.

If the variable is a calculated variable, that is, the definition starts with an equals sign, the calculation is made in the default state, unless you specify an alternate state in the variable definition.

For example, if you have a state named `Mystate`, and a variable named `vmyvar`:

```
vMyvar: =only({Mystate}MyField)
```

The variable definition content, with an explicit reference to the alternate state name, determines in which state the variable content will be evaluated.

Dollar-sign expansion using parameters

Parameters can be used in dollar-sign expansions. This makes it possible to have a dynamic evaluation of the variable.

The variable created for use in the expansion must contain formal parameters. A formal parameter is a placeholder for the input to the evaluation and is written with a dollar sign: \$1, \$2, \$3. The number indicates the parameter number.

When using the variable, the actual parameters should be stated in a comma separated list.

Examples: Parameters in dollar-sign expansions

If you want to define a multiplication between two numbers, you can write:

```
Set MUL= $1*$2 ;
```

This defines that \$1 and \$2 should be multiplied with each other. When used, this variable must be dollar expanded with values for \$1 and \$2 inserted into the expression:

```
Set X= $( MUL(3,7) ) ;
```

The values (3 and 7) are the actual parameters that replace \$1 and \$2 in the expansion. The expansion is made before the Set statement is parsed and executed, which means that the script parser sees the following:

```
Set X= 3*7 ;
```

As a result, the variable x will be assigned the string value: 3*7.

If you use a Let instead of set statement:

4 Loading and transforming data with scripting

```
Let x= $( MUL(3,7) ) ;
```

The parser will see the following:

```
Let x= 3*7 ;
```

Here an evaluation will be made, and x will be assigned the numeric value 21.

Number of parameters

If the number of formal parameters exceeds the number of actual parameters, only the formal parameters corresponding to actual parameters will be expanded. If the number of actual parameters exceeds the number of formal parameters, the superfluous actual parameters will be ignored.

Examples: Formal parameters versus actual parameters

```
Set MUL= '$1*$2' ;
Set x= $(MUL) ;           // returns $1*$2 in x
Set x= $(MUL(10)) ;      // returns 10*$2 in x
Let x= $(MUL(5,7,8)) ;  // returns 35 in x
```

The \$0 parameter

The parameter \$0 returns the number of parameters actually passed by a call.

Example: How to introduce error handling

```
Set MUL= If($0=2, $1*$2, 'Error') ;
```

Dollar-sign expansion using an expression

Expressions can be used in dollar-sign expansions.

The content between the brackets must start with an equal sign:

```
$(=expression)
```

The expression will be evaluated and the value will be used in the expansion. For example:

```
// returns a string with the current year
$(=Year(Today()));
// returns the year before the selected one
$(=Only(Year)-1);
```

Example: Select a measure by referencing the measure in a chart expression

Example: Load script

Load script

Load the following data as an inline load in the data load editor:

```
// Load string “=MinString(Measure)” into variable vSelectMeasure
Let vSelectMeasure = '=MinString(Measure)';
MyTable1: // Create table and load values for Dim and Sales
Load * inline [
Dim, Sales
A, 150
A, 200
```

```
B, 240  
B, 230  
C, 410  
C, 330 ];
```

```
MyTable2: // Create table and load aggregations as field values for Measure  
Load * Inline [  
Measure  
avg(sales)  
sum(sales)  
count(distinct Dim)];
```

Explanation

The script allows you to select a measure by referencing the measure in a chart expression. The chart expression contains the variable expansion `$(=MinString(Measure))`. `MinString()` finds string values in the expression and returns the first text value sorted alphabetically, in this case `avg(sales)`. This makes it possible to interactively decide (and select) which expression to use in a visualization without accessing and manipulating the properties of the object.

Output

Create the following table in Qlik Sense by using:

- Dimension: `Dim`
- Measures:
 - `='$vSelectMeasure'`
 - `=$(=MinString(Measure))`
 - `Avg(Sales)`

Table - Dollar-sign expansion using an expression

Dim	<code>='\$vSelectMeasure'</code>	<code>=\$(=MinString(Measure))</code>	<code>Avg(Sales)</code>
Totals	-	260	260
A	avg(Sales)	175	175
B	avg(Sales)	235	235
C	avg(Sales)	370	370

Example: Calculate the quota between two dimensions

Example: Load script

Load script

Load the following data as an inline load in the data load editor:

4 Loading and transforming data with scripting

```
Let vDivision = '=MinString(Numerator) / MinString(Denominator)';
Load recno() as Numerator autogenerate 100;
Load recno() as Denominator autogenerate 100;
```

Explanation

This script generates a pair of fields, containing 100 values each, that function as Numerators and Denominators in a calculation of the quota between the two. The chart measure contains the variable expansion ='\$(<vDivision>) ', making it possible to interactively decide (and select) which quota to be calculated.

Output

Create the following table in Qlik Sense by using:

- Dimensions:
 - Numerator
 - Denominator
- Measure: ='\$(vDivision)'

The resultant quota (Totals) is affected by the selected values in the fields. Numerator (=5) and Denominator (=3) and will be recalculated with each new selection in either field.

Table - Dollar-sign expansion as expression

Numerator	Denominator	='\$(<vDivision>)'
Totals	-	1,66666666666667
5	3	1,66666666666667

File inclusion

File inclusions are made using dollar-sign expansions. The syntax is:

```
$(include=filename)
```

The above text will be replaced by the content of the file specified after the equal sign. This feature is useful when storing scripts or parts of scripts in text files.

Dollar expansions and alternate states

A dollar expansion is normally not sensitive to alternate states. However, if the dollar expansion has an expression, this is evaluated in the state relevant to the object where the dollar expansion is made.

For example:

```
$(=Sum(Amount))
```

The calculation will return the sum of amount based on the selection in the state of the object.

Using quotation marks in the script

You can use quotation marks in script statements in a number of different ways.

Inside LOAD statements

In a **LOAD** statement the following symbols should be used as quotation marks:

Field names

Description	Symbol	Code point	Example
double quotation marks	" "	34	"string"
square brackets	[]	91, 93	[string]
grave accents	` `	96	`string`

String literals

Description	Symbol	Code point	Example
single quotation marks	' '	39	'string'

In SELECT statements

For a **SELECT** statement interpreted by an ODBC driver, usage may vary. Usually, you should use the straight double quotation marks (Alt + 0034) for field and table names, and the straight single quotation marks (Alt + 0039) for literals, and avoid using grave accents. However, some ODBC drivers not only accept grave accents as quotation marks, but also prefer them. In such a case, the generated **SELECT** statements contain grave accent quotation marks.

Microsoft Access quotation marks example

Microsoft Access ODBC Driver 3.4 (included in Microsoft Access 7.0) accepts the following quotation marks when analyzing the **SELECT** statement:

Field names and table names:

- []
- " "
- ` `

String literals:

- ''

Other databases may have different conventions.

Outside LOAD statements

Outside a **LOAD** statement, in places where Qlik Sense expects an expression, double quotation marks denote a variable reference and not a field reference. If you use double quotation marks, the enclosed string will be interpreted as a variable and the value of the variable will be used.

Out-of-context field references and table references

Some script functions refer to fields that have already been created, or are in the output of a **LOAD** statement, for example **Exists()** and **Peek()**. These field references are called out-of-context field references, as opposed to source field references that refer to fields that are in context, that is, in the input table of the **LOAD** statement.

Out-of-context field references and table references should be regarded as literals and therefore need single quotation marks.

Difference between names and literals

The difference between names and literals becomes clearer comparing the following examples:

Example:

'Sweden' as Country

When this expression is used as a part of the field list in a **LOAD** or **SELECT** statement, the text string "Sweden" will be loaded as field value into the Qlik Sense field "Country".

Example:

"land" as Country

When this expression is used as a part of the field list in a **LOAD** or **SELECT** statement, the content of the database field or table column named "land" will be loaded as field values into the Qlik Sense field "Country". This means, that *land* will be treated as a field reference.

Difference between numbers and string literals

The difference between numbers and string literals becomes clearer comparing the following examples.

Example:

'12/31/96'

When this string is used as a part of an expression, it will in a first step be interpreted as the text string "12/31/96", which in turn may be interpreted as a date if the date format is 'MM/DD/YY'. In that case it will be stored as a dual value with both a numeric and a textual representation.

Example:

12/31/96

When this string is used as a part of an expression, it will be interpreted numerically as 12 divided by 31 divided by 96.

Using quotation marks in a string

When a string contains characters that can be used as quotation marks, it is important to clearly indicate where a string begins and where it ends when quoting the string. If the string is not properly quoted, the script will fail or it will load data incorrectly.

There are two methods for quoting a string that contains quotation marks.

4 Loading and transforming data with scripting

Use a specific quotation mark to quote the string

Choose a quotation mark that is not used inside the string, and use it to quote the entire string. Qlik Sense will use that specific quotation mark to determine where the string begins and ends.

Any of the following quotation marks can be used to quote the entire string:

- Double quotation marks "
- Square brackets []
- Grave accents ` `
- Single quotation marks '

Example:

```
[Table '1 "2"]
```

Square brackets are used to quote the string. The string loads as : *Table '1 "2"*

```
'string `Name1` "Name2'
```

Single quotation marks are used to quote the string. The string loads as: *string `Name1` "Name2"*

Use escape characters

Escape characters are an additional instance of the quotation mark that is used to quote the string. They must be added beside every instance of the quotation mark that appears within the string. When all quotation marks are used inside a string, you need to add escape characters beside the same type of quotation mark used to quote the string. Escape characters can also be used if you want to use a quotation mark that is already in use in a string.

Only the following marks can be used as escape characters:

- Double quotation marks "
- Square brackets []
- Single quotation marks '

Example:

```
"Michael said """It's a beautiful day""."
```

If you quote the string using the double quotation marks " ", then you must add an extra double quotation mark beside every double quotation mark used inside the string.

This string is loaded as *Michael said "It's a beautiful day"*. By using the escape character "", the Qlik Sense Data load editor understands which double quotation marks are part of the string and which quotation mark indicates the end of the string. The single quotation mark ' used in the abbreviation *It's* does not need to be escaped because it is not the mark used to quote the string.

Example:

```
'Michael said: "It''s a beautiful day".'
```

If you quote this string using single quotation marks, then you must add an extra single quotation mark beside each single quotation mark used inside the string.

4 Loading and transforming data with scripting

This string is loaded as *Michael said "It's a beautiful day"*. The double quotation mark " used for quoting what Michael said does not need to be escaped because it is not the mark used to quote the string.

Example:

```
[Michael said [It's a "beautiful day"]].]
```

Square brackets [] behave differently from the other two quotation marks. If you want to use brackets as an escape character, you must add an extra bracket beside the right square bracket] only, and not beside the left square bracket [.

This string is loaded as *Michael said [It's a "beautiful day"]*. Only the right square bracket] is escaped. The single quotation mark ' and the double quotation mark " used in the string do not need to be escaped as they are not used to quote the string.

Wild cards in the data

You can use wild cards in the data. Two different wild cards exist: the star symbol, interpreted as all values of this field, and an optional symbol, interpreted as all remaining values of this field.

The star symbol

The star symbol is interpreted as all (listed) values of this field, that is, a value listed elsewhere in this table. If used in one of the system fields (*USERID*, *PASSWORD*, *NTNAME* or *SERIAL*) in a table loaded in the access section of the script, it is interpreted as all (also not listed) possible values of this field.

There is no star symbol available unless explicitly specified. .

OtherSymbol

In many cases a way to represent all other values in a table is needed, that is, all values that were not explicitly found in the loaded data. This is done with a special variable called **OtherSymbol**. To define the **OtherSymbol** to be treated as "all other values", use the following syntax:

```
SET OTHERSYMBOL=<sym>;  
before a LOAD or SELECT statement. <sym> may be any string.
```

The appearance of the defined symbol in an internal table will cause Qlik Sense to define it as all values not previously loaded in the field where it is found. Values found in the field after the appearance of the **OtherSymbol** will thus be disregarded.

In order to reset this functionality use:

```
SET OTHERSYMBOL=;
```

Example:

Table Customers

CustomerID	Name
1	ABC Inc.

4 Loading and transforming data with scripting

CustomerID	Name
2	XYZ Inc.
3	ACME INC
+	Undefined

Table Orders

CustomerID	OrderID
1	1234
3	1243
5	1248
7	1299

Insert the following statement in the script before the point where the first table above is loaded:

```
SET OTHERSYMBOL=+;
```

Any reference to a *CustomerID* other than 1, 2 or 3, e.g. as when clicking on *OrderID* 1299 will result in *Undefined* under *Name*.



OtherSymbol is not intended to be used for creating outer joins between tables.

NULL value handling

When no data can be produced for a certain field as a result of a database query and/or a join between tables, the result is normally a NULL value.

Overview

The Qlik Sense logic treats the following as real NULL values:

- NULL values returned from an ODBC connection
- NULL values created as a result of a forced concatenation of tables in the data load script
- NULL values created as a result of a join made in the data load script
- NULL values created as a result of the generation of field value combinations to be displayed in a table



*It is generally impossible to use these NULL values for associations and selections, except when the **NullAsValue** statement is being employed.*

Text files per definition cannot contain NULL values.

4 Loading and transforming data with scripting

Associating/selecting NULL values from ODBC

It is possible to associate and/or select NULL values from an ODBC data source. For this purpose a script variable has been defined. The following syntax can be used:

```
SET NULLDISPLAY=<sym>;
```

The symbol <sym> will substitute all NULL values from the ODBC data source on the lowest level of data input. <sym> may be any string.

In order to reset this functionality to the default interpretation, use the following syntax:

```
SET NULLDISPLAY=;
```



*The use of **NULLDISPLAY** only affects data from an ODBC data source.*

If you wish to have the Qlik Sense logic interpret NULL values returned from an ODBC connection as an empty string, add the following to your script before any **SELECT** statement:

```
SET NULLDISPLAY=";
```



Here " is actually two single quotation marks without anything in between.

Creating NULL values from text files

It is possible to define a symbol, which when it occurs in a text file or an **inline** clause will be interpreted as a real NULL value. Use the following statement:

```
SET NULLINTERPRET=<sym>;
```

The symbol <sym> is to be interpreted as NULL. <sym> may be any string.

In order to reset this functionality to the default interpretation, use:

```
SET NULLINTERPRET=;
```



*The use of **NULLINTERPRET** only affects data from text files and inline clauses.*

Propagation of NULL values in expressions

NULL values will propagate through an expression according to a few logical and quite reasonable rules.

Functions

The general rule is that functions return NULL when the parameters fall outside the range for which the function is defined.

Examples

Expression	Result
asin(2)	returns NULL

4 Loading and transforming data with scripting

Expression	Result
log(-5)	returns NULL
round(A,0)	returns NULL

As a result of the above follows that functions generally return NULL when any of the parameters necessary for the evaluation are NULL.

Examples

Expression	Result
sin(NULL)	returns NULL
chr(NULL)	returns NULL
if(NULL, A, B)	returns B
if(True, NULL, A)	returns NULL
if(True, A, NULL)	returns A

The exception to the second rule are logical functions testing for type.

Examples

Expression	Result
isnull(NULL)	returns True (-1)
isnum(NULL)	returns False (0)

Arithmetic and string operators

If NULL is encountered on any side of these operators NULL is returned.

Examples

Expression	Result
A + NULL	returns NULL
A - NULL	returns NULL
A / NULL	returns NULL
A * NULL	returns NULL
NULL / A	returns NULL
0 / NULL	returns NULL
0 * NULL	returns NULL
A & NULL	returns A

Relational operators

If NULL is encountered on any side of relational operators special rules apply.

Examples	
Expression	Result
NULL (any relation operator) NULL	returns NULL
A <> NULL	returns True (-1)
A < NULL	returns False (0)
A <= NULL	returns False (0)
A = NULL	returns False (0)
A >= NULL	returns False (0)
A > NULL	returns False (0)

4.4 Guidelines for data and fields

There are certain conventions and limitations you need to be aware of when working with Qlik Sense. For example: the upper limit for data tables and fields as well as maximum amount of loaded data in Qlik Sense.

Guidelines for amount of loaded data

The amount of data that can be loaded into Qlik Sense is primarily, but not exclusively, limited by the amount of primary memory of the computer.

Upper limits for data tables and fields

Be aware when building very large apps that a Qlik Sense app cannot have more than 2,147,483,648 distinct values in one field.

The number of fields and data tables as well as the number of table cells and table rows that can be loaded, is mainly limited by RAM.

When importing a dataset file into a Qlik Sense app or space with Data manager (drag-and-drop or other direct uploads), the maximum number of fields that can be loaded is 5000.

Recommended limit for load script sections

The recommended maximum number of characters to be used per load script section is 50,000 characters.

Descriptions for number and time formats

In many interpretation and formatting functions it is possible to set the format for numbers and dates by using a format code. This topic describes the formats for the number, date, time, and timestamp functions. These formats apply both to script and chart functions.

Number formats

To denote a specific number of digits, use the symbol "0" for each digit.

To denote a possible digit to the left of the decimal point, use the symbol "#".

4 Loading and transforming data with scripting

To mark the position of the thousands separator or the decimal separator, use the applicable thousands separator and the decimal separator.

The format code is used for defining the positions of the separators. It is not possible to set the separator in the format code. Use the **DecimalSep** and **ThousandSep** variables for this in the script.

It is possible to use the thousand separator to group digits by any number of positions, for example, a format string of "0000-0000-0000" (thousand separator="-") could be used to display a twelve-digit part number as "0012-4567-8912".

Examples:

Example of number formats

Number format	Description
# ##0	describes the number as an integer with a thousands separator. In this example " " is used as a thousands separator.
###0	describes the number as an integer without a thousands separator.
0000	describes the number as an integer with at least four digits. For example, the number 123 will be shown as 0123.
0.000	describes the number with three decimals. In this example "." is used as a decimal separator.

Special number formats

Qlik Sense can interpret and format numbers in any radix between 2 and 36 including binary, octal and hexadecimal. It can also handle roman formats.

Special number formats

Format	Description
Binary format	To indicate binary format the format code should start with (bin) or (BIN).
Octal format	To indicate octal format the format code should start with (oct) or (OCT).
Hexadecimal format	To indicate hexadecimal format the format code should start with (hex) or (HEX). If the capitalized version is used A-F will be used for formatting (for example 14FA). The non-capitalized version will result in formatting with a-f (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code.
Decimal format	The use of (dec) or (DEC) to indicate decimal format is permitted but unnecessary.

4 Loading and transforming data with scripting

Format	Description
Custom radix format	To indicate a format in any radix between 2 and 36 the format code should start with (rxx) or (Rxx) where xx is the two-digit number denoting the radix to be used. If the capitalized R is used letters in radices above 10 will be capitalized when Qlik Sense is formatting (for example 14FA). The non-capitalized r will result in formatting with non-capital letters (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code. Note that (r02) is the equivalent of (bin), (R16) is the equivalent of (HEX), and so on.
Roman format	To indicate roman numbers the format code should start with (rom) or (ROM). If the capitalized version is used capital letters will be used for formatting (for example MMXVI). The non-capitalized version will result in formatting with lower cap letters (mmxvi). Interpretation will work for both variants regardless of the capitalization of the format code. Roman numbers are generalized with minus sign for negative numbers and 0 for zero. Decimals are ignored with roman formatting.

Examples:

Examples of special number formats

Example	Result
num(199, '(bin)')	returns 11000111
num(199, '(oct)')	returns 307
num(199, '(hex)')	returns c7
num(199, '(HEX)')	returns C7
num(199, '(r02)')	returns 11000111
num(199, '(r16)')	returns c7
num(199, '(R16)')	returns C7
num(199, '(R36)')	returns 5J
num(199, '(rom)')	returns cxcix
num(199, '(ROM)')	returns CXCIX

Dates

You can use the following symbols to format a date. Arbitrary separators can be used.

Symbols to format a date

Symbols	Description
D	To describe the day, use the symbol "D" for each digit.

4 Loading and transforming data with scripting

Symbols	Description
M	To describe the month number, use the symbol "M". Use "M" or "MM" for one or two digits. "MMM" denotes short month name in letters as defined by the operating system or by the override system variable MonthNames in the script. "MMMM" denotes long month name in letters as defined by the operating system or by the override system variable LongMonthNames in the script.
Y	To describe the year, use the symbol "Y" for each digit.
W	To describe the weekday, use the symbol "W". "W" will return the number of the day (for example 0 for Monday) as a single digit. "WW" will return the number with two digits (e.g. 02 for Wednesday). "WWW" will show the short version of the weekday name (for example Mon) as defined by the operating system or by the override system variable DayNames in the script. "WWWW" will show the long version of the weekday name (for example Monday) as defined by the operating system or by the override system variable LongDayNames in the script.

Examples: (with 31st March 2013 as example date)

Examples of date formats

Example	Result
YY-MM-DD	describes the date as 13-03-31.
YYYY-MM-DD	describes the date as 2013-03-31.
YYYY-MMM-DD	describes the date as 2013-Mar-31.
DD MMMM YYYY	describes the date as 31 March 2013.
M/D/YY	describes the date as 3/31/13.
W YY-MM-DD	describes the date as 6 13-03-31.
WWW YY-MM-DD	describes the date as Sat 13-03-31.
WWWW YY-MM-DD	describes the date as Saturday 13-03-31.

Times

You can use the following symbols to format a time. Arbitrary separators can be used.

4 Loading and transforming data with scripting

Symbols to format a time

Symbols	Description
h	To describe the hours, use the symbol "h" for each digit.
m	To describe the minutes, use the symbol "m" for each digit.
s	To describe the seconds, use the symbol "s" for each digit.
f	To describe the fractions of a second, use the symbol "f" for each digit.
tt	To describe the time in AM/PM format, use the symbol "tt" after the time.

Examples: (with 18.30 as example time):

Examples of time formats

Example	Result
hh:mm	describes the time as 18:30
hh.mm.ss.ff	describes the time as 18.30.00.00
hh:mm:tt	describes the time as 06:30:pm

Time stamps

The same notation as that of dates and times above is used in time stamps.

Examples: (with 31th March 2013 18.30 as example time stamp):

Examples of time stamp formats

Example	Result
YY-MM-DD hh:mm	describes the time stamp as 13-03-31 18:30.
M/D/Y hh.mm.ss.ffff	describes the time stamp as 3/31/13 18.30.00.0000.

4.5 Working with QVD files

A QVD (QlikView Data) file is a file containing a table of data exported from Qlik Sense. QVD is a native Qlik format and can only be written to and read by Qlik Sense or QlikView. The file format is optimized for speed when reading data from a script but it is still very compact. Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.

QVD files can be read in two modes: standard (fast) and optimized (faster). The selected mode is determined automatically by the script engine.

There are some limitations regarding optimized loads. It is possible to rename fields, but any of the operations mentioned here will disable the optimized load and result in a standard load.

- Any transformations on the fields that are loaded.
- Using a **where** clause causing Qlik Sense to unpack the records.
- Using **Map** on a field that is loaded.

Purpose of QVD files

QVD files can be used for many purposes and more than one may apply in any given situation. At least four major uses can be easily identified:

- Increasing load speed
By buffering non-changing or slowly-changing blocks of input data in QVD files, script execution becomes considerably faster for large data sets.
- Decreasing load on database servers
The amount of data fetched from external data sources can also be greatly reduced. This reduces the workload on external databases and network traffic. Furthermore, when several scripts share the same data, it is only necessary to load it once from the source database into a QVD file. Other apps can make use of the same data through this QVD file.
- Consolidating data from multiple apps
With the **binary** script statement, data can be loaded from a single app into another app, but with QVD files a script can combine data from any number of apps. This makes it possible for apps to consolidate similar data from different business units, for example.
- Incremental
In many common cases, the QVD functionality can be used for incremental load by loading only new records from a growing database.

Creating QVD files

A QVD file can be created in two ways:

- Explicit creation and naming using the **store** command in the script. State in the script that a previously-read table, or part thereof, is to be exported to an explicitly-named file at a location of your choice.
- Automatic creation and maintenance from script. When you precede a **LOAD** or **SELECT** statement with the **buffer** prefix, Qlik Sense will automatically create a QVD file, which, under certain conditions, can be used instead of the original data source when reloading data.

There is no difference between the resulting QVD files with regard to reading speed.

Reading data from QVD files

A QVD file can be read or accessed by the following methods:

- Loading a QVD file as an explicit data source. QVD files can be referenced by a **LOAD** statement in the script, just like any other type of text files (csv, fix, dif, biff etc).

For example:

- `LOAD * from xyz.qvd (qvd)`
- `LOAD Name, RegNo from xyz.qvd (qvd)`

- LOAD Name as a, RegNo as b from xyz.qvd (qvd)
- Automatic loading of buffered QVD files. When you use the **buffer** prefix on **LOAD** or **SELECT** statements, no explicit statements for reading are necessary. Qlik Sense will determine the extent to which it will use data from the QVD file as opposed to acquiring data using the original **LOAD** or **SELECT** statement.
- Accessing QVD files from the script. A number of script functions (all beginning with **qvd**) can be used for retrieving various information on the data found in the XML header of a QVD file.

QVD format

A QVD file holds exactly one data table and consists of three parts:

- Header.



If the QVD file was generated with QlikView the header is a well-formed XML header (in UTF-8 char set) describing the fields in the table, the layout of the subsequent information and other metadata.

- Symbol tables in a byte-stuffed format.
- Actual table data in a bit-stuffed format.

4.6 Configuring analytic connections in Qlik Sense Desktop

With analytic connections you are able to integrate external analysis with your business discovery. An analytic connection extends the expressions you can use in load scripts and charts by calling an external calculation engine (when you do this, the calculation engine acts as a server-side extension (SSE)). For example, you could create an analytic connection to R, and use statistical expressions when you load the data.

For Qlik Sense Desktop, the configuration must be done in the *Settings.ini* file.

Do the following:

1. Open the file *Settings.ini*.

For Qlik Sense Desktop it is located in *C:/Users/<User ID>/Documents/Qlik/Sense/* or in *C:/Users/AppData/Local/Programs/Qlik/Sense/Engine*.

For Qlik Sense it is found in: *C:/ProgramData/Qlik/Sense/Engine/*.

2. Add the following configuration (note the empty line at the end):

```
[settings 7]  
SSEPlugin=<PluginConfig>[;<PluginConfig>...]
```

Where *<PluginConfig>* is a comma-separated list of configuration elements containing the following:

```
<EngineName>,<Address>[,<PathToCertFile>,<RequestTimeout>,<ReconnectTimeout>]
```



After adding new connections or changing existing connections, a restart of Qlik Sense Desktop is required for the changes to take effect.



Note that the server-side extension (SSE) plugin server must be running before you start Qlik Sense, otherwise the connection will not be established.

Qlik open source SSE repositories

The following two Qlik SSE repositories are open source:

- <https://github.com/qlik-oss/server-side-extension>
Contains the SSE protocol, general documentation, and examples written in Python and C++.
- <https://github.com/qlik-oss/sse-r-plugin>
Contains an R-plugin written in C#, only the source code. You must create the plugin before it can be used.

Description of the elements

<EngineName>: Mapping/alias to the plugin that will be used from within the expressions in the app using the plugin functions, for example, *SSEPython* for a Python plugin.

<Address>: colon-separated list with two elements, and

- <Host>: DNS name (or IP-address) of the plugin.
- <Port>: Port on which the plugin listens, typically 50051.

<PathToCertFile>: File system path to folder containing client certificates required for secure communication with the plugin. Optional. If omitted, insecure communication will be invoked. This path just points to the folder where the certificates are located. You have to make sure that they are actually copied to that folder. The names of the three certificate files must be the following: *root_cert.pem*, *sse_client_cert.pem*, *sse_client_key.pem*. Only mutual authentication (server and client authentication) is allowed.

<RequestTimeout>: Integer (seconds). Optional. Default value is 0 (infinite). Timeout for message duration.

<ReconnectTimeout>: Integer (seconds). Optional. Default value is 20 (seconds). Time before the client tries to reconnect to the plugin after the connection to the plugin was lost.

Examples:

- Example where one SSE plugin server is defined: `SSEPlugin=SSEPython,localhost:50051`
- Example where two SSE plugin servers are defined:
`SSEPlugin=SSEPython,localhost:50051;R,localhost:50053`
- Example where one SSE plugin server is defined without certificate path but with timeouts set:
`SSEPlugin=SSEPython,localhost:50051,,0,20`

5 Managing data security with Section Access

Section Access is used to control the security of an application. It is basically a part of the data load script where you add a security table to define who gets to see what. Qlik Sense uses this information to reduce data to the appropriate scope when the user opens the application, that is, some of the data in the app is hidden from the user based on their identity. Section Access is tightly integrated with the data in the app and relies upon it to control access. This form of dynamic data reduction can target table rows, columns, or a combination of both. For more information, see [Trust and Security at Qlik](#).

5.1 Sections in the load script

Data access control is managed through one or more security tables loaded in the same way that data is normally loaded. This makes it possible to store these tables in a standard database or in a spreadsheet. The script statements managing the security tables are given within an authorization section, which in the script is initiated by the statement `Section Access`.

If an authorization section is defined in the script, the part of the script loading the app data must be put in a different section, initiated by the statement `Section Application`.

Example:

```
Section Access;
Load * INLINE [
    ACCESS,   USERID,           REDUCTION
    USER,     AD_DOMAIN\ADMIN,   *
    USER,     AD_DOMAIN\A,       1
    USER,     AD_DOMAIN\B,       2
    USER,     AD_DOMAIN\C,       3
    ADMIN,    INTERNAL\SA_SCHEDULER,
];
Section Application;
T1:
Load *, 
NUM AS REDUCTION;
LOAD
Chr(RecNo()+ord('A')-1) AS ALPHA,
RecNo() AS NUM
AUTOGENERATE 3;
```

Note that after making changes to the load script, you must always reload the data for the changes to take effect.

Section Access system fields

The access levels are assigned to users in one or more security tables loaded within the Section Access part of the script. These tables must contain, as a minimum, two system fields: ACCESS, which is the field defining the access level, and USERID or USER.EMAIL . Other optional system fields can be added depending on the use case. The full set of Section Access system fields is described below.

ACCESS

Defines what access the corresponding user should have.

Access to Qlik Sense apps can be authorized for specified users. In the security table, users can be assigned to the access levels ADMIN or USER. A user with ADMIN privileges has access to all data in the app unless limited by the security table. A user with USER privileges can only access data as defined in the security table. If no valid access level is assigned, the user cannot open the app.

If Section Access is used in a reload scenario, INTERNAL\SA_SCHEDULER, which is the scheduler service user, needs ADMIN access to perform reloads. For example:

```
Section Access;
LOAD * inline [
    ACCESS,    USERID
    ADMIN,     INTERNAL\SA_SCHEDULER
];

```

If you do not want to use the INTERNAL\SA_SCHEDULER account, see *Using impersonation to reload data* (page 171) for an alternate method.

If Section Access is used in an on-demand app generation (ODAG) scenario in the template app, the INTERNAL\SA_API user must be included as ADMIN in the Section Access table. For example:

```
Section Access;
LOAD * inline [
    ACCESS,    USERID
    ADMIN,     INTERNAL\SA_API
];

```

USERID

Contains a string corresponding to a Qlik Sense domain name and user name. Qlik Sense will get the login information from the proxy service and compare it to the value in this field.

A wildcard character (*) is interpreted as all users, subject to further conditions specified in the security table. For example, in the following security table, users who are in the Qlik Sense Tenant Admins can see all listed REDUCTION values.

```
Section Access;
LOAD * INLINE [
    ACCESS,    USERID,          GROUP,          REDUCTION
    ADMIN,     *,              Qlik Sense Tenant Admins, *
    USER,      QLIK-POC\SOMEOTHERUSER1,  *,          1
    USER,      QLIK-POC\SOMEOTHERUSER2,  *,          2
    ...
];

```



USERID and the NTNAME both use the same authentication information, so it is not necessary to check both on the same row in the security table. The difference between the two fields is that NTNAME checks groups also.

NTNAME



NTNAME is a legacy QlikView field and it is recommended to use USERID if QlikView is not using the same security table.

A field that should contain a string corresponding to a Windows NT Domain user name or group name. If a different authentication system is used, it should contain the name of an authenticated user. Qlik Sense will fetch the login information from the OS and compare it to the value in this field.

GROUP

Contains a string corresponding to a group in Qlik Sense. Qlik Sense will resolve the user supplied by the proxy service against this group.

SERIAL



SERIAL is a legacy QlikView field and is not used if you are only using Qlik Sense.

Contains a string corresponding to the platform. If the field contains the string ‘QLIKSENSE’ or a wildcard ‘*’, access may be granted, depending on the other fields in the security table.



If the field SERIAL contains a license number the Section Access row will deny access to the document. This setting is only valid in QlikView.

OMIT

Contains the name of the field that is to be omitted for this specific user. Wildcards may be used and the field may be empty.



It is recommended that you do not apply OMIT on key fields. Key fields that are omitted are visible in the data model viewer, but the content is not available, which can be confusing for a user. Additionally, applying OMIT on fields that are used in a visualization can result in an incomplete visualization for users that do not have access to the omitted fields.

5.2 Managing user access to an app

Section access, in its simplest form, can be used to restrict specific users from accessing an app. Users are denied access to an app through exclusion. In other words, if a specific user ID is not listed in the security table, they will not be able to access the app. The only exception to this rule is if a wildcard (*) is assigned to the USERID field in one of the rows in the security table. A wildcard, in this case, means that all authenticated users can access the app. Here is an example of a security table with a list of user IDs:

```
Section Access;  
LOAD * inline [  
    ACCESS,    USERID
```

```
ADMIN,    AD_DOMAIN\ADMIN
USER,     AD_DOMAIN\A
USER,     AD_DOMAIN\B
];
Section Application;
```

5.3 Managing user access to specific data in an app

Dynamic data reduction limits access to rows and columns in the data tables within Qlik Sense apps after a user has been authorized to access the app itself.

Managing access to row-level data

Restrict access to row-level data by adding a data reduction column to the security table in the access section of the load script. Specific records (rows) can be hidden from users by linking the Section Access data with the real data. The selection of data to be shown or excluded is controlled by having one or more reduction fields with common names in Section Access and Section Application parts of the script. After user login, Qlik Sense matches the selections in reduction fields in the access section to any fields in the application section with exactly the same field names (the field names must be written in uppercase). After the selections have been made, Qlik Sense permanently hides all data excluded by these selections from the user. If a wildcard (*) is used as a field value in the data reduction column, it is interpreted as allowing the user to access records associated with all selected reduction fields in the security table.

When Qlik Sense is comparing the reduction field in Section Access to fields in the data model, the following behaviors are expected:

- If a field value in the data model matches the reduction field in Section Access, the app will open showing data associated with the match for the specified user. Other data will be hidden.
- If the reducing field value does not match any of the values in the data model, the app will not open for a normal USER. It will, however, open unreduced for a user marked as ADMIN.

Using several reducing fields in Section Access is not recommended, since it will allow other access combinations than the intended ones.



*The wildcard character * in the data reduction column refers only to all values in the security table. If there are values in Section Application that are not available in the reduction column of the security table, they will be reduced.*



All field names used in the transfer described above and all field values in these fields must be uppercase because all field names and field values are, by default, converted to uppercase in the access section.



By default, if you want to enable reload of the script in a Qlik Management Console task, the INTERNAL\SA_SCHEDULER account user with ADMIN access is required. If you do not want to use the INTERNAL\SA_SCHEDULER account, see [Using impersonation to reload data \(page 171\)](#) for an alternate method.

Example: Row-level data reduction based on user identity

```
Section Access;
Authorization:
LOAD * inline [
    ACCESS,   USERID,           REDUCTION
    ADMIN,    AD_DOMAIN\ADMIN,   *
    USER,     AD_DOMAIN\A,      1
    USER,     AD_DOMAIN\B,      2
    USER,     AD_DOMAIN\C,      *
    ADMIN,    INTERNAL\SA_SCHEDULER, *
];
Section Application;
T1:
LOAD *,
NUM AS REDUCTION;
LOAD
RecNo() AS NUM
AUTOGENERATE 3;
```

In this example, the field REDUCTION (uppercase) now exists in both Section Access and Section Application (all field values are also uppercase). The two fields would normally be different and separated, but using Section Access, these fields are linked and the number of records displayed to the user is reduced.

The result will be:

- User ADMIN can see all fields and only those records other users can see when REDUCTION = 1 or REDUCTION =2.
- User A can see all fields, but only those records associated with REDUCTION=1.
- User B can see all fields, but only those records associated with REDUCTION=2.
- User C can see all fields and only those records other users can see when REDUCTION = 1 or REDUCTION =2.

Managing access to column-level data

Restrict access to column-level data by adding the OMIT system field to the security table in the Section Access script. The following example builds on the previous example where row data reduction is already in place.

Example: Column data reduction based on user identity

```
Section Access;
LOAD * inline [
    ACCESS,   USERID,           REDUCTION,   OMIT
    ADMIN,    AD_DOMAIN\ADMIN,   *,           *
    USER,     AD_DOMAIN\A,      1,           1,
```

```
USER,      AD_DOMAIN\B,          2,          NUM
USER,      AD_DOMAIN\C,          3,          ALPHA
ADMIN,     INTERNAL\SA_SCHEDULER, *,          ,
];
Section Application;
T1:
LOAD *,          NUM AS REDUCTION;
LOAD
Chr( RecNo()+ord('A')-1) AS ALPHA,
RecNo() AS NUM
AUTOGENERATE 3;
```

The field OMIT in Section Access defines the fields that should be hidden from the user.

The result will be:

- User ADMIN can see all fields and only those records other users can see in this example when REDUCTION is 1, 2, or 3.
- User A can see all fields, but only those records associated with REDUCTION=1.
- User B can see all fields except NUM, and only those records associated with REDUCTION=2.
- User C can see all fields except ALPHA, and only those records associated with REDUCTION=3.



Some visualizations have minimum data requirements that must be met in order to render. As a result, "Incomplete visualization" might be displayed when a column-level field is omitted from a user's data view.

Managing access to user groups

Section Access offers you the option to limit the scope of data visible to users through group membership. To restrict your data using user groups, add the GROUP field name to the security table in the access section and define values for the GROUP field.

Example: Data reduction based on user groups

```
Section Access;
LOAD * inline [
    ACCESS,   USERID,          GROUP,    REDUCTION,  OMIT
    USER,      *,              ADMIN,    *,           ,
    USER,      *,              A,        1,           ,
    USER,      *,              B,        2,           NUM
    USER,      *,              C,        3,           ALPHA
    USER,      *,              GROUP1,   3,           ,
    ADMIN,     INTERNAL\SA_SCHEDULER, *,           *,
];

section application;
T1:
LOAD *,          NUM AS REDUCTION;
LOAD
Chr( RecNo()+ord('A')-1) AS ALPHA,
```

```
RecNo() AS NUM  
AUTOGENERATE 3;
```

The result will be:

- Users belonging to the ADMIN group are allowed to see all data and all fields.
- Users belonging to the A group are allowed to see data associated with REDUCTION=1 across all fields.
- Users belonging to the B group are allowed to see data associated with REDUCTION=2, but not in the NUM field
- Users belonging to the C group are allowed to see data associated with REDUCTION=3, but not in the ALPHA field
- Users belonging to the GROUP1 group are allowed to see data associated with REDUCTION=3 across all fields

Qlik Sense compares the user with UserID and resolves the user against groups in the table. If the user belongs to a group that is allowed access, or the user matches, they can access the app.

5.4 Using impersonation to reload data

By default, the internal system account, SA_SCHEDULER, is used to run reload tasks. This account has elevated privileges and, technically, can use any data source. There is a setting, however, in the QMC that uses impersonation to run reload tasks with the permissions of the app owner instead of the internal system account. By configuring this setting, the app owner and not SA_SCHEDULER is used for reloads, meaning that you do not add SA_SCHEDULER in the Section Access table but instead add the app owner. Within a task chain, apps can have different owners with permissions to sources dependent on each owner's access rights.

5.5 Managing user access in a multi-cloud environment

A Qlik Sense multi-cloud environment involves a mix of user authentication mechanisms. Typically, with Qlik Sense Enterprise on Windows, the USERID in the Section Access security table is verified by the proxy service. In Qlik Cloud, an Identity Provider assumes that authentication role. Consequently, Section Access that is set up for an on-premises environment such as Qlik Sense Enterprise on Windows will not work in a cloud environment.

When using an OIDC Identity Provider (Qlik IdP or custom IdP) with Qlik Cloud, the `subject claim` is used to identify users when logging in. With Section Access, the value of the USERID field in the security table is compared to the value of the `subject claim`. When you set up your tenant, make sure that the SAM account name is mapped to the `subject claim` of your identity provider. So, for example, if your SAM account name is AD_DOMAIN\Dev, set the `subject claim` to AD_DOMAIN\Dev. If you want to see the value of the `subject claim` of the IdP, append `/api/v1/diagnose-claims` to the tenant URL in the browser, for example, `your-tenant.us.qlikcloud.com/api/v1/diagnose-claims`. In the JSON response, the `subject claim` is called `sub`.

If you are unable to use the SAM account name, there is an alternate way to authenticate a user. Since e-mail addresses tend to remain the same in different environments, you can use the USER.EMAIL field instead of USERID in the security table. Here is an example of what the security table could look like:

ACCESS	USERID	USER.EMAIL	Comment	COUNTRY
USER	ABC\Joe	*	Access-on-prem	United States
USER	*	joe.smith@example.com	Access-in-cloud	United States
USER	ABC\Ursula	*	Access-on-prem	Germany
USER	*	ursula.schultz@example.com	Access-in-cloud	Germany
USER	ABC\Stefan	*	Access-on-prem	Sweden
USER	*	stefan.svensson@example.com	Access-in-cloud	Sweden

Authorization script:

```
Section Access;
LOAD * INLINE [
    ACCESS, USERID,     USER.EMAIL,          COUNTRY
    USER,   ABC\Joe,      *,                 United States
    USER,   *,           joe.smith@example.com, United States
    USER,   ABC\Ursula,   *,                 Germany
    USER,   *,           ursula.schultz@example.com, Germany
    USER,   ABC\Stefan,   *,                 Sweden
    USER,   *,           stefan.svensson@example.com, Sweden
];

```

Note that each user has two records: One for on-premises access and one for cloud access. The wildcards ensure that only the relevant authenticating fields are used. In this example, COUNTRY is used as a data reduction field.

5.6 Using Section Access and Insight Advisor Chat

To make apps using section access to be available in Insight Advisor Chat, you must ensure the following service users have admin access in the section access script:

- INTERNAL/sa_repository: This makes the section access script available with the repository service for controlling user access.
- INTERNAL/sa_scheduler: This allows the app to reload using QMC tasks.



If you have sensitive information in app names, field names, or master item names, these may be exposed by making apps using Section Access available for Insight Advisor Chat. App suggestions for queries include app in streams to which users have access. These may include apps to which users do not have access in an app's Section Access. Selecting these apps will do nothing, however. When clicking **Dimensions** or **Measures** to view the available items from an app using Section Access, users may see items to which they do not have access. Clicking on these items will not provide any data to the users, however.

For example:

```
Section Access;
LOAD * inline [
```

```
USERID ,ACCESS  
INTERNAL\sa_repository ,ADMIN  
INTERNAL\sa_scheduler ,ADMIN  
DOMAINNAME\user1 ,ADMIN  
DOMAINNAME\user2 ,USER  
DOMAINNAME\user3 ,USER  
];
```

Once these users are in the Section Access script, you can make the app available for Insight Advisor Chat.

Once the app is reloaded, the app will be available in Insight Advisor Chat.

5.7 Guidelines and tips for using Section Access

Here are some important facts and helpful hints to know about Section Access.

- All the field names and values listed in **LOAD** or **SELECT** statements in the access section must be written in uppercase. Convert any field name containing lowercase letters in the database to uppercase using the **Upper** function before reading the field by the **LOAD** or **SELECT** statement.
- You cannot use the Section Access system field names listed as field names in your data model.
- Apps must be published before Section Access controls will be applied. Reloading the app does not apply any new or changed Section Access scripts.
- A snapshot shows data according to the access rights of the user who takes the snapshot, and the snapshot can then be shared in a story. However, when users return to a visualization from a story to see the live data in the app, they are restricted by their own access rights.
- Do not assign colors to master dimension values if you use section access or work with sensitive data, because the values might be exposed by the color configuration.
- To avoid exposing restricted data, remove all attached files with section access settings before publishing the app. Attached files are included when the app is published. If the published app is copied, the attached files are included in the copy. However, if section access restrictions have been applied to the attached data files, the section access settings are not retained when the files are copied, so users of the copied app will be able to see all the data in the attached files.
- A wildcard (*) is interpreted as all (listed) values of the field in the table. If used in one of the system fields (USERID, GROUP) in a table loaded in the access section of the script, it is interpreted as all (also not listed) possible values of this field.
- Security fields can be put in different tables.
- When loading data from a QVD file, the **Upper** function slows down the loading speed.
- If you have locked yourself out of an app by setting Section Access, you can open the app without data, and edit the access section in the data load script. This requires that you have access to edit and reload the data load script.
- A binary load will cause the access restrictions to be inherited by the new Qlik Sense app.

6 Managing big data with on-demand apps

On-demand apps enable you to load and analyze big data sources in Qlik Sense Enterprise. Trying to analyze an entire big data store at one time is highly inefficient. Nevertheless, to make representative visualizations, all the data must be discoverable. Qlik Sense on-demand apps give users aggregate views of big data stores and allow them to identify and load relevant subsets of the data for detailed analysis.

On-demand apps expand the potential use cases for data discovery, enabling business users to conduct associative analysis on larger data sources. They allow users to first select data they are interested in discovering insights about and then interactively generate an on-demand app with which they can analyze the data with the full Qlik in-memory capabilities.

6.1 On-demand app components

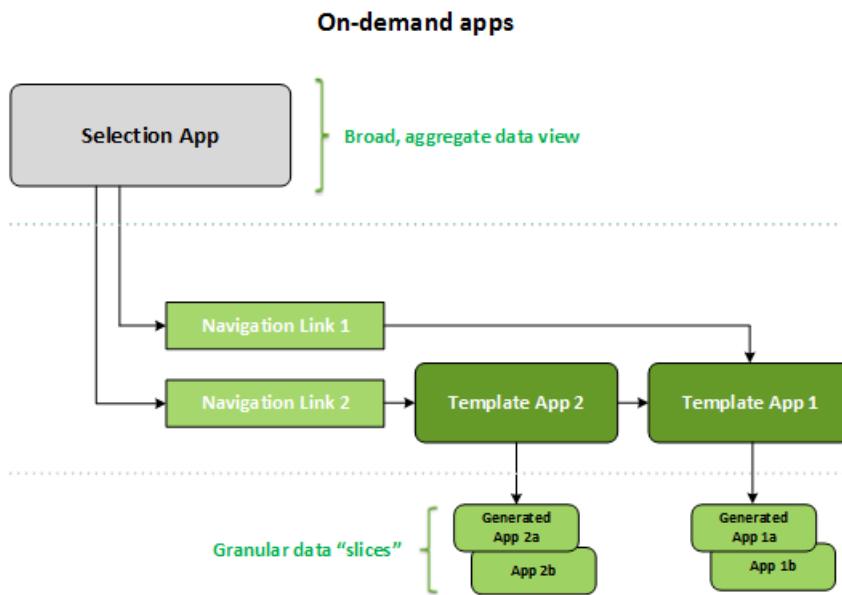
Qlik Sense manages the loading of big data sources with selection apps that provide aggregated views of the big data and also enable a user to zoom in and analyze finer-grained data. Embedded in each selection app are on-demand app navigation links to one or more template apps used as the basis for creating on-demand apps. Properties of the navigation links and template apps enable you to tightly control the shape and volume of data loaded into on-demand apps.

Apps can be generated repeatedly from the template app to track frequently changing data sets. While the data is filtered according to selections made in the selection app, the on-demand app content is dynamically loaded from the underlying data source. The same on-demand app can be generated multiple times to make fresh analyses of the data as they change.



On-demand app generation is controlled by the On-demand app service. The service is disabled by default and must be enabled before selection and template apps can be linked and on-demand apps generated. The On-demand app service is managed in the Qlik Management Console.

Relations between on-demand app components.



6.2 Constructing on-demand apps

Because on-demand selection and template apps require special load scripting, they are usually created by users with experience writing Qlik Sense load scripts. On-demand selection apps, for example, must load data with a modest level of dimension granularity. On-demand template apps contain load scripts with data binding expressions used to formulate the queries made on the data sources.

A selection app can be linked to multiple template apps, and a single template app can be linked to by multiple selection apps. But the template app's data binding expressions must correspond to fields in the selection apps that link to it. For that reason, selection and template apps tend to be created in conjunction with one another and often by the same experienced script writer.



There are sample on-demand selection and template apps included in the Qlik Sense Enterprise installation at ProgramData\Qlik\Examples\OnDemandApp\sample.

Creating navigation links also requires an understanding of the fields in the selection app that have corresponding bindings in the template app. That is because each navigation link requires an expression that computes the total number of detail records. That total represents the aggregate records accessible by way of the selection state in the selection app. To create that expression requires that the user know how to compute the template app's total record count using fields available in the selection app.

Using selection apps to generate on-demand apps does not require a user to understand the load script. Once an on-demand app navigation link has been created, a user can drag that navigation link onto the selection app's **App navigation** bar to create an app navigation point. On-demand apps are then generated from the navigation point.

Navigation points become available for on-demand app generation when the maximum row calculation from the expression in the navigation link comes within the required range. At that point, the user can generate an on-demand app. The user can also make another set of selections and generate additional apps based on those different selections.

Navigation links have a limit on the number of on-demand apps that can be generated from the link. When the maximum number of apps has been generated, the user who is generating apps from the navigation point must delete one of the existing apps before generating a new on-demand app. The maximum number of generated apps applies to the on-demand app navigation link. If one on-demand app navigation point is created from the navigation link, then that navigation point would be able to create up to the maximum number. When multiple navigation points are created from the same navigation link, together those navigation points are limited to the maximum number set for the navigation link.

Navigation links also set a retention time for generated apps. On-demand apps are automatically deleted when their retention period expires.

6.3 Publishing on-demand apps

Most users will use on-demand and selection apps after they have been published. When selection apps are published to a stream, users who have proper permission on that stream can use them to make aggregate selections and generate on-demand apps from the navigation points included with the selection apps. As with all published apps, they cannot be changed after they have been published. To add navigation points, for example, the user would have to make a copy of the selection app.

In many cases, users only use generated on-demand apps. Each generated app can be published separately. In fact, the app navigation link can specify that apps generated from it be published to a specific stream automatically. Users would then explore the selected slices of data loaded with those generated on-demand apps on the stream to which the app was published.

6.4 Advantages of on-demand apps

On-demand apps help business users and IT departments derive value from big data environments in numerous ways. On-demand apps:

- Provide users with a "shopping list" experience that enables them to interactively populate their apps with a subset of data such as time period, customer segment, or geography.
- Provide full Qlik Sense functionality on a latent subset that is hosted in memory.
In contrast, Direct Discovery, which can also manage large data sources, does not keep all relevant data in memory. With Direct Discovery, measure data resides at the source until execution.
- Enable IT to govern how large an app can be and invoke apps based on data volume or dimensional selections.
- Provide access to non-SQL data sources such as Teradata Aster, MapR, SAP BEx, and the PLACEHOLDER function in SAP HANA.
Performing non-SQL queries is in contrast to Direct Discovery, which can only be used with SQL data sources.
- Allow customizable SQL and load script generation.
- Allow section access in all cases.

6.5 Limitations

You cannot use Qlik NPrinting with on-demand apps.

6.6 Creating an on-demand selection app

An on-demand selection app provides the means for selecting subsets of large data sets so that the Qlik associative engine can make associations efficiently and effectively. In very large data volume environments, we recommend that you have the selection app load only a modest level of dimension granularity. For example, a selection app whose data is based on sales data aggregated by quarter, region, and product category could use an **SQL SELECT** statement such as the following:

```
SELECT SUM(S.UNIT_COST) AS TOTAL_UNIT_COST,  
      SUM(S.QUANTITY) AS TOTAL_QUANTITY,  
      SUM(S.UNIT_PRICE * S.QUANTITY) AS TOTAL_SALE,  
      SUM( (S.UNIT_PRICE - S.UNIT_COST) * QUANTITY) AS TOTAL_PROFIT,  
      SUM(1) AS TOTAL_LINE_ITEMS,  
      S.REGION,  
      S.YEARQUARTER,  
      S.PRODCAT,  
  FROM SALE_DETAIL S  
 GROUP BY S.REGION, S.YEARQUARTER, S.PRODCAT
```

The on-demand measure expression property is typically based on a computed aggregate result from an SQL GROUP BY query used to load the data. Because the selection app uses a **GROUP BY** query to aggregate the SALE_DETAIL records, an aggregation function--in this case **SUM**--must be used on the measure fields of UNIT_COST, QUANTITY and the computed values for TOTAL_SALE and TOTAL_PROFIT.

The **SUM(1) AS TOTAL_LINE_ITEMS** provides a way to precisely measure the total number of sale line items for every distinct combination of region, quarter, and product category. When creating a link used to produce on-demand apps, a measure expression must be supplied as way to control the number of records loaded into the on-demand apps. In the SALE_DETAIL example, when a user selects multiple product categories, regions, and/or quarters, a sum can be computed for TOTAL_LINE_ITEMS to determine whether or not the selection exceeds the record limit for the on-demand app.



There is a sample on-demand selection app included in the Qlik Sense Enterprise on Windows installation at ProgramData\Qlik\Examples\OnDemandApp\sample.

Record limits are specified when the selection app is linked to a template app to create an app navigation link. Each app navigation link has a record limit. Multiple navigation links can be created from the selection app. Multiple app navigation links are commonly made linking a selection app to different template apps in order to produce multiple views of data.

Individual on-demand app navigation links can be included in a selection app for publication. Once included in the selection app, an app navigation link is used to create one or more app navigation points that make it possible for users of specific sheets to create on-demand apps based on that link's template app.

6.7 Creating an on-demand template app

An on-demand template app is a regular Qlik Sense app with one important difference: its load script contains data binding expressions used to formulate the queries made on the data sources. These data binding expressions are used at on-demand app generation time to read values from the selection state of the selection app and bind them to the template app script so that the generated app is loaded with a user-controlled subset of the data.

The template app typically connects to the same data source as the selection app. The load script of a selection app typically loads aggregated data to reduce data volumes while still offering interactive visualizations of important dimensions and measures. The load script of a template app uses queries that load a controlled subset of more granular data.



An on-demand template app does not load data directly. The template app connection must be valid, but to test whether the connection works correctly, you must generate an on-demand app. When an on-demand app is generated, the load script is modified by the On-demand app service to load the selection state of the on-demand selection app. If the on-demand app generates without error, then you know the connection in template app work correctly.

Structure of a template app

A template app is linked to a selection app using an on-demand app navigation link. The app navigation link includes properties that control the number of records queried when the on-demand app is loaded with data. The load script of the template app contains data binding expressions that specify which field data from the selection app is used to formulate the queries issued when loading data into the on-demand app.



There is a sample on-demand template app included in the Qlik Sense Enterprise on Windows installation at ProgramData\Qlik\Examples\OnDemandApp\sample.



A new syntax for data binding expression was introduced in June 2020. The previous syntax and prefixes od_, ods_, odo_, odso_, and odb_ behave as before including quantity constraints, _n suffix, and format specifications. If your app should work on Qlik Sense versions prior to June 2020, use the old syntax. For the old syntax, see [Creating an on-demand template app \(old version\)](#).

The _n suffix is not supported when using the new prefixes.

Basic data binding expressions have the form:

`$(odag_FIELDNAME)`

The odag_ prefix is used to bind the selection state of the selection app to the load script of the on-demand app, which is created by copying the template app. The part of the data binding expression that follows the odag prefix must be a name that matches a field in the selection app. When the on-demand app is generated,

the current selection state of the selection app is used to obtain the desired values to bind for each field. Each occurrence of a **\$(odag_FIELDNAME)** expression in the load script of the newly created on-demand app is replaced with the list of values selected for the corresponding field in the selection state of the selection app.

Other prefixes for more specialized data binding are available. To learn more about tailoring for special cases and optimizing load statements, see *Binding expressions in on-demand template apps (page 180)*.

On-demand bindings can be inserted directly into **SELECT** and **WHERE** statements in your load script. When you add bindings directly into your **WHERE** statements, it is easy to combine them with other conditions in the statement.

You can add a placeholder variable **\$(odagActive)** when making your load script. This enables you to load sample data into the template app so that master charts for dynamic views can be created without loading all data.

The following examples illustrate a sample on-demand template load script.

Example: Adding some sample data

This example adds sample values so the app can be loaded even if the bindings are not complete.

```
IF '$(odagActive)'=''' THEN
trace ODAG variables not inserted! Loading sample data. ;
SET 'odag_Origin Code' = '''LAX''' ;
SET 'odag_Destination Code' = '''JFK''' ;
SET odagn_Year = 2015;
SET odag_Quarter = '''1''' ;
SET 'odag_Ticket Carrier Code' = '''CA''' ;
SET 'odag_Fare Class' = '''X''' ;
END IF;
```

Example: Loading data in the template app

The following is a sample load script for loading the sample data and filtering it with the generated **odag_FIELDNAME** bindings. The **odagn_<Field Name>** bindings pick the numbers in the duals and uses by default no quoting.

```
SQL SELECT *
FROM FlightDB.Flights
WHERE "Origin Code" IN $(odag_Origin Code)
AND "Destination Code" IN $(odag_Destination Code)
AND "Year" IN $(odagn_Year)
AND "Quarter" IN $(odag_Quarter)
AND "Ticket Carrier Code" IN $(odag_Ticket Carrier Code)
AND "Fare Class" IN $(odag_Fare Class);
```

Single Sign-On (SSO)

On-demand apps can use single sign-on (SSO) with data sources that support SSO. The engine and the data source must be configured to allow SSO.

Once the engine and data source have been configured for SSO, the template app must enable SSO by adding the following syntax to the template app script:

///!ODAG_SSO

The On-Demand App Service parses the script when an on-demand app is generated and each time it is reloaded.

When an on-demand app is loaded with SSO, the identity of the end user is sent to the data source. The end user must have access to the sources used in the template app's data connections. Only data that user has access to in those sources is loaded, even if a larger set of data is selected.



On-demand apps generated from template apps that use single sign-on (SSO) cannot be published.

Reload nodes for template apps

Administrators can control where on-demand apps are reloaded in a multi-node environment by setting load balancing rules on template apps. When a load balancing rule is set, all apps generated from links to the template app will be loaded according to the rule that applies to the template app.

Binding expressions in on-demand template apps

Data bindings in a template app specify which data from a corresponding selection app is used to formulate the queries issued when loading data into an on-demand app.

The basic form of binding expressions--**\$(odag_FIELDNAME)**--is not the only method available to bind expressions. Other prefixes can be used to refine selections and to ensure that the template app loads data correctly.



Template apps originally created using the Qlik Sense extension for On-demand App Generation should be changed to use the approach illustrated below for binding a large number of selections from a field.

Available binding prefixes

The general prefix form is **odag[s|o][n][cnt]** where:

- s - include only selected values
- o - include only optional values
- n - pick the numeric version, by default unquoted
- cnt - insert the number of values instead of actual values

The following table provides a list of all versions of the binding prefixes available. The examples assume a field named *MyField* with 1,2,3 as the selected values (green values) and 4 as an optional selected value (white value).

Binding prefixes

Prefix	Description	Example	Replaced with
odag_	Replaced by the selected (green) and optional (white) values. Picks the text version of the values. This is the standard prefix for string values.	<code>\$(odag_MyField)</code>	'1','2','3','4'
odagcnt_	Replaced by the number of values in the corresponding odag_ binding. This prefix is used for optimization of queries.	<code>\$(odagcnt_MyField)</code>	4
odagn_	Replaced by the selected (green) and optional (white) values. Picks the numeric version of the values. This is the standard prefix for numeric values.	<code>\$(odagn_MyField)</code>	1,2,3,4
	<p> <i>If the data model is such that there can be no selected or optional values of the field, a noValue must be specified in the expression. For example, \$(odagn_MyField){"noValue": "-99999"}.</i></p> <p><i>For more information, see Changing the value quotation and delimiter characters (page 184).</i></p>		
odagncnt_	Replaced by the number of values in the corresponding odagn_ binding. This is for optimization of queries.	<code>\$(odagncnt_MyField)</code>	4
odago_	Replaced by the optional (white) values. Picks the text version of the values. This is for optimization of queries.	<code>\$(odago_MyField)</code>	'4'
odagocnt_	Replaced by the number of values in the corresponding odago_ binding. This is for optimization of queries.	<code>\$(odagocnt_MyField)</code>	1
odagon_	Replaced by the optional (white) values. Picks the numeric version of the values. This is for optimization of queries.	<code>\$(odagon_MyField)</code>	4
odagoncnt_	Replaced by the number of values in the corresponding odagon_ binding. This is for optimization of queries.	<code>\$(odagoncnt_MyField)</code>	1

Prefix	Description	Example	Replaced with
odags_	Replaced by the selected (green) values. Picks the text version of the values. This is for optimization of queries. <i>Optimizing for large database (page 182).</i>	<code>\$(odags_ MyField)</code>	'1','2','3'
odagscnt_	Replaced by the number of values in the corresponding odags_ binding. This is for optimization of queries.	<code>\$(odagscnt_ MyField)</code>	3
odagsn_	Replaced by the selected (green) values. Picks the numeric version of the values. This is for optimization of queries. <i>Optimizing for large database (page 182).</i>	<code>\$(odagsn_ MyField)</code>	1,2,3
odagsncnt_	Replaced by the number of values in the corresponding odagsn_ binding. This is for optimization of queries.	<code>\$(odagsncnt_ MyField)</code>	3



Empty values are filtered away in the text versions. Non numeric and NaN values are filtered away in the numeric versions.

Optimizing for large database

The odags_ and odagsn_ prefixes are intended for optimization of queries. When there are no selections in the bound fields, odag_ includes all the values while odags_ includes no values. In some cases, it is more efficient to use the odags_ and odagscnt_ prefixes. This enables you to test if the set of values are empty. For example, the following is more efficient when no selections are made in *MyField* than testing for all values in *odag_ MyField*:

```
WHERE ($(odagscnt_MyField)=0 OR MyColumn IN ($(odags_MyField)))
```

odags_ cannot be used when there is an alternate field to select from in the selection app that is not an on-demand field. For example, if the user makes selections in *CountryName*, but the binding expression is on the associated field *CountryCode*, odags_ could not be used. In these cases, odago_ can be used instead. If there are no values in a odago_ binding, it could either mean that either all values should be included or that no values should be included.

Binding numeric values

When the data to be bound to the on-demand app consists of numbers instead of strings, it is useful to disable the quote wrapping behavior on the numeric fields. For example, if the sales records include a numeric DAY_OF_WEEK column and you want the user of the selection app to choose arbitrary combinations of DAY_OF_WEEK, you would augment the aggregation query used for loading the selection app to include DAY_OF_

WEEK in both the **SELECT** list as well as the **GROUP BY** list. If quotation marks are wrapped around DAY_OF_WEEK values when they are selected, a runtime query error could result if the database does not support automatic type conversion from string to numeric.

To handle this situation, you can use a numeric version of the binding expression suffix. This forces the field binding to use the numeric values from the selection app rather than string values. The following numeric version are available:

- odagn_
- odagon_
- odagsn_

By using numeric versions, the values are picked up from the numeric part in the duals that store selected values and the values are unquoted by default.

Requiring a certain number of selections

In some situations, it may be necessary to require that the on-demand app query contain a specific number or range of values for a specific field. For example, if the on-demand app's query contains a **BETWEEN** clause to obtain all sales between a start and end date, the bind expression for the YEARQUARTER field can have a suffix syntax of [2] that will require exactly two values be selected for YEARQUARTER, as in:

```
$(odag_YEARQUARTER) [2]
```

The on-demand app navigation point on the selection app will remain disabled as long as there are not exactly two values of YEARQUARTER selected. A message will display to indicate that exactly two values of YEARQUARTER must be selected.

Selection quantity constraints create a prerequisite linkage between the selection app and the on-demand app. This is different from bind expressions that do not use quantity constraints. For example, when the template app's script contains a bind expression without a quantity constraint, as in:

```
$(odag_MYFIELD)
```

there is no requirement that the selection app contain a field named MYFIELD nor for there to be any selected values of that field if it does exist. If the selection app does not contain a field named MYFIELD or if the user simply neglects to make any selections from it, the on-demand app navigation point can still become enabled when other selections are made to fulfill the record-limit value condition.

If on the other hand, the bind expression is:

```
$(odag_MYFIELD) [1+]
```

there are now two requirements placed on the selection app:

- The selection app must contain a field named MYFIELD.
- The user must select at least one value for MYFIELD.

This type of bind expression must be used carefully because it limits which selection apps can be used with the template app. You should not use this quantity constraint on bindings of a template app unless you are certain you want to impose that selection quantity requirement on all selection apps that link to that template app.

6 Managing big data with on-demand apps

To perform the data binding process, the On-demand app service uses a string substitution approach that is insensitive to comments in the script. This means you should not use bind expressions in comments unless you want those comments to contain the list of bound values following app generation.

Other quantity constraints are possible. The following table summarizes the different combinations of selection quantity constraints.

Different combinations of selection quantity constraints

Constraint pattern	Selection requirement
<code>\$(odag_YEARQUARTER)[2]</code>	Exactly 2 values of YEARQUARTER must be selected.
<code>\$(odag_YEARQUARTER)[2-4]</code>	Between 2 and 4 values of YEARQUARTER must be selected.
<code>\$(odag_YEARQUARTER)[2+]</code>	At least 2 values of YEARQUARTER must be selected.
<code>\$(odag_YEARQUARTER)[2-]</code>	At most 2 values of YEARQUARTER can be selected.



The check to determine if all the quantity constraints in the template app have been met is performed during the app generation process. If a quantity constraint is violated, the request to generate the app will be rejected and an error message will be displayed.

Changing the value quotation and delimiter characters

When a list of values from a field selected in a selection app is inserted into the script of a template app, the values are surrounded by single quotation marks and separated by commas. These are the default characters for quotations and delimiters. These values can be changed in syntax appended to binding statement for each field. For example:

```
$(odag_ORIGIN){"quote": "|", "delimiter": ";"}
```

These new values are then used when formulating the list of binding values taken from the selection app. For example, if the selected values are the first three months of the year, the list would be constructed as:

```
|January| ; |February| ; |March|
```

The default values for the quotation and delimiter characters work for most standard SQL databases. But they might not work for some SQL databases and do not work for many dynamic data sources such as NoSQL and REST. For those sources, you need append this binding expression to change the quotation and delimiter characters.

The following table outlines format parameters for changing quotation and delimiter characters.

6 Managing big data with on-demand apps

Format parameters		
Parameter	Default value	Definition
quote	' (single quote) for text prefixes empty for numeric prefixes	Will be added before and after each value
delimiter	, (comma)	Will be added between all values
quoteReplace	" (double single quotes)	When the value is not empty and the quotation is not empty, then all occurrences of the of the quote inside the values will be replaced by the specified string.
		 <i>quoteReplace is not supported as a parameter for numeric prefixes such as odagn_. quoteReplace is ignored by numeric prefixes.</i>
noValue	(empty)	<p>When there are no values selected for a field, this value will be used instead. This parameter is useful when there can be no values of a particular field in the selection.</p> <p>The value should be set to a value that does not exist in the source data. For numeric values, for example, use a negative value if all values in the database are positive.</p>
		 <i>For unquoted values, noValue must be specified if selected values of the field can be an empty set.</i>

The following tables outline the format specification and generated values for odag_ and odagn_. The generated inserted values are based on the default data values of VAL1, VAL2.

odag_ example format specifications and generated values

Format specification	Description	Generated inserted values
not specified	Comma separated list of values, quoted with ' .	'VAL1','VAL2'
{"quote": "", "delimiter": ""}	Concatenated values	VAL1VAL2
{"quote": "X", "delimiter": "Y"}	Values quoted by X and delimited by Y.	XVAL1XYXVAL2X

Format specification	Description	Generated inserted values
{"quote": "XX", "delimiter": "YY"}	Values quoted by XX and delimited by YY.	XXVAL1XXXXXXVAL2XX
{"quote": "X"}	Values quoted by X and delimited by , (default).	XVAL1X,XVAL2X
{"delimiter": "YY"}	Values quoted by ' (default) and delimited by YY	'VAL1'YY'VAL2'
{"quote": ""}	Unquoted values delimited by ,.	VAL1,VAL2
{"quote": "A", "quoteReplace": "\A"}	Values quoted by A and delimited by comma (default). A values inside the field will be replaced by \A.	AV\AL1A,AV\AL2A
 <i>In this example, there needs to be double since is the escape character in json format.</i>		

odagn_ example format specifications and generated values

Format specification	Description	Generated inserted values
not specified	Comma separated list of unquoted values. Note that the numeric values will be used.	VAL1,VAL2
{"delimiter": "YY"}	Unquoted values delimited by YY	VAL1YYVAL2
{"quote": "A", "quoteReplace": "\A"}	Compared to the odag_ prefix the quoteReplace parameter will be ignored.	AVAL1A,AVAL2A

Processing individual values

When individual processing of field values is required, you can use an inline method to generate values in the variable Values and perform arbitrary processing with **Replace** or another function. In the example below, **Replace** is used with placeholder values.

```
MyTempBindingData:
LOAD * INLINE [VAL
$(odag_MyField){"quote": "", "delimiter": "\n"};
];

(TempTable:
LOAD Concat(chr(39) & Replace(text, from_str, to_str) & chr(39), ',') as CombinedData Resident
MyTempBindingData;
LET values = Peek('CombinedData',0,'_TempTable');
drop table _TempTable;
drop table MyTempBindingData;
```

6.8 Building an on-demand app

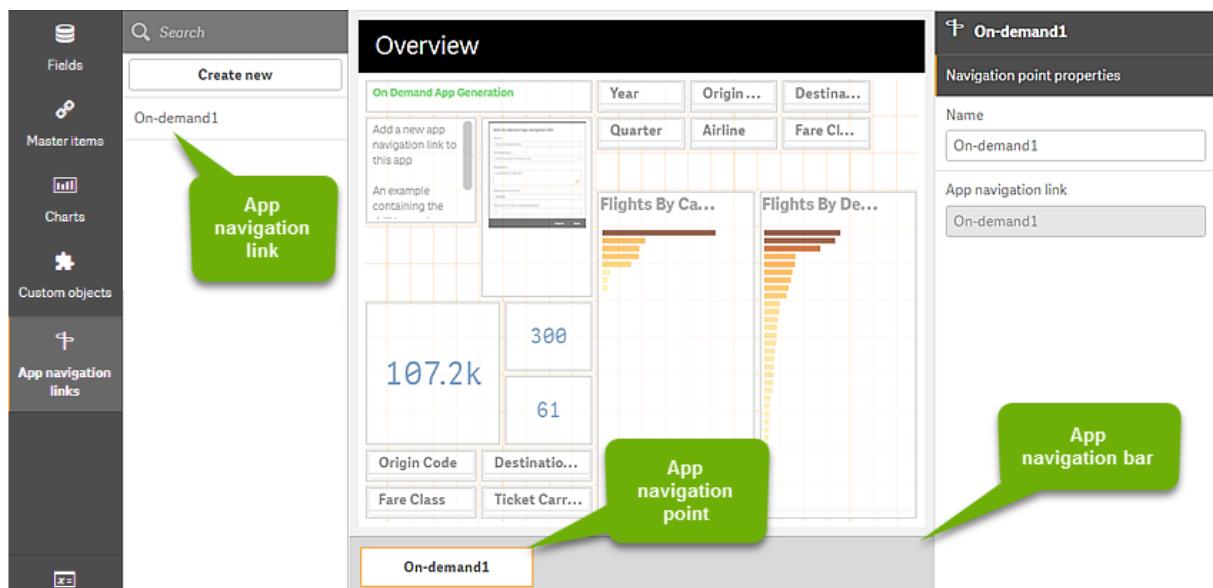
An on-demand app loads a subset of the data that is loaded in aggregate form by a selection app. An on-demand app is constructed by linking an on-demand selection app to an on-demand template app. Selection apps and template apps are the fundamental building blocks of on-demand apps.

To build an on-demand app, selection and template apps that can be linked together must first be created. To be linked, selection and template apps must have data fields in common that can be bound together.

A selection app can be linked to multiple template apps, and a single template app can be linked to by multiple selection apps. But the template app's data binding expressions must correspond to fields in the selection apps that link to it.

An on-demand app navigation link joins a selection app to a template app. On-demand app navigation links are created in selection apps. Once a navigation link has been defined, it can be added to the selection app's **App navigation** bar as an on-demand app navigation point. Each sheet in an app contains its own **App navigation** bar. Users then generate on-demand apps from the app navigation point.

Example the On-demand app building view.



Multiple on-demand apps, each containing a different combination of selected data, can be generated from the same app navigation point.

Pointers to a single app navigation link can be added to multiple sheets in the same selection app. Also, sheets can have multiple app navigation points, created from multiple app navigation links.

When a selection app is complete with navigation links and navigation points, on-demands can be generated.

Do the following:

1. Open an on-demand selection app and select **Edit**.
2. Select **App navigation links** from the panel on the left side.
3. Click the **Create new** button to open the **Create new On-demand app navigation link** dialog.
4. Name the new on-demand app navigation link.
5. Select an **On-demand template app**.

Not all the apps in the **Template app** drop-down list are valid template apps. You must select an app that has been constructed as a template app and whose data binding expressions correspond to fields in the selection app you are working with. Otherwise, the on-demand apps generated from the app navigation link will produce errors.
6. Write an expression that computes the total number of detail records that are represented by the aggregate records accessible by way of the selection state in the selection app.

The expression usually uses the **SUM** function to obtain a total of the records selected. The result is used to determine when the amount of data to load is within the range specified for generating the on-demand app.
7. Specify the **Maximum row count**.

The **Maximum row count** value sets the upper limit on the number of records, computed by the function in the **Expression** entry, that the on-demand app can load. As long as the number of records as computed by the row estimate expression in the selection app is greater than the **Maximum row count** value, the on-demand app cannot be generated. The app can only be generated when the number of records computed by the row estimate expression is at or below the upper limit set by the **Maximum row count** value.

To create the expression used for **Maximum row count**, you must know how the total record count is computed from fields available in the selection app.
8. Specify the **Maximum number of generated apps**.

Multiple on-demand apps can be generated from the same on-demand app navigation point on the selection app's **App navigation** bar. The reason for generating multiple apps is that each one can contain a different selection of data. When the maximum number of apps has been generated, the user who is generating apps from the navigation point must delete one of the existing apps before generating a new on-demand app.

The maximum number of generated apps applies to the on-demand app navigation link. If one on-demand app navigation point is created from the navigation link, then that navigation point would be able to create up to the maximum number. But if multiple navigation points are created from the same navigation link, then the total number of on-demand apps generated from those navigation points is limited to the setting for **Maximum number of generated apps**.
9. Enter a numeric value in the **Retention time** field for the length of time apps generated from the navigation link will be retained before they are deleted.
10. In the drop-down menu to the right of the **Retention time** field, select the unit of time for the retention period.

The options for retention time are hours, days, or **Never expires**.

All on-demand apps generated from the navigation link will be retained according to this setting. The age of a generated on-demand app is the difference between the current time and the time of the last data load. This calculation of an on-demand app's age is the same for published and unpublished apps. And if an on-demand app is published manually after it has been generated, the age calculation remains the same: it is based on the last data load of the generated app.



*There is also a retention time setting in the On-Demand App Service that applies to apps generated by anonymous users. That setting does not affect the retention time for users who are logged in with their own identity. For apps generated by anonymous users, the retention time is the shorter of the **Retention time** setting on the navigation link and the On-Demand App Service setting, which is set in the Qlik Management Console.*

11. In the **Default view when opened** drop-down menu, select the sheet to display first when the apps generated from the navigation link are opened.
You can select **App overview** or one of the sheets in the selection app from which the navigation link is created.
12. Select a stream from the **Publish to** drop-down menu where apps generated from the navigation link will be published.
You must have permission to publish on the stream you select. If you do not have Publish privileges on the selected stream, attempts to generate on-demand apps from the navigation link will fail.
When selecting a stream to publish generated apps to, you must be sure the intended users of the on-demand app have Read privileges on the stream.
You can also select **Not published (saved to workspace)** to save the generated apps in the users workspace without publishing them.



If anonymous users will be allowed to use a published selection app, the on-demand app navigation links should be configured to publish to a stream that anonymous users can access. If on-demand apps generated from the navigation link are not published automatically, anonymous users will get an error message when they try to generate those apps.

After an app has been generated, it can be published manually.

13. Click **Create** and the new on-demand app navigation link will appear in the list of **App navigation links**.
14. Drag the app navigation link to the **App navigation** bar on the selection app.
Dragging the app navigation link onto the selection app creates an on-demand app navigation point. The properties of the new on-demand app navigation point are displayed in the panel on the right side. You can change the name of the navigation point there if you wish.
15. Click **Done** in the sheet editor.
The on-demand selection app is now ready to use or publish. Users of the selection app will be able to generate on-demand apps from the navigation points on the **App navigation** bar in the selection app.

7 Managing data with dynamic views

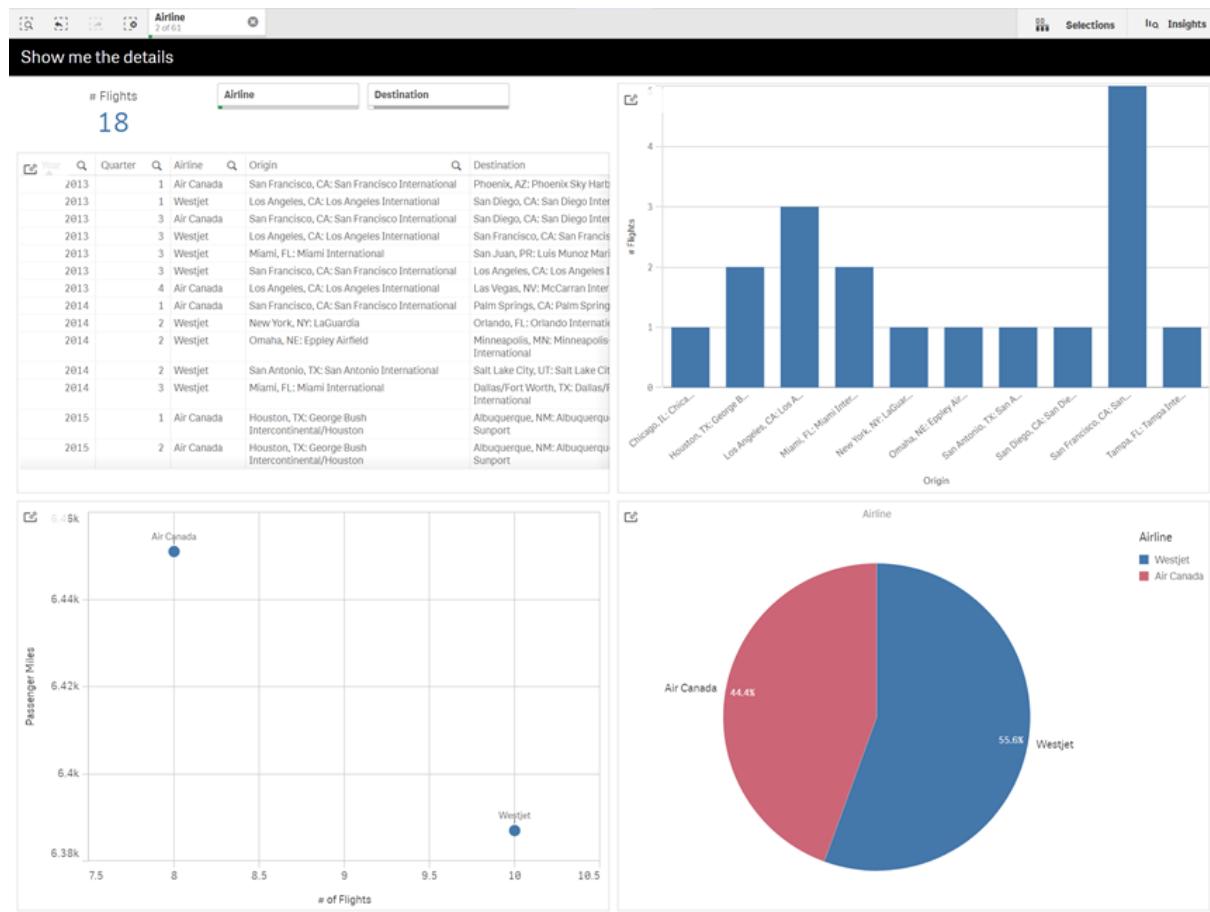
Dynamic views offer users the ability to directly control both the analytic sources they want to explore and when data is refreshed in visualizations.

Dynamic views enable you to query and view relevant subsets of large data sets in charts that can be dynamically refreshed as selections are made. This enables up-to-date visualizations for high data volume or fast-changing data scenarios.

7.1 Dynamic views overview

Dynamic views enable you to connect a base app to another app. Master visualizations from that app can then be used in the base app. This enables app creators to use master visualizations from the template app as dynamic charts in other apps. There is no limit to the number of dynamic views you can add to your base app.

Sheet view with dynamic view and dynamic charts



Dynamic views are made from three main components:

- Dynamic views: A mechanism added to base apps that connect to template apps and enable app creators to add master visualizations from the template app to the base app.

- Dynamic view template apps: A Qlik Sense app containing connections to data sources, such as cloud databases.
- Dynamic charts: Master visualizations in the dynamic view template app that can be added to base apps and that can be manually refreshed by users.

The template app and the base app do not need to use the same data. If you have a data set covering customer purchases, you could add a dynamic view to a template app containing weather data to look at any correlations.

If the data queried from the template app's source can be filtered using values in your base app, you can use binding expressions in the template app's script. This enables the dynamic view to only query a subset of data specifically related to the selections in the base app from the data sources of the template app. For example, you could bind the field SalesDate in the base app to the field DailyTemperatureReadingDate in the template app.

This subset capability is useful if your base app contains aggregated data and the dynamic view data is from the same source but is more granular than the base app data (e.g. the base app contains sales by month and product brand while the template app contains sales by day and product name). For more information on adding binding expressions to template apps, see *Binding expressions in on-demand template apps (page 180)*.

Dynamic views can be used with any kind of data. Dynamic views are particularly useful when dealing with large volumes of data or fast changing data scenarios, where it is better to perform data aggregations on the database. This helps avoid latency in data transfers from the data source.

Dynamic views are accessible from the **Assets** panel. Dynamic views are enabled by administrators in QMC. For information about enabling dynamic views, see [Managing on-demand apps](#).

Dynamic views are similar to on-demand app generation. Both use template apps to offer on-demand data, but dynamic views allow you to use individual charts in sheets rather than generating whole on-demand apps. If you are also using on-demand apps, you can create dynamic views using your on-demand template apps. For more information about generating on-demand apps, see *Managing big data with on-demand apps (page 174)*.

Dynamic views

When you create a dynamic view, you select a template app and optionally apply a row limit expression to control how much data the dynamic view will access. Once the dynamic view is created, you can then add master visualizations from the template app to your sheets.

Multiple dynamic views can use the same template app. Each dynamic view is refreshed individually. If bind expressions are used in a dynamic view's template app script, selections made in the base app can control which data is loaded into each dynamic view that uses that template app. Two dynamic views using the same template can be used to compare side-by-side charts of two separate subsets of granular data. For example, you have two dynamic views using the same template app. You could select Jan 1, 2018 from the base app *Saledate* field and refresh one dynamic view. You could then change the selection to Jan 1, 2019, refresh the other dynamic view, and then compare the dynamic charts.

When a user accesses an app containing a dynamic view, an on-demand app is added to their **Work**. This app contains the dynamic view template app with the current data and is used to populate the base app with the dynamic view. It is replaced by a new version every time the dynamic view refreshes. If the user is not the owner of the dynamic view template app, the load script will be removed. These apps are deleted 24 hours after the last refresh.

For information on creating and editing dynamic views, see *Managing data with dynamic views (page 190)*.

For information on using dynamic views, see *Using dynamic views and charts (page 195)*.

Dynamic view template apps

A dynamic view template app is a Qlik Sense app used to supply dynamic views with data and master visualizations.

Dynamic view templates can have a load script that contains data binding expressions used to formulate queries made on the data sources based on selections made in the base app. Binding expressions are usually created by users with experience writing Qlik Sense load scripts. Template apps can have query filter conditions that are based on input parameters supplied during the activation of dynamic charts.

Once the data model of a dynamic view template app is complete, master visualizations can be added to the template app. These master visualizations can then be accessed through dynamic views and added as dynamic charts in other apps.

For information on creating template apps, see *Creating an on-demand template app (page 178)*.

Dynamic charts

Dynamic charts are derived from the master charts of a dynamic view's template app. Dynamic charts can be added to the sheets of other apps using dynamic views. Unlike other Qlik Sense charts, users can control when the source data in a dynamic view is refreshed using a refresh option in the charts. When a dynamic view's data is controlled by bind expressions, Qlik Sense tracks the base app selection state. A stale data icon appears on each chart of a dynamic view whenever the base app's selection state changes so that the new value sets for and off the dynamic view's bound fields no longer match the values used for the view's last refresh.

For information about using dynamic charts, see *Using dynamic views and charts (page 195)*.

7.2 Dynamic views limitations

Dynamics views have the following limitations:

- Dynamic views are not supported with stories. You can add snapshots of dynamic charts to stories, but cannot use go to source with a dynamic chart.
- Dynamic views are not supported with Qlik NPrinting.
- Dynamic views support the dashboard and visualization bundle extensions. No other extensions are supported.
- Dynamic view ownership does not change with app ownership.
- Dynamic views cannot be created in apps in a managed space.

- Dynamic views are not supported in Qlik Sense Desktop.
- Dynamic charts cannot be placed in a container.
- Trellis containers and containers are not supported as master charts in dynamic views.
- Dynamic views are not supported for anonymous users.

7.3 Streams and dynamics views

You can create dynamic views to any app to which you have access. You can make dynamic views to your published apps in streams and to your unpublished apps in **Work**. You can also make dynamic views to published apps owned by other users in streams to which you have access.

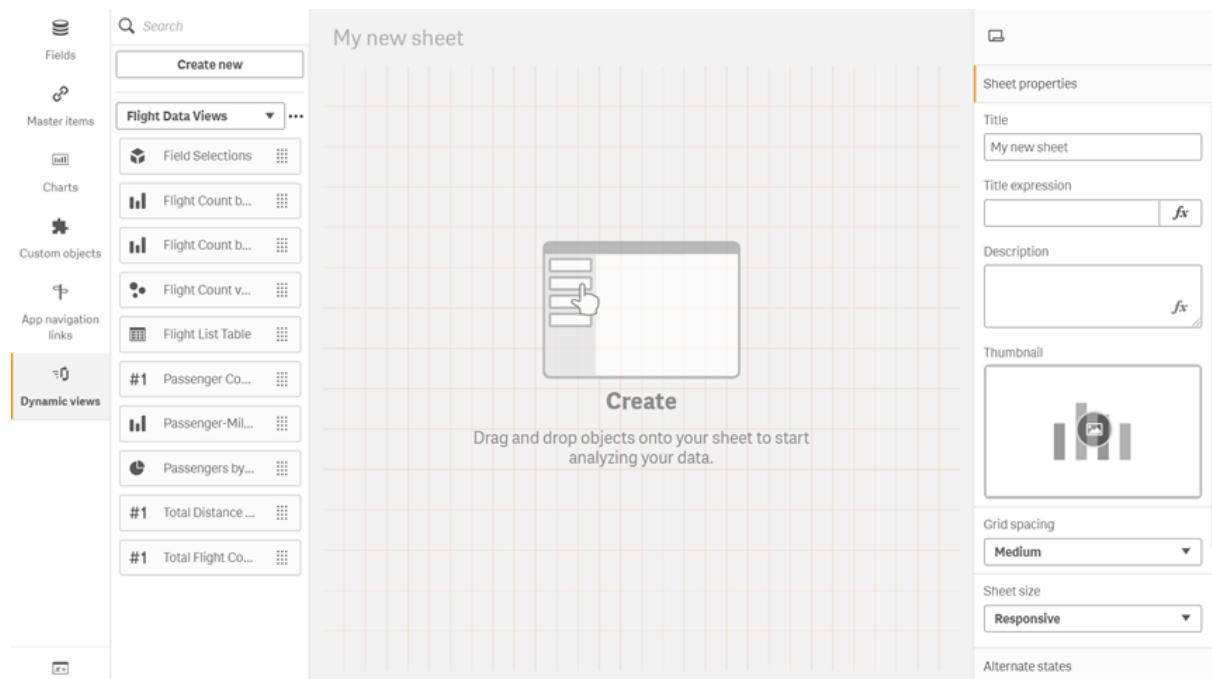
Users with access to an app with a dynamic view can use the dynamic view even if they do not have access to the template app.

Dynamic views cannot be added to published apps. Apps can be duplicated and republished to add new dynamic views.

7.4 Creating dynamic views and charts

Dynamic views can be added from the **Assets** panel in sheet view. You can add dynamic charts from your dynamic views to your sheets.

*Dynamic views in the **Assets** panel of a new sheet*



Creating dynamic views

When you create a dynamic view, you can limit how many rows are brought in as a part of the dynamic view. This can help prevent returning too many rows if your dynamic views are connecting to a very large data set.

Do the following:

1. In the **Edit** mode of a sheet, click **Dynamic views**.
2. Click **Create new**.
3. After **Name**, enter a name for the view.
4. After **Template app**, select a template app.
5. After **Row limit**, optionally select to use a **Row limit expression** and optionally set a **Maximum row count**.
6. Click **Create**.

Considerations for multi-node environments

To enable dynamic views in multi-node deployments of Qlik Sense, the app developer should specify a stream to publish to. This can be configured in the dynamic view creation dialog.

Do the following:

1. In the dynamic view creation dialog, under **Publish to**, specify a stream for publishing.



*If you have a single-node deployment of Qlik Sense, select **Not published (saved to workspace)**. Selecting a stream for a single-node environment might result in significantly longer processing time.*

2. Click **Create**.

Adding dynamic charts to sheets

Dynamic charts can be added to sheets from the **Assets** panel.

If you have editing permissions in the template app, you can edit the master chart upon which the dynamic chart is based.

To edit a dynamic chart, select a dynamic chart while the sheet is in **Edit** mode and select **Edit source app**.

Editing dynamic views

You can edit your dynamic views and change your row limits. You cannot change the template app used by a dynamic view.

Do the following:

1. Select a dynamic view from the dynamic views drop-down.
2. Click and click .

7.5 Using dynamic views and charts

Dynamic charts can be interacted with like other Qlik Sense visualizations. Users can also directly control when to refresh data in dynamic views. If a dynamic view's template app uses bind expressions in its load script, the query that refreshes the dynamic view's data will usually include filter conditions that use values selected in the selection state of the base app. When a user refreshes a dynamic view, all dynamic charts associated with the view are updated with the new data.

To access a context menu for chart options, such as for taking snapshots and opening the exploration menu, right-click on the chart and select **Dynamic chart**.

You can interact with the dynamic view through the dynamic charts. By right-clicking a dynamic chart and selecting **Dynamic view**, you can refresh the charts, view constraints, and view the values of the dynamic view's binding fields used on the last refresh of the dynamic view.

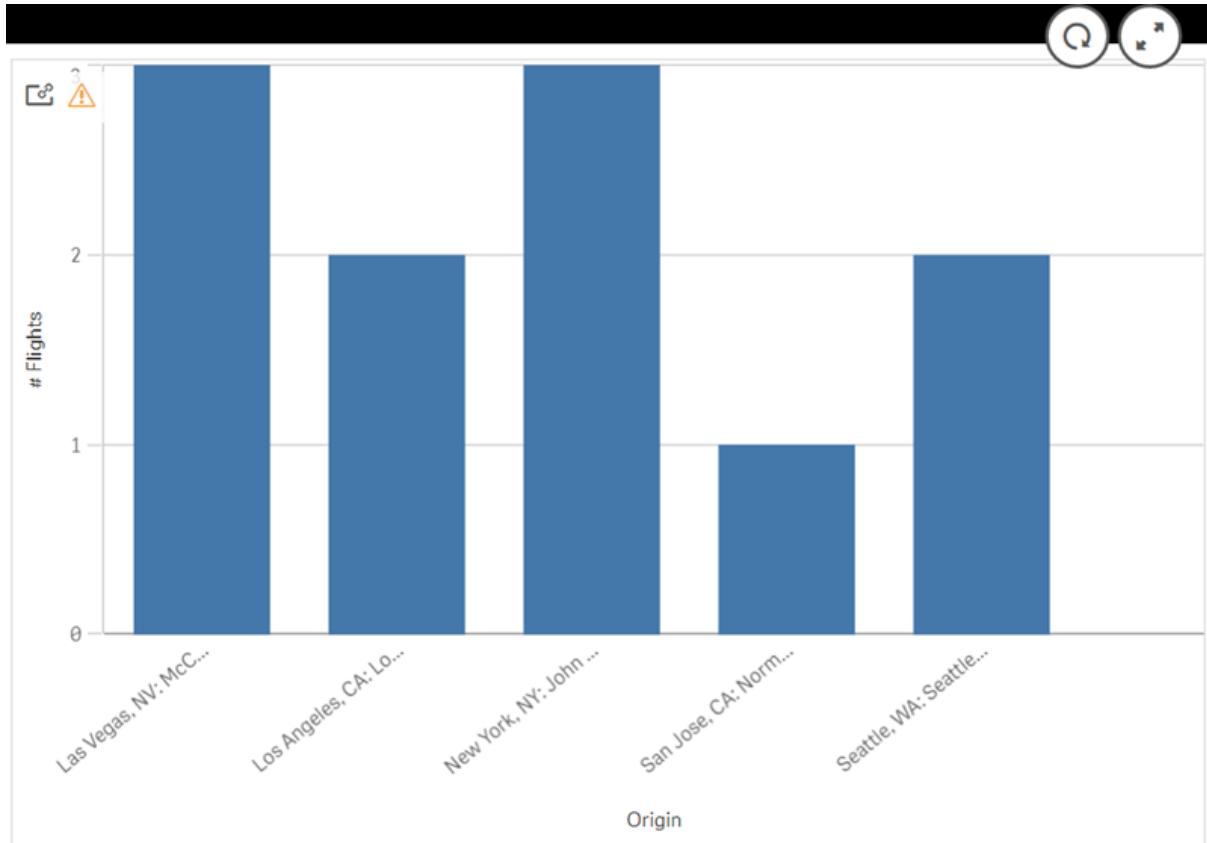
Selections in dynamic views

Dynamic charts use the same associative selection model as standard Qlik Sense charts. Selections made in dynamic charts do not impact the selections in the base app and do not appear in the base app selection bar. You can clear the selections you have made within a dynamic view by right-clicking a dynamic chart from that view, selecting **Dynamic view** and then selecting **Clear selections**.

If bind expressions are in the dynamic view's template app, selections made in the base app impact the data loaded into a dynamic view when it is next refreshed. Qlik Sense keeps track of the selection state that existed on the last refresh of each dynamic view. The data in the dynamic view is considered stale whenever selections change the values used by any of the bound fields in the dynamic view from when the dynamic view was last refreshed. A stale data icon displays on all of the charts of a dynamic view whenever data is stale. Refreshing a stale chart refreshes all charts from dynamic view using the updated selected value sets for each of the view's bound fields. If you want to restore the base app's selection state to match what it was when the dynamic view was last refreshed instead of refreshing it, use the context menu on any of the charts from that dynamic view and select **Dynamic View, Show last selections**, and click **Apply**.

7 Managing data with dynamic views

Dynamic chart with stale data notification



If any of the constraints have not been met for a dynamic view, such as if the current selections result in a number of rows that exceeds the maximum row count, no data will be shown for any of that view's charts. It is not be possible to refresh the dynamic view until the selection state of the base app changes so that all constraints are met.

Dynamic chart with selections exceeding the constraints



Your current selections exceed the constraints set for this view.

Viewing dynamic view details

You can view the refresh history, constraints, and selections used with dynamic view in **Dynamic view details**. You can access the details by clicking the **Dynamic view details** icon in the corner of a dynamic chart.

Refresh shows the last time the data in the dynamic view was refreshed.

Refresh details in **Dynamic view details**

Dynamic view of Flight Details with 70k Limit

🔗 Linked from ODAG Sample Detail

Refresh	Constraints	Selections
---------	-------------	------------

Last completed run: Nov 25, 2019, 2:59:46 PM

Refresh

Constraints shows the field and row constraints applied for generating content from the dynamic view.

Current constraints in **Dynamic view details**

Dynamic view of Flight Details with 70k Limit

🔗 Linked from ODAG Sample Detail

Refresh	Constraints	Selections
Field and row constraints for generating an app from this dynamic view.		
Row count		
✓	Current 18	Constraint < 70000
Fields and value count		
Field	Current	Constraint
Fare Class	✓ 3	None
Origin Code	✓ 10	None
Destination Code	✓ 13	None

Selections shows the selections that were applied to generate data for this view. If you make new selections that would change the data in the dynamic view and have not refreshed your dynamic view, you can click **Apply** to restore the original selections of the dynamic view.

Current selections in **Dynamic view details**

Dynamic view of Flight Details with 70k Limit

🔗 Linked from ODAG Sample Detail

Refresh	Constraints	Selections
Selections made when this app was generated		
Field	Values	Apply
Airline	Air Berlin PLC and CO, Iberia Air Lines Of Spain, Japan Air Lines Co. Ltd., Korean Air Lines Co. Ltd., Klm Royal Dutch Airlines ... (2 more)	
Destination Name	Atlanta, GA: Hartsfield-Jackson Atlanta International, Billings, MT: Billings Logan International, Boston, MA: Logan International, Burlington, VT: Burlington International, Baltimore, MD: Baltimore/Washington International Thurgood Marshall ... (1 more)	

Refreshing dynamic views

You can refresh the data in your dynamic view by selecting a dynamic chart from that view and clicking . Using the refresh option on a dynamic chart will refresh all charts from the same dynamic view. Refreshing removes any selections you have made within the dynamic charts of the dynamic view.

8 Connecting to data sources

Qlik Sense lets you to connect to your data, wherever it is stored, with a wide range of Qlik Connectors and other data connection types.

When you create a data connection it is saved, so you can quickly select and load data from the data sources that you commonly use. Connect to databases, social media data, local files, remote files, and web files.

8.1 Create a connection

To select data from a data source, you can either create a new data connection or use a saved data connection. You can create data connections and access your saved connections from:

- **Add data** in the data manager.
Add new data to your app quickly and get assistance creating associations.
- **Data connections** in the data load editor.
Select data from a new or existing data connection, or use the script editor to load data from a data connection. You can also edit existing data connections.



You can only see data connections that you own, or have been given access to, for reading or updating. Please contact your Qlik Sense system administrator to acquire access if required.

8.2 Data connection types

Qlik Sense allows you to access your data wherever it resides. The following data connection types are available with Qlik Sense. You can download connectors from [Product Downloads](#) to use with Qlik Sense.

Many of the connectors that access these data sources are built into Qlik Sense, while others can be added. Each type of data connection has specific settings that you need to configure.

Attached files

Attach data files directly to your app by using drag and drop.

Database connectors



Qlik Sense Enterprise only.

Connect to an ODBC data source with preconfigured ODBC database connectors.

- Amazon Athena
- Amazon Redshift
- Apache Drill

- Apache Hive
- Apache Phoenix
- Apache Spark
- Azure SQL
- Azure Synapse
- Cloudera Impala
- Databricks
- Google BigQuery
- IBM DB2
- Microsoft SQL Server
- MongoDB
- MySQL Enterprise
- Oracle
- PostgreSQL
- Presto
- Snowflake
- Sybase ASE
- Teradata

Essbase

Connect to an Essbase dataset.

Local or network files



Qlik Sense Enterprise only.

Connect to a local or network file to select and load data.

ODBC connections through DSN



Qlik Sense Enterprise only.

Connect to a Database Management System (DBMS) with ODBC. Install an ODBC driver for the DBMS in question, and create a data source DSN.

Qlik Web Connectors

Connect to social media or web-based data sources. Requires separate installation.

- Amazon S3
- AYLIEN News V2

- AYLIEN Text Analysis
- Azure Storage
- Bitly V2
- Box
- Dropbox
- Facebook Insights
- GitHub
- Google Ads
- Google Ad Manager
- Google AdWords
- Google Analytics
- Google Calendar
- Google Drive and Spreadsheets
- Google Search Console
- Helper Connector
- JIRA
- Mailbox IMAP
- Mailbox POP3
- MeaningCloud
- Microsoft Dynamics CRM V2
- OData
- Outlook 365
- Office 365 SharePoint
- RegEx Connector
- Repustate
- Sentiment140
- SMTP Connector
- Strava
- SugarCRM
- SurveyMonkey
- Watson Natural Language Understanding
- Twitter
- YouTube Analytics

REST

Connect to a REST data source. The REST connector is not tailored for a specific REST data source and can be used to connect to any data source exposed through the REST API.

Salesforce



Qlik Sense Enterprise only.

Connect to your Salesforce.com account.

SAP



Qlik Sense Enterprise only.

Load data from SAP NetWeaver.

Web files



Qlik Sense Enterprise only

Connect to a web-based data source on a web URL.

Web Storage Provider Connectors

Connect to your file-based data from web storage providers.

- Dropbox
- Google Drive

Third-party connectors

With third-party connectors, you can connect to data sources that are not directly supported by Qlik Sense. Third-party connectors are developed using the QVX SDK or supplied by third-party developers. In a standard Qlik Sense installation you will not have any third-party connectors available.

8.3 Where is the data connection stored?

Connections are stored in the repository database by the Qlik Sense Repository Service. In a Qlik Sense server deployment, you manage data connections with the Qlik Management Console. The Qlik Management Console allows you to delete data connections, set access rights, and perform other system administration tasks.

In Qlik Sense Desktop, all connections are saved in the app without encryption.



Because Qlik Sense Desktop connections store any details about user name, password, and the file path that you entered when creating the connection, these stored details are available in plain text if you share the app with another user. You need to consider this when you design an app for sharing.

8.4 Loading data from files

Qlik Sense can read data from a variety of file formats.

File formats

There are several supported data file formats:

- Text files: Data in fields must be separated by delimiters such as commas, tabs, or semicolons. For example: comma-separated variable (CSV) files.
- HTML tables
- Excel files:



You cannot load data from password-protected Excel files, or Excel Binary Workbook files (.xlbs).

- XML files
- Qlik native QVD and QVX files
- Fixed record length files
- Data Interchange Format (DIF) files: DIF files can only be loaded with the data load editor.

Connection types

You can load files from different data connection types:

- Local and network file folders: For more information, see *Loading files from local and network file folders (page 206)*.
- The **Attached files** folder: You cannot delete or edit this folder. It contains files that are uploaded and attached to the app. (Not available in Qlik Sense Desktop). For more information, see *Adding data to the app (page 16)*.
- Files on the web: For more information, see *Loading files from web resources (page 206)*.



The file extension of DataFiles connections is case sensitive. For example: .qvd.

How do I load data from files?

There are several ways to load data from files.



Users with edit permissions in a space can read, write, and load DataFiles in that space. Other users will not see the DataFiles.

Selecting data from a data connection in the data load editor

You can go to **Data Connections**, and use the **Select data** dialog to select data to load.

Loading data from a file by writing script code

Files are loaded using a **LOAD** statement in the script. **LOAD** statements can include the full set of script expressions. To read in data from another Qlik Sense app, you can use a **Binary** statement.

Loading files from local and network file folders

You can load files from local and network file folders with a folder connection:

Settings for the data connection

UI item	Description
Path	<p>Path to the folder containing the data files. You can either: Select the folder, type a valid local path, or type a UNC path.</p> <p>Example of valid local path: <code>C:\data\DataFiles\</code></p> <p>Example of a UNC path: <code>\myserver\filedir\</code></p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  <i>You cannot use a mapped network drive in the path.</i> </div>
Name	Name of the data connection.

Loading files from web resources

You can load files from web resources, such as FTP, HTTP, or HTTPS, with a web file data connection. The file can be of any type supported by Qlik Sense:

Settings for a web file data connection

UI item	Description
URL	<p>Full URL to the web file you want to connect to, including the protocol identifier.</p> <p>Example: <code>http://unstats.un.org/unsd/demographic/products/socind/Dec.%202012/1a.xls</code></p> <p>If you connect to an FTP file you may need to use special characters, for example : or @, in the user name and password part of the URL. In this case you need to replace special characters with a percent character and the ASCII hexadecimal code of the character. For example, you should replace : with '%3a', and @ with '%40'.</p>
Name	Name of the data connection.

The URL set in the web file data connection is static by default, but you can override the URL with the format specification setting **URL is**. This is useful if you need to load data from dynamically created URLs.

Loading data from a dynamically created URL

In this example we want to load forum posts from the first 10 pages of the New to Qlik Sense forum of Qlik Community. The forum page contains 20 posts on each page, and the final parameter of the URL ,start, sets which post to show as the first post of the page. In the example URL here, the page will show posts starting with post number 20, and the following 20 posts.

```
https://community.qlik.com/community/qlik-sense/new-to-qlik-sense/content?filterID=contentstatus%5Bpublished%5D~objecttype~objecttype%5Bthread%5D&itemView=detail&start=20
```

With the counter *i* we step through the pages with a step of 20 until 180, which means the **For** loop executes 10 times.

To load the page, we substitute the start page with \$(i) at the end of the URL in the **URL is** setting,.

```
For i = 0 to 180 step 20
LOAD
    Title1,
    "Author",
    F6 As Replies,
    Views,
    "Latest activity"
FROM [lib://x2]
(URL IS [https://community.qlik.com/community/qlik-sense/new-to-qlik-sense/content?filterID=contentstatus%5Bpublished%5D~objecttype~objecttype%5Bthread%5D&itemView=detail&start=$(i)], html, utf8, embedded labels, table is @1);
Next i;
```

This will load the 200 most recent posts of the forum in a table, with title, author, number of replies and views, and time of latest activity.

Loading data from Microsoft Excel spreadsheets

Qlik Sense can read data from Microsoft Excel spreadsheets. The supported file formats are *XLS*, *XLSX*, *XLW* and *XLSM*.

You can either use Add data in data manager, or select data in the data load editor. In both cases you can select named areas of a sheet, a single sheet, selected sheets, or all sheets from the spreadsheet file. Each sheet is loaded as a separate table, except if they have the same field structure, in which case they are concatenated into one table.



When you load a Microsoft Excel spreadsheet, you are using the spreadsheet as a data source for Qlik Sense apps. That is, Microsoft Excel sheets become tables in Qlik Sense, not sheets in a Qlik Sense app.

You may find it useful to make some changes in Microsoft Excel before you load the spreadsheet.

Selecting data from Microsoft Excel sheets

When you select data from Microsoft Excel sheets, there are some settings to assist you with interpreting the table data correctly:

Settings to assist you with interpreting the table data correctly

UI item	Description
Field names	Set to specify if the table contains Embedded field names or No field names . Typically in an Excel spreadsheet, the first row contains the embedded field names. If you select No field names , fields will be named A,B,C...
Header size	Set to the number of rows to omit as table header, typically rows that contain general information that is not in a columnar format.

Example

My spreadsheet looks like this:

Spreadsheet example

Machine:	AEJ12B	-	-
Date:	2015-10-05 09	-	-
Timestamp	Order	Operator	Yield
2015-10-05 09:22	00122344	A	52
2015-10-05 10:31	00153534	A	67
2015-10-05 13:46	00747899	B	86

In this case you probably want to ignore the two first lines, and load a table with the fields Timestamp, Order, Operator, and Yield. To achieve this, use these settings:

Settings to ignore the two first lines and load the fields

UI item	Description
Header size	2 This means that the first two lines are considered header data and ignored when loading the file. In this case, the two lines starting with Machine: and Date: are ignored, as they are not part of the table data.
Field names	Embedded field names. This means that the first line that is read is used as field names for the respective columns. In this case, the first line that will be read is the third line because the two first lines are header data.

Preparing Microsoft Excel spreadsheets for easier loading with Qlik Sense

If you want to load Microsoft Excel spreadsheets into Qlik Sense, there are many functions you can use to transform and clean your data in the data load script, but it may be more convenient to prepare the source data directly in the Microsoft Excel spreadsheet file. This section provides a few tips to help you prepare your spreadsheet for loading it into Qlik Sense with minimal script coding required.

Use column headings

If you use column headings in Microsoft Excel, they will automatically be used as field names if you select **Embedded field names** when selecting data in Qlik Sense. It is also recommended that you avoid line breaks in the labels, and put the header as the first line of the sheet.

Formatting your data

It is easier to load an Microsoft Excel file into Qlik Sense if the content is arranged as raw data in a table. It is preferable to avoid the following:

- Aggregates, such as sums or counts. Aggregates can be defined and calculated in Qlik Sense.
- Duplicate headers.
- Extra information that is not part of the data, such as comments. The best way is to have a column for comments, that you can easily skip when loading the file in Qlik Sense.
- Cross-table data layout. If, for instance, you have one column per month, you should, instead, have a column called “Month” and write the same data in 12 rows, one row per month. Then you can always view it in cross-table format in Qlik Sense.
- Intermediate headers, for example, a line saying “Department A” followed by the lines pertaining to Department A. Instead, you should create a column called “Department” and fill it with the appropriate department names.
- Merged cells. List the cell value in every cell, instead.
- Blank cells where the value is implied by the previous value above. You need to fill in blanks where there is a repeated value, to make every cell contain a data value.

Use named areas

If you only want to read a part of a sheet, you can select an area of columns and rows and define it as a named area in Microsoft Excel. Qlik Sense can load data from named areas, as well as from sheets.

Typically, you can define the raw data as a named area, and keep all extra commentary and legends outside the named area. This will make it easier to load the data into Qlik Sense.

Remove password protection

Password protected files are not supported by Qlik Sense, so you need to remove password protection from the spreadsheet before loading it into Qlik Sense.

Loading Excel Binary Workbook files (.xlsb)

It is not possible to load Excel Binary Workbook files (.xlsb) directly into Qlik Sense. The workaround is to use an ODBC connection.

8.5 Loading data from databases

You can load data from commercial database systems into Qlik Sense using the following connectors:

- Connectors specifically developed to load data directly from databases through licensed ODBC drivers, without the need for DSN connections. For more information, see [Qlik Connectors: Database](#).
- Connectors that use the Microsoft ODBC interface or OLE DB. To use Microsoft ODBC, you must install a driver to support your DBMS, and you must configure the database as an ODBC data source in the **ODBC Data Source Administrator** in Windows **Control Panel**.

Loading data from an ODBC database

There are two ways to load data from a database.

To connect directly to a database through one of the Qlik-licensed ODBC drivers, see the instructions for Database connectors on the Qlik Connectors help site.

For more information, see [ODBC Connector Package](#).

Qlik-licensed ODBC drivers support the following databases:

- Amazon Redshift
- Apache Drill
- Apache Hive
- Apache Phoenix
- Apache Spark
- Azure SQL
- Cloudera Impala
- Google BigQuery
- Microsoft SQL Server
- MongoDB
- MySQL Enterprise
- Oracle
- PostgreSQL
- Presto
- Sybase ASE
- Teradata

To use the Microsoft ODBC interface, do the following:

1. You need to have an ODBC data source for the database you want to access. This is configured in the **ODBC Data Source Administrator** in Windows **Control Panel**. If you do not have one already, you need to add it and configure it, for example pointing to a Microsoft Access database.
2. Open the data load editor.

3. Create an **ODBC** data connection, pointing to the ODBC connection mentioned in step 1.
4. Click  on the data connection to open the data selection dialog.

Now you can select data from the database and insert the script code required to load the data.

ODBC

Loading data from ODBC data sources

- You can use the Database connectors in the Qlik ODBC Connector Package that supports the most common ODBC sources. This lets you define the data source in Qlik Sense without the need to use the Microsoft Windows **ODBC Data Source Administrator**. To connect directly to a database through one of the Qlik-licensed ODBC drivers in the ODBC Connector Package, see the instructions for Database connectors on the Qlik Connectors help site.
- You can install an ODBC driver for the DBMS in question, and create a data source DSN. This is described in this section.

You can access a DBMS (Database Management System) via ODBC with Qlik Sense:



The **Create new connection (ODBC)** dialog displays the **User DSN** connections that have been configured. When you are using the Qlik Sense Desktop, the list of DSN connections displays the ODBC drivers included in the ODBC Connector Package. They are identified by the "Qlik-" attached to the name (for example, Qlik-db2). These drivers cannot be used to create a new ODBC connection. They are used exclusively by the database connectors in the ODBC Connector Package. The ODBC drivers from the ODBC Connector Package are not displayed when you are using Qlik Sense in a server environment.

The alternative is to export data from the database into a file that is readable to Qlik Sense.

Normally, some ODBC drivers are installed with Microsoft Windows. Additional drivers can be bought from software retailers, found on the Internet or delivered from the DBMS manufacturer. Some drivers are redistributed freely.



In a server environment, the Microsoft Access Database driver has limitations. To avoid issues, use SQL Server Express Edition.
[Microsoft Access Database Engine 2016 Redistributable](#).

The ODBC interface described here is the interface on the client computer. If the plan is to use ODBC to access a multi-user relational database on a network server, additional DBMS software that allows a client to access the database on the server might be needed. Contact the DBMS supplier for more information on the software needed.

ODBC data connection settings

ODBC data connection settings

UI item	Description
User DSN System DSN	Select which type of DSN to connect to. For User DSN sources you need to specify if a 32-bit driver is used with Use 32-bit connection . System DSN connections can be filtered according to 32-bit or 64-bit .
Single Sign-On	You can enable Single Sign-On (SSO) when connecting to SAP HANA data sources. If this option is not selected, Engine service user credentials are used, unless you specify credentials in Username and Password . If this option is selected, Engine service user or Username / Password credentials are used to do a Windows logon, followed by a subsequent logon to SAML (SAP HANA) using current user credentials.
Username	User name to connect with, if required by the data source. Leave this field empty if you want to use Engine service user credentials, or if the data source does not require credentials.
Password	Password to connect with, if required by the data source. Leave this field empty if you want to use Engine service user credentials, or if the data source does not require credentials.
Name	Name of the data connection.

Adding ODBC drivers

An ODBC driver for your DBMS (Database Management System) must be installed on for Qlik Sense to be able to access your database. Best practice is always to install the latest driver. Please refer to the vendor's documentation for download and installation instructions.

An ODBC driver for your DBMS must be installed for Qlik Sense to be able to access your database. This is external software. Therefore the instructions below may not match the software of all vendors. For details, refer to the documentation for the DBMS you are using.

Do the following:

1. Double-click the **Administrative Tools** icon in the **Control Panel**.
2. Double-click the **ODBC Data Sources (64-bit)** icon.
The **ODBC Data Source Administrator** dialog appears.
3. In the **User DSN** tab, select the database to use with Qlik Sense.
4. Select the **Drivers** tab in the **Data Sources** dialog.

In the **Drivers** tab you can see a list of installed ODBC drivers. If your DBMS is not listed you must install a driver for it. Run the install program delivered with the ODBC driver, for example the Microsoft ODBC install program.

64-bit and 32-bit versions of ODBC configuration

A 64-bit version of the Microsoft Windows operating system includes the following versions of the Microsoft Open DataBase Connectivity (ODBC) Data Source Administrator tool (*Odbcad32.exe*):

- The 32-bit version of the *Odbcad32.exe* file is located in the %systemdrive%\Windows\SysWOW64 folder.
- The 64-bit version of the *Odbcad32.exe* file is located in the %systemdrive%\Windows\System32 folder.

Creating ODBC data sources

An ODBC data source must be created for the database you want to access. This can be done during the ODBC installation or at a later stage.



*Before you start creating data sources, a decision must be made whether the data sources should be **User DSN** or **System DSN** (recommended). You can only reach user data sources with the correct user credentials. On a server installation, typically you need to create system data sources to be able to share the data sources with other users.*

Do the following:

1. Open *Odbcad32.exe*.
2. Go to the tab **System DSN** to create a system data source.
3. Click **Add**.

The **Create New Data Source** dialog appears, showing a list of the ODBC drivers installed.

4. If the correct ODBC driver is listed, select it and click **Finish**.

A dialog specific to the selected database driver appears.

5. Select **Microsoft Access Driver (*.mdb, *.accdb)** and click **Finish**.



If you cannot find this driver in the list you can download it from Microsoft's downloads website and install it.

6. Name the data source and set the necessary parameters.
7. Name the data source *Scripting tutorial ODBC*.
8. Under **Database:**, click **Select....**
9. Under **Directories**, navigate to the location of your *Sales.accdb* file (a tutorial example file).
10. When the file *Sales.accdb* is visible in the text box on the left, click on it to make it the database name.
11. Click **OK** three times to close all the dialogs.
12. Click **OK**.

Best practices when using ODBC data connections

Moving apps with ODBC data connections

If you move an app between Qlik Sense sites/Qlik Sense Desktop installations, data connections are included.

If the app contains ODBC data connections, you need to make sure that the related ODBC data sources exist on the new deployment as well. The ODBC data sources need to be named and configured identically, and point to the same databases or files.

Security aspects when connecting to file based ODBC data connections

ODBC data connections using file based drivers will expose the path to the connected data file in the connection string. The path can be exposed when the connection is edited, in the data selection dialog, or in certain SQL queries.

If this is a concern, it is recommended to connect to the data file using a folder data connection if it is possible.

Stopping preview of large datasets in tables

If you have large data sets and you do not want to see a data preview while adding ODBC data sources to **Data manager** or **Data load editor**, hold down the Shift key while selecting your ODBC data connection.

OLE DB

Qlik Sense supports the OLE DB (Object Linking and Embedding, Database) interface for connections to external data sources. A great number of external databases can be accessed via OLE DB.

OLE DB data connection settings

OLE DB data connection settings

UI item	Description
Provider	Select Provider from the list of available providers. Only available when you create a new connection.

UI item	Description
Data source	Type the name of the Data source to connect to. This can be a server name, or in some cases, the path to a database file. This depends on which OLE DB provider you are using. Only available when you create a new connection. Example: If you selected Microsoft Office 12.0 Access Database Engine OLE DB Provider, enter the file name of the Access database file, including the full file path: <code>C:\Users\{user}\Documents\Qlik\Sense\Apps\Tutorial source files\Sales.accdb</code>
	 If a connection to the data source fails, a warning message is displayed.
Connection string	The connection string to use when connecting to the data source. This string contains references to the Provider and the Data source . Only available when you edit a connection.
Windows integrated security	With this option you use the existing Windows credentials of the user running the Qlik Sense service.
Specific user name and password	With this option you need to enter User name and Password for the data source login credentials.
Username	User name to connect with, if required by the data source. Leave this field empty if you use Windows integrated security or the data source does not require credentials.
Password	Password to connect with, if required by the data source. Leave this field empty if you use Windows integrated security or the data source does not require credentials.
Load Select database...	If you want to test the connection, click Load and then Select database... to use for establishing the data connection.  You are still able to use all other available databases of the data source when selecting data from the data connection.
Name	Name of the data connection.

Security aspects when connecting to file based OLE DB data connections

OLE DB data connections using file based drivers will expose the path to the connected data file in the connection string. The path can be exposed when the connection is edited, in the data selection dialog, or in certain SQL queries.

If this is a concern, it is recommended that you connect to the data file using a folder data connection if it is possible.

Stopping preview of large datasets in tables

If you have large data sets and you do not want to see a data preview while adding OLE DB data sources to **Data manager** or **Data load editor**, hold down the Shift key while selecting your OLE DB data connection.

Logic in databases

Several tables from a database application can be included simultaneously in the Qlik Sense logic. When a field exists in more than one table, the tables are logically linked through this key field.

When a value is selected, all values compatible with the selection(s) are displayed as optional. All other values are displayed as excluded.

If values from several fields are selected, a logical AND is assumed.

If several values of the same field are selected, a logical OR is assumed.

In some cases, selections within a field can be set to logical AND.

8.6 Accessing large data sets with Direct Discovery

Direct Discovery enables you to load big data sets from certain SQL sources that have simple star schema structures and combine them with in-memory data.

Selections can be made on in-memory and Direct Discovery data to see associations across the data sets with the Qlik Sense association colors: green, white, and gray. Certain visualizations can analyze data from both data sets together, though there are a number of limitations with this approach. It is not designed to be a real-time solution.



No new development of Direct Discovery will be undertaken to overcome the limitations.

Qlik Sense on-demand apps provide a more flexible approach to loading and analyzing big data sources.

Direct Discovery expands the associative capabilities of the Qlik Sense in-memory data model by providing access to additional source data through an aggregated query that seamlessly associates larger data sets with in-memory data. Direct Discovery enhances business users' ability to conduct associative analysis on big data

sources without limitations. Selections can be made on in-memory and Direct Discovery data to see associations across the data sets with the same Qlik Sense association colors - green, white, and gray. Visualizations can analyze data from both data sets together.

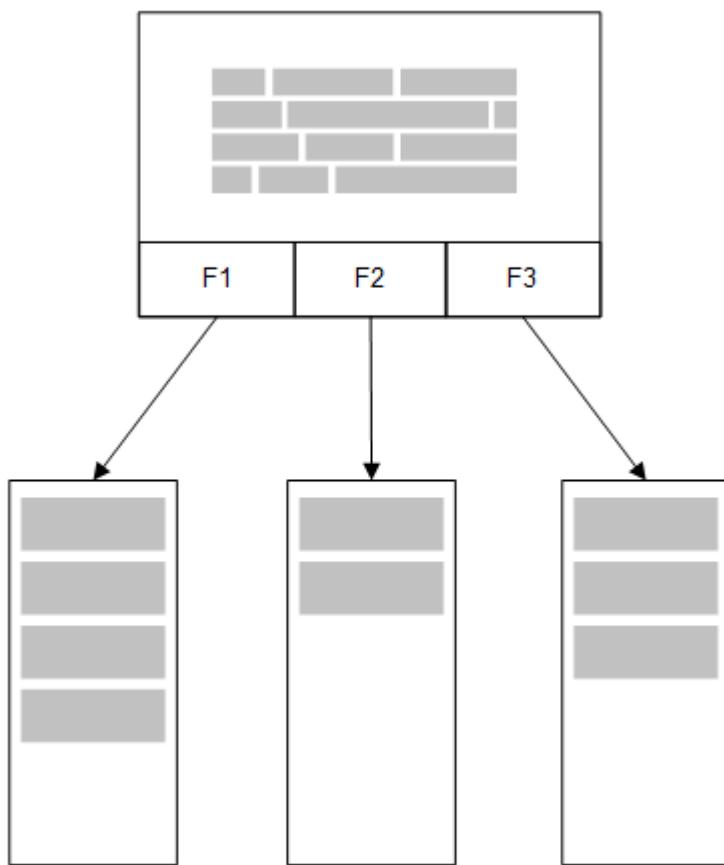
Data is selected for Direct Discovery using a special script syntax, **DIRECT QUERY**. Once the Direct Discovery structure is established, Direct Discovery fields can be used along with in-memory data to create Qlik Sense objects. When a Direct Discovery field is used in a Qlik Sense object, an SQL query is run automatically on the external data source.

On-demand apps provide another method for accessing large data sets. In contrast to Direct Discovery, on-demand apps provide full Qlik Sense functionality on a latent subset that is hosted in memory.

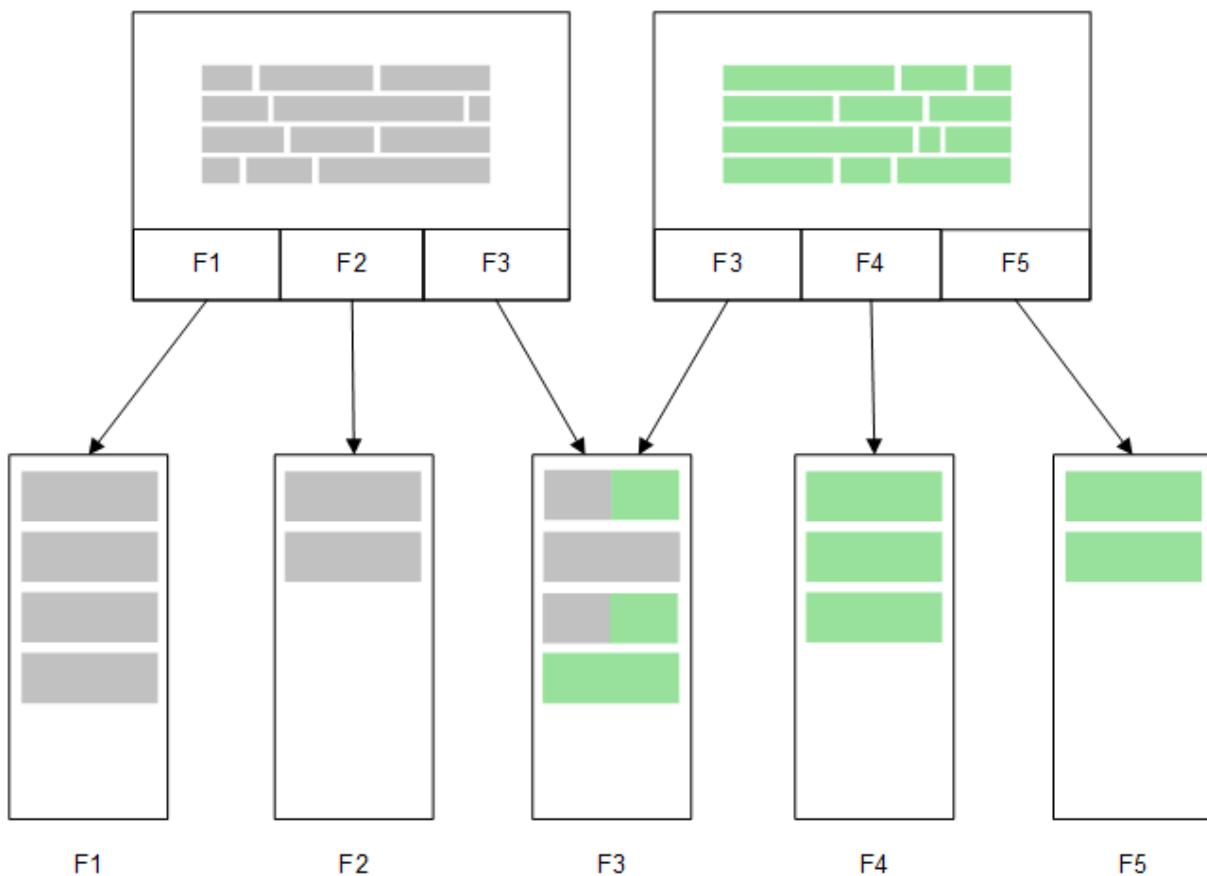
Differences between Direct Discovery and in-memory data

In-memory model

In the Qlik Sense in-memory model, all unique values in the fields selected from a table in the load script are loaded into field structures, and the associative data is simultaneously loaded into the table. The field data and the associative data is all held in memory.

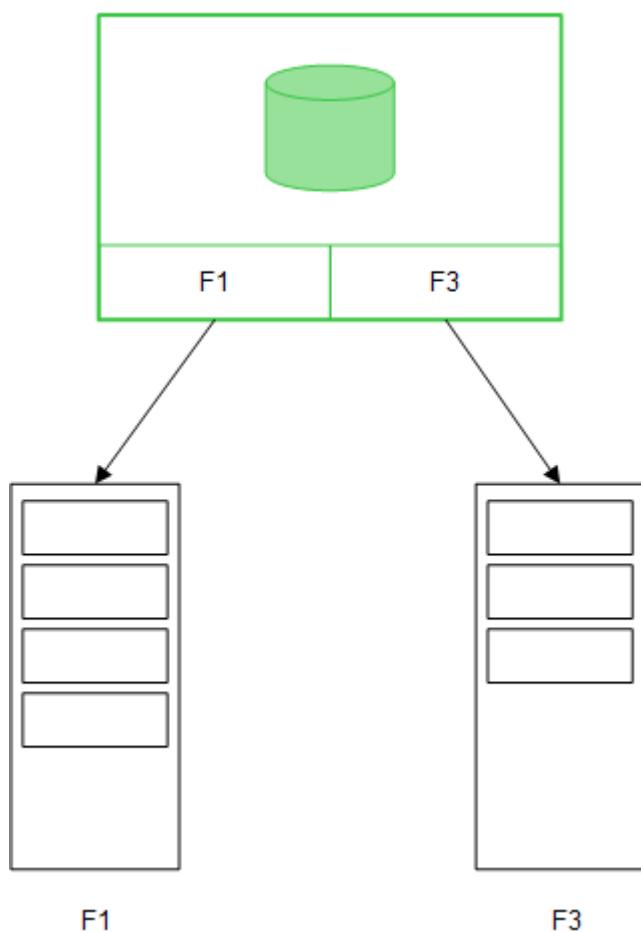


A second, related table loaded into memory would share a common field, and that table might add new unique values to the common field, or it might share existing values.

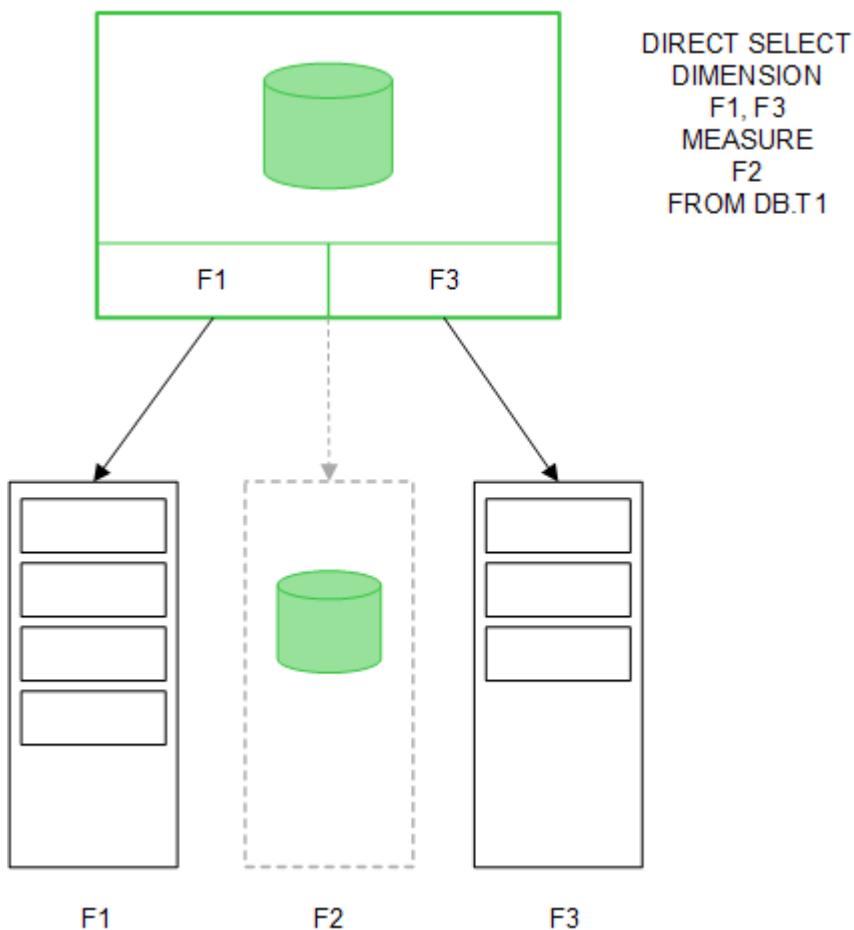


Direct Discovery

When table fields are loaded with a Direct Discovery **LOAD** statement (**Direct Query**), a similar table is created with only the **DIMENSION** fields. As with the In-memory fields, the unique values for the **DIMENSION** fields are loaded into memory. But the associations between the fields are left in the database.



MEASURE field values are also left in the database.



Once the Direct Discovery structure is established, the Direct Discovery fields can be used with certain visualization objects, and they can be used for associations with in-memory fields. When a Direct Discovery field is used, Qlik Sense automatically creates the appropriate SQL query to run on the external data. When selections are made, the associated data values of the Direct Discovery fields are used in the WHERE conditions of the database queries.

With each selection, the visualizations with Direct Discovery fields are recalculated, with the calculations taking place in the source database table by executing the SQL query created by Qlik Sense. The calculation condition feature can be used to specify when visualizations should be recalculated. Until the condition is met, Qlik Sense does not send queries to recalculate the visualizations.

Performance differences between in-memory fields and Direct Discovery fields

In-memory processing is always faster than processing in source databases. Performance of Direct Discovery reflects the performance of the system running the database processing the Direct Discovery queries.

It is possible to use standard database and query tuning best practices for Direct Discovery. All of the performance tuning should be done on the source database. Direct Discovery does not provide support for query performance tuning from the Qlik Sense app. It is possible, however, to make asynchronous, parallel calls to the database by using the connection pooling capability. The load script syntax to set up the pooling capability is:

```
SET DirectConnectionMax=10;
```

Qlik Sense caching also improves the overall user experience. See *Caching and Direct Discovery (page 222)* below.

Performance of Direct Discovery with **DIMENSION** fields can also be improved by detaching some of the fields from associations. This is done with the **DETACH** keyword on **DIRECT QUERY**. While detached fields are not queried for associations, they are still part of the filters, speeding up selection times.

While Qlik Sense in-memory fields and Direct Discovery **DIMENSION** fields both hold all their data in memory, the manner in which they are loaded affects the speed of the loads into memory. Qlik Sense in-memory fields keep only one copy of a field value when there are multiple instances of the same value. However, all field data is loaded, and then the duplicate data is sorted out.

DIMENSION fields also store only one copy of a field value, but the duplicate values are sorted out in the database before they are loaded into memory. When you are dealing with large amounts of data, as you usually are when using Direct Discovery, the data is loaded much faster as a **DIRECT QUERY** load than it would be through the **SQL SELECT** load used for in-memory fields.

Differences between data in-memory and database data

DIRECT QUERY is case-sensitive when making associations with in-memory data. Direct Discovery selects data from source databases according to the case-sensitivity of the database fields queried. If the database fields are not case-sensitive, a Direct Discovery query might return data that an in-memory query would not. For example, if the following data exists in a database that is not case-sensitive, a Direct Discovery query of the value "Red" would return all four rows.

Example table

ColumnA	ColumnB
red	one
Red	two
rED	three
RED	four

An in-memory selection of "Red," on the other hand, would return only:

Red two

Qlik Sense normalizes data to an extent that produces matches on selected data that databases would not match. As a result, an in-memory query may produce more matching values than a Direct Discovery query. For example, in the following table, the values for the number "1" vary by the location of spaces around them:

Table with different values for
the number "1" because of
different location of spaces
around them

ColumnA	ColumnB
' 1'	space_before
'1'	no_space

ColumnA	ColumnB
'1'	space_after
'2'	two

If you select "1" in a **Filter pane** for ColumnA, where the data is in standard Qlik Sense in-memory, the first three rows are associated:

Associated rows

ColumnA	ColumnB
'1'	space_before
'1'	no_space
'1'	space_after

If the **Filter pane** contains Direct Discovery data, the selection of "1" might associate only "no_space". The matches returned for Direct Discovery data depend on the database. Some return only "no_space" and some, like SQL Server, return "no_space" and "space_after".

Caching and Direct Discovery

Qlik Sense caching stores selection states of queries and associated query results in memory. As the same types of selections are made, Qlik Sense leverages the query from the cache instead of querying the source data. When a different selection is made, an SQL query is made on the data source. The cached results are shared across users.

Example:

1. User applies initial selection.
SQL is passed through to the underlying data source.
2. User clears selection and applies same selection as initial selection.
Cache result is returned, SQL is not passed through to the underlying data source.
3. User applies different selection.
SQL is passed through to the underlying data source.

A time limit can be set on caching with the **DirectCacheSeconds** system variable. Once the time limit is reached, Qlik Sense clears the cache for the Direct Discovery query results that were generated for the previous selections. Qlik Sense then queries the source data for the selections and recreates the cache for the designated time limit.

The default cache time for Direct Discovery query results is 30 minutes unless the **DirectCacheSeconds** system variable is used.

Direct Discovery field types

Within Direct Discovery, there are three types of data fields: DIMENSION, MEASURE, and DETAIL. The types are set on data fields when the Direct Discovery selection is made using the **Direct Query** statement in the load script.

All Direct Discovery fields can be used in combination with in-memory fields. Typically, fields with discrete values that will be used as dimensions should be loaded with the DIMENSION keyword, whereas numeric data that will be used in aggregations only should be marked as MEASURE fields.

The following table summarizes the characteristics and usage of the Direct Discovery field types:

Direct Discovery field types			
Field Type	In memory?	Forms association?	Used in chart expressions?
DIMENSION	Yes	Yes	Yes
MEASURE	No	No	Yes
DETAIL	No	No	No

DIMENSION fields

DIMENSION fields are loaded in memory and can be used to create associations between in-memory data and the data in Direct Discovery fields. Direct Discovery DIMENSION fields are also used to define dimension values in charts.

MEASURE fields

MEASURE fields, on the other hand, are recognized on a "meta level." MEASURE fields are not loaded in memory (they do not appear in the data model viewer). The purpose is to allow aggregations of the data in MEASURE fields to take place in the database rather than in memory. Nevertheless, MEASURE fields can be used in expressions without altering the expression syntax. As a result, the use of Direct Discovery fields from the database is transparent to the end user.

The following aggregation functions can be used with MEASURE fields:

- **Sum**
- **Avg**
- **Count**
- **Min**
- **Max**

DETAIL fields

DETAIL fields provide information or details that you may want to display but not use in chart expressions. Fields designated as DETAIL commonly contain data that cannot be aggregated in any meaningful way, like comments.

Any field can be designated as a DETAIL field.

Data sources supported in Direct Discovery

Qlik Sense Direct Discovery can be used against the following data sources, both with 32-bit and 64-bit connections:

- ODBC/OLEDB data sources - All ODBC/OLEDB sources are supported, including SQL Server, Teradata and Oracle.
- Connectors that support SQL – SAP SQL Connector, Custom QVX connectors for SQL compliant data stores.

Both 32-bit and 64-bit connections are supported.

SAP

For SAP, Direct Discovery can be used only with the Qlik SAP SQL Connector, and it requires the following parameters in **SET** variables:

```
SET DirectFieldColumnDelimiter=' ';  
SET DirectIdentifierQuoteChar=' ';
```

SAP uses OpenSQL, which delimits columns with a space rather than a comma, so the above set statements cause a substitution to accommodate the difference between ANSI SQL and OpenSQL.

Google Big Query

Direct Discovery can be used with Google Big Query and requires the following parameters in the set variables:

```
SET DirectDistinctSupport=false;  
SET DirectIdentifierQuoteChar='[]';  
SET DirectIdentifierQuoteStyle='big query'
```

Google Big Query does not support **SELECT DISTINCT** or quoted column/table names and has non-ANSI quoting configuration using '[]'.

MySQL and Microsoft Access

Direct discovery can be used in conjunction with MySQL and Microsoft Access but may require the following parameters in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='`';
```

DB2, Oracle and PostgreSQL

Direct discovery can be used in conjunction with DB2, Oracle and PostgreSQL but may require the following parameter in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='"';
```

Sybase and Microsoft SQL Server

Direct discovery can be used in conjunction with Sybase and Sybase and Microsoft SQL Server but may require the following parameter in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='[]';
```

Apache Hive

Direct discovery can be used in conjunction with Apache Hive but may require the following parameter in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='';
```

Cloudera Impala

Direct discovery can be used in conjunction with Cloudera Impala but may require the following parameter in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='[]';
```

This parameter is required when using the Cloudera Impala Connector in the Qlik ODBC Connector Package. It may not be required when using ODBC through DSN.

Limitations when using Direct Discovery



No new development of Direct Discovery will be undertaken to overcome the limitations.

Supported data types

Not all data types are supported in Direct Discovery. There may be cases in which specific source data formats need to be defined in Qlik Sense. You define data formats in the load script by using the "SET Direct...Format" syntax. The following example demonstrates how to define the date format of the source database that is used as the source for Direct Discovery:

Example:

```
SET DirectDateFormat='YYYY-MM-DD';
```

There are also two script variables for controlling how the Direct Discovery formats currency values in the generated SQL statements:

```
SET DirectMoneyFormat (default '#.0000')
SET DirectMoneyDecimalSep (default '.')
```

The syntax for these two variables is the same as for **MoneyFormat** and **MoneyDecimalSep**, but there are two important differences in usage:

- This is not a display format, so it should not include currency symbols or thousands separators.
- The default values are not driven by the locale but are tied to the values. (Locale-specific formats include the currency symbol.)

Direct Discovery can support the selection of extended Unicode data by using the SQL standard format for extended character string literals (N'<extended string>') as required by some databases, such as SQL Server. This syntax can be enabled for Direct Discovery with the script variable **DirectUnicodeStrings**. Setting this variable to "true" enables the use of "N" in front of the string literals.

Security

The following behaviors that could affect security best practice should be taken into consideration when using Direct Discovery:

- All of the users using the same app with the Direct Discovery capability use the same connection. Authentication pass-through and credentials-per-user are not supported.
- Section Access is supported in server mode only.
- Section access is not supported with high-cardinality joins.
- It is possible to execute custom SQL statements in the database with a NATIVE keyword expression, so the database connection set up in the load script should use an account that has read-only access to the database.
- Direct Discovery has no logging capability, but it is possible to use the ODBC tracing capability.
- It is possible to flood the database with requests from the client.
- It is possible to get detailed error messages from the server log files.

Qlik Sense functionality that is not supported

Because of the interactive and SQL syntax-specific nature of Direct Discovery, several features are not supported:

- Advanced calculations (set analysis, complex expressions)
- Calculated dimensions
- Comparative analysis (alternate state) of the objects that use Direct Discovery fields
- Direct Discovery **MEASURE** and **DETAIL** fields are not supported in smart search.
- Search of Direct Discovery **DETAIL** fields
- Binary load from an application that is accessing a Direct Discovery table.
- Synthetic keys on the Direct Discovery table
- Table naming in a script does not apply to the Direct Discovery table.
- The wild card character * after a **DIRECT QUERY** keyword in the load script

Example: (DIRECT QUERY *)

- Oracle database tables with LONG data type columns.
- Big integers in scientific notation, outside the range [-9007199254740990, 9007199254740991]. These may cause rounding errors and undefined behavior.
- Snowflake database schemas
- Data preparation in the Data manager
- Qlik Sense Enterprise SaaS
- Download to Microsoft Excel
- Offline mobile iOS app
- Advanced Analytics Integration
- Extensions
- Qlik GeoAnalytics

- Assigning colors to master dimensions and measures
- New visualizations included in Qlik Sense Enterprise on Windows June 2017 and later
- Non-SQL sources and non-SQL statements (for example, the PLACEHOLDER function in SAP HANA).
- The following connectors are not supported:
 - Qlik Salesforce Connector
 - Qlik REST Connector
 - Qlik Web connectors
 - Qlik Connector for use with SAP NetWeaver
- Optimizing the SQL generated by the Direct Discovery queries.
- High-cardinality joins in combination with in-memory tables can produce large IN clauses that may exceed the SQL buffer limit of the data source.
- Qlik Visualization and Dashboard bundle objects
- Insight Advisor
- Alerting
- Dynamic Views
- Custom tooltips

Multi-table support in Direct Discovery

You can use Direct Discovery to load more than one table or view using ANSI SQL join functionality.

In a single chart, all measures must be derived from the same logical table in Qlik Sense, but this can be a combination of several tables from source linked via join statements. However, you can use dimensions sourced from other tables in the same chart.

For example, you can link the tables loaded with Direct Discovery using either a **Where** clause or a **Join** clause.

- Direct Discovery can be deployed in a single fact/multi-dimension in memory scenario with large datasets.
- Direct Discovery can be used with more than one table which match any of the following criteria:
 - The cardinality of the key field in the join is low.
 - The cardinality of the key field in the join is high, **DirectEnableSubquery** is set to true and all tables have been joined with Direct Discovery.
- Direct Discovery is not suitable for deployment in a Third Normal Form scenario with all tables in Direct Discovery form.

Linking Direct Discovery tables with a **Where** clause

In this example script, we load data from the database AW2012. The tables Product and ProductSubcategory are linked with a **Where** clause using the common ProductSubCategoryID field.

```
Product_Join:  
DIRECT QUERY  
DIMENSION  
[ProductID],
```

```
[AW2012].[Production].[Product].[Name] as [Product Name],  
[AW2012].[Production].[ProductSubcategory].[Name] as [Sub Category Name],  
Color,  
[AW2012].[Production].[Product].ProductSubcategoryID as [SubcategoryID]  
MEASURE  
[ListPrice]  
FROM [AW2012].[Production].[Product],  
[AW2012].[Production].[ProductSubcategory]  
WHERE [AW2012].[Production].[Product].ProductSubcategoryID =  
[AW2012].[Production].[ProductSubcategory].ProductSubcategoryID ;
```

Linking Direct Discovery tables with **Join On** clauses

You can also use **Join On** clauses to link Direct Discovery tables. In this example statement we join the SalesOrderHeader table to the SalesOrderDetail table via the SalesOrderID field, and also join the Customer table to the SalesOrderHeader table via the Customer ID field.

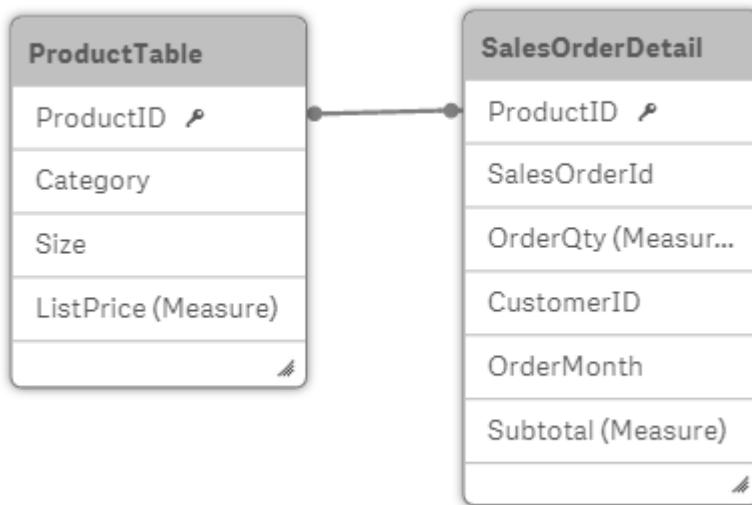
In this example we create measures from the same logical table, which means we can use them in the same chart. For example, you can create a chart with SubTotal and OrderQty as measures.

```
Sales_Order_Header_Join:  
DIRECT QUERY  
DIMENSION  
    AW2012.Sales.Customer.CustomerID as CustomerID,  
    AW2012.Sales.SalesOrderHeader.SalesPersonID as SalesPersonID,  
    AW2012.Sales.SalesOrderHeader.SalesOrderID as SalesOrderID,  
    ProductID,  
    AW2012.Sales.Customer.TerritoryID as TerritoryID,  
    OrderDate,  
    NATIVE('month([OrderDate])') as orderMonth,  
    NATIVE('year([OrderDate])') as orderYear  
MEASURE  
    SubTotal,  
    TaxAmt,  
    TotalDue,  
    OrderQty  
DETAIL  
    DueDate,  
    ShipDate,  
    CreditCardApprovalCode,  
    PersonID,  
    StoreID,  
    AccountNumber,  
    rowguid,  
    ModifiedDate  
FROM AW2012.Sales.SalesorderDetail  
JOIN AW2012.Sales.SalesOrderHeader  
ON (AW2012.Sales.SalesorderDetail.SalesOrderID =  
    AW2012.Sales.SalesOrderHeader.SalesOrderID)  
JOIN AW2012.Sales.Customer  
ON(AW2012.Sales.Customer.CustomerID =  
    AW2012.Sales.SalesOrderHeader.CustomerID);
```

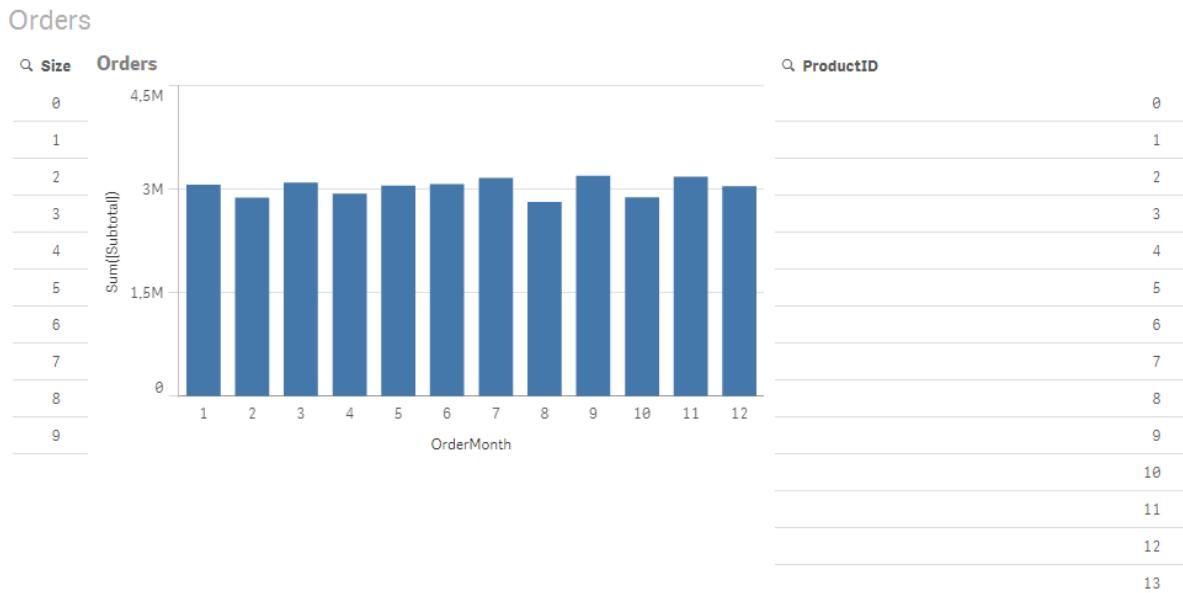
Using subqueries with Direct Discovery

If the cardinality of the key field joining the table is high, that is, it contains a large number of distinct values, a selection in Qlik Sense may generate a very large SQL statement as the **WHERE key_field IN** clause can contain a large number of values. In this case, a possible solution is to let Qlik Sense create subqueries instead.

To illustrate this, we use an example where a products table (ProductTable) is linked to a sales order table (SalesOrderDetail) using a product id (ProductID), with both tables used in Direct Discovery mode.

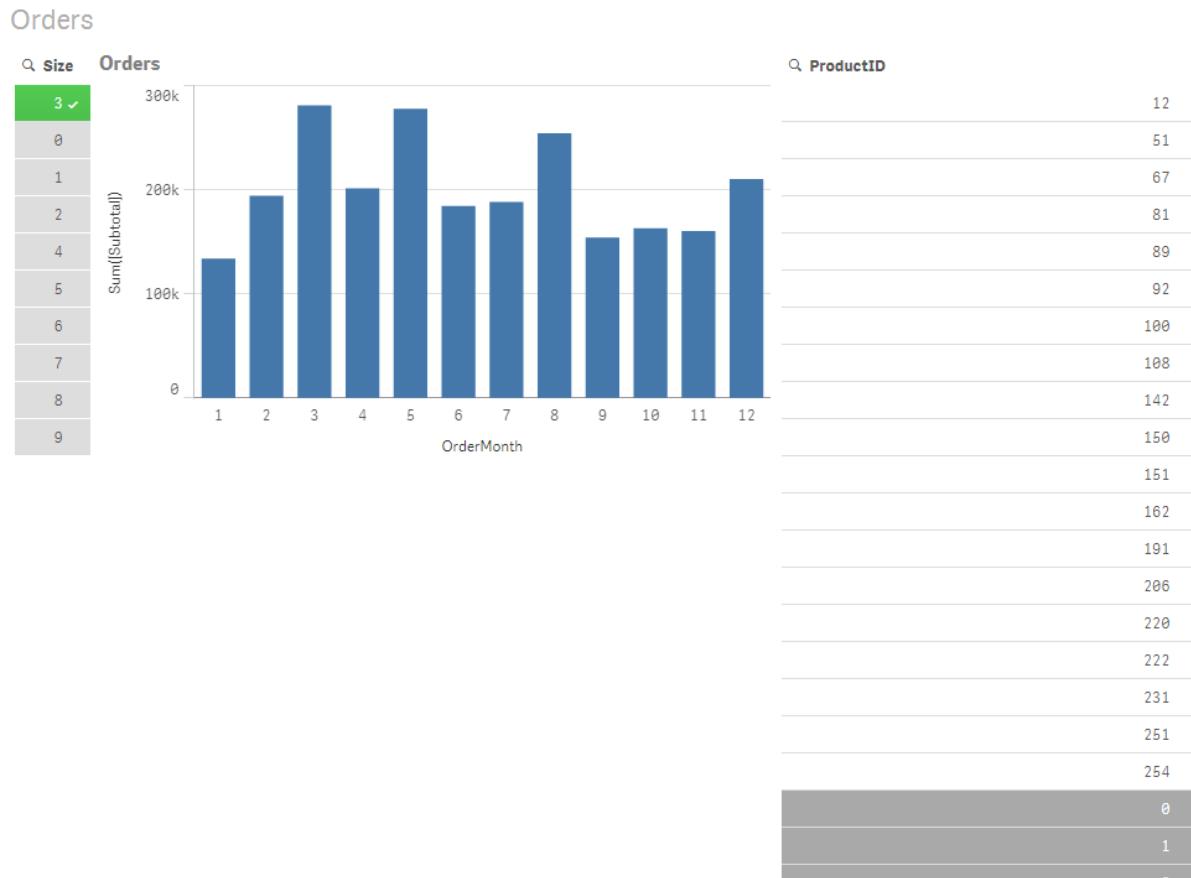


We create a chart with OrderMonth as dimension, and Sum(Subtotal) as measure, and a filter box for selecting Size.



Scenario 1: Low cardinality

In this scenario, the product table contains a low number of distinct products, 266. If we make a selection in Size, Direct Discovery generates a SQL statement to return the data, using a **WHERE ProductID IN** clause containing the product IDs matching the selected size, in this case 19 products.



The SQL that is generated looks like this:

```
SELECT ProductID, month([OrderDate]), SUM(OrderQty), SUM(SubTotal)
FROM SalesTable
WHERE ProductID IN ( 12, 51, 67, 81, 89, 92, 100, 108, 142, 150, 151, 162, 191, 206, 220, 222,
251, 254)
GROUP BY ProductID, month([OrderDate])
```

Scenario 2: Using subqueries

If the same example contains a high number of distinct products, for example 20.000, selecting a dimension filter, Size for example, would generate a SQL statement with a **WHERE ProductID IN** clause containing thousands of product IDs. The resulting statement could be too large to be handled by the data source due to limitations or issues with memory or performance.

The solution is to let Qlik Sense create subqueries instead, by setting the **DirectEnableSubquery** to true. The generated SQL statement could look like this instead:

```
SELECT ProductID, month([OrderDate]), SUM(OrderQty), SUM(SubTotal)
FROM SalesTable
```

```

WHERE ProductID IN
( SELECT DISTINCT "AW2012"."dbo"."PRODUCT"."PRODUCTID" WHERE "AW2012"."dbo"."PRODUCT"."SIZE"
IN (3))
GROUP BY ProductID, month([OrderDate])

```

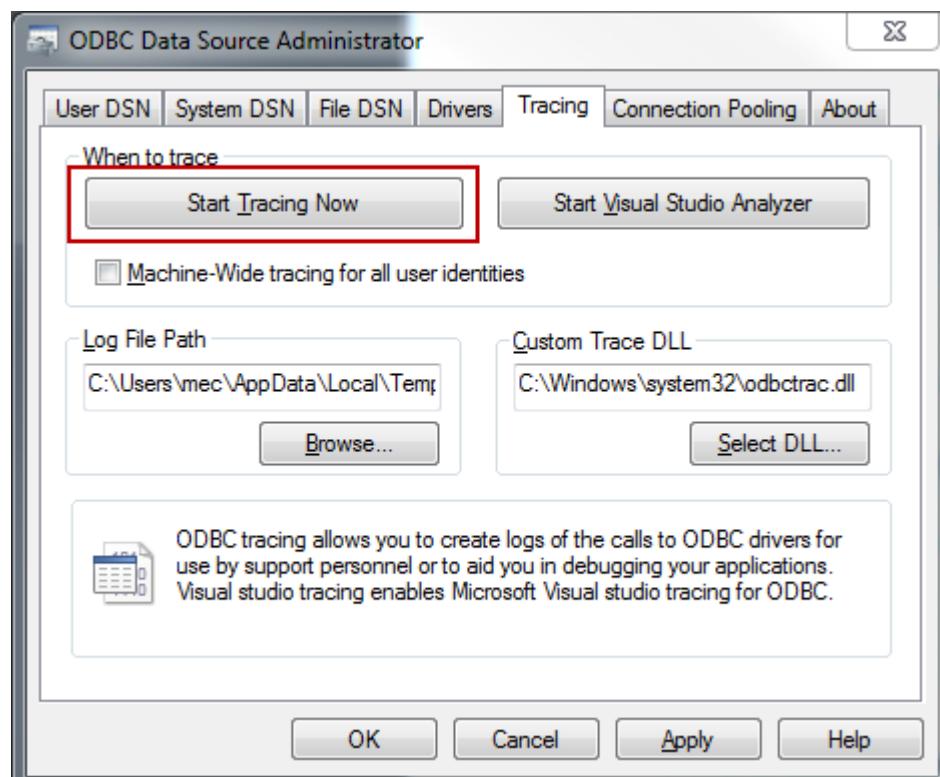
The **WHERE ProductID IN** clause size is not dependent on the number of keys resulting from the selection anymore.

The following limitations apply when using subqueries:

- Subquery syntax is only invoked when you select data which involves filtering a chart using data from another table.
- The amount of data within the keys is the determining factor, not the number of keys.
- Subqueries are only invoked if all tables involved are in Direct Discovery mode. If you filter the chart using data from a table included in memory mode, an **IN** clause will be generated.

Logging Direct Discovery access

Direct DiscoverySQL statements passed to the data source can be recorded in the trace files of the database connection. For a standard ODBC connection, tracing is started with the **ODBC Data Source Administrator**:



The resulting trace file details SQL statements generated through the user selections and interactions.

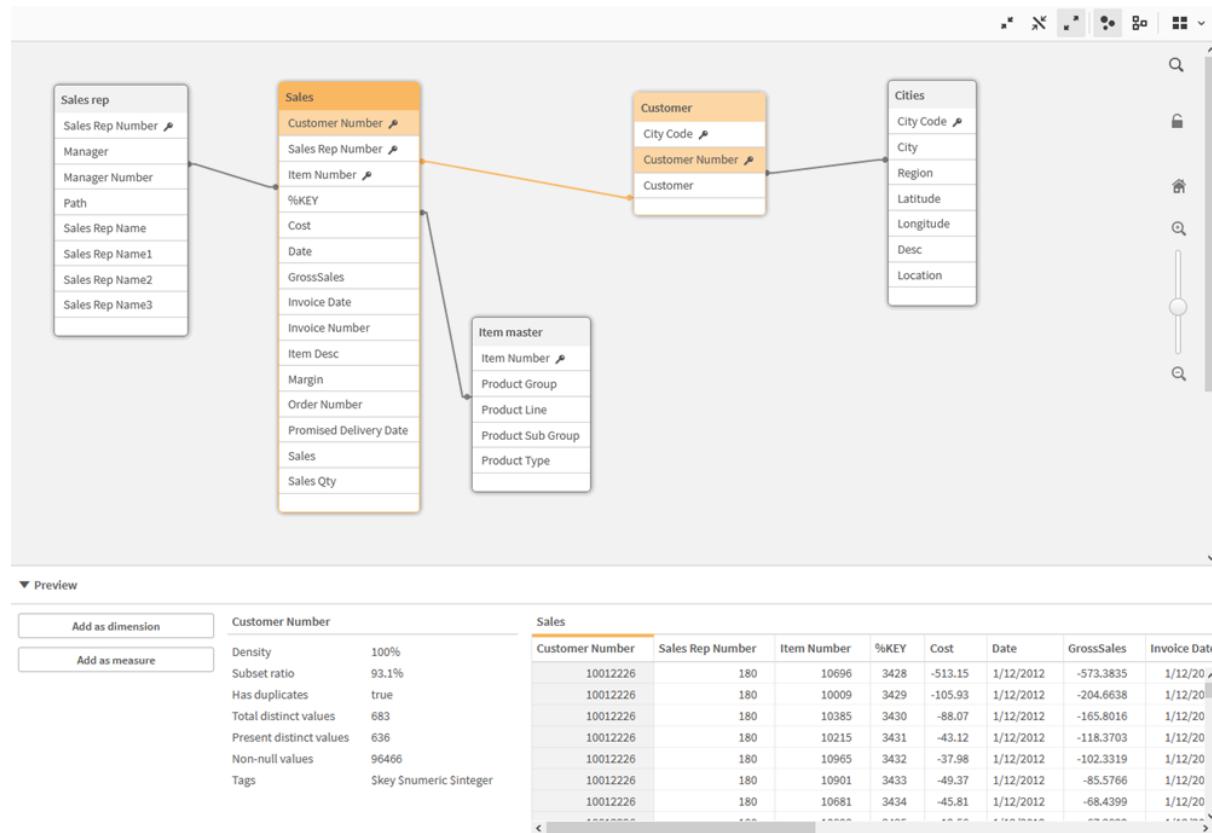
9 Viewing and transforming the data model

The data model viewer is an overview of the data structure of an app. You can view detailed metadata about the tables and fields. You can also create dimensions and measures from the data fields.

Click **Data model viewer** under the **Prepare** tab in the navigation bar to open the data model viewer.

Each data table is represented by a box, with the table name as title and with all fields in the table listed. Table associations are shown with lines, with a dotted line indicating a circular reference. When you select a table or a field, the highlighting of associations instantly gives you a picture of how fields and tables are related. You can search for specific tables and fields by clicking .

The data model displays the data structure of the app.



You can change the zoom level by clicking   or using the slider. Click  to restore the zoom level to 1:1.

In the preview pane, you can inspect the contents of a table or field. You can also add dimensions and measures to the app if you select a field. For more information, see *Previewing tables and fields in the data model viewer (page 236)*.

9.1 Moving tables

You can move tables by dragging them on the canvas. The table positions will be saved when the app is saved.

9 Viewing and transforming the data model

You can lock the table layout (positions and sizes), by clicking  in the right part of the canvas. To unlock the table layout, click .

You can also arrange the layout automatically using the options under  in the toolbar:

Options for moving tables

UI item	Name	Description
	Grid layout	To arrange the tables in a grid.
	Auto layout	To arrange the tables to fit in the window.
	Restore layout	To revert to the layout state present when the data model viewer was last opened.

9.2 Resizing tables

You can adjust the display size of a table with the arrow in the bottom right corner of the table. The display size will not be saved when the app is saved.

You can also use the automatic display size options in the toolbar:

Options for resizing tables

UI item	Name	Description
	Collapse all	To minimize all tables to show the table name only.
	Show linked fields	To reduce the size of all tables to show the table name and all fields with associations to other tables.
	Expand all	To maximize all tables to show all fields in the table.

9.3 Data model performance

These are indicators that can impact data model performance. Each one is a best practice that will improve app usability.

Data model performance best practices

Action	Description
Synthetic keys removed	Qlik Sense creates synthetic keys when two or more data tables have two or more fields in common. This may mean that there is an error in the script or the data model.

9 Viewing and transforming the data model

Action	Description
Circular references removed from data model	<p>Circular references occur when two fields have more than one association. Qlik Sense will attempt to resolve these by changing the connection to one of the tables.</p> <p>However, all circular reference warnings should be resolved.</p>
Appropriate granularity of data	<p>You should only load data that is necessary. For example: a group of users only need data divided by week, month, and year. You can either load in the aggregated data or aggregate the data within the load script to save memory. If a user does need to visualize data at a lower level of granularity, you can use ODAG or document chaining.</p>
QVDs used where possible	<p>A QVD is a file containing a table of data exported from Qlik Sense. This file format is optimized for speed when reading data from a script, but is still very compact.</p> <p>Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.</p>
QVD files optimized on load	<p>QVD files can be read in two modes: standard (fast) and optimized (faster). The selected mode is determined automatically by the script engine.</p> <p>There are some limitations regarding optimized loads. It is possible to rename fields, but any of these operations will result in a standard load:</p> <ul style="list-style-type: none">• Any transformations on the fields that are loaded.• Using a where clause causing Qlik Sense to unpack the records.• Using Map on a field that is loaded.
Incremental loads leveraged	<p>If your app connects to a large amount of data from databases that are continuously updated, reloading the entire data set can be time consuming. Instead, you should use incremental load to retrieve new or changed records from the database.</p>

9 Viewing and transforming the data model

Action	Description
Snowflake model consolidated	If you have a snowflake data model, you may be able to reduce the number of data tables by joining some of them using the Join prefix or other mapping. This is especially important for large fact tables. A good rule of thumb is to have only one large table.
Tables that have a small number of fields are denormalized	If you have two tables with few fields, it may improve performance to join them. .
Denormalized lookup (leaf) tables with mapping loads	You should not use the Join prefix if you only need to add one field from a table to another. You should use the ApplyMap lookup function.
Time stamps removed or decoupled from date field	Date fields can fill up space when the timestamp is present as the string representation is larger, and the number of distinct values is larger. If the precision is not necessary for your analysis, you can round the timestamp to e.g. the nearest hour using <i>Timestamp(Floor(YourTimestamp,1/24))</i> or remove the time component completely using <i>Date(Floor(YourTimestamp))</i> . If you want the timestamp, you can decouple it from the date itself. You can use the same Floor() function, and then create a new field with the extracted time by using something along the lines of: <i>Time(Frac(YourTimestamp))</i> .
Unnecessary fields removed from data model	You should only load necessary fields in your data model. Avoid using Load * and SELECT. Make sure you keep: <ul style="list-style-type: none">• Fields that are necessary for your analysis.• Fields that are actually being used in the app.

Action	Description
Link tables avoided when dealing with high data volumes	You should use link tables where possible. However, if you are dealing with large data volumes, concatenated tables can outperform link tables.
Concatenated dimensions broken to new fields	You should break apart concatenated dimensions into separate fields. This reduces the number of unique occurrences of values in your fields. This is similar to how timestamps can be optimized.
AutoNumber used where possible	You can create an optimized load by loading your data from a QVD file first, and then using the AutoNumber statement to convert values to symbol keys.
Data islands avoided	Data islands can be useful, but they usually affect performance. If you are creating islands for selection values, use variables.
QVDs are stored based on incremental timeframes	You should store QVD in segments, such as monthly. These smaller monthly QVD can then support many different apps that might not need all of the data.

For more best practices, see *Best practices for data modeling* (page 239).

9.4 Previewing tables and fields in the data model viewer

In the data model viewer, you can preview any data table in the panel at the bottom of the screen. In the preview, you can quickly inspect the contents of a table or field. You can also add dimensions and measures to the app if you select a field.

Additionally, metadata for the selected table or field are displayed in the preview panel.

You can show and hide the preview panel in two ways:

- Click  in the toolbar.
- Click the **Preview** header.



Direct Discovery data is not displayed in the preview .

Showing a preview of a table

Do the following:

- Click a table header in the data model viewer.

The preview panel is displayed with fields and values of the selected table.

Preview of data				
Item Number	Product Group	Product Line	Product Sub Group	Product Type
10001	Beverages	Drink	Juice	Pure Juice Beverages
10002	Beverages	Drink	Flavored Drinks	Drinks
10003	Beverages	Drink	Flavored Drinks	Drinks
10004	Beverages	Drink	Soda	Carbonated Beverages
10005	Beverages	Drink	Soda	Carbonated Beverages

Showing a preview of a field

Do the following:

- Click a table field in the data model viewer.

The preview panel is displayed with the selected field and its values, and metadata for the field. You can also add the field as a master dimension or measure.

Add as dimension		Preview of data			
Add as measure		Product Group	Item Number	Product Group	Product Line
Density	100%	10001	Beverages	Drink	Juice
Subset ratio	100%	10002	Beverages	Drink	Flavored Drinks
Has duplicates	true	10003	Beverages	Drink	Flavored Drinks
Total distinct values	15	10004	Beverages	Drink	Soda
Present distinct values	15	10005	Beverages	Drink	Soda
Non-null values	827	Tags		\$ascii, \$text	

- Density** is the number of records that have non-NULL values in this field, as compared to the total number of records in the table.
- Subset ratio** is the number of distinct values of the field found in this table, as compared to the total number of distinct values of this field in other tables in the data model. This is only relevant for key fields.
- If the field is marked with **[Perfect key]**, every row contains a key value that is unique.

9.5 Creating a master dimension from the data model viewer

When you are working with an unpublished app, you can create master dimensions so that they can be reused. Users of a published app will have access to the master dimensions, but will not be able to modify them. The data model viewer is not available in a published app.

Do the following:

1. In the data model viewer, select a field and open the **Preview** panel.
2. Click **Add as dimension**.
The **Create new dimensions** dialog opens, with the selected field. The name of the selected field is also used as the default name of the dimension.
3. Change the name if you want to, and optionally add a description, color, and tags.
4. Click **Create**.
5. Click **Done** to close the dialog.

The dimension is now saved to the master items tab of the assets panel.



*You can quickly add several dimensions as master items by clicking **Add dimension** after adding each dimension. Click **Done** when you have finished.*



Direct Discovery tables are indicated by in the data model viewer.

9.6 Creating a master measure from the data model viewer

When you are working with an unpublished app, you can create master measures so that they can be reused. Users of a published app will have access to the master measures, but will not be able to modify them.

Do the following:

1. In the data model viewer, select a field and open the **Preview** panel.
2. Click **Add as measure**.
The **Create new measure** dialog opens, with the selected field. The name of the selected field is also used as the default name of the measure.
3. Enter an expression for the measure.
4. Change the name if you want to, and optionally add a description, color, and tags.
5. Click **Create**.

The measure is now saved to the master items tab of the assets panel.

10 Best practices for data modeling

This section describes a number of different ways you can load your data into a Qlik Sense app, depending on how the data is structured and which data model you want to achieve.

10.1 Turning data columns into rows

My data probably looks like this, and I want to have the sales figures in a separate field:

Original data table				
Year	Q1	Q2	Q3	Q4
2013	34	54	53	52
2014	47	56	65	67
2015	57	56	63	71

Proposed action

Use the **Crosstable** prefix when you load the table.

The result will look like this:

Table after applying Crosstable prefix

Year	Quarter	Sales
2013	Q1	34
2013	Q2	54
2013	Q3	53
2013	Q4	52
2014	Q1	47
...

10.2 Turning data rows into fields

I have a generic table with three fields similar to this, and I want to have each attribute as a separate table:

Generic table with three fields

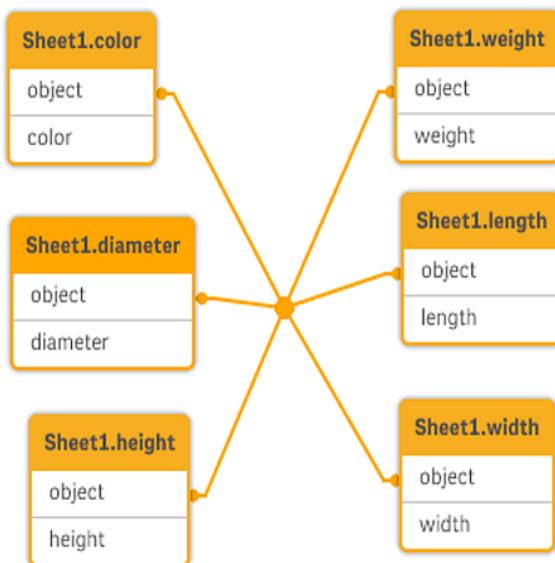
Object	Attribute	Value
ball	color	red
ball	diameter	25
ball	weight	3

Object	Attribute	Value
box	color	56
box	height	30
box	length	20
box	width	25

Proposed action

Create a generic data model using the **Generic** load prefix.

You will get a data model that looks like this:



10.3 Loading data that is organized in hierarchical levels, for example an organization scheme

My data is stored in an adjacent nodes table that looks like this:

Adjacent nodes table

NodeID	ParentNodeID	Title
1	-	General manager
2	1	Country manager
3	2	Region manager

Proposed action

Load the data with the Hierarchy prefix to create an expanded nodes table:

Expanded nodes table

NodeID	ParentNodeID	Title	Level1	Level2	Level3
1	-	General manager	General manager	-	-
2	1	Country manager	General manager	Country manager	-
3	2	Region manager	General manager	Country manager	Region manager

10.4 Loading only new or updated records from a large database

I have a database with a large number of records, and I don't want to reload the entire database to refresh the data in my app. I only want to load new or updated records, and remove records that are deleted from the database.

Proposed action

Implement an incremental load solution using QVD files.

10.5 Combining data from two tables with a common field

Qlik Sense will associate tables with a common field automatically, but I want to control how the tables are combined.

Proposed action : Join / Keep

You can combine two tables into a single internal table with the **Join** or **Keep** prefixes.

Proposed action : Mapping

An alternative to joining tables is to use mapping, which automates lookup of associated values in a mapping table. This can reduce the amount of data to load.

10.6 Matching a discrete value to an interval

I have a table of discrete numeric values (Event), and I want to match it to one or more intervals (Start and End).

Table of discrete numeric values (Event)

Time	Event	Comment
00:00	0	Start of shift 1
01:18	1	Line stop
02:23	2	Line restart 50%

Time	Event	Comment
04:15	3	Line speed 100%
08:00	4	Start of shift 2
11:43	5	End of production

Table with intervals (Start and End)

Start	End	Order
01:00	03:35	A
02:30	07:58	B
03:04	10:27	C
07:23	11:43	D

Proposed action

Use the **IntervalMatch** prefix to link the Time field with the interval defined by Start and End.

If the interval is not defined explicitly with start and end, only with a change timestamp like in the table below, you need to create an interval table.

Table with a change timestamp

Currency	Change Data	Rate
EUR	-	8.59
EUR	28/01/2013	8.69
EUR	15/02/2013	8.45
USD	-	6.50
USD	10/01/2013	6.56
USD	03/02/2013	6.30

10.7 Handling inconsistent field values

My data contains field values that are not consistently named in different tables. For example, one table contains the value US in Country while another table contains United States. This situation will prevent associations.

Table 1

Country	Region
US	Maryland
US	Idaho

Country	Region
US	New York
US	California

Table 2

Country	Population
United States	304
Japan	128
Brazil	192
China	1333

Proposed action

Perform data cleansing using a mapping table, that will compare field values and enable correct associations.

10.8 Handling inconsistent field value capitalization

My data contains field values that are not consistently formatted in different tables. For example, one table contains the value single in Type while another table contains Single in the same field. This situation will prevent associations, as the Type field will contain both single and Single values, capitalization matters.

Table 1

Type	Price
single	23
double	39

Table 2

Type	Color
Single	Red
Single	Blue
Double	White
Double	Black

Proposed action

If you loaded the data with **Add data**, you can fix this in the data manager.

Do the following:

1. In the data manager, open Table2 in the table editor.
2. Rename the Type field to Table2.Type.

If you just added the table with **Add data** with data profiling enabled, the field may already be named Table2.Type to prevent automatic association. In this case, this procedure will associate the two tables.

3. Create a calculated field using the expression `Lower(Table2.Type)` and name it Type.
4. Click **Load data**.

Table1 and Table2 should now be associated by the field Type, which only contains values in lowercase, like single and double.

If you want to use different capitalization, you can also achieve this with similar procedures, but remember that the tables will associate using the fields with the same name.

- To get all values capitalized, like Single, create the calculated Type field in Table1 instead, and use the expression `Capitalize(Table1.Type)`.
- To get all values in uppercase, like SINGLE, create the calculated Type field in both tables, and use the expressions `Upper(Table1.Type)` and `Upper(Table2.Type)` respectively.

10.9 Loading geospatial data to visualize data with a map

I have data that I want to visualize using a map, for example sales data per country, or per store. To use the map visualization I need to load area or point data.

Proposed action

You can load area or point data that match your data value locations from a KML file or an Excel file. Additionally, you need to load the actual map background.

10.10 Loading new and updated records with incremental load

If your app contains a large amount of data from database sources that are continuously updated, reloading the entire data set can be time consuming. In this case, you want to load new or changed records from the database. All other data should already be available in the app. Incremental load using QVD files, makes it possible to achieve this.

The basic process is described below:

1. Load new or updated data from the database source table.
This is a slow process, but only a limited number of records are loaded.
2. Load data that is already available in the app from the QVD file.
Many records are loaded, but this is a much faster process.
3. Create a new QVD file.
This is the file you will use the next time you do an incremental load.
4. Repeat the procedure for every table loaded.

The following examples show cases where incremental load is used. However, a more complex solution might be necessary, depending on the source database structure and mode of operation.

- Append only (typically used for log files)
- Insert only (no update or delete)
- Insert and update (no delete)
- Insert, update and delete

You can read QVD files in either optimized mode or standard mode. (The method employed is automatically selected by the Qlik Sense engine depending on the complexity of the operation.) Optimized mode is about 10 times faster than standard mode, or about 100 times faster than loading the database in the ordinary fashion.

Append only

The simplest case is the one of log files; files in which records are only appended and never deleted. The following conditions apply:

- The database must be a log file (or some other file in which records are appended and not inserted or deleted) which is contained in a text file (ODBC, OLE DB or other databases are not supported).
- Qlik Sense keeps track of the number of records that have been previously read and loads only records added at the end of the file.

Example:

```
Buffer (Incremental) Load * From LogFile.txt (ansi, txt, delimiter is '\t', embedded labels);
```

Insert only (no update or delete)

If the data resides in a database other than a simple log file, the append approach will not work. However, the problem can still be solved with a minimum amount of extra work. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted in the database after the last script execution.
- A ModificationTime field (or similar) is required for Qlik Sense to recognize which records are new.

Example:

```
QV_Table:  
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE  
WHERE ModificationTime >= #$(LastExecTime)#  
AND ModificationTime < #$(BeginningThisExecTime)#;  
  
Concatenate LOAD PrimaryKey, X, Y FROM File.QVD;  
STORE QV_Table INTO File.QVD;
```

The hash signs in the SQL WHERE clause define the beginning and end of a date. Check your database manual for the correct date syntax for your database.

Insert and update (no delete)

The next case is applicable when data in previously loaded records may have changed between script executions. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted into the database or updated in the database after the last script execution.
- A ModificationTime field (or similar) is required for Qlik Sense to recognize which records are new.
- A primary key field is required for Qlik Sense to sort out updated records from the QVD file.
- This solution will force the reading of the QVD file to standard mode (rather than optimized), which is still considerably faster than loading the entire database.

Example:

```
QV_Table:  
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE  
WHERE ModificationTime >= #$(LastExecTime)#;  
  
Concatenate LOAD PrimaryKey, X, Y FROM File.QVD  
WHERE NOT Exists(PrimaryKey);  
  
STORE QV_Table INTO File.QVD;
```

Insert, update and delete

The most difficult case to handle is when records are actually deleted from the source database between script executions. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted into the database or updated in the database after the last script execution.
- Qlik Sense removes records deleted from the database after the last script execution.
- A field ModificationTime (or similar) is required for Qlik Sense to recognize which records are new.
- A primary key field is required for Qlik Sense to sort out updated records from the QVD file.
- This solution will force the reading of the QVD file to standard mode (rather than optimized), which is still considerably faster than loading the entire database.

Example:

```
Let ThisExecTime = Now( );  
  
QV_Table:  
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE  
WHERE ModificationTime >= #$(LastExecTime)#  
AND ModificationTime < #$(ThisExecTime)#;  
  
Concatenate LOAD PrimaryKey, X, Y FROM File.QVD  
WHERE NOT EXISTS(PrimaryKey);  
  
Inner Join SQL SELECT PrimaryKey FROM DB_TABLE;  
  
If ScriptErrorCount = 0 then  
STORE QV_Table INTO File.QVD;  
Let LastExecTime = ThisExecTime;  
End If
```

10.11 Combining tables with Join and Keep

A join is an operation that uses two tables and combines them into one. The records of the resulting table are combinations of records in the original tables, usually in such a way that the two records contributing to any given combination in the resulting table have a common value for one or several common fields, a so-called natural join. In Qlik Sense, joins can be made in the script, producing logical tables.

It is possible to join tables already in the script. The Qlik Sense logic will then not see the separate tables, but rather the result of the join, which is a single internal table. In some situations this is needed, but there are disadvantages:

- The loaded tables often become larger, and Qlik Sense works slower.
- Some information may be lost: the frequency (number of records) within the original table may no longer be available.

The Keep functionality, which has the effect of reducing one or both of the two tables to the intersection of table data before the tables are stored in Qlik Sense, has been designed to reduce the number of cases where explicit joins need to be used.



In this documentation, the term join is usually used for joins made before the internal tables are created. The association made after the internal tables are created, is however essentially also a join.

Joins within a SQL SELECT statement

With some ODBC drivers it is possible to make a join within the **SELECT** statement. This is almost equivalent to making a join using the **Join** prefix.

However, most ODBC drivers are not able to make a full (bidirectional) outer join. They are only able to make a left or a right outer join. A left (right) outer join only includes combinations where the joining key exists in the left (right) table. A full outer join includes any combination. Qlik Sense automatically makes a full outer join.

Further, making joins in **SELECT** statements is far more complicated than making joins in Qlik Sense.

Example:

```
SELECT DISTINCTROW
[Order Details].ProductID, [Order Details].
UnitPrice, Orders.OrderID, Orders.OrderDate, Orders.CustomerID
FROM Orders
RIGHT JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID;
This SELECT statement joins a table containing orders to a fictive company, with a table containing order details. It is a right outer join, meaning that all the records of OrderDetails are included, also the ones with an OrderID that does not exist in the table Orders. Orders that exist in Orders but not in OrderDetails are however not included.
```

Join

The simplest way to make a join is with the **Join** prefix in the script, which joins the internal table with another named table or with the last previously created table. The join will be an outer join, creating all possible combinations of values from the two tables.

Example:

```
LOAD a, b, c from table1.csv;
join LOAD a, d from table2.csv;
```

The resulting internal table has the fields a, b, c and d. The number of records differs depending on the field values of the two tables.



*The names of the fields to join over must be exactly the same. The number of fields to join over is arbitrary. Usually the tables should have one or a few fields in common. No field in common will render the cartesian product of the tables. All fields in common is also possible, but usually makes no sense. Unless a table name of a previously loaded table is specified in the **Join** statement the **Join** prefix uses the last previously created table. The order of the two statements is thus not arbitrary.*

Keep

The explicit **Join** prefix in the data load script performs a full join of the two tables. The result is one table. In many cases such joins will result in very large tables. One of the main features of Qlik Sense is its ability to make associations between tables instead of joining them, which reduces space in memory, increases speed and gives enormous flexibility. The keep functionality has been designed to reduce the number of cases where explicit joins need to be used.

The **Keep** prefix between two **LOAD** or **SELECT** statements has the effect of reducing one or both of the two tables to the intersection of table data before they are stored in Qlik Sense. The **Keep** prefix must always be preceded by one of the keywords **Inner**, **Left** or **Right**. The selection of records from the tables is made in the same way as in a corresponding join. However, the two tables are not joined and will be stored in Qlik Sense as two separately named tables.

Inner

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **Inner**.

If used before **Join**, it specifies that the join between the two tables should be an inner join. The resulting table contains only combinations between the two tables with a full data set from both sides.

If used before **Keep**, it specifies that the two tables should be reduced to their common intersection before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

Table 1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

Inner Join

First, we perform an **Inner Join** on the tables, resulting in VTable, containing only one row, the only record existing in both tables, with data combined from both tables.

VTable:

```
SELECT * from Table1;
inner join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx

Inner Keep

If we perform an **Inner Keep** instead, you will still have two tables. The two tables are associated via the common field A.

VTab1:

```
SELECT * from Table1;
```

VTab2:

```
inner keep SELECT * from Table2;
```

VTab1

A	B
1	aa

VTab2

A	C
1	xx

Left

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **left**.

10 Best practices for data modeling

If used before **Join**, it specifies that the join between the two tables should be a left join. The resulting table only contains combinations between the two tables with a full data set from the first table.

If used before **Keep**, it specifies that the second table should be reduced to its common intersection with the first table before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

First, we perform a **Left Join** on the tables, resulting in VTable, containing all rows from Table1, combined with fields from matching rows in Table2.

VTable:

```
SELECT * from Table1;
left join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx
2	cc	-
3	ee	-

If we perform an **Left Keep** instead, you will still have two tables. The two tables are associated via the common field A.

VTab1:

```
SELECT * from Table1;
VTab2:
left keep SELECT * from Table2;
```

VTab1

A	B
1	aa

A	B
2	cc
3	ee

VTab2

A	C
1	xx

Right

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **right**.

If used before **Join**, it specifies that the join between the two tables should be a right join. The resulting table only contains combinations between the two tables with a full data set from the second table.

If used before **Keep**, it specifies that the first table should be reduced to its common intersection with the second table before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

First, we perform a **Right Join** on the tables, resulting in VTable, containing all rows from Table2, combined with fields from matching rows in Table1.

VTable:

```
SELECT * from Table1;
right join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx
4	-	yy

If we perform an **Right Keep** instead, you will still have two tables. The two tables are associated via the common field A.

```
VTab1:  
SELECT * from Table1;  
VTab2:  
right keep SELECT * from Table2;
```

VTab1	
A	B
1	aa

VTab2	
A	C
1	xx
4	yy

10.12 Using mapping as an alternative to joining

The **Join** prefix in Qlik Sense is a powerful way of combining several data tables in the data model.

One disadvantage is that the combined tables can become large and create performance problems. An alternative to **Join** in situations where you need to look up a single value from another table is to use mapping instead. This can save you from loading unnecessary data that slows down calculations and potentially can create calculation errors, as joins can change the number of records in the tables.

A mapping table consists of two columns: a comparison field (input) and a mapping value field (output).

In this example we have a table of orders (Orders), and need to know the countries of the customers, which are stored in the customer table (Customers).

Orders data table

OrderID	OrderDate	ShipperID	Freight	CustomerID
12987	2007-12-01	1	27	3
12988	2007-12-01	1	65	4
12989	2007-12-02	2	32	2
12990	2007-12-03	1	76	3

Customers data table

CustomerID	Name	Country	...
1	DataSales	Spain	...
2	BusinessCorp	Italy	...

CustomerID	Name	Country	...
3	TechCo	Germany	...
4	Mobecho	France	...

In order to look up the country (Country) of a customer, we need a mapping table that looks like this:

Mapping table

CustomerID	Country
1	Spain
2	Italy
3	Germany
4	France

The mapping table, which we name MapCustomerIDtoCountry, is defined in the script as follows:

```
MapCustomerIDtoCountry:
  Mapping LOAD CustomerID, Country From Customers ;
```

The next step is to apply the mapping, by using the **ApplyMap** function when loading the order table:

Orders:

```
LOAD *,
  ApplyMap('MapCustomerIDtoCountry', CustomerID, null()) as Country
  From Orders ;
```

The third parameter of the **ApplyMap** function is used to define what to return when a value is not found in the mapping table, in this case **Null()**.

The resulting table will look like this:

Result table

OrderID	OrderDate	ShipperID	Freight	CustomerID	Country
12987	2007-12-01	1	27	3	Germany
12988	2007-12-01	1	65	4	France
12989	2007-12-02	2	32	2	Italy
12990	2007-12-03	1	76	3	Germany

10.13 Working with crosstables in the data load script

A crosstab is a common type of table featuring a matrix of values between two orthogonal lists of header data. It is usually not the optimal data format if you want to associate the data to other data tables.

This topic describes how you can unpivot a crosstab, that is, transpose parts of it into rows, using the **crosstable** prefix to a **LOAD** statement in the data load script.

Unpivoting a crosstab with one qualifying column

A crosstab is often preceded by a number of qualifying columns, which should be read in a straightforward way. In this case there is one qualifying column, Year, and a matrix of sales data per month.

Crosstab with one qualifying column

Year	Jan	Feb	Mar	Apr	May	Jun
2008	45	65	78	12	78	22
2009	11	23	22	22	45	85
2010	65	56	22	79	12	56
2011	45	24	32	78	55	15
2012	45	56	35	78	68	82

If this table is simply loaded into Qlik Sense, the result will be one field for *Year* and one field for each of the months. This is generally not what you would like to have. You would probably prefer to have three fields generated:

- The qualifying column, in this case *Year*, marked with green in the table above.
- The attribute field, in this case represented by the month names Jan - Jun marked with yellow. This field can suitably be named *Month*.
- The data matrix values, marked with blue. In this case they represent sales data, so this can suitably be named *Sales*.

This can be achieved by adding the **crosstable** prefix to the **LOAD** or **SELECT** statement, for example:

```
crosstable (Month, Sales) LOAD * from ex1.xlsx;
```

This creates the following table in Qlik Sense:

Table with crosstable prefix added to the LOAD or SELECT statement

Year	Month	Sales
2008	Jan	45
2008	Feb	65
2008	Mar	78
2008	Apr	12
2008	May	78
2008	Jun	22
2009	Jan	11
2009	Feb	23
...

Unpivoting a crosstab with two qualifying columns

In this case there are two qualifying columns to the left, followed by the matrix columns.

Crosstab with two qualifying columns

Salesman	Year	Jan	Feb	Mar	Apr	May	Jun
A	2008	45	65	78	12	78	22
A	2009	11	23	22	22	45	85
A	2010	65	56	22	79	12	56
A	2011	45	24	32	78	55	15
A	2012	45	56	35	78	68	82
B	2008	57	77	90	24	90	34
B	2009	23	35	34	34	57	97
B	2010	77	68	34	91	24	68
B	2011	57	36	44	90	67	27
B	2012	57	68	47	90	80	94

The number of qualifying columns can be stated as a third parameter to the **crosstable** prefix as follows:

```
crosstable (Month, Sales, 2) LOAD * from ex2.xlsx;
```

This creates the following result in Qlik Sense:

Table with qualifying columns stated as a third parameter to the crosstable prefix

Salesman	Year	Month	Sales
A	2008	Jan	45
A	2008	Feb	65
A	2008	Mar	78
A	2008	Apr	12
A	2008	May	78
A	2008	Jun	22
A	2009	Jan	11
A	2009	Feb	23
...

10.14 Generic databases

A generic database is a table in which the field names are stored as field values in one column, while the field values are stored in a second. Generic databases are usually used for attributes of different objects.

Look at the example GenericTable below. It is a generic database containing two objects, a ball and a box. Obviously some of the attributes, like color and weight, are common to both the objects, while others, like diameter, height, length and width are not.

GenericTable

object	attribute	value
ball	color	red
ball	diameter	10 cm
ball	weight	100 g
box	color	black
box	height	16 cm
box	length	20 cm
box	weight	500 g
box	width	10 cm

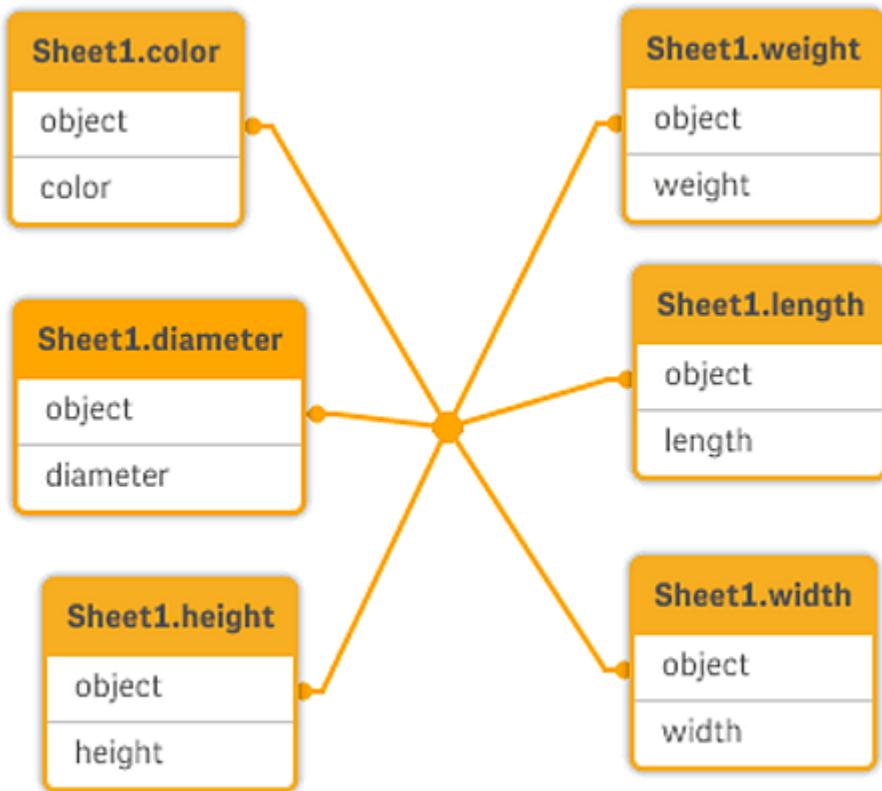
On one hand it would be awkward to store the data in a way giving each attribute a column of its own, since many of the attributes are not relevant for a specific object.

On the other hand, it would look messy displaying it in a way that mixes lengths, colors and weights.

If this database is loaded into Qlik Sense using the standard way and the data is displayed in a table, it looks like this:

object	attribute	value
ball	color	red
ball	diameter	10 cm
ball	weight	100 g
box	color	black
box	height	16 cm
box	length	20 cm
box	weight	500 g
box	width	10 cm

However, if the table is loaded as a generic database, column two and three will be split up into different tables, one for each unique value of the second column:



The syntax for doing this is simple, as shown in the following example.

Example

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```

Sheet1:
Generic Load * inline [
object, attribute, value
ball, color, red
ball, diameter, 10 cm
ball, weight, 100 g
box, color, black
box, height, 16 cm
box, length, 20 cm
box, weight, 500 g
box, width, 10 cm ];
  
```

Result

Resulting table

object	color	diameter	length	height	width	weight
ball	red	10 cm	-	-	-	100 g
box	black	-	20 cm	16 cm	10 cm	500 g

10.15 Matching intervals to discrete data

The **intervalmatch** prefix to a **LOAD** or **SELECT** statement is used to link discrete numeric values to one or more numeric intervals. This is a very powerful feature which can be used, for example, in production environments.

Intervalmatch example

Look at the two tables below. The first table shows the start and end of production of different orders. The second table shows some discrete events. How can we associate the discrete events with the orders, so that we know, for example, which orders were affected by the disturbances and which orders were processed by which shifts?

Table OrderLog

Start	End	Order
01:00	03:35	A
02:30	07:58	B
03:04	10:27	C
07:23	11:43	D

Table EventLog

Time	Event	Comment
00:00	0	Start of shift 1
01:18	1	Line stop
02:23	2	Line restart 50%
04:15	3	Line speed 100%
08:00	4	Start of shift 2
11:43	5	End of production

First, load the two tables as usual and then link the field *Time* to the intervals defined by the fields *Start* and *End*:

```
SELECT * from OrderLog;
SELECT * from EventLog;
```

`Intervalmatch (Time) SELECT Start,End from OrderLog;`

You can now create a table in Qlik Sense as below:

Table with Time field linked to the intervals defined by the Start and End

Time	Event	Comment	Order	Start	End
0:00	0	Start of shift 1	-	-	-
1:18	1	Line stop	A	1:00	3:35
2:23	2	Line restart 50%	A	1:00	3:35
4:15	3	Line speed 100%	B	2:30	7:58
4:15	3	Line speed 100%	C	3:04	10:....
8:00	4	Start of shift 2	C	3:04	10:....
8:00	4	Start of shift 2	D	7:23	11:....
11:43	5	End of production	E	7:23	11:....

We can now easily see that mainly order A was affected by the line stop but that the reduced line speed affected also orders B and C. Only the orders C and D were partly handled by *Shift 2*.

Note the following points when using **intervalmatch**:

- Before the **intervalmatch** statement, the field containing the discrete data points (*Time* in the example above) must already have been read into Qlik Sense. The **intervalmatch** statement does not read this field from the database table.
- The table read in the **intervalmatch LOAD** or **SELECT** statement must always contain exactly two fields (*Start* and *End* in the example above). In order to establish a link to other fields you must read the interval fields together with additional fields in a separate **LOAD** or **SELECT** statement (the first **SELECT** statement in the example above).
- The intervals are always closed. That is, the end points are included in the interval. Non-numeric limits render the interval to be disregarded (undefined) while NULL limits extend the interval indefinitely (unlimited).
- The intervals may be overlapping and the discrete values will be linked to all matching intervals.

Using the extended **intervalmatch** syntax to resolve slowly changing dimension problems

The extended **intervalmatch** syntax can be used for handling of the well-known problem of slowly changing dimensions in source data.

Sample script:

```
SET NullInterpret='';

IntervalTable:
LOAD Key, ValidFrom, Team
FROM 'lib://dataqv/intervalmatch.xlsx' (ooxml, embedded labels, table is IntervalTable);
```

Key:

```

LOAD
Key,
ValidFrom as FirstDate,
date(if(Key=previous(Key),
previous(ValidFrom) - 1)) as LastDate,
Team
RESIDENT IntervalTable order by Key, ValidFrom desc;

drop table IntervalTable;

Transact:
LOAD Key, Name, Date, Sales
FROM 'lib://dataqv/intervalmatch.xlsx' (ooxml, embedded labels, table is Transact);

INNER JOIN intervalmatch (Date,Key) LOAD FirstDate, LastDate, Key RESIDENT Key;
The nullinterpret statement is only required when reading data from a table file since missing values are
defined as empty strings instead of NULL values.

```

Loading the data from *IntervalTable* would result in the following table:

Table with data loaded from IntervalTable

Key	FirstDate	Team
000110	2011-01-21	Southwest
000110	-	Northwest
000120	-	Northwest
000120	2013-03-05	Southwest
000120	2013-03-05	Northwest
000120	2013-03-05	Southwest

The **nullasvalue** statement allows NULL values to map to the listed fields.

Create *Key*, *FirstDate*, *LastDate*, (attribute fields) by using **previous** and **order by** and thereafter the *IntervalTable* is dropped having been replaced by this key table.

Loading the data from *Transact* would result in the following table:

Table with data loaded from Transact

Key	Name	Date	Sales
000110	Spengler Aaron	2009-08-18	100
000110	Spengler Aaron	2009-12-25	200
000110	Spengler Aaron	2011-02-03	300
000110	Spengler Aaron	2011-05-05	400
000120	Ballard John	2011-06-04	500
000120	Ballard John	2013-01-20	600

Key	Name	Date	Sales
000120	Ballard John	2013-03-10	700
000120	Ballard John	2013-03-13	800
000120	Ballard John	2013-09-21	900

The **intervalmatch** statement preceded by the **inner join** replaces the key above with a synthetic key that connects to the *Transact* table resulting in the following table:

Table with the **intervalmatch** statement preceded by the inner join

Key	Team	Name	FirstDate	LastDate	Date	Sales
000110	Northwest	Spengler Aaron	-	2011-01-20	2009-08-18	100
000110	Northwest	Spengler Aaron	-	2011-01-20	2009-12-25	200
000110	Southwest	Spengler Aaron	2011-01-21		2011-02-03	300
000110	Southwest	Spengler Aaron	2011-01-21		2011-05-05	400
000120	Northwest	Ballard John		2013-01-05	2011-06-04	500
000120	Southwest	Ballard John	2013-01-06	2013-03-04	2013-01-20	600
000120	Southwest	Ballard John	2013-03-05		2013-03-10	700
000120	Southwest	Ballard John	2013-03-05		2013-03-13	800
000120	Southwest	Ballard John	2013-03-05		2013-09-21	900

10.16 Creating a date interval from a single date

Sometimes time intervals are not stored explicitly with a beginning and an end. Instead they are implied by only one field – the change timestamp.

It could be as in the table below where you have currency rates for multiple currencies. Each currency rate change is on its own row; each with a new conversion rate. Also, the table contains rows with empty dates corresponding to the initial conversion rate, before the first change was made.

Currency rates

Currency	Change Date	Rate
EUR	-	8.59
EUR	28/01/2013	8.69
EUR	15/02/2013	8.45
USD	-	6.50
USD	10/01/2013	6.56
USD	03/02/2013	6.30

The table above defines a set of non-overlapping intervals, where the begin date is called *Change Date* and the end date is defined by the beginning of the following interval. But since the end date isn't explicitly stored in a column of its own, we need to create such a column, so that the new table will become a list of intervals.

Do the following:

1. Create a new app and give it a name.
2. Add a new script section in the **Data load editor**.
3. Add the following inline table. Make sure that the dates in the *Change Date* column are in the same format as the local date format.

```
In_Rates:  
LOAD * Inline [  
Currency,Change Date,Rate  
EUR,,8.59  
EUR,28/01/2013,8.69  
EUR,15/02/2013,8.45  
USD,,6.50  
USD,10/01/2013,6.56  
USD,03/02/2013,6.30  
];
```

4. Determine which time range you want to work with. The beginning of the range must be before the first date in the data and the end of the range must be after the last.

Add the following to the top of your script:

```
Let vBeginTime = Num('1/1/2013');  
Let vEndTime = Num('1/3/2013');  
Let vEpsilon = Pow(2,-27);
```

5. Load the source data, but change empty dates to the beginning of the range defined in the previous bullet. The change date should be loaded as "From Date".

Sort the table first according to *Currency*, then according to the "From Date" descending so that you have the latest dates on top.

Add the following after the *In_Rates* table:

```
Tmp_Rates:  
LOAD Currency, Rate,  
Date(IF(IsNum([Change Date]), [Change Date], $(#vBeginTime))) as FromDate  
Resident In_Rates;
```

6. Run a second pass through data where you calculate the "To Date". If the current record has a different currency from the previous record, then it is the first record of a new currency (but its last interval), so you should use the end of the range defined in step 1. If it is the same Currency, you should take the "From Date" from the previous record, subtract a small amount of time, and use this value as "To Date" in the current record.

Add the following after the *Tmp_Rates* table:

```
Rates:  
LOAD Currency, Rate, FromDate,  
Date(IF( Currency=Peek('Currency'),  
Peek('FromDate') - $(#vEpsilon),  
$(#vEndTime)  
)) as ToDate  
Resident Tmp_Rates  
Order By Currency, FromDate Desc;
```

```
Drop Table Tmp_Rates;
```

Your script should look like this:

```
Let vBeginTime = Num('1/1/2013');
Let vEndTime = Num('1/3/2013');
Let vEpsilon = Pow(2,-27);

In_Rates:
LOAD * Inline [
Currency,Change Date,Rate
EUR,,8.59
EUR,28/01/2013,8.69
EUR,15/02/2013,8.45
USD,,6.50
USD,10/01/2013,6.56
USD,03/02/2013,6.30
];

Tmp_Rates:
LOAD Currency, Rate,
      Date(If(IsNum([Change Date]), [Change Date], $(#vBeginTime))) as FromDate
Resident In_Rates;

Rates:
LOAD Currency, Rate, FromDate,
      Date(If( Currency=Peek('Currency'),
              Peek('FromDate') - $(#vEpsilon),
              $(#vEndTime)
            )) as ToDate
Resident Tmp_Rates
Order By Currency, FromDate Desc;

Drop Table Tmp_Rates;
```

The script will update the source table in the following manner:

Updated source table

Currency	Rate	FromDate	ToDate
EUR	8.45	15/02/2013	vEndTime
EUR	8.69	28/01/2013	14/02/2013 23:59:59
EUR	8.59	vBeginTime	28/01/2013 23:59:99
USD	6.30	03/02/2013	vEndTime
USD	6.56	10/01/2013	2/02/2013 23:59:59
USD	6.50	vBeginTime	9/01/2013 23:59:59

In your app, the table appears as follows:

Preview of data

Currency	Rate	FromDate	ToDate
EUR	8.45	15/02/2013	01/03/2013
EUR	8.69	28/01/2013	14/02/2013
EUR	8.59	01/01/2013	28/01/2013
USD	6.30	03/02/2013	01/03/2013
USD	6.56	10/01/2013	2/02/2013
USD	6.50	01/01/2013	9/01/2013

This table can subsequently be used in a comparison with an existing date using the Intervalmatch method.

10.17 Loading hierarchy data

Unbalanced n -level hierarchies are often used to represent geographical or organizational dimensions in data, among other things.

These types of hierarchies are usually stored in an adjacent nodes table, which is in a table where each record corresponds to a node and has a field that contains a reference to the parent node.

Nodes table

NodeID	ParentNodeID	Title
1	-	General manager
2	1	Region manager
3	2	Branch manager
4	3	Department manager

In such a table, the node is stored on one record only but can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

An adjacent nodes table is optimal for maintenance, but difficult to use in everyday work. Instead, in queries and analysis, other representations are used. The expanded nodes table is one common representation, where each level in the hierarchy is stored in a separate field. The levels in an expanded nodes table can easily be used e.g. in a tree structure. The **hierarchy** keyword can be used in the data load script to transform an adjacent nodes table to an expanded nodes table.

Example:

```
Hierarchy (NodeID, ParentNodeID, Title, 'Manager') LOAD
    NodeID,
    ParentNodeID,
    Title
FROM 'lib://data/hierarchy.txt' (txt, codepage is 1252, embedded labels, delimiter is ',', msq);
```

Expanded nodes table

NodeID	ParentNodeID	Title	Title1	Title2	Title4	Title4
1	-	General manager	General manager	-	-	-
2	1	Region manager	General manager	Region manager	-	-
3	2	Branch manager	General manager	Region manager	Branch manager	-
4	3	Department manager	General manager	Region manager	Branch manager	Department manager

A problem with the expanded nodes table is that it is not easy to use the level fields for searches or selections, since prior knowledge about which level to search or select in is needed. An ancestors table is a different representation that solves this problem. This representation is also called a bridge table.

An ancestors table contains one record for every child-ancestor relation found in the data. It contains keys and names for the children as well as for the ancestors. That is, every record describes which node a specific node belongs to. The **hierarchybelongsto** keyword can be used in the data load script to transform an adjacent nodes table to an ancestors table.

10.18 Loading your own map data

To be able to create a map visualization, you need access to geographical data that connects to the data in your app.

Qlik Sense can use:

- Name data in fields to place locations in map layers.
- Fields containing geopoints (latitude and longitude) in WGS-84.
- Fields containing geopoints, polygons, or lines from a geographic data source such as a KML file.
- Fields containing geodata in GeoJSON, LineString, or MultiLineString formats.
- Fields containing non-WGS-84 coordinates (when using a custom map as the base map).

When loading map data in **Data manager** with data profiling enabled, the data profiling service will identify country names, city names, and latitude and longitude fields and load the corresponding geometries into new fields. In **Data load editor**, you can optionally combine coordinate fields into a single field for convenience.

Supported name data for fields in a map visualization

The map visualization can use name data in fields to place locations in map layers. The following location types can be used:

- Continent names
- Country names
- ISO alpha 2 country codes

- ISO alpha 3 country codes
- First-level administrative area names, such as a state or province names
- Second-level administrative area names
- Third order administrative area names
- Fourth order administrative area names
- City, village, or other populated place names
- Postal codes or ZIP Codes
- IATA airport codes
- ICAO airport codes



Availability of locations may vary by country. If the named location is not available, use coordinate or area data for the location.

Qlik Sense uses map and location data obtained from recognized field leaders who use accepted methodologies and best practices in marking borders and naming countries within their mappings. Qlik Sense provides flexibility to enable users to integrate their own, separate background maps. If the standard maps do not fit, Qlik Sense offers the option to load customer provided background maps, borders, and areas.

Loading point and area data from a KML file

You can add data from a KML file into your map in **Data manager** and **Data load editor**. By default, all fields are selected in the data selection dialog, even if they do not contain any data. A KML file could contain, for example, area data but no point data. When adding data from a KML file that contains an empty point or area field to Qlik Sense, you can exclude the empty field without running the risk of creating map dimensions without any data.

When adding a field from a KML field to a map layer, if the name field contains meaningful name data, it should be added as the dimension of the layer. The area or point field should then be added as the **Location field**. There will be no difference in how the data is visualized in the layer and the text in the name field will be shown as a tooltip.



If the KML file does not contain point data, line data, or area data, you cannot load data from that file. If the KML file is corrupt, an error message is displayed, and you will not be able to load the data.

Loading map data with data profiling

When you load geographical data using **Add data** in **Data manager** with data profiling enabled, Qlik Sense will try to recognize if your data contains:

- Country and city names from your data
- Geopoint data (latitude, longitude) for a single location, such as a city
- Area data (polygons of geopoints) to represent regions or countries

If successful, a new field containing geographical information is created automatically.



*When using **Add data**, data profiling must be enabled. This is the default selection. If you disable data profiling, the geographical data is not detected and the new field containing geographical information is not created.*

If cities are recognized during data preparation, the new field contains geopoints, and if countries are recognized the new field contains area polygon data. This field is named <data field>_GeoInfo. For example, if your data contains a field named **Office** containing city names, a field with geopoints named **Office_GeoInfo** is created.



Qlik Sense analyzes a subset of your data to recognize fields containing cities or countries. If the matching is less than 75 percent, a field with geographical information will not be created. If a field is not recognized as geographical data, you can manually change the field type to geographical data.

For more information, see [Changing field types \(page 54\)](#).

Fields with geographical information do not display the geopoint or polygon data in the **Associations** preview panel or in the **Tables** view. Instead, the data is indicated generically as [GEO DATA]. This improves the speed with which the **Associations** and **Tables** views are displayed. The data is available, however, when you create visualizations in the **Sheet** view.

Loading and formatting point data

You can create a map by using point data (coordinates). Two formats are supported:

- The point data is stored in two fields, one for latitude and one for longitude. You can add the fields to a point layer in the **Latitude** and **Longitude** fields in the point layer. Optionally, you can combine them into a single field. To combine them into a single field:
 - If you used **Add data** with data profiled enabled to load the table, the latitude and longitude fields are recognized, and a geopoint field is created automatically.
 - If you loaded the data using the data load script, you can create a single field with point data in [x, y] format, using the function `GeoMakePoint()`.
For more information, see *Example: Loading point data from separate latitude and longitude columns with the data load script (page 268)*.
- The point data is stored in one field. Each point is specified as an array of x and y coordinates: [x, y]. With geospatial coordinates, this would correspond to [longitude, latitude].
When using this format and loading the data in **Data load editor**, it is recommended that you tag the point data field with `$geopoint;`.
For more information: *Example: Loading point data from a single column with the data load script (page 268)*.

In the following examples we assume that the files contain the same data about the location of a company's offices, but in two different formats.

Example: Loading point data from separate latitude and longitude columns with the data load script

The Excel file has the following content for each office:

- Office
- Latitude
- Longitude
- Number of employees

The load script could look as follows:

```
LOAD
  Office,
  Latitude,
  Longitude,
  Employees
FROM 'lib://Maps/offices.xls'
(biff, embedded labels, table is (Sheet1$));
```

Combine the data in the fields `Latitude` and `Longitude` to define a new field for the points.

Run the script and create a map visualization. Add the point dimension to your map.

You can choose to create the dimension `Location` in the script by adding the following string above the **LOAD** command:

```
LOAD *, GeoMakePoint(Latitude, Longitude) as Location;
```

The function `GeoMakePoint()` joins the longitude and latitude data together.

It is recommended that you tag the field `Office` with `$geoname` so that it is recognized as the name of a geopoint. Add the following lines after the last string in the **LOAD** command:

```
TAG FIELDS Office WITH $geoname;
```

The complete script then is as follows:

```
LOAD *, GeoMakePoint(Latitude, Longitude) as Location;
LOAD
  Office,
  Latitude,
  Longitude,
  Employees
FROM 'lib://Maps/offices.xls'
(biff, embedded labels, table is (Sheet1$));

TAG FIELDS Office WITH $geoname;
```

Run the script and create a map visualization. Add the point dimension to your map.

Example: Loading point data from a single column with the data load script

The Excel file has the following content for each office:

- Office
- Location
- Number of employees

The load script could look as follows:

```
LOAD
  Office,
  Location,
  Employees
FROM 'lib://Maps/offices.xls'
(biff, embedded labels, table is (sheet1$));
```

The field `Location` contains the point data and it is recommended to tag the field with `$geopoint` so that it is recognized as a point data field. It is recommended that you tag the field `Office` with `$geoname` so that it is recognized as the name of a geopoint. Add the following lines after the last string in the **LOAD** command:

```
TAG FIELDS Location WITH $geopoint;

TAG FIELDS Office WITH $geoname;
```

The complete script then looks as follows:

```
LOAD
  Office,
  Location,
  Employees
FROM 'lib://Maps/offices.xls'
(biff, embedded labels, table is (sheet1$));

TAG FIELDS Location WITH $geopoint;

TAG FIELDS Office WITH $geoname;
```

Run the script and create a map visualization. Add the point dimension to your map.

10.19 Data cleansing

When loading data from different tables, note that field values denoting the same thing are not always consistently named. Since this lack of consistency is not only annoying, but also hinders associations, the problem needs to be solved. This can be done in an elegant way by creating a mapping table for the comparison of field values.

Mapping tables

Tables loaded via **mapping load** or **mapping select** are treated differently from other tables. They will be stored in a separate area of the memory and used only as mapping tables during script execution. After the script execution they will be automatically dropped.

Rules:

- A mapping table must have two columns, the first one containing the comparison values and the second the desired mapping values.
- The two columns must be named, but the names have no relevance in themselves. The column names have no connection to field names in regular internal tables.

Using a mapping table

When loading several tables listing countries, you may find that one and the same country has several different names. In this example, the U.S.A. are listed as US, U.S., and United States.

To avoid the occurrence of three different records denoting the United States in the concatenated table, create a table similar to that shown and load it as a mapping table.

The entire script should have the following appearance:

```
CountryMap:  
Mapping LOAD x,y from MappingTable.txt  
(ansi, txt, delimiter is ',', embedded  
labels);  
Map Country using CountryMap;  
LOAD Country,City from CountryA.txt  
(ansi, txt, delimiter is ',', embedded labels);  
LOAD Country, City from CountryB.txt  
(ansi, txt, delimiter is ',', embedded labels);  
The mapping statement loads the file MappingTable.txt as a mapping table with the label CountryMap.
```

The **map** statement enables mapping of the field *Country* using the previously loaded mapping table *CountryMap*.

The **LOAD** statements load the tables *CountryA* and *CountryB*. These tables, which will be concatenated due to the fact that they have the same set of fields, include the field *Country*, whose field values will be compared with those of the first column of the mapping table. The field values US, U.S., and United States will be found and replaced by the values of the second column of the mapping table, i.e. USA.

The automatic mapping is done last in the chain of events that leads up to the field being stored in the Qlik Sense table. For a typical **LOAD** or **SELECT** statement the order of events is roughly as follows:

1. Evaluation of expressions
2. Renaming of fields by as
3. Renaming of fields by alias
4. Qualification of table name, if applicable
5. Mapping of data if field name matches

This means that the mapping is not done every time a field name is encountered as part of an expression but rather when the value is stored under the field name in the Qlik Sense table.

To disable mapping, use the **unmap** statement.

For mapping on expression level, use the **applymap** function.

For mapping on substring level, use the **mapsubstring** function.

11 Customizing logical models for Insight Advisor

Business logic enables you to customize how Insight Advisor interprets your app data when generating analysis from queries.

Business logic defines how Insight Advisor interprets your data and handles alternative terms for values in your data model with:

- Insight Advisor Search
- Insight Advisor Analysis Types
- Insight Advisor Chat
- Associative insights

Insight Advisor relies on the Qlik cognitive engine and learned precedents to understand the relationships and uses of fields in your data model. You can optionally customize the logical model used by Insight Advisor for an app. You can add vocabulary to your business logic that helps Insight Advisor handle alternate terminology users might use when querying Insight Advisor.

You can customize the following areas of business logic in Qlik Sense:

- **Logical model:** Customize the fields and groups used in the model, create packages, define field hierarchies, and set behavior.
Building logical models for Insight Advisor with Business logic (page 272)
- **Vocabulary:** Add terms and associate them to fields and values in your data to enable users to use alternative terminology when using natural language questions with Insight Advisor.
Creating vocabularies for Insight Advisor (page 292)

Business logic options are available in the **Prepare** tab.

As of August 2022, Insight Advisor, including business logic, is no longer supported with Qlik Sense Desktop. In November 2022, Insight Advisor will be upgraded to a new experience. This will only be available on Qlik Sense Enterprise on Windows. Users who want to continue using Insight Advisor and business logic on Qlik Sense Desktop should not upgrade to August 2022.

11.1 Building logical models for Insight Advisor with **Business logic**

Insight Advisor uses a logical model based on learned precedents to generate analyses based on your queries. You can define your own logical model for your apps with **Business logic**.

Building logical models for Insight Advisor with Business logic



Insight Advisor relies on the Qlik cognitive engine and learned precedents to understand the relationships and uses of fields in your data model. Optionally, you can customize the logical model to improve Insight Advisor results. You can customize your logical model in **Logical model** under **Business logic** in the **Prepare** tab.



When business logic is enabled in an app, precedent-based learning is disabled for the app.

Understanding logical models

The logical model of an app is the conceptual model Insight Advisor uses when generating visualizations. It is built from the underlying data model of an app. Each app has a single logical model. Fields and master items are the core components of the logical model. They are organized into groups. Groups indicate a conceptual association or relationship between fields or master items. The logical model also contains information about possible relationships between groups.

The logical model directly influences how Insight Advisor functions. For example, when a user selects a field to show a trend analysis, Insight Advisor tries to find a date field that is part of a primary calendar group. If the field was *Sales*, Insight Advisor would prioritize a field like *Order Date* over the field *Employee Birth Date*.

Business logic also affects how the system chooses between fields in natural language questions. For example, the fields *Product Name* and *Product Code* are grouped as a single group. If '*what are sales by product*' is used as a natural language question, *Product Name* would be used for '*product*' as it is a better choice for that group.

The default logical model used for business logic is a star schema. Business logic enables you to construct different modeling for your app if a star schema is not optimal. Business logic can also help constrain aggregation in logical models containing semi-additive measures or very large measure tables. This improves the exploration of app data in Insight Advisor.

Customizing logical models

Logical model is divided into the following sections for customizing the logical model of an app:

- **Overview:** **Overview** provides a summary of your business logic. Clicking the cards for **Fields & groups**, **Packages**, **Hierarchies**, or **Behaviors** opens the corresponding section.
- **Fields & groups:** **Fields & groups** enables you to define the groups to which your fields and master items belong in the logical model.
- **Packages:** **Packages** enables you to create collections of related groups. This prevents groups from being used together that are not in the same package.
- **Hierarchies:** **Hierarchies** enables you to define drill-down relationships between groups.
- **Behaviors:** **Behaviors** enables you to specify prefer or deny relationships between fields. Behaviors can also enforce required selections.
- **Calendar periods:** **Calendar periods** enables you to create default periods of analysis for Insight Advisor.

To customize a logical model, do the following:

1. Enable the customization of the business logic of your app.
2. Define your fields and groups.
Defining fields and groups (page 275)
3. Optionally, add your groups to packages.
Setting logical model scope with packages (page 279)
4. Optionally, define hierarchies between groups
Creating drill-down analysis with hierarchies (page 281)
5. Optionally, apply behaviors.
Applying behaviors to logical models (page 282)
6. Optionally, create calendar periods.
Defining analysis periods with calendar periods (page 284)

You can reset your logical model to the default. You can also disable business logic temporarily.

Enabling custom business logic

Do the following:

1. In an app, click **Prepare** and select **Logical model**.
2. Click **Continue**.

Custom business logic is now enabled for your app. Precedent-based learning is now disabled.

Resetting business logic

You can reset your logical model to the default model. Resetting disables custom business logic and enables precedent-based learning in Insight Advisor.

Do the following:

1. In **Logical model**, click **Reset to default**.
2. Click **Confirm**.

Disabling business logic

You can disable custom logical models. Unlike resetting business logic, you can enable your custom business logic again later. While your business logic is disabled, it will use the default business logic for your app.

Do the following:

1. In **Logical model**, click **Disable logic**.
2. Click **Confirm**.

Defining fields and groups

Field and groups are the primary component of logical models in business logic. By defining the groups to which your fields and master items belong, you can define how Insight Advisor should use them.



You define how business logic should handle fields and groups in the **Fields & groups** section of **Logical model**. Fields and master items can be grouped together to indicate a relationship in Insight Advisor analysis. For each item in a group, you can define how Insight Advisor should treat it in analysis.

Groups enable you to organize fields and master items into related conceptual groupings. For example, you can group all the fields related to customers in a single group, regardless of which tables in the data model

from which they are loaded. Insight Advisor uses this information to determine which fields to show together in visualizations. Groups can also be used to create packages, which limits scope for Insight Advisor to only use related groups together. There are three kinds of groups:

- **Dimension:** A dimension group commonly consists of fields that are classified as dimensions. Dimension groups can also contain fields classified as measures or dates.
- **Measure:** A measure group consists of related measure fields. Only measures can belong to a measure group.
- **Calendar:** A calendar group contains a time dimension in your logical model. Calendar groups can only contain dimensions and are expected to have at least one temporal fields (such as date, timestamp, or year). Calendar groups are useful if you have separate fields defining your calendar, such as *year*, *month*, and *day*. If you can also group other data-related fields, such as *fiscal quarter* or *fiscal year*.

By default, fields and master items are ordered by group. If you disable this, an additional field, **In group**, is added to the table.

In addition to defining groups, you can also define the properties of individual fields and master items belonging to your groups.

Creating groups

Once you create a group, the group type cannot be changed. You can rename the group and add or remove fields from the group. Groups can also be deleted. Deleting a group ungroups all items in the group.

To edit an existing group, click  in the group row or after the group name in the **In group** column.

Do the following:

1. Click **Create group**.
2. Enter a group name.
3. Select a group type.
4. Add fields to the group from **Available fields**.
5. Click **Create**.

Defining fields and master items

Fields & groups consists of a table containing fields and master items from your app, along with the groups to which they belong. You can edit the properties of your fields and master items by adjusting the column value in that item's row. For additional options, you can click  in the row. You can:

- **Move item:** Move the item to a different group.
- **Create behavior:** Create a behavior for the group to which the current item belongs.
- **Ungroup:** Remove the item from its current group. Ungrouped items are excluded from Insight Advisor.

You can also select multiple rows using the row checkboxes to make the same changes to multiple items. When selected in this way, properties settings and options are available above the table.

Field and master item properties

The table is divided into the following columns:

- Name
- Group
- Visibility
- Classification
- Data value lookup
- Default aggregation
- Favorable trend
- Favorite
- Overall aggregation
- Default period grain

Name lists the field names. **Group** lists the group name to which the field belongs. The following section outlines the different values and settings of the other fields. You can set which columns display in the table by clicking .

Visibility

Visibility controls if an item is available in Insight Advisor. There are two possible values:

- **Visible**: The item is available for use in Insight Advisor.
- **Hidden**: The item is not available for use in Insight Advisor. Hidden fields should not be enabled for data value lookup.

You can hide all hidden items in **Field & groups** by selecting **Show visible only**.

Classification

Classification defines the default role that attribute can play in an analysis. The following types can be used to classify fields and groups:

- **dimension**: A field that is only to be used as a dimension
- **measure**: A field that is to only be used as a measure.
- **boolean**: A binary dimension.
- **date**: A temporal dimension containing dates.
- **timestamp** : A temporal field containing timestamps.
- **year**: A temporal dimension containing year data.
- **week**: A temporal dimension containing week data.
- **quarter**: A temporal dimension containing quarter data.
- **month**: A temporal dimension containing month data.
- **weekDay**: A temporal dimension containing data for the day of the week, either short form (Mon., Tues.), long form (Monday, Tuesday), or a number between 1-7.
- **monthDay**: A temporal dimension containing a number between 1-31 indicating the day of the month.
- **yearDay**: A temporal dimension containing a number for the day of the year (between 1-366).
- **hour**: A temporal dimension containing hour data.
- **email**: A dimension containing email addresses.
- **address**: A dimension containing addresses.

- **country**: A dimension containing country names.
- **stateProvince**: A dimension representing first-level administrative areas, such as states and provinces.
- **city**: A dimension representing cities.
- **geoPoint**: A dimension containing geographic point data.
- **geoPolygon**: A dimension containing geographic polygon data
- **geographical**: A dimension representing a geographic location, such as country or region.
- **postalCode**: A dimension containing postal codes.
- **longitude**: A dimension containing longitude data.
- **latitude**: A dimension containing latitude data.
- **percentage**: A measure field representing percentage values such as employment rate or inflation.
- **monetary**: A monetary measure such as revenue, cost, or salary.
- **ordinal**: A dimension whose values have inherent order.
- **temporal**: A time-related dimension.

You may have fields that might be considered a dimension in one question and a measure in another question. As a best practice, it is recommended to create a second field or master item for the alternate use case of the field.

You may have fields that might be considered a dimension in one query and a measure in another query. As a best practice, it is recommended to create a second field or master item for the alternate use case of the field.

Data value lookup

Data value lookup controls whether or not Insight Advisor can look up values from fields when a user asks a natural language question.

Reducing the number of fields that have data value lookup enabled can help you avoid false positive results and to reduce the time to generate results. For example, you may have three fields containing names in your data model: *First Name*, *Last Name*, and *Full Name*. If **Data value lookup** was enabled for all three fields, users might get confusing results from all three fields if they search for ‘Drew’.



Data value lookup should be disabled for measures and hidden fields.

Default aggregation

Default aggregation sets the standard aggregation for measures in Insight Advisor. The following aggregations can be used:

- **sum**
- **avg**
- **min**
- **max**
- **count**
- **countDistinct**

When field has a default aggregation, Insight Advisor always applies that aggregation when using it as a measure. Users can edit charts to change the aggregation to a different type in Insight Advisor.

Default aggregations cannot be assigned to master items.

Favorable trend

Favorable trend sets whether the desired trend for a measure is to increase or decrease. By default, the favorable trend for measures is to increase. The following trends can be used:

- **Up**
- **Down**

The favored trend is used in visualizations created by Insight Advisor to ranking results.

Favorite

Favorite marks a measure as being of particular interest for Insight Advisor. Insight Advisor will use this measure more often when generating visualizations without user queries or selections, such as with **Generate**.

Overall aggregation

Overall aggregation indicates to Insight Advisor what aggregation should be used when Insight Advisor cannot determine for itself what aggregation to use in natural language insights for results involving master measures. As some master measures use complex expressions, Insight Advisor may not know the aggregation to use when calculating the contribution of the master measure in the narratives. Setting an overall aggregation allows Insight Advisor to calculate narratives for these complex master measures.

You can only set overall aggregation for master measures.

Default period grain

You can set the default period grain, such as month or year, to be used with a date field in a calendar group when generating analyses in Insight Advisor. The grains are either derived from the autocalendar or you can pick a field containing a grain from the same calendar group.

You can use the default calendar setting with the default calendar period behavior. Default calendar properties set for fields and master items are overruled by default calendar period behaviors set for groups.

Setting logical model scope with packages

Packages enable you to define and limit the scope of insights for specific areas of interest in your logical model.

Setting logical model scope with packages



A package is a collection of related groups. Insight Advisor only uses groups in the same package together when generating results. Groups can belong to multiple packages. Logical model packages are created and managed in **Packages**.

For example, your data model may contain tables from separate business areas. If there are connections between the tables, a date field from *Sales* might be used in an Insight Advisor analysis with a date field from *Support*. By putting *Sales* groups and *Support* groups into separate packages, you could ensure that analysis for those business areas only use fields in the associated package.

The use of packages is optional. If you define no packages, then all fields and groups are considered to be in the same package for analysis.

You can edit or delete packages by clicking **...** and selecting **Edit** or **Delete**.

Creating packages

Do the following:

1. Click **Create package**.
2. Enter a package name.
3. Add groups to the package from **Available groups**



Packages must include at least one measure group.

4. Click **Create**.

Creating drill-down analysis with hierarchies

Defining which groups in your logical model have a hierarchical relationship enables Insight Advisor to offer breakdown analysis along specified hierarchies and better detect dependencies among groups.



Hierarchies enable drill-down breakdowns in recommendations (such as *Sales by Product category, Product sub-category, and Product*). Hierarchies also help Insight Advisor avoid mixing geographic items in recommendations. For example, creating hierarchies for *Supplier County* with *Supplier City* and for *Customer Country* and *Customer City* prevents them from being used in erroneous combinations such as *Supplier Country* and *Customer City*.

Hierarchies are also used in associative insights to detect dependencies when making selections. For example, selecting *Sweden* in an Insight Advisor analysis and then having only Swedish city values be selected in another chart that uses the *City* field.

There two kinds of logical model hierarchies: learned and defined. Learned hierarchies are detected from how fields are linked and used in the data model. For example, the logical model may see the hierarchical link between a city field and a country field and treat them as having a hierarchy. Defined hierarchies are user-created. Defined hierarchies are created and managed in **Hierarchies**. Hierarchies are optional for business logic.

You can edit or delete defined hierarchies by clicking  and selecting **Edit** or **Delete**.



*You cannot view or modify learned hierarchies in **Hierarchies**.*

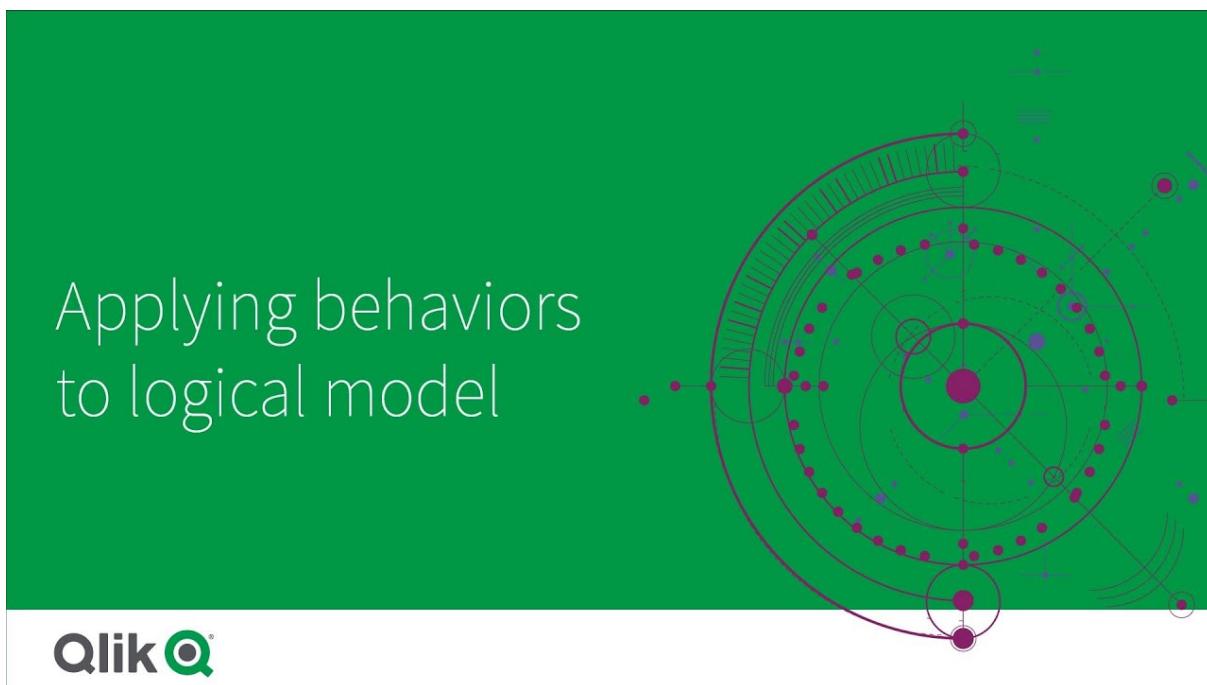
Creating hierarchies

Do the following:

1. Click **Create hierarchy**
2. Enter a hierarchy name.
3. Add groups to the hierarchy in order from lowest level in the hierarchy to the highest level of the hierarchy.
4. Click **Create**.

Applying behaviors to logical models

Behaviors enable you to set prefer or deny relationships between measure groups and other groups. You can also use behaviors to enforce value selection in Insight Advisor.



You may have groups that either should be always used together in analysis or never used together in analysis. You also maybe have field values that you would prefer always been included as selected when fields from a group are being used by Insight Advisor. Behaviors enable you to set these preferences on a group by group basis. Logical model behaviors are created and managed in **Behaviors**.

The following behaviors are available in logical models:

- Required selections
- Prefer relationship
- Deny relationship
- Default calendar period

You can edit or delete behaviors by clicking *** and selecting **Edit** or **Delete**.

Required selections

Required selection behavior enables you to specify field values that must be included when using fields from a group. A behavior can include multiple required selections.

For example, you may have fields for *Country*, *Population*, and *Year*. When generating recommendations in Insight Advisor, you might get charts that use *Country* and *Population*, but include the sum of data for all years in the charts. You can use required selection behavior to limit Insight Advisor results to use the current year instead of all time when generating analysis for *Population* or *Country*.

Prefer relationships

Prefer relationship behavior guides Insight Advisor into selecting groups that should be used more often together when generating analysis . When you specify a prefer relationship, Insight Advisor uses the preferred group when generating results. Prefer relationships are useful when there might be ambiguity .

Prefer relationships does not prevent a group from being used with other groups. It only selects the preferred group when all group choices are equal in the analysis. In analysis where other groups are more appropriate, the non-preferred groups may be used instead.

For example, there are four groups:

- *Sales*
- *Customer*
- *Product*
- *Sales Person*

Sales has a preferred relationship with *Customer*. For a breakdown analysis with *Sales*, *Customer* is selected over the other groups as it is preferred. For a trend analysis for *Sales*, *Product* might be used in the analysis instead.

Deny relationships

Deny relationship behavior prevents Insight Advisor from using the selected groups together when generating insights. This is useful when some groups in the same package may not be useful to use together in analysis. Deny relationships can also be used to block groups from being used together that could impact app performance. With star schema data models, there may be groups that have one-to-many and many-to-one relationships that complicate analysis. Deny relationships can prevent association between these groups.

Deny relationships are overruled when a user specifically requests insights from the groups that have a deny relationship. For example, *Sales* and *Supplier* have a deny relationship. If someone searches 'show me sales', no analysis will be generated that contains *Sales* and *Supplier*. If someone searches 'show me sales by supplier', results including *Sales* and *Supplier* are generated.

Default calendar period

Default calendar period behaviors assign calendar periods to be used as the default time period in visualizations for the selected group. The default calendar period is applied whenever Insight Advisor creates visualizations for fields from that group. Groups can have single default calendar period. Default calendar periods behaviors overrule any default period grains set for a date field.

Calendar periods are generally applied to analyses in the following way:

- Fact and rank analyses use the current or first selected period from the calendar period. The previous or second selected period from the calendar period is added to the analyses for comparison.
- Trend analyses and similar analyses only use the aggregated grain from the calendar period.
- Other analyses use the current or first selected period from the calendar period.

For example, you have the group *Client Satisfaction* that contains client satisfaction data. By default, many of your app's users are only interested in seeing data for the current month. By creating a calendar period for the current month and making it the default calendar period, Insight Advisor visualizations for *Client Satisfaction* will only show data from the current month.

Ignoring default calendar periods for select analysis types

You can configure analysis types to ignore parts or all of a default calendar period. This is useful for controlling how default calendar periods are applied more specifically. For example, you may not want a comparison in your fact analyses and only want to view the current period.

You can pick analysis types to ignore period 1 (the previous or second selected period), period 2 (the current or first selected period), or the grain of your calendar period. Ignoring period 1 stops using earlier period in analyses that would compare two periods so that only period 2 is displayed. Ignoring period 2 or the grain stops the calendar period from being used by default with that analysis type.

Creating behaviors

Do the following:

1. Click **Create behavior**.
2. Under **Applies to**, select a group.
3. Select a behavior type
4. Do one of the following:
 - If configuring a **Prefer relationship** or a **Deny relationship**, select the groups to which to apply that relationship.
 - If configuring a **Required selection**, select if this is a single value. From **Require for**, select the fields and their required values.
You can add additional required selections by clicking **Add another**.
 - If configuring a **Default calendar period**, select a calendar group and a calendar period from that group. Optionally, specify which analysis types should have exceptions from the default calendar period under **Ignore Period 1**, **Ignore Period 2**, and **Ignore grain**.
5. Click **Create**.

Defining analysis periods with calendar periods

Define calendar periods for your calendar groups to control the time frames that Insight Advisor uses when creating visualizations.

Defining analysis periods with calendar periods



Calendar periods define time periods of interest for Insight Advisor to use in analysis in the logical model. For example, a KPI showing total sales for all time may be less useful than a KPI showing total sales for the current month. You could create a calendar period for the current month and set that as the default period to use with total sales. Calendar periods for logical model are created and managed in **Calendar periods**.

Calendar periods define single periods of times or comparisons of relative periods of time. For example, you can create calendar periods to:

- Show only the values from this month
- Compare this month to last month
- Compare the current quarter to the current quarter last year

Insight Advisor uses calendar periods when creating charts. Calendar periods enable additional analysis types. You can select and apply calendar periods when editing a chart in Insight Advisor. You can set a default calendar period to be used with a group when Insight Advisor creates charts for fields and master items in that group. Default calendar periods are set in **Behaviors**.

Creating calendar periods

Create calendar periods from the calendar groups defined in **Fields & groups**. You can create calendar periods with or without fields that use the `autoCalendar` function in the data load script.



You can use the **Use Autocalendar** option with a custom calendar if it uses the same declared fields as `autoCalendar`.

Creating calendar periods using autocalendar

Select the grain for analysis when creating calendar periods using the autocalendar. The grain is for example, month of year or quarter of year. You can then choose to use most recent value, or create a comparison.

Do the following:

1. Click **Create calendar period**.
2. Select a calendar group, and then select **Use Autocalendar**.
3. Enter a name for the calendar period.
4. Select the calendar period grain.
5. Do one of the following:
 - To use the most recent value in the period grain field, select **Use last sorted value**.
 - To make a comparative calendar period, do one of the following:
 - In **Compare**, select the period for comparison. Select **Last complete period** if Insight Advisor should use the last complete period instead of the current period.
 - In **Custom**, select the period for analysis in **Offset**, and then select the period for comparison in **Compare offset**.
Offset and **Compare offset** use numeric values, with 0 being the current period.
For example, if you used quarter of year as your period grain, 0 would be the current period and 3 would be three quarters ago.
6. Click **Create**.

Creating calendar periods using a custom calendar

You can create calendar periods without using the autocalendar in the load script. For example, you may have a custom calendar, or you may want to use custom calendar flags that do not exist in the autocalendar.

To create a calendar period using a custom calendar, select a calendar group and choose a time field to aggregate. You then choose to use most recent value or create a comparison. Two comparison methods are supported:

- **Relative:** A relative comparison that uses a field containing the relative number of periods ago. The relative period field must contain numeric values that define the relative period from the current date for each value in the aggregated field. You can then select the offset and comparison offset for analysis.
For example, to do a comparison of this month to last month, select *Date* as your aggregate field. Next, select *MonthsAgo* as your relative period field. *MonthsAgo* uses an expression to evaluate how many months ago each value in *Date* is from the current month. If the current month is July, values from July would be 0, values from June would be 1, values from May would be 2, and so on. You can then select 0 for your offset and 1 for your compare offset.
- **Flag:** A flag comparison uses two fields that define two separate flagged periods of time.
For example, first quarter and second quarter could be two periods used for a flag comparison. The comparison fields must contain binary values indicating which aggregated field values are in the flagged period.
To compare Quarter 1 to Quarter 2, select *Date* as the aggregated field. As flag fields, you can then select *InQuarter1* and *InQuarter2*. These fields use expressions to evaluate if values are or are not in Quarter 1 or Quarter 2.

Do the following:

1. Click **Create calendar period**.
2. Select a calendar group, and then clear the **Use Autocalendar** check box.
3. Enter a name for the calendar period.
4. Select the aggregated date field to use.
5. Do one of the following:
 - To use the most recent value in the period grain field, select **Use last sorted value**.
 - To make a comparative calendar period, do one of the following:
 - In **Relative**, for **Relative periods ago**, select the field defined in your load script containing the relative time period data for the field selected in **Aggregated date**. Set the time period for analysis in **Offset**, and then set time period for comparison in **Compare offset**.
Offset and **Compare offset** use numeric values, with 0 being the current period.
 - In **Flag**, select the field containing the flag for the current period, and then select the field containing the flag for the comparison period.
6. Click **Create**.

Limitations

Business logic calendar periods have the following limitations:

- Default calendar periods are not applied to time-based master measures that include specific time periods in their expression.

Step-by-step – Creating calendar periods using a custom calendar

This step-by-step walkthrough shows you how to create calendar periods using custom calendar fields and flags.

Calendar periods can be made either by using the autocalendar or by using individual date/time fields from your data. You can also use fields containing binary data to flag periods of time for comparative analysis.

There are three kinds of calendar periods you can make with custom calendar data:

- Last sorted value: Last sorted value calendar periods show the most recent period in the selected aggregated field. In Insight Advisor analyses supporting comparisons, such as rank analysis, the last sorted value shows the previous period as well.
- Relative comparison: Relative calendar periods use a field containing the relative periods of data from the current date. It provides a comparison between the current or previous period and an older period.
- Flag comparison: Flags use boolean classified fields containing binary data to flag two periods of time for comparative analysis.



To demonstrate relative time period comparisons, the data source for this app contains data for future dates. The load script loads data from the data source up to the current date. The results in the images may differ from your results as the time periods shown in the images will have changed.

Getting started

Download the example package and unzip it:

[Calendar periods example app](#)

The QVF file contains the following data file:

- TutorialCustomCalendarData.xlsx
To demonstrate relative comparisons possible through calendar periods,
TutorialCustomCalendarData.xlsx contains data for future dates. The app load script updates the in app data for the current date when loaded.

Import the QVF file in Qlik Sense and attach the XLSX file to your app. Once you have imported the app and attached the data file to the app, load the app data in **Data load editor**.

Example data

The data used in this example is loaded with the following load script:

```
Sales:  
LOAD  
City,  
Country,  
Customer,  
OrderDate,  
Sales,  
"Q4-2020",  
"Q1-2021",  
"Q2-2021",  
"Q3-2021",  
"Q4-2021",  
"Q1-2022",  
"Q2-2022",  
"Q3-2022",  
"Q4-2022",  
"Q1-2023",  
"Q2-2023",  
"Q3-2023",  
Month([OrderDate]) AS [Month],  
Year([OrderDate]) AS [Year],  
Day([OrderDate]) AS [Day],  
Dual(Year(OrderDate)& '-'&Month(OrderDate), monthstart(OrderDate)) AS [YearMonth],  
12*Year(Today())+Month(Today())-12*Year(OrderDate)-Month(OrderDate) AS [MonthsAgo]  
FROM [lib://AttachedFiles/TutorialCustomCalendarData.xlsx]  
(ooxml, embedded labels, table is Sales) where OrderDate <= Today(1);
```

The load script creates separate fields for *Year*, *Month*, and *Day*. These three fields are used to create the following calculated fields:

- *YearMonth*, which has year and month information. This is the primary field for aggregation in the example.
- *MonthsAgo*, which calculates if months are from a specific month relative to the current date.

The data also contains several fields for the different quarters covered in the data. These fields contain binary data that indicates to which fiscal quarter each value in the *Sales* table belongs.

Tasks

This walkthrough takes you through creating three different kinds of calendar periods:

- Create a custom calendar period without autocalendar
- Create a relative calendar period
- Creating a flag comparison calendar period

Creating a calendar period using the last sorted value

For the first calendar period, you will create a calendar period for *YearMonth* using the last sorted value.

Do the following:

1. In the example app, click **Prepare**.
2. Under **Business logic**, select **Logical model**.
3. Click **Create calendar period**.
4. Select *OrderDate*.
5. For **Calendar period name**, enter *Last sorted month*.
6. For **Aggregated date**, select *YearMonth*.
7. Select **Use last sorted value**.
8. Click **Create**.

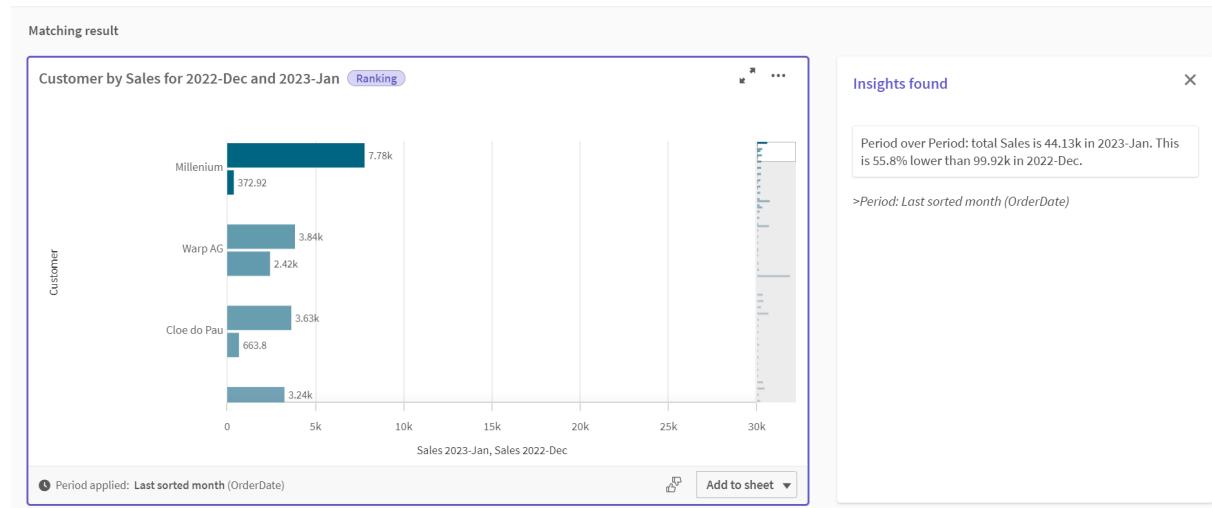
Result

Navigate to **Sheet** and search for *show me sales by customer*.

Select the chart *sum(Sales) by customer* and apply the analysis period *Last sorted month*. The chart updates to display a comparison between the current month and the previous month.

11 Customizing logical models for Insight Advisor

Last sorted month applied to Customer by sum(Sales) between 2022-Dec and 2023-Jan



Creating a relative comparison calendar period

Next, you will make a relative calendar period. A relative calendar period requires:

- An aggregation field containing a period of time (year, month, quarter, etc).
- A field containing the relative positions of dates from that field to today's date.

From these fields, you then define the offset. The offset is the relative difference, in the selected period of time, from the current date for the two periods of comparison. You can compare the current or previous period (set as 0 or 1 in **Offset**) to an older period up to 12 periods ago (set as a number from 1 to 12 in **Compare offset**).

For this calendar period, we will use *YearMonth* as the aggregation field and *MonthsAgo* as the relative periods field. We want to compare the current month to the same month last year.

Do the following:

1. Click **Create calendar period**.
2. Select *OrderDate*.
3. For **Calendar period name**, enter *This month to this month last year*.
4. For **Aggregated date**, select *YearMonth*.
5. Under **Relative periods ago**, select *MonthsAgo*.
6. Under **Offset**, select 0.
7. Under **Compare offset**, select 12.
8. Click **Create**.

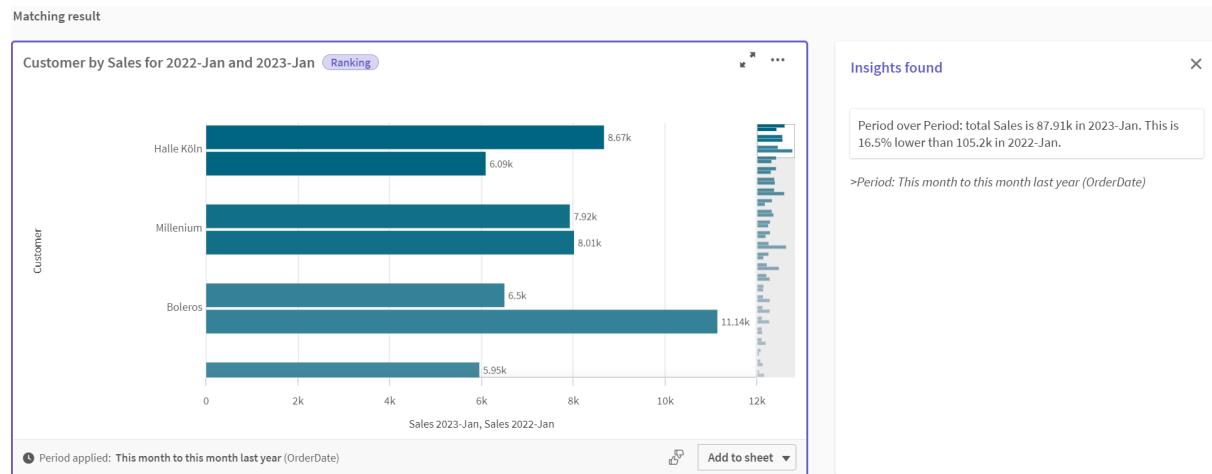
Result

Navigate to **Sheet** and search for *show me sales by customer*.

Select the chart *sum(Sales)* by *customer* and apply the analysis period *This month to this month last year*. The chart updates to display a comparison between the current month and the current month last year.

11 Customizing logical models for Insight Advisor

Customer by sum(Sales) between 2022-Jan and 2023-Jan



Creating a flag comparison calendar period

Flag comparison calendar periods use two fields to flag two separate periods for analysis from an aggregated date field.

In the example app data, there are separate fields for the different fiscal quarters. Each field has binary data indicating if the corresponding dates are or are not in the quarter. You will use these with *YearMonth* to make a flag comparison calendar period.

Do the following:

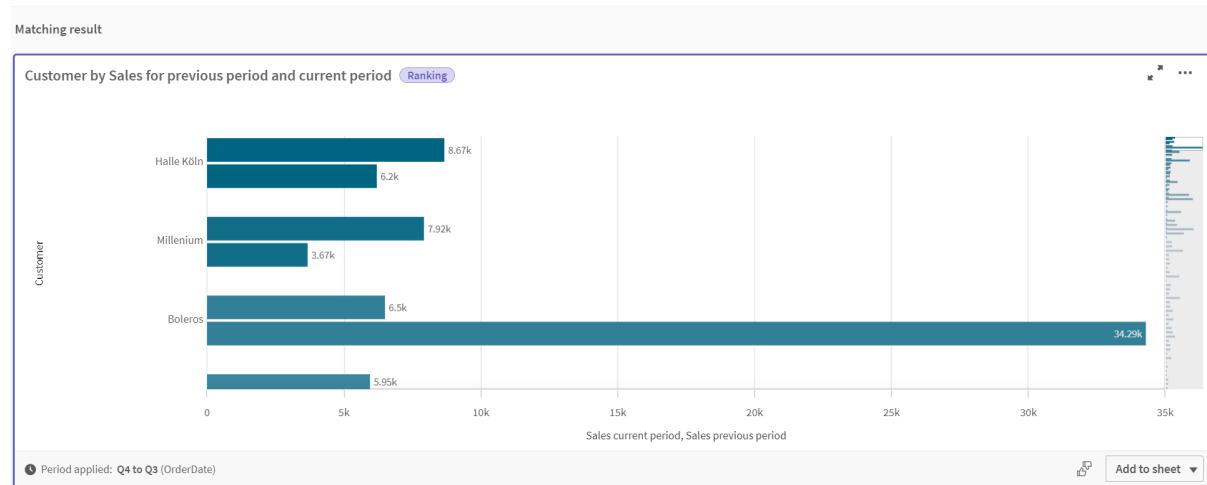
1. Click **Create calendar period**.
2. Select *OrderDate*.
3. For **Calendar period name**, enter *Q4 to Q3*
4. For **Aggregated date**, select *YearMonth*.
5. Click **Flags**.
6. Under **Current period flag**, select *Q4-2020*.
7. Under **Compare period flag**, select *Q3-2020*.
8. Click **Create**.

Result

Navigate to **Sheet** and search for *show me sales by customer*.

Select the chart *sum(Sales) by customer* and apply the analysis period *Q4 to Q3*. The chart updates to display a comparison between the fourth fiscal quarter of 2020 and the third fiscal quarter of 2020.

Customer by sum(Sales) between Q4 and Q3



11.2 Creating vocabularies for Insight Advisor

You can create vocabularies for Insight Advisor. This allows you to define synonyms for Insight Advisor and define analyses to be used with specific terms.

You create vocabularies in **Vocabulary** under **Business logic** in the **Prepare** tab. Vocabularies help improve the success of natural language questions. For example, you can use vocabulary to:

- Add alternative names for fields, master items, and values.
- Define names for coded values.
- Define the Insight Advisor analysis type to be used with certain terms or questions.
- Provide sample questions for users to ask.

Vocabulary supports the following kinds of vocabulary:

- **Synonyms:** Synonyms allow you to add alternate terms.
For more information, see *Adding synonyms to Insight Advisor* (page 293).
- **Custom analysis:** Custom analysis allow you to define what analyses should be used with select terms.
For more information, see *Adding custom analyses to Insight Advisor* (page 294).
- **Example questions:** Add sample questions users can select to help guide analysis of your app.
For more information, see *Adding example questions to Insight Advisor* (page 299).

Limitations

Vocabularies have the following limitations:

- Vocabularies are not supported in associative insights.
- Vocabularies do not support creating terms for fields classified as dates.
- The following words are reserved for use in Qlik Sense and cannot be used as a vocabulary term:
 - App
 - Application

- Dimensions
 - Dim
 - Measure
 - Msr
- Insight Advisor does not lemmatize or stem vocabulary terms. For example, if the term is *open*, a question using *opened* or *opening* will not be treated as having entered *open*. Similarly, if the term was *opened*, *open* and *opening* would not be recognized as the term.
 - If a vocabulary term also exists as a field value, and a question includes the field containing that value and the shared term, Insight Advisor won't use the vocabulary term and will use the field value instead when processing the question as the question included the field.

Adding synonyms to Insight Advisor

You can create vocabularies for Insight Advisor. This enables you to define terms and values that may be used in questions that are not present in your data model.

You can create vocabularies in **Vocabulary** under **Business logic** in the **Prepare** tab. Vocabularies help improve the success of natural language queries. For example, you can use vocabularies to define:

- Alternative names or synonyms for fields, master items, and values.
For example, *Earnings*, *Proceeds*, and *Revenue* for the field *Income*.
- Names for coded values
For example, names for medical classifications codes.
- Names for groups of values from a field
For example, defining named age ranges for a field containing ages.
- Common acronyms and abbreviations not in the data.
For example, *yr* for *year* or *num* for *number*.

When you create a vocabulary, you define the terms associated to that vocabulary. You then associate the terms to fields and master items in your app. Optionally, terms can be associated to individual values from the selected fields and master items. The following conditions are available when defining terms for values from fields:

- Greater than
- Greater than or equal to
- Less than
- Less than or equal to
- In
- Not in
- In range



The availability of conditions for fields varies depending on the values in the fields.

Vocabularies are created by language for an app. Each language's vocabulary is separate.



Qlik Sense supports English, French, Russian, and Spanish for vocabularies.

Do the following:

1. In an app, click **Prepare** and select **Vocabulary**.
2. Select a language from the language drop-down.
3. Click **Create terms**.
4. Enter terms for the vocabulary.
5. Select the field or master item to which the vocabulary applies.
6. Optionally, select a condition and the values to which the condition applies.
7. Click **Create**.

Limitations

Synonyms have the following limitations:

- Master items and temporal fields do not support conditions.

Adding custom analyses to Insight Advisor

You can define specific analyses to be used for certain terms. This helps you control what analyses users receive in Insight Advisor and create preferred analyses.

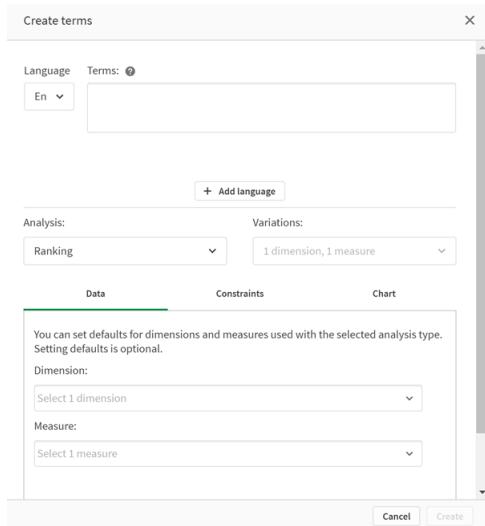
With custom analyses, you can define the response for specific terms and control the analyses returned by Insight Advisor. You can define multiple terms for a custom analysis. You can also define different terms in different languages for a custom analysis.



In Qlik Sense Client-Managed, vocabularies are only supported in English.

11 Customizing logical models for Insight Advisor

Custom analysis



When you create a custom analysis, you add terms to the analysis and then select the analysis type. You can then specify default dimensions or measures. When no defaults are set, Insight Advisor determines the appropriate dimensions or measures to use in the analysis based on the user's question.

Optionally, you can set constraints on your analysis type, to limit the scope. For example, you may want only values that exceed a certain measure value to be included in the analysis.



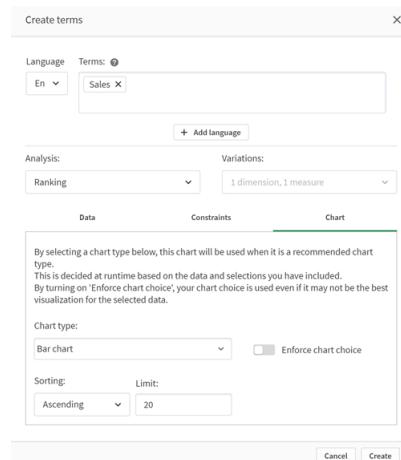
The availability of conditions for fields varies depending on the values in the fields.

You can optionally choose the chart type to use in your analysis from those available to your analysis type. If you select a chart type, the custom analysis will always use that chart type unless it determines another chart is more appropriate for the selected data. You can choose to enforce your chart so that only that chart type will be used, even if there are more appropriate charts for the question. For example, you can enforce a pie chart. If a search uses a dimension that contains 200 distinct values, the pie chart will be used even if it is not the optimal chart for the data.

Some analysis types also let you set sorting options for the chart and optionally apply limits for how many values are displayed. For example, you could configure a custom analysis with the Ranking analysis type, sorted by ascending values and with a limit of 20 dimension values shown. The image below shows how this can be set up.

11 Customizing logical models for Insight Advisor

Custom analysis dialog with options for chart sorting



Analysis types

The following table outlines the analysis types available when making custom analyses. Some analyses have different variations from which to choose. These determine what kind of field are used in the analysis.

Analysis types

Analysis type	Use	Variations
Breakdown	Break down a measure by multiple dimensions.	2-3 dimensions, 1 measure
Breakdown (geospatial)	Group data by simple and hierarchical geographic divisions.	1-2 geographic dimensions, 0-2 measures
Calculated measure (KPI)	Display aggregated totals.	1 measure
Clustering (k-means)	Compare 2 measures over a dimension by clustering the results using the KMeans2D function.	1 dimension, 2 measures
Comparison	Compare two to four measures over one or two dimensions. Two variations are supported: <ul style="list-style-type: none">• 1 dimension, 2-4 measures• 2 dimensions, 2-3 measures	1 dimension, 2-4 measures 2 dimensions, 2-3 measures
Correlation	Show the relative correlation between 2 measures over up to 2 optional dimensions.	0-2 dimensions, 2 measures

11 Customizing logical models for Insight Advisor

Analysis type	Use	Variations
Mutual information	<p>Display the statistical dependency between the target and the selected items.</p> <p>The dependency indicator ranges between 0% (no dependency) and 100% (strong dependency).</p> <p>You can select one field (measure or dimension) as the target and then select 1-10 dimensions or measures as drivers.</p> <p>Mutual information uses a randomly selected sample of data. Results for this analysis type for the same fields or selections may vary as a result.</p>	2-11 fields
Overview	Show the overview of a measure with multiple dimensions.	1-2 dimensions, 1 measure
Period over period	Compare a measure over the current period versus the previous period.	1 measure
Process control (mean)	Show an indication of a measure's performance between two calculated control limits.	1 measure, 1 temporal dimension
Process control (rolling mean)	Show an indication of a measure's performance between two calculated control limits over the last seven periods.	1 measure, 1 temporal dimension
Ranking	Rank dimension values by a measure, with optional grouping.	1 dimension, 1 measure
Ranking (grouped)	Show the nested ranking of one or more dimensions against a measure over time.	3 dimensions, 1 measure
Relative importance	Create a rank with cumulative contribution (Pareto chart).	1 dimension, 1 measure
Trend over time	<p>Show the performance of a measure over time, optionally broken down by a dimension with low cardinality.</p> <p>Two variations are supported:</p> <ul style="list-style-type: none"> • 0-1 dimensions, 1 measure, 1 date/time dimension • 2-3 measures, 1 date/time dimension 	0-1 dimension, 1 measure, 1 temporal dimension 2-3 measures, 1 temporal dimension
Year to date	Break down a measure (with results from this year and last year) by a dimension.	1 dimension, 1 measure, 1 temporal dimension

Creating custom analyses

Do the following:

1. In an app, click **Prepare** and select **Vocabulary**.
2. Click **Custom analysis**.
3. Click **Create terms**.
4. Select a language from the language drop-down and enter the terms for your custom analysis.
5. Optionally, add another language and add the terms for that language.
6. From **Analysis type**, select your analysis type.
7. If available, select the variation to use.
8. If you want to specify the dimensions and measures for the analysis. in **Data**, select the default fields to use.
9. If you want to apply constraints to the analysis, in **Constraints**, select a field, condition, and value.
10. In **Chart**, under **Chart type**, optionally select the primary chart to use with your custom analysis. To use this chart type even when other charts might be more appropriate, select **Enforce chart choice**.
11. If available for your analysis type, set the sorting and limit for the chart.
12. Click **Create**.
13. If your app is available for Insight Advisor Chat, click the app name in the navigation bar and then under **Use Insight Advisor**, turn **In hub** off then on.

Limitations

Custom analyses have the following limitations:

- The setting **Enforce chart choice** is not followed in the following scenarios:
 - When an unsupported number of fields are added to the custom analysis than the selected chart type supports.
For example, some charts support two dimensions, but not three dimensions. If you create a breakdown analysis with two dimensions and enforce the mekko chart, that functions as expected. If you create a breakdown analysis with three dimensions and select the mekko chart, the mekko chart will not be used as it does not support three dimensions.
 - When a user adds a constraint on a dimension that causes the limit of dimensions to be exceeded.
For example, you create a breakdown analysis with the dimensions category and product and select and enforce a mekko chart. This works as expected. If you add a constraint on the dimension country, there are now three dimensions being used in the chart, so the mekko chart will not be enforced.
 - When a variance waterfall chart is selected, but there are no calendar periods applied.
- Constraints are not available with all analysis types.
- In Insight Advisor Chat, follow-up questions do not work with custom analyses. For example, if you ask *sales summary*, a custom analysis, and then ask a follow up question, such as *what about profit*, Insight Advisor Chat will treat it as a new question.

11 Customizing logical models for Insight Advisor

- Fields tagged \$hidden in the load script will not be available in custom analysis, even if they are set to visible in the logical model.

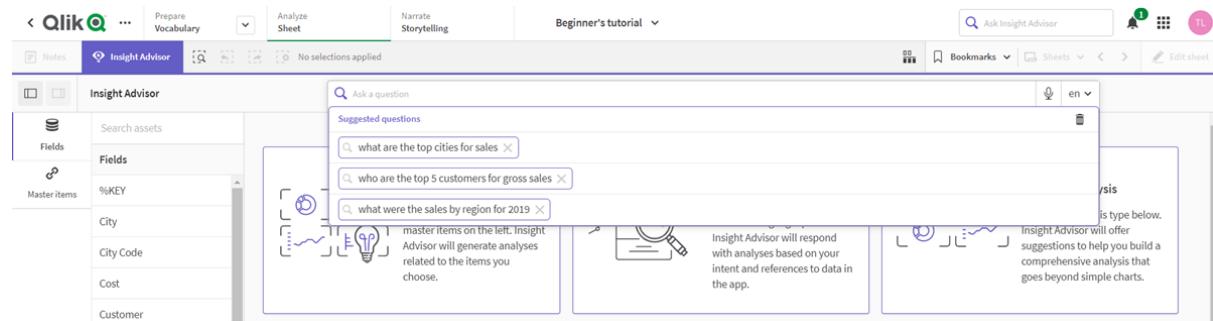
Adding example questions to Insight Advisor

You can add example question to Insight Advisor. These help give your app users example question to help guide or encourage their analysis.

With example questions, you can add a curated list of questions that are available in Insight Advisor Search and Insight Advisor Chat. These questions can help show your app users what is possible in this app and help prompt them on how to use your app.

In Insight Advisor Search, the questions are available when the user selects the search box. Selecting a question asks Insight Advisor that question. In Insight Advisor Chat, users can click **Questions** to see a list of questions from apps to which they have access. Selecting the question in Insight Advisor Chat enters the question in the user's chat box.

Example questions in Insight Advisor Search



When you add an example question, you pick a locale and enter the question. You can add entries for the same question in other languages. Questions you add are only available in the language you added them in. Once added, the question is available in Insight Advisor. New example questions become available in Insight Advisor Chat after the app reloads.

Do the following:

- In an app, click **Prepare** and select **Vocabulary**.
- Click **Example questions**.
- Click **Add question**.
- Under **Language**, select the language for the question.
- Under **Question**, enter the question.
- Optionally, click **Add language** to add the question in another language.
- Click **Add**.

11.3 Tutorial – Customizing how Insight Advisor interprets data

Welcome to this tutorial, which will introduce you to customizing how Insight Advisor interprets data in Qlik Sense.

After you load your data into an app, you can customize how Insight Advisor interprets your data. Insight Advisor can learn how to interpret relationships in your data model through precedent-based learning. If you are familiar with your data model and know how you want fields used, you can instead define a logical model in business logic for Insight Advisor. This enables you to ensure Insight Advisor offers the most relevant results to the users of your app.

What you will learn

After completing this tutorial, you should be comfortable using business logic to define your app's logical model. You will understand how business logic impacts Insight Advisor results.

Who should complete this tutorial

You should be familiar with Qlik Sense basics. For example, you have loaded data, created apps, and created visualizations in sheets.

You will need access to the Data manager and should be allowed to load data in Qlik Sense Enterprise on Windows.

What you need to do before you start

Download this app and upload the QVF file to Qlik Sense:

[Insight Advisor and business logic tutorial app](#)

This app contains the data file *TutorialData.xlsx*. Upload the app Qlik Sense. Open the app and navigate to Data load editor. Click **Load data** to load the app data.



To demonstrate relative comparisons possible through calendar periods, TutorialData.xlsx contains data for future dates. The app load script updates the in app data for the current date when loaded. Screenshots in this tutorial will differ depending on the date you load data.

Lessons in this tutorial

The topics in this tutorial are designed to be completed in sequence. However, you can step away and return at any time. The screenshots in this tutorial are taken in Qlik Sense SaaS. This tutorial is focused on Qlik Sense SaaS, so some differences might occur if you are using Qlik Sense in a different environment.

- *What is Insight Advisor and business logic? (page 301)*
- *Enabling a custom logical model (page 304)*

- *Customizing fields and groups (page 305)*
- *Configuring your packages (page 311)*
- *Reviewing your hierarchies (page 312)*
- *Configuring your calendar periods (page 314)*
- *Configuring your behaviors (page 319)*
- *Creating vocabularies (page 321)*

Further reading and resources

- [Qlik](#) offers a wide variety of resources when you want to learn more.
- [Qlik online help](#) is available.
- Training, including free online courses, is available in the [Qlik Continuous Classroom](#).
- Discussion forums, blogs, and more can be found in [Qlik Community](#).

What is Insight Advisor and business logic?

Insight Advisor is a suite of Qlik Sense features. Insight Advisor helps you build your data model, create visualizations, and analyze data. Business logic helps Insight Advisor interpret your data.

Insight Advisor

Insight Advisor comprises the following Qlik Sense features:

- **Insight Advisor Search:** Insight Advisor Search is available from **Sheet** in the **Analyze** tab of an app. Insight Advisor Search creates visualizations based on natural language searches or selections of fields and master items. Insight Advisor Search can also generate charts of potential interest.
- **Insight Advisor Analysis Types:** Insight Advisor Analysis Types is available from **Sheet** in the **Analyze** tab of an app. Insight Advisor Analysis Types allows you to select an analysis type and the data to include. Insight Advisor then generates charts based on your parameters.
- **Insight Advisor Chat:** Insight Advisor Chat is a chat-based interface for conversational analytics. Insight Advisor Chat enables you to make natural language searches from the hub to apps to which you have access. Insight Advisor Chat then returns relevant visualizations.
- **Associative insights:** Associative insights helps you uncover blind spots and reveal relationships you may have missed. Associative insights compares the contributions of your selections and excluded values against your measures.
- **Chart suggestions:** Chart suggestions enable you to select data fields when editing a sheet and let Qlik Sense choose the dimensions, measures, and visualization types. The suggested chart adjusts itself based on your changes. You can customize a suggested visualization with a focused set of properties.
- **Recommended associations:** Insight Advisor can recommend associations between your data tables in the **Associations** view in **Data manager**. The **Recommended associations** panel lets you view and apply these recommendations.

This tutorial will focus on improving search-based analysis in Insight Advisor Search with business logic.

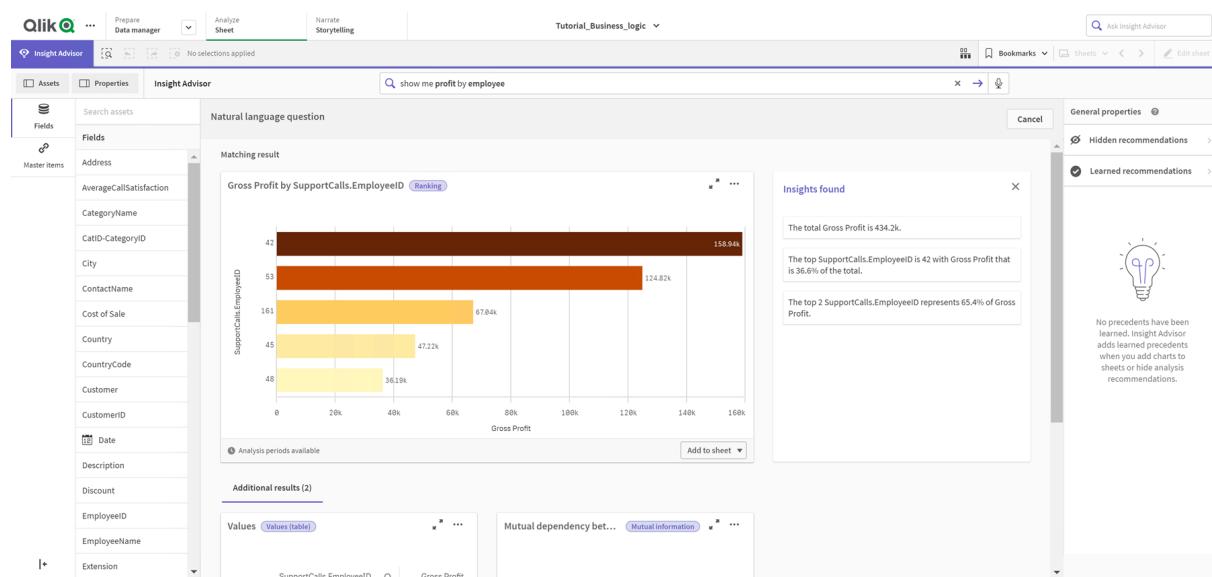
11 Customizing logical models for Insight Advisor

Insight Advisor Search

Insight Advisor Search is available from **Sheet** in the **Analyze** tab in an app. You can enter natural language searches in the search box, such as '*what is profit by employee for 2020*'. If you click **Insight Advisor**, you can also select specific data assets. Insight Advisor then generates charts or finds matching charts in the app. If you are creating apps, Insight Advisor Search helps you making apps based on the questions you want answered. If you are using an app, Insight Advisor helps you find the right visualization to answer your question, either in the app or by creating it for you.

For example, you are interested in viewing some visualizations that show profit earned by each employee. Open the tutorial app in Qlik Sense. In **Sheet**, search for *show profit by employee*. Insight Advisor generates results.

Results for *show profit by employee*



You have one matching result, that shows the sum of *Gross Profit* by *EmployeeID*, and some additional results. You can add these charts to sheets in our app.

While these results matched what we were looking for, the fields selected could be improved. In trying to pick a field to match *employee* from the data model, Insight Advisor selected *EmployeeID*. The field *EmployeeName* is more useful in analysis, however. The sum of *Gross Profit* in the chart is also for all time. While this is not wrong, you may want to see more recent data by default.

In **Analysis properties**, you can teach Insight Advisor that you prefer *EmployeeName* by selecting that dimension. These precedents are learned on a user-by-user basis. An app developer, however, can create a model that shows Insight Advisor exactly how to interpret the data model and use the fields within it. This logical model is used by Insight Advisor for all users using the app. You can do this using business logic.

Business logic

Business logic is an optional part of data preparation that defines the logical model used by some Insight Advisor features to interpret your data model. Business logic comprises two main features:

- **Logical model:** The data model of the app used when generating visualizations.
- **Vocabulary:** Alternative terms for natural language questions.

Insight Advisor uses business logic in the following Insight Advisor features:

- Insight Advisor Analysis Types
- Insight Advisor Search
- Insight Advisor Chat
- Associative insights

Logical model

The logical model is the conceptual model Insight Advisor uses when creating visualizations. It is built from the data model of an app. Each app has a single logical model. Fields and master items are the core components of this model. They are organized into groups. Groups indicate a conceptual association or relationship between fields or master items. Once you define these, you can then specify other relationships and handling behavior. These include:

- **Packages:** **Packages** enables you to create collections of related groups. This prevents groups from being used together that are not in the same package.
- **Hierarchies:** **Hierarchies** enables you to define drill-down analytical relationships between groups.
- **Calendar periods:** **Calendar periods** enables you to create default periods of analysis for Insight Advisor.
- **Behaviors:** **Behaviors** enables you to specify prefer or deny relationships between fields. Behaviors can also enforce required selections and default calendar periods.

Vocabulary

Business logic vocabulary enables you to improve the success of natural language questions. With vocabulary, you can define terms and values that may be used in natural language questions that are not present in your data model. You can also set the analysis types to be used for questions containing specific terms. For example, you can use vocabulary to:

- Add alternative names for fields, master items, and values.
For example, *Earnings*, *Proceeds*, and *Revenue* for the field *Income*.
- Define names for coded values.
For example, names for medical classifications codes.
- Common acronyms and abbreviations not in the data.
For example, *yr* for *year* or *num* for *number*.
- Define the Insight Advisor analysis type to be used with certain terms or questions.
For example, setting it so the map distribution analysis type is always used by Insight Advisor when a question includes the term *regional view*.
- Add example questions to help guide app users with their analysis.
For example, provide questions such as *what are the sales for this quarter compared to last quarter*.

Enabling a custom logical model

The first step to customizing how Insight Advisor interprets your data is enabling the business logic logical model.

Once enabled, Qlik Sense creates a logical model for business logic based on your data model. You can then start customizing this logical model.



*You can toggle the logical model on and off by clicking **Disable logic/Enable logic**. This is useful when you want to compare results with the model applied to results without the model applied.*

Do the following:

1. In the tutorial app, click **Prepare**.
2. Under **Business logic**, select **Logical model**.
3. Click **Continue**.

Results

Business logic analyzes your data model and automatically creates groups, hierarchies, and behaviors.

Overview provides a look at your new logical model.

*The new logical model in **Overview***

The screenshot shows the Qlik Sense Overview page for the 'Tutorial_Business_logic' app. The top navigation bar includes 'Prepare Logical model' (which is selected), 'Analyze Sheet', and 'Narrate Storytelling'. The main content area is titled 'Tutorial_Business_logic'. It features several sections: 'Fields and master items' (29 Visible, 14 Hidden, 0 Excluded), 'Groups and their types' (9 Groups in business logic, 1 calendar type, 1 measure type, 7 dimension type), 'Packages' (No packages created as app is currently using a generated business logic), 'Hierarchies' (2 Hierarchies defined, 3 groups added to hierarchies), 'Behaviors' (No behaviors set as app is currently using a generated business logic), and 'Calendar periods' (3 Calendar periods defined, 3 using autoCalendar, 0 using aggregated date). A note at the bottom left states: 'An overview of the metadata model used by Insights. Much of Insights functionality is grounded in a conceptual model based on an app's underlying data model. Fields and master library items are the building blocks of this conceptual model. These are organized into different types of groups. The model also contains information about possible relationships between these groups.' There are also 'Reset to default' and 'Disable logic' buttons at the top right.

Qlik Sense has hidden 14 fields from analysis, so they are no longer available in Insight Advisor. 9 groups have been created from the data model, associating related fields together. 2 hierarchies have been defined, linking groups for drill-down analysis. 3 calendar periods have been defined, providing periods of time for use in analyses.

Next, you will edit this default logical model, starting by modifying the default fields and groups.

Customizing fields and groups

The next step to customizing your logical model is to define your fields and groups. Fields and groups are the basic level of the logical model. All other logical model elements use these groups.

When you enable a custom logical model, Qlik Sense automatically creates default groups from your data model. It also sets the properties of all your fields. Some of these groups and field properties need to be adjusted. Qlik Sense will make a best interpretation of your data model, but it is not always correct in which fields should be hidden or which fields are measure, for example.

Fields & groups, with default groups

Name	Visibility	Classification	Data value lookup	Default aggregation
Offices (Fields: 2, Master items: 0, Group type: Dimension)				
Office	hidden	dimension	No	Not specified
SalesOffice	visible	city	Yes	Not specified
Categories (Fields: 3, Master items: 0, Group type: Dimension)				
CategoryID	hidden	dimension	No	Not specified
CategoryName	visible	dimension	Yes	Not specified
Description	visible	dimension	Yes	Not specified
Suppliers (Fields: 4, Master items: 0, Group type: Dimension)				
SupplierID	hidden	dimension	No	Not specified
Supplier	visible	dimension	Yes	Not specified
SupplierContact	visible	dimension	Yes	Not specified
SupplierCountry	visible	country	Yes	Not specified
Products (Fields: 2, Master items: 0, Group type: Dimension)				
ProductID	hidden	dimension	No	Not specified
ProductName	visible	dimension	Yes	Not specified
SupportCalls (Fields: 4, Master items: 0, Group type: Dimension)				
SupportCallsEmployeeID	visible	dimension	Yes	Not specified
SupportCalls	visible	measure	Yes	sum

There are three kinds of groups:

- **Dimension:** A dimension group consists of related fields that are classified as dimensions. Dimension groups can also contain fields classified as measures or dates.
- **Measure:** A measure group consists of related measure fields. Only measures can belong to a measure group.
- **Calendar:** A calendar group contains a time dimension in your logical model. Calendar groups can only contain dimensions and must have at least one temporal fields (such as date, timestamp, or year).

Each group has field with the following properties:

- **Visibility:** Defines if a field is visible or not in Insight Advisor. Hidden fields can still be used in analysis, such as when a user searches for that field.
- **Classification:** Defines the default role the field plays in analysis. Classifications can be broad, such as **dimension**, or specific, such as **city**.

- **Data value lookup:** Controls whether or not users can search for specific values from this field. Reducing the number of fields that have data value lookup enabled can help you avoid false positive results and reduce the time to generate results.
- **Default aggregation:** Sets the standard aggregation for measures in Insight Advisor. When a field has a default aggregation, Insight Advisor always applies that aggregation when using it as a measure. Users can edit Insight Advisor analyses to change the aggregation to a different type in Insight Advisor.
- **Favorable trend:** Defines whether the favorable trend is for the measure is to go up or down.
- **Overall aggregation:** Defines which aggregation should be used when Insight Advisor cannot determine for itself what aggregation to use when generating results involving master measures with complex expressions. You can only set overall aggregation for master measures.
- **Favorite:** Defines a measure that should be used more often in analysis when Insight Advisor generates analyses without a question or field selections, such as with **Generate**.
- **Default period grain:** Sets a default grain to use with a date field in analyses.

You can set which columns display in the table by clicking .

Fields can be ungrouped. Ungrouped fields belong to no group and are excluded from analysis.

Why define fields and groups?

Defining fields and groups enables you to set the defaults of how fields and master items are handled. When you define fields, you set the preferences on how they are handled by Insight Advisor. For example:

- Should this field be available for analysis in Insight Advisor?
- Should this field be used as a dimension or measure? What kind of dimension or measure?
- Can Insight Advisor look up individual values from the field in searches?
- What should the default aggregation for the field be when used as a measure?

When you define groups, you show Insight Advisor which fields are closely related and should be used together in analysis. Groups are used to create other business logic features, such as packages, calendar periods, and behaviors.

Customizing your fields and groups

You are going to start by cleaning up some of the field properties to change visibility and adjust a few classifications. You are then going to ungroup some of the fields from the groups. With these fields, you will make the following new groups:

- *Customer*
- *SalesCity*
- *SalesCountry*
- *SupportCalls*
- *SupportDate*

Some fields you will leave ungrouped. Ungrouped fields are not used by Insight Advisor in any analysis. This will remove fields that are not relevant to analysis, such as ID and GeoInfo fields.

Customizing field properties

Do the following:

1. Navigate to **Fields & groups**.
2. In *Employees*, adjust the following fields:
 - For *Extension*, set the following properties:
 - **Visibility: visible**
 - **Classification: dimension**
 - **Data value lookup: No**
 - **Default aggregation: Not specified**
 - For *EmployeeName*, set the following properties:
 - **Visibility: visible**
 - **Classification: dimension**
 - **Data value lookup: Yes**
 - **Default aggregation: Not specified**
 - For *Hire Date*, set the following properties:
 - **Visibility: visible**
 - For *Reports To*, set the following properties:
 - **Visibility: hidden**
 - **Classification: dimension**
 - **Data value lookup: Yes**
 - **Default aggregation: Not specified**
 - For *Title*, set the following properties:
 - **Visibility: visible**
 - For *Year Salary*, set the following properties:
 - **Visibility: visible**
 - **Classification: monetary**
 - **Data value lookup: No**
 - **Default aggregation: Sum**

Ungrouping fields

Do the following:

1. In *Categories*, select *CatID-CategoryID*.
2. In *Suppliers*, select *SupplierID*.
3. In *SupportCalls*, select the following:
 - *SupportCalls*
 - *AverageCallSatisfaction*
 - *Date*

4. In *Products*, select *ProductID*.
5. In *Employees*, select *EmployeeID*.
6. In *Sales_ENT*, select the following fields:
 - *Address*
 - *City*
 - *ContactName*
 - *Country*
 - *CountryCode*
 - *Customer*
 - *CustomerID*
 - *Latitude*
 - *Longitude*
 - *Longitude_Latitude*
 - *OrderID*
 - *Phone*
 - *PostalCode*
 - *ShipperID*
7. Click **Ungroup**.

Creating *Customer* group

This group will let you group together dimension fields relating to customer information.

Do the following:

1. Click **Create group**.
2. For **Group name**, enter *Customer*.
3. For **Group type**, select **Dimension**.
4. Add the following fields to the group:
 - *Address*
 - *ContactName*
 - *Customer*
 - *Phone*
 - *PostalCode*
5. Click **Create**.

Creating *SalesCity* group

This group will let you group together dimension fields relating to city information. You will use this later when making a hierarchy.

Do the following:

1. Click **Create group**.
2. For **Group name**, enter *SalesCity*.
3. For **Group type**, select **Dimension**.
4. Add the following fields to the group:
 - *City*
 - *Longitude_Latitude*
5. Click **Create**.

Creating *SalesCountry* group

This group will let you group together dimension fields relating to country information. You will use this later when making a hierarchy.

Do the following:

1. Click **Create group**.
2. For **Group name**, enter *SalesCountry*.
3. For **Group type**, select **Dimension**.
4. Add the followings fields to the group:
 - *Country*
 - *CountryCode*
5. Click **Create**.

Creating *SupportCalls* group

You removed all fields from the previous *SupportCalls* group as it was classified as a dimension group. You will use the fields to make a measure group.

Do the following:

1. Click **Create group**.
2. For **Group name**, enter *SupportCalls*.
3. For **Group type**, select **Measure**.
4. Add the following fields to the group:
 - *AverageCallSatisfaction*
 - *SupportCalls*
5. Click **Create**.

Creating *SupportDate* group

Finally, you will make a calendar group for the *Date* field from the support calls data.

11 Customizing logical models for Insight Advisor

Do the following:

1. Click **Create group**.
2. For **Group name**, enter *SupportDate*.
3. For **Group type**, select **Calendar**.
4. Add the following fields to the group:
 - *Date*
5. Click **Create**.

Results

Go to **Sheet** in the **Analyze** tab and click **Insight Advisor**. Look at the **Assets** panel. There is a significantly reduced number of fields in **Assets**. The ID fields are no longer there, for example.

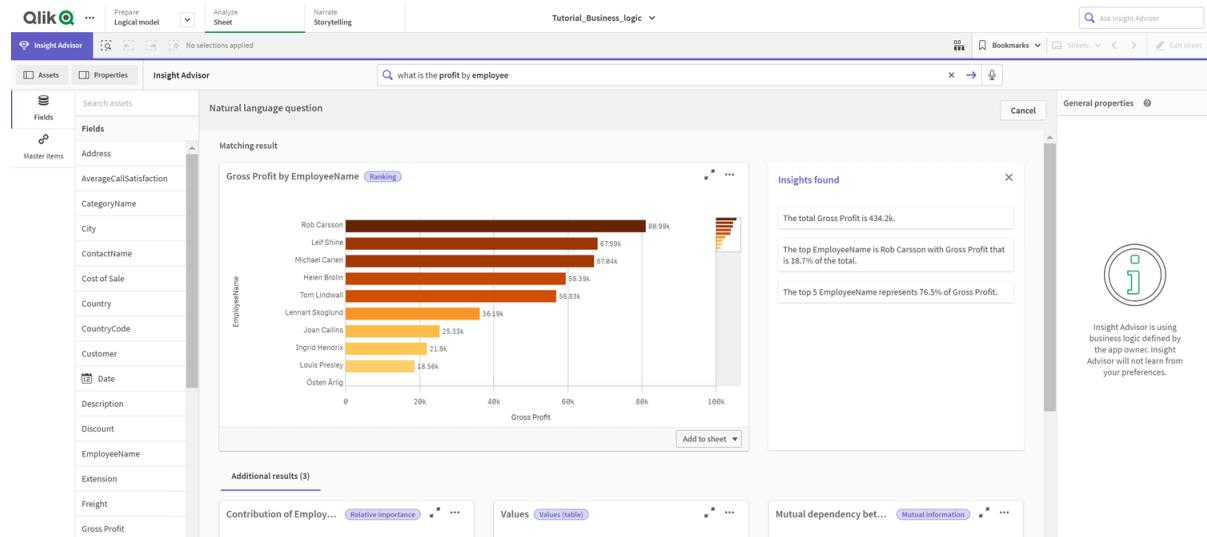
Fields in Insight Advisor assets panel

The screenshot shows the Qlik Sense interface with the 'Insight Advisor' tab selected. On the left, the 'Assets' panel lists various fields: Address, AverageCallSatisfaction, CategoryName, City, ContactName, Cost of Sale, Country, CountryCode, Customer, Date, Description, Discount, EmployeeName, Extension, Freight, Gross Profit, and Hire Date. The 'Fields' section is expanded, showing sub-fields for each category. In the center, there are three main sections: 'Explore your data' (with a description of generating analyses based on selected items), 'Ask a question' (with a search bar icon), and 'Create an analysis' (with a description of choosing an analysis type). Below these is a grid of eight analysis types: Calculated measure (KPI), Ranking, Breakdown (geospatial), Breakdown, Overview, Trend over time, Comparison, and Relative importance. Each type has a small icon and a brief description.

Now, search for *what is the profit by employee*. This time, instead of *EmployeeID*, Insight Advisor uses *EmployeeName*, a more useful field to use as a dimension in visualizations.

11 Customizing logical models for Insight Advisor

Results for what is the profit by employee



Configuring your packages

The next step is to start working with the other elements of the logical model, starting with packages.

Packages are collections of groups that should be used together in analysis. This ensures only relevant fields are used together by Insight Advisor. Groups can belong to multiple packages. Packages are an optional part of the logical model.

Why create packages?

Packages define the scope of which groups should be used together in analysis. For example, your logical model may have groups that cover different areas of an organization. Mixing these may not always be desirable for your Insight charts.

For example, the tutorial app has data related to the handling of support calls, which is not related to the sales and supplier data. If you navigate to **Sheet** and select *AverageCallSatisfaction* from the Insight Advisor Assets panel, you get results where the support call data is mixed with sales data fields.

By using packages, you can keep the sales and supplier data together for analysis and keep the support call data together for analysis, preventing Insight Advisor from trying to use the data together.

Creating packages

You are going to create packages for the sales-related groups and support-related groups. Both will include the relevant *Employees* and *Offices* groups.

Creating the *Support* package

Do the following:

1. Navigate to **Packages**.
2. Click **Create package**.
3. For **Package name**, enter *Support*.
4. Add the following groups to the package:
 - *SupportCalls*
 - *SupportDate*
 - *Employees*
 - *Offices*
5. Click **Create**.

Creating the *Sales* package

Do the following:

1. Navigate to the **Packages** section.
2. Click **Create package**.
3. For **Package name**, enter *Sales*.
4. Add the following groups to the package:
 - *Categories*
 - *Customer*
 - *SalesCity*
 - *SalesCountry*
 - *Employees*
 - *Offices*
 - *Sales_OrderDate*
 - *Products*
 - *Sales*
 - *Suppliers*
5. Click **Create**.

Results

Navigate to **Sheet** and click **Insight Advisor**. In the assets panel, select *AverageCallSatisfaction*. Now, none of the generated Insight Advisor analyses use fields from groups that are not part of the *Support* package.

Reviewing your hierarchies

Next, you will review the hierarchies in your logical model. **Hierarchies** is an optional business logic feature. It defines drill-down relationships between groups.

11 Customizing logical models for Insight Advisor

When you enable business logic, some hierarchies may be automatically created by Qlik Sense from your data model. If you navigate to **Hierarchies**, you can see that business logic has created two hierarchies.

Hierarchies in the logical model

The screenshot shows the Qlik Sense Logical model interface. At the top, there are tabs for Overview, Fields & groups, Packages, **Hierarchies**, Behaviors, and Calendar periods. The Hierarchies tab is selected. Below the tabs, there is a message: "Hierarchies create drill-down paths in analyses ⓘ Each hierarchy defines a drill-down relationship between groups. Hierarchies can contain multiple levels." Two hierarchies are listed: "Categories-Products" and "Suppliers-Products". Each hierarchy has a tree structure with nodes like "Categories" or "Suppliers" at the top level, followed by "Products". On the right side of the interface, there are buttons for "Reset to default" and "Disable logic", and a "Create hierarchy" button.

Hierarchies indicate groups that can be used to break each other down in analysis. For example, the two hierarchies created by Qlik Sense correctly identify two drill-down relationships in our logical model:

- The data in the *Category* fields can be broken down into the data in the *Products* fields.
- The data in the *Suppliers* fields can be broken down into the data in the *Products* fields.

If you navigate to **Sheet**, click **Insight Advisor**, and select *CategoryName*, Insight Advisor includes a tree map that breaks down *CategoryName* by *ProductName*.

Category and products breakdown

The screenshot shows the Qlik Sense Insight Advisor interface. The left sidebar lists various dimensions: Address, AverageCallSatisfac..., CategoryName (with a checked checkbox), City, ContactName, Cost of Sale, Country, CountryCode, Customer, Date, Description, Discount, EmployeeName, and Extension. In the center, there is an "Auto-analysis" section with a "Selected fields:" dropdown containing "CategoryName". Below it, there is a table titled "Top ProductName by Cost of Sale for Cate..." with columns for "CategoryName" and "Values". To the right, there is a treemap chart titled "Sales by CategoryName and ProductName". The treemap is divided into several categories: Womenswear (teal), Sportswear (light blue), Mens' footwear (yellow), Ladies' footwear (dark green), Beachwear (light green), Childrenswear (purple), Womenswear (pink), Babywear (dark pink), and Mens' footwear (dark grey). Each category contains smaller rectangles representing individual products. The overall interface includes a search bar, a question asking "Ask a question", and various analysis properties like "Chart type: Treemap" and "Analysis type: Breakdown".

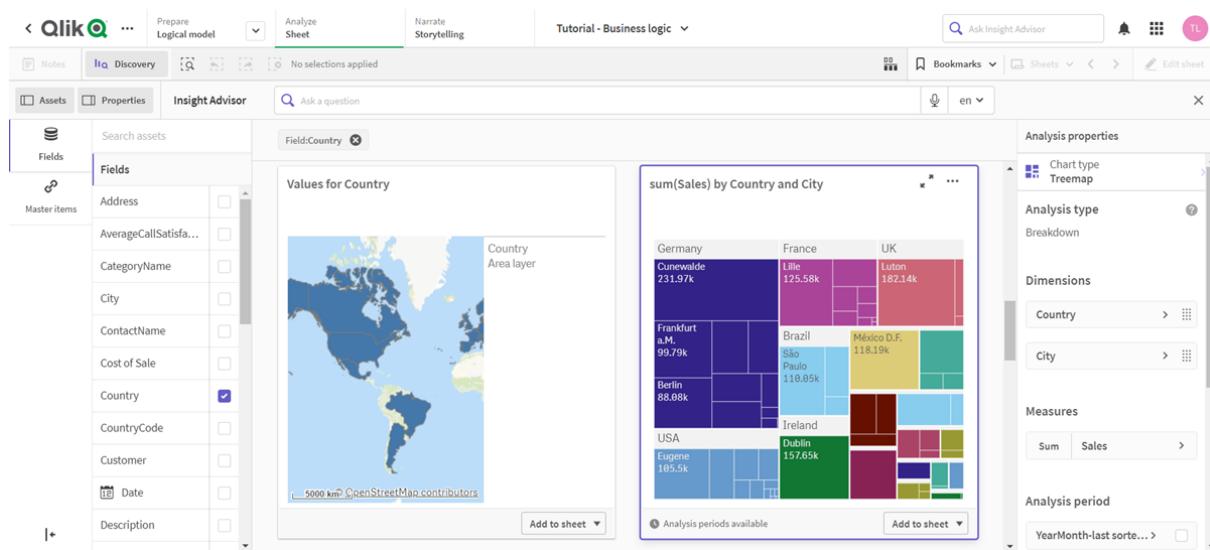


Hierarchies do not enable drill-down selections in the generated charts. That requires drill-down dimension master items. For example, if you want map charts that drill-down from cities after a country is selected, you need to create a corresponding drill-down dimension master item.

11 Customizing logical models for Insight Advisor

In addition to defined hierarchies, the logical model can contain learned hierarchies. These are learned automatically from how fields are used and defined in the data model. For example, navigate to **Sheet** and click **Insight Advisor**. From the assets panel, select *Country*. You now have results that reflect a *City-Country* hierarchy, including a tree map that shows $\text{sum}(\text{Sales})$ by *Country and City*. This hierarchy is a learned hierarchy detected from the data model.

New Insight Advisor results for *Country*



Configuring your calendar periods

Calendar periods use your calendar groups to create default periods of time for analysis in Insight Advisor.

Calendar periods are used to define time periods for analysis in Insight Advisor analysis. You create calendar periods from your calendar groups. Depending on your data, business logic may automatically create calendar periods from your data model. If you navigate to **Calendar periods**, you can see calendar periods have already been made. Calendar periods can be relative, covering a period relative to the current date. For example, you could create a relative period to cover the current month. Calendar periods can also use the last recorded value in your data. For example, the three calendar periods created when you activated business logic all use last recorded values to make calendar periods for the last year, month, and quarter. App users can apply the calendar periods to Insight Advisor analysis in **Sheets**.

You can also define default calendar periods for groups in behaviors. This ensures that by default, Insight Advisor uses that calendar period when creating any charts for fields in that group.

Why define calendar periods?

Calendar periods are useful because they create specific periods of analysis for your data. If Insight Advisor is not given a specified time frame, such as *in 2020* or *for December*, it will use the entirety of the data available. If you provide calendar periods, app users can more easily view and compare data for specific periods.

For example, you want to view sales by product. Navigate to **Sheet**, and search for *show me sales by product*. If you want to view specific periods of sales, such as a data for this month compared to last month or a comparison of the current quarter to a previous quarter, you would need to enter a new query. By making calendar periods, you can view these periods for your query without making new queries. Business logic has already made last recorded value calendar periods for the last month, quarter, and year in the data. It does not have relative periods, however. You could create a comparison for the current month to the same month last year. You could also make a comparison between the current quarter (x) and the quarter two quarters ago (x-2).

Creating the *Month comparison* calendar period

This calendar period will display results for the most recent month in the data and compare it to the same month last year.

Do the following:

1. Click **Create calendar period**.
2. Select *Sales_Order Date*.
3. For **Calendar period name**, enter *Month comparison*.
4. For **Calendar period grain**, select **Month of year**.
5. Under **Period comparison**, select **Year over year**.
6. Select **Last complete period**.
7. Click **Create**.

Creating the *Quarter comparison* calendar period

This calendar period will display results for the current quarter (x) and compares it to the quarter two quarters ago (x-2).

Do the following:

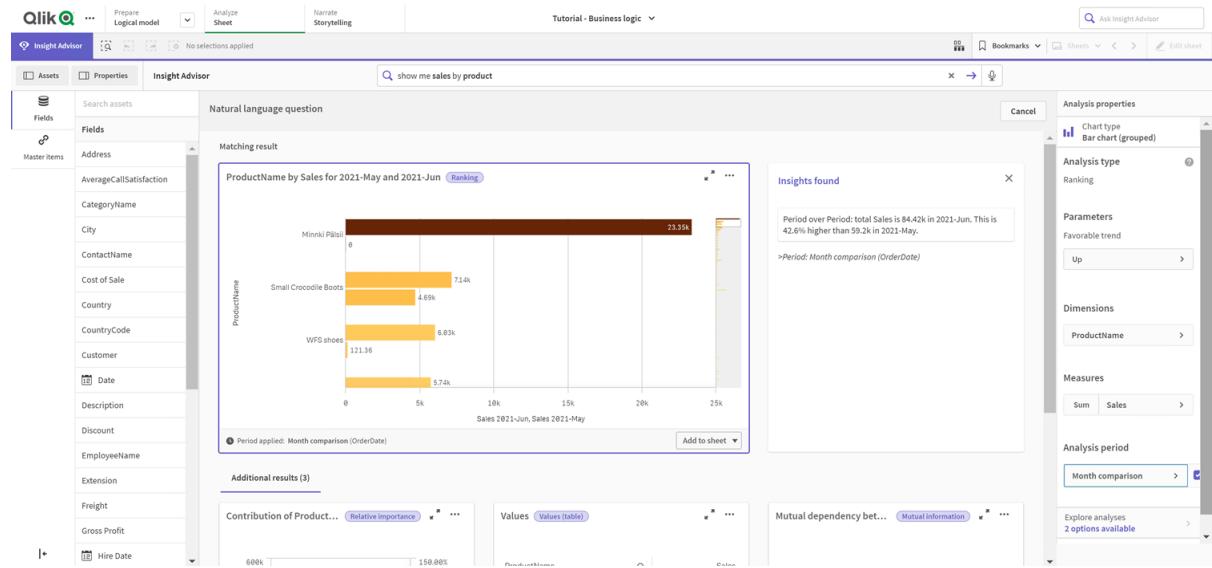
1. Click **Create calendar period**.
2. Select *Sales_Order Date*.
3. For **Calendar period name**, enter *Quarter comparison*.
4. For **Calendar period grain**, select **Quarter of year**.
5. Click **Custom**.
6. For **Offset**, select 0.
7. For **Compare offset**, select 2.
8. Click **Create**.

Results

Navigate to **Sheet** and search for *show me sales by product*. Select the matching result. In **Analysis properties**, there is a new property available, **Analysis period**. Select it and select *Month comparison* to apply the period to the chart.

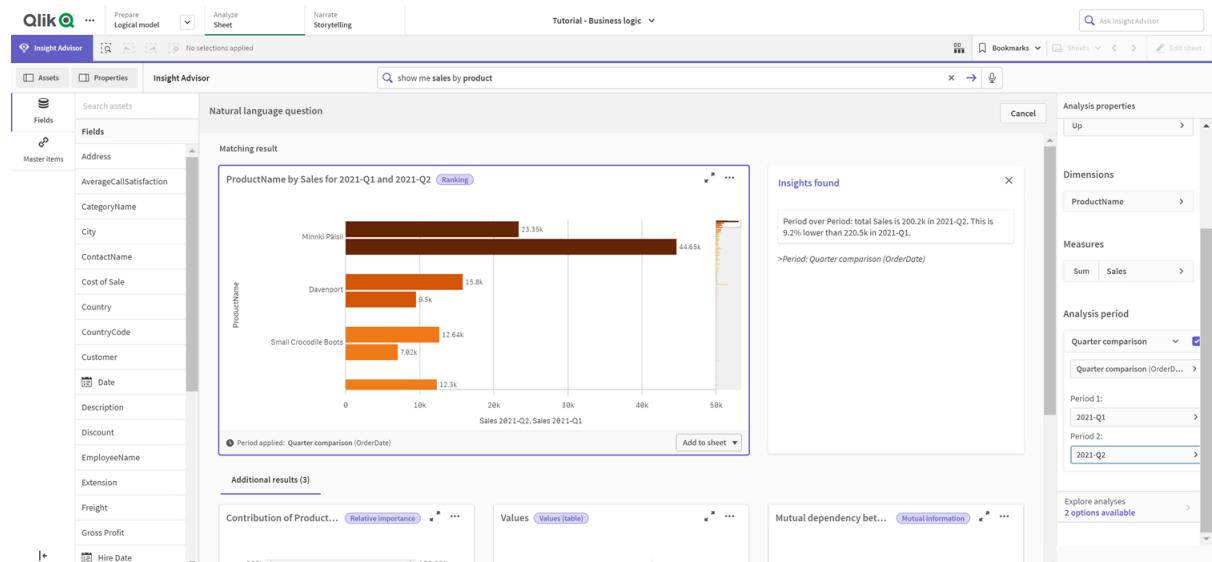
11 Customizing logical models for Insight Advisor

Month comparison calendar period applied to a chart



Next, apply the *Quarter comparison* calendar period.

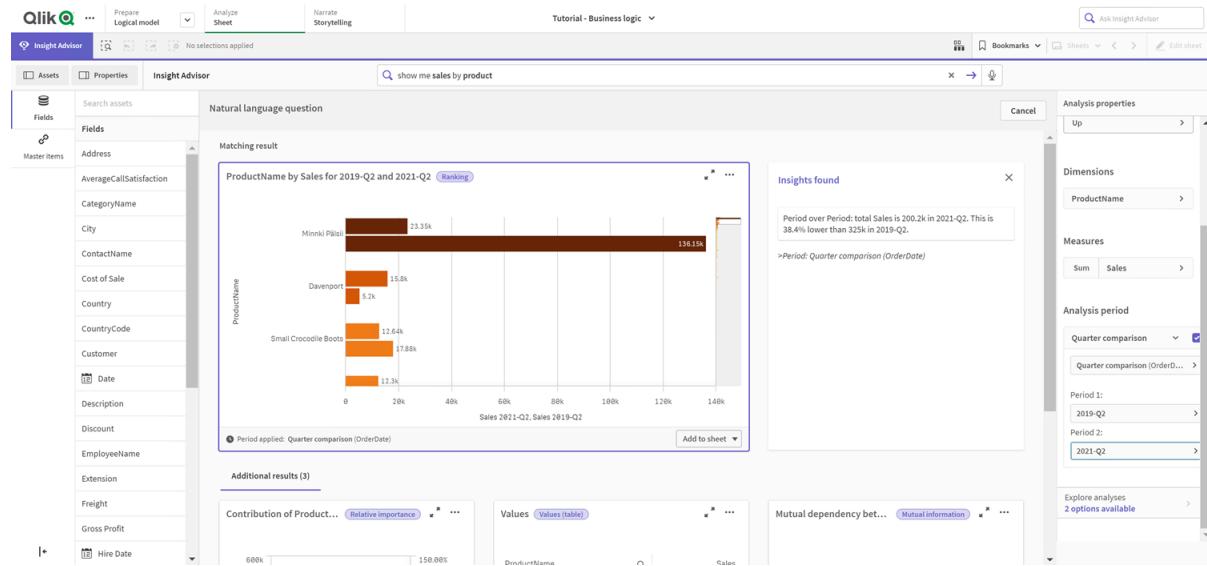
Quarter comparison calendar period applied to a chart



You can also adjust the time periods in an analysis period manually for individual charts in Insight Advisor by changing the values in **Period 1** and **Period 2**. For example, under **Period 1**, change the value to 2019-Q2. The chart updates for the newly defined period.

11 Customizing logical models for Insight Advisor

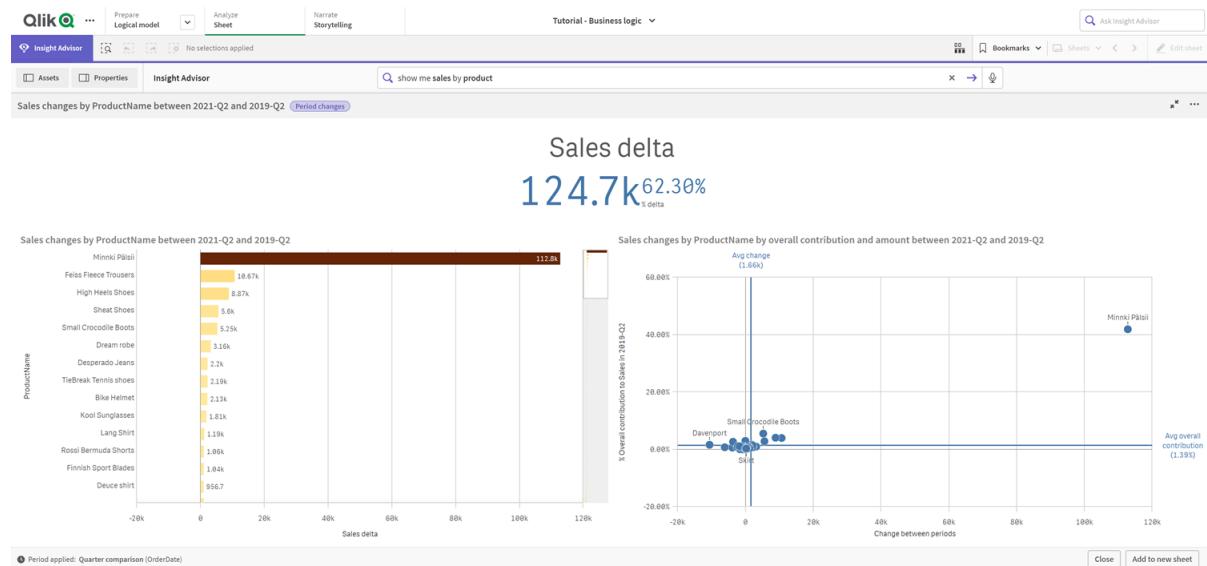
Quarter comparison calendar period applied to a chart, now using 2019-Q2 and 2021-Q2



Additional analysis types are available in Insight Advisor when calendar periods are available: period changes, period changes (detailed), and period over period. These provide a suite of charts for analyzing the period and its changes.

Click **Explore analyses** under **Analysis period**. There are two analyses to select from. First, choose **Period changes**. Period changes shows a KPI and bar chart for the changes of the sum(Sales) delta between the two periods. When there is a small number of dimensions, a variance waterfall chart is displayed instead of a bar chart.

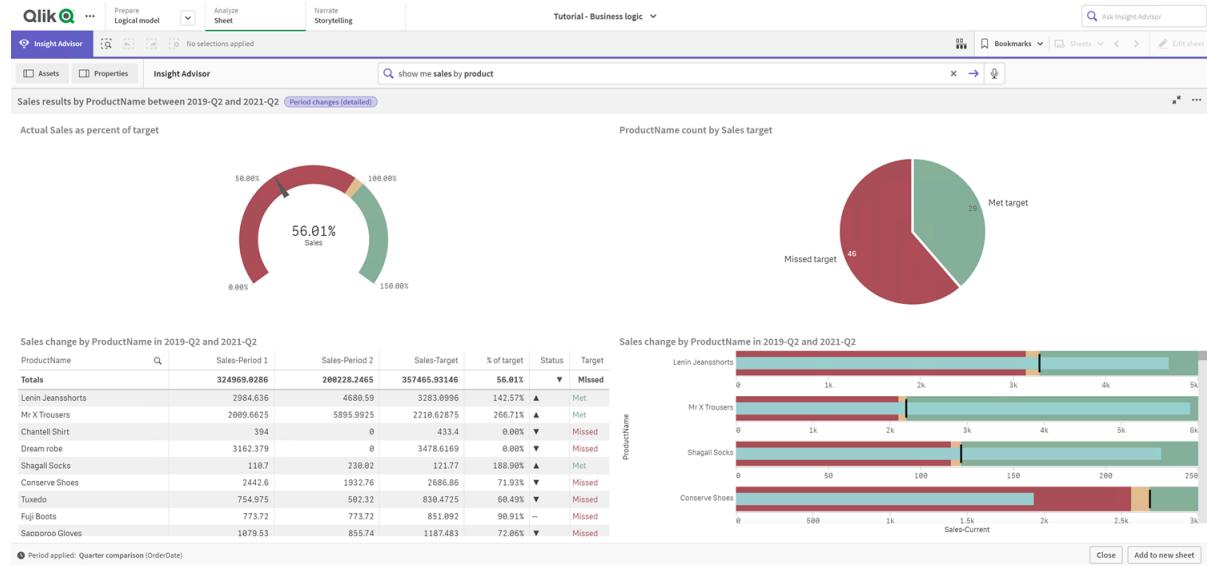
Period changes analysis



11 Customizing logical models for Insight Advisor

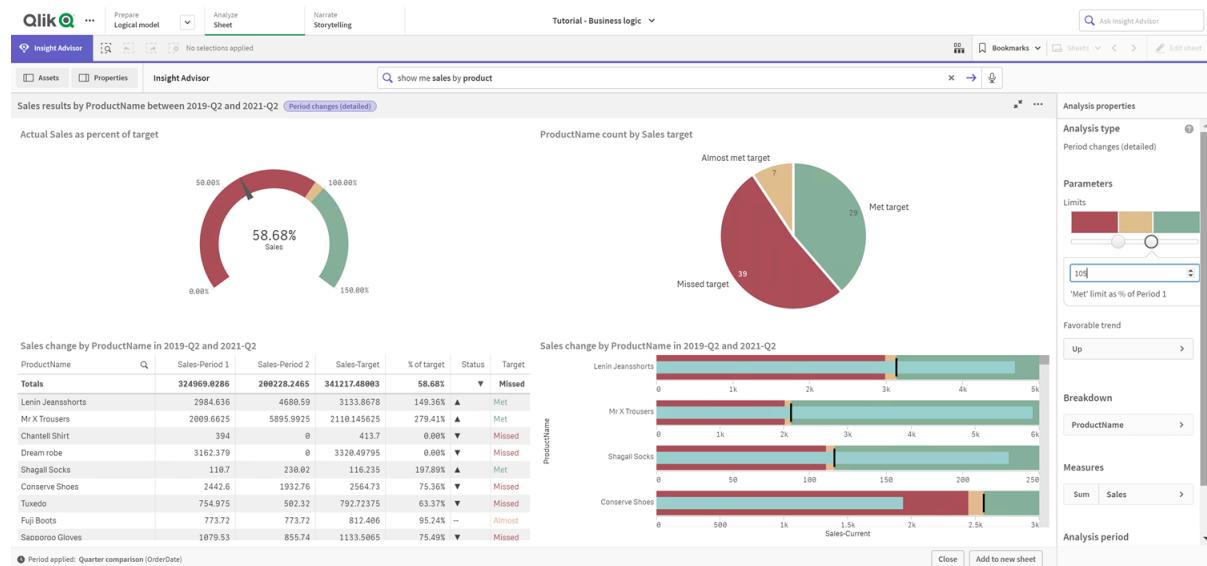
Click **Close** and scroll back to the bar chart. Click **Explore analyses** and now select **Period analysis (detailed)**. This provides the period performance against a target. It displays four charts that measure if targets were met for this period. You can adjust the parameters to define what met, what almost met, and what did not meet the target in **Analysis properties**.

Period performance against target analysis



Under **Parameters**, adjust the limit for missing the target to 100 and the limit for meeting the target to 105. The detailed period analysis updates for the new targets.

Updated period performance against target analysis



As you can see, calendar periods enable app users to quickly view different periods of interest in Insight Advisor.

Configuring your behaviors

Now you will create a behavior. Behaviors enable you to set custom individual scope for how measure groups are used.

With behaviors, you can:

- Enforce required selections of values from a field with a measure group.
- Set which groups you prefer to be used with a measure group.
- Set which groups you do not want used with a measure group.
- Set a calendar period to be used by default with a measure group.

Behaviors are optional.

Why configure behaviors?

Like packages, behaviors help you set the scope for how groups are used together. Behaviors enable you to set narrow scope for how individual measure groups are used with other groups.

For example, when we search for fields from the **Sales** group in **Sheet** measure results for **Gross Profits** are aggregated across all time. It would be more useful if you had a more recent period of time applied. You can create a behavior to use one of your calendar periods by default.

Creating a default calendar period behavior

Do the following:

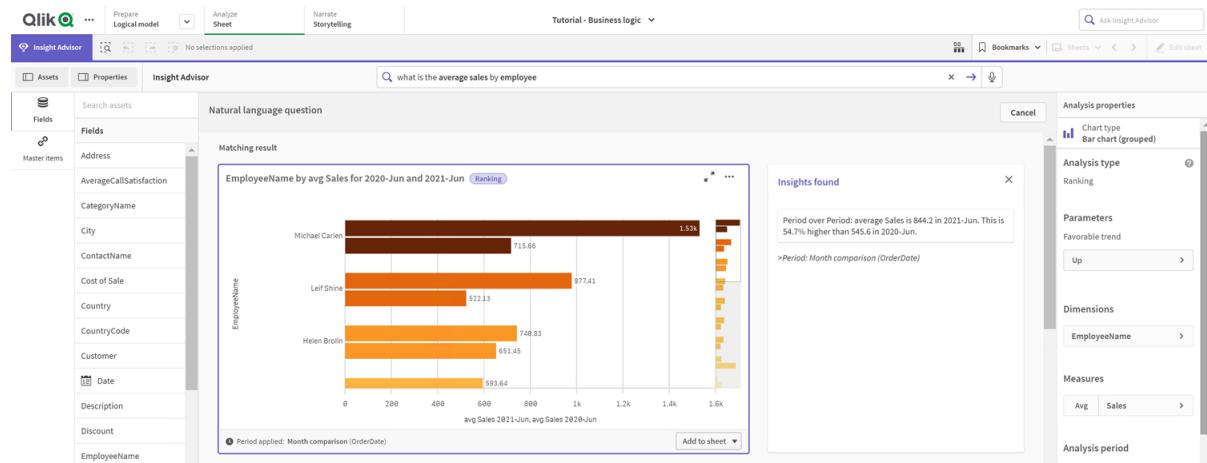
1. Click **Create behaviors**.
2. Under **Applies to**, select **Sales**.
3. Under **Behavior type**, select **Default calendar period**.
4. Under **Calendar group**, select **Sales_OrderDate**.
5. Under **Period**, select **Month comparison**.
6. Click **Create**.

Results

Navigate to **Sheet** and search for *what is the average sales by employee*. The matching results now displays a bar chart that compares this month's sales for each employee to the sales from the same month last year. For charts that do not support comparisons, only the results for the current month are shown.

11 Customizing logical models for Insight Advisor

Insight Advisor analysis using the default calendar period



Configure a behavior to use a default calendar period for Period over period analysis

You can also set a default calendar period behavior for a group that is applied to **Period over period** analysis.

Period over period analysis type uses a Line chart to compare a measure over the current period versus the previous period. It requires a default calendar period set for the group containing the measure in the logical model.

First, create a *Quarter over quarter* calendar period.

Do the following:

1. Click **Create calendar period**.
2. Select *Sales_Order Date*.
3. For **Calendar period name**, enter *Quarter over quarter*.
4. For **Calendar period grain**, select **Quarter of year**.
5. Select **Use last sorted value**.
6. Click **Create**.

Now in the **Behaviors** tab, set this calendar period as the default for the group *Sales*.

Do the following:

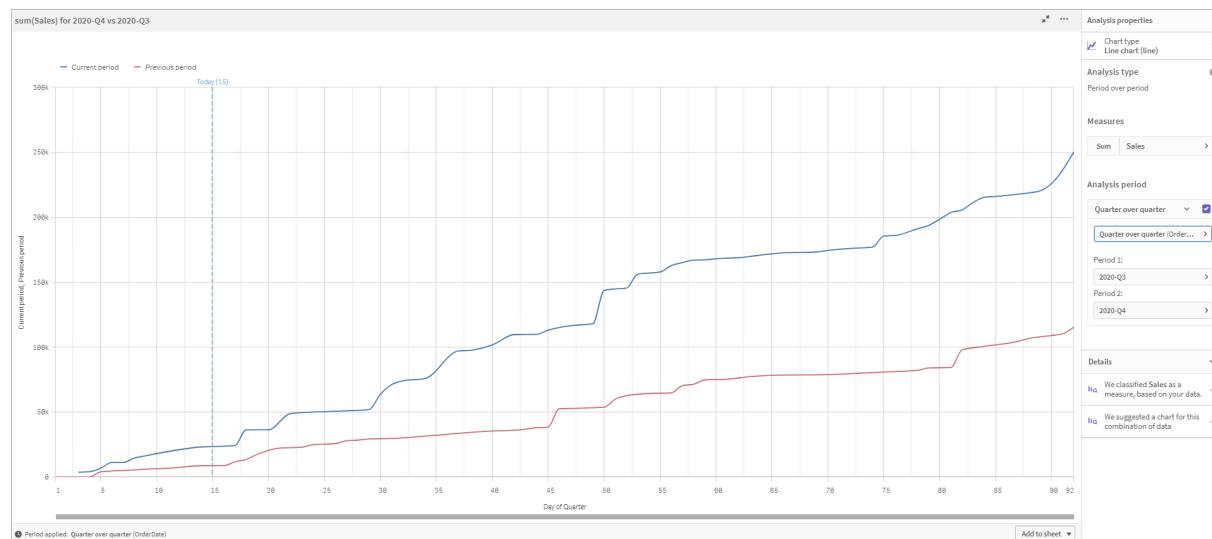
1. Click **Create behavior**.
2. Under **Applies to**, select *Sales*.
3. Under **Behavior type**, select **Default calendar period**.
4. Under **Calendar group**, select *Sales_OrderDate*.
5. Under **Period**, select *Quarter over quarter*.
6. Click **Create**.

Navigate to **Discovery** and search on *show me sales*. The first chart that appears below the KPI is a **Period over period** analysis. Open this chart (*sum (Sales)* for 2020-Q4 vs 2020-Q3) and navigate to the property **Analysis period**. Select it and note that the default calendar period *Quarter over quarter* was correctly

11 Customizing logical models for Insight Advisor

selected to compare the most recent period to the previous period. In this tutorial those periods are 2020-Q3 (Period 1) and 2020-Q4 (Period 2). These selections can be changed by selecting different quarters from the drop-downs. A reference line indicates the current day in the current calendar period grain, which in this case is a quarter.

*Period over period analysis applied to a **Line chart***



You can change this default calendar period in **Analysis period**.

Creating vocabularies

With vocabularies, you can define synonyms, custom analysis, and example questions. With synonyms, you can add terms to define values or selections of values from your data model. With custom analyses, you can define the response from custom questions or partial questions and indicate the specific results returned by Insight Advisor. Example questions let you add default questions users can select in Insight Advisor Search to help guide analysis.

With vocabularies, you can define synonyms and custom analysis. With synonyms, you can add terms to define values or selections of values from your data model. With custom analyses, you can define the response from custom questions or partial questions and indicate the specific results returned by Insight Advisor.

Why create vocabularies?

Vocabulary is a key tool for preparing your app for natural language questions. Insight Advisor attempts to link all natural language questions to field names and values in your data. It cannot know all possible search terms that your app users might enter. Vocabulary lets you fill these terminology gaps, linking terms to fields and field values.

For example, in the tutorial app, you have some products that are swimwear. These items are in the category *Beachwear* in the data. In **Sheet**, if you search for *swimwear*, you will not get any results. Similarly, you cannot search for *footwear* and get results for both men's and women's footwear.

Custom analyses are useful when you know what analyses you want your app consumers to receive based on their search terms. For example, you know that your app users have a preference to viewing regional data as maps. You can use a custom analysis to make sure that maps are offered when users include *regional* in their searches.

Example questions can be used to help encourage users in their analysis and help guide them to insights. Users may not know what is possible in your app. Example questions can help make their analyses and exploration an easier experience.

Creating beachwear vocabulary

Do the following:

1. From **Prepare**, click **Vocabulary**.
2. In **Synonyms**, click **Create terms**.
3. In **Terms**, enter the following terms:
 - *swimwear*
 - *swimsuits*
4. Under **Applies to**, select *CategoryName*.
5. Under **Condition**, select **In**.
6. Under **In**, select the following values:
 - *Beachwear*
7. Click **Create**.

Creating footwear vocabulary

Do the following:

1. In **Synonyms**, click **Create terms**.
2. In **Terms**, enter *footwear*.
3. Under **Applies to**, select *CategoryName*.
4. Under **Condition**, select **In**.
5. Under **In**, select the following values:
 - *men's footwear*
 - *women's footwear*
6. Click **Create**.

Creating regional custom analysis

For this custom analysis, you will not specify a measure. This will allow Insight Advisor to pick a measure based on how the someone uses *regional* in their question.

Do the following:

1. Click **Custom analysis**.
2. Click **Create terms**.
3. Under **Terms**, enter *regional*.
4. Under **Analysis**, select **Breakdown (Geospatial)**.
5. In **Data**, under **Geographicals**, select *Country*.
6. Click **Chart**.
7. Under **Chart type**, select **Map**.
8. Click **Create**.

Creating example questions

Do the following:

1. Click **Example questions**.
2. Click **Add question**.
3. Under **Language**, select your language.
4. Under **Question**, enter *Who are the top customers for sales*.
5. Click **Add**.

Results

Navigate to **Sheet** and open **Insight Advisor**. Select the search box. *Who are the top customers for sales* is now available there. Search for *what is the average freight for footwear*. You now receive results for both men's and women's footwear.

Next, search for *gross profit for swimwear in 2019*. You now get results for beachwear while searching with swimwear.

Now, search for *show me regional sales*. You now get a map distribution of the sales by country.

Thank you!

Now you have finished this tutorial, and hopefully you have gained some more knowledge about business logic and Insight Advisor in Qlik Sense.

12 Troubleshooting - Loading data

This section describes problems that can occur when loading and modeling data in Qlik Cloud Analytics and Qlik Sense.

12.1 Attaching a file by dropping it in **Add data** does not work

You are trying to attach a file by dragging and dropping it on the **Add data/Attach files** dialog, but the file is not uploaded.

Possible cause

The file is stored in a ZIP archive. It is not possible to attach individual files from a ZIP archive.

Proposed action

Extract the files from the ZIP archive before attaching them.

12.2 Character set problems with non-ANSI encoded data files

You may experience problems with character encoding in non-ANSI encoded data files when using an ODBC data connection.

Possible cause

ODBC data connections do not provide full capabilities for character set encoding.

Proposed action

Do the following:

- If possible, import the data files using a folder data connection, which supports more options for handling character codes. This is probably the best option if you are loading a Microsoft Excel spreadsheet or a text data file.

12.3 Circular references warning when loading data

Possible cause

If you have loaded more than two tables, the tables can be associated in such a way that there is more than one path of associations between two fields, causing a loop in the data structure.

Proposed action

12.4 Columns are not lining up as expected when selecting data from a fixed record file

Possible cause

The file uses tab characters to pad the columns. Typically, you will see that the field headings do not line up with the expected data if you select **Field breaks** in the select dialog.

In this case, the tab character is usually equivalent to a number of characters.

Proposed action

Do the following:

1. Select **No field names** in **Field names**.
2. Select **Field breaks**.
3. Increase the setting of **Tab size** until you see the columns lining up with the header.
4. Insert field breaks by clicking at the appropriate column positions.
5. Select **Data preview**.
6. Select **Embedded field names** in **Field names**.

The columns are now lined up properly, and each field should have the correct field name.

12.5 Connector is not working

You are trying to create a data connection to a separately installed connector in the data load editor, but the connection fails, or an existing connection is labeled as unknown.

The connector is not properly installed

Possible cause

The connector is not properly installed according to installation instructions. If an app uses a connector on a multi-node site, the connector needs to be installed on all nodes.

Proposed action

Do the following:

- Verify that the connector is installed according to instructions on all nodes of the site.

The connector is not adapted for Qlik Sense

Possible cause

QlikView connectors need to be adapted for Qlik Sense if you want to be able to select data.

Proposed action (if you developed the connector yourself with the QVX SDK)

Do the following:

- You need to adapt the connector for Qlik Sense with an interface to select data.

Proposed action (if the connector was supplied to you)

Do the following:

- Contact the connector supplier to acquire a Qlik Sense adapted connector.

12.6 Data connection stops working after SQL Server is restarted

Possible cause

If you create a data connection to a SQL Server, and then restart the SQL Server, the data connection may stop working, and you are not able to select data. Qlik Sense has lost connection to the SQL Server and was not able to reconnect.

Proposed action

Qlik Sense:

Do the following:

- Close the app, and open it again from the hub.

Qlik Sense Desktop:

Do the following:

1. Close all apps.
2. Restart Qlik Sense Desktop.

12.7 Data load editor does not display the script

When Data load editor is opened, the content of the editor is blank, and the script cannot be edited.

Possible cause

The script contains very complex constructions, for example, a large number of nested if statements.

Proposed action

Open the data load editor in safe mode by adding `/debug/dle_safe_mode` to the URL. This will disable syntax highlighting and auto-complete functions, but you should be able to edit and save the script.



*Consider moving the complex parts of the script to a separate text file, and use the **include** variable to inject it into the script at runtime.*

12.8 Data load script is executed without error, but data is not loaded

The script is executed without syntax or load errors, but data is not loaded as expected. A general recommendation is to activate debug to step through the script and examine execution results, but here are some common causes of error.

A statement is not terminated with a semicolon

Possible cause

You have forgotten to terminate a statement with a semicolon.

Proposed action

Do the following:

- Terminate all statements with a semicolon.

Single quote character inside a string

Possible cause

A string contains a single quote character in, for example, a SET variable statement.

Proposed action

Do the following:

- If a string contains a single quote character, it needs to be escaped with an extra single quote.

12.9 Data manager does not show tables in app that contains data

When opening an app created in a Qlik Sense version earlier than 3.0, Data manager shows no tables and a message is displayed that the app contains no data.

Possible cause

The improved data model in Qlik Sense 3.0 and later requires a data reload to complete data profiling and preparation.

Proposed action

Click **Load data** in Data manager. This requires that the app can access the data sources that are used in the app.

12.10 Data manager work flows are broken for all users creating apps on a server

Users get errors when trying to use **Add data** or **Load data** in **Data manager**, or when refreshing the app in the browser.

Possible cause

The **Data manager** uses QVD files to cache loaded data. These files are deleted automatically when they are no longer used, but if a large number accumulate, or they become corrupted, they can cause errors.

Proposed action

Delete the folder containing the QVD files. On a Qlik Sense server, the cache is located at:

<Qlik Sense shared folder>\Apps\DataPrepAppCache

On a Qlik Sense Desktop, the cache is located at:

C:\Users\<username>\Documents\Qlik\Sense\Apps\DataPrepAppCache

12.11 Data selection problems with an OLE DB data source

Possible cause

If you are not able to select data from an OLE DB data connection, you need to check how the connection is configured.

Proposed action

Do the following:

1. Verify that the connection string is correctly designed.
2. Verify that you are using appropriate credentials to log on.

12.12 Date fields are not recognized as date fields in sheet view

You have fields containing date or timestamp data, but they are not recognized as date fields in sheet view, that is, they are not indicated with  in the assets panel and other field lists.

Data profiling was disabled when the table was added

Possible cause

When you added the tables, you disabled data profiling from  beside the **Add data** button.

With this option, date and timestamp fields that are recognized will function correctly, but they are not indicated with  in the assets panel and other field lists, and expanded property fields are not available.

Proposed action

Open **Data manager** and click **Load data**.

Now, all date and timestamp fields should be indicated with  in the assets panel of sheet view. If they are still not indicated with , the field data is probably using a format that is not recognized as a date.

Date format was not recognized

Possible cause

The input format of the date field was not recognized when the table was loaded. Usually, Qlik Sense recognizes date fields automatically, based on locale settings and common date formats, but in some cases you may need to specify the input format.

Proposed action

Open **Data manager** and edit the table containing the field that was not recognized as a date. The field is most likely indicated with  as a general field. Change the field type to **Date** or **Timestamp**, with an input format that matches the field data.

12.13 Error message "**Invalid path**" when attaching a file

Possible cause

The file name is too long. Qlik Sense only supports file names up to 171 characters.

Proposed action

Rename the file to a name that contains less than 172 characters.

12.14 Errors when loading an app converted from a QlikView document

You may receive errors when reloading an app that was converted from a QlikView document due to differences between the two products.

Absolute file path references are used in the script

Possible cause

The load script refers to files using absolute paths, which is not supported in Qlik Sense standard mode. Examples of error messages are "Invalid Path" and "LOAD statement only works with lib:// paths in this script mode".

Proposed action

Do the following:

- Replace all file references with **lib://** references to data connections in Qlik Sense.

Unsupported functions or statements are used in the script

Possible cause

If you get a syntax error when running the script in the data load editor, it may be related to using QlikView script statements or functions that are not supported in Qlik Sense.

Proposed action

Do the following:

- Remove the invalid statement or replace it with a valid one.

12.15 Microsoft Excel: Loading data from files in data manager or data load editor fails

Possible cause

Excel spreadsheet has **Freeze Panes** or **Split** screen enabled, and there are empty cells in a table.

Proposed action

Disable **Freeze Panes** or **Split** screen, or clean the spreadsheet, and then reload the data.

12.16 Microsoft Excel: Problems connecting to and loading data from files through ODBC

Possible cause

You may encounter problems when setting up an ODBC data connection to a Microsoft Excel file, or loading data from Microsoft Excel files through an ODBC data connection. This is commonly due to issues with the ODBCDSN configuration in Windows, or problems with the associated ODBC drivers.

Proposed action

Qlik Sense has native support for loading Microsoft Excel files. If possible, replace the ODBC data connection with a folder data connection that connects to the folder containing the Microsoft Excel files.

12.17 Running out of disk space

There are several reasons why a system may run low on disk space, and the data manager's method of caching loaded data in QVD files is one possible cause.

Proposed action

Delete the folder containing the QVD files. On a Qlik Sense server, the cache is located at:

<Qlik Sense shared folder>|Apps|DataPrepAppCache

On a Qlik Sense Desktop, the cache is located at:

C:\Users\<username>\Documents\Qlik\Sense\Apps\DataPrepAppCache

12.18 Synthetic keys warning when loading data

If you have loaded several files, you may receive a warning that synthetic keys have been created after loading the data.

Possible cause

If two tables contain more than one common field, Qlik Sense creates a synthetic key to resolve the linking.

Proposed action

In many cases, you do not need to do anything about synthetic keys if the linking is meaningful, but it is a good idea to review the data structure in the data model viewer.

12.19 Tables with common fields are not automatically associated by field name

You have added two or more tables using **Add data**. The tables have fields with a common field name, but they are not automatically associated.

Possible cause

When you added the tables, you kept the default option to enable data profiling in the **Add data** dialog. This option auto-qualifies all field names that are common between tables. For example, if you add table A and table B with a common field F1 using this option, the field will be named F1 in table A, and B.F1 in table B. This means that the tables are not automatically associated.

Proposed action

Open **Data manager** and select the **Associations** view. Now you can associate the tables based on data profiling recommendations.

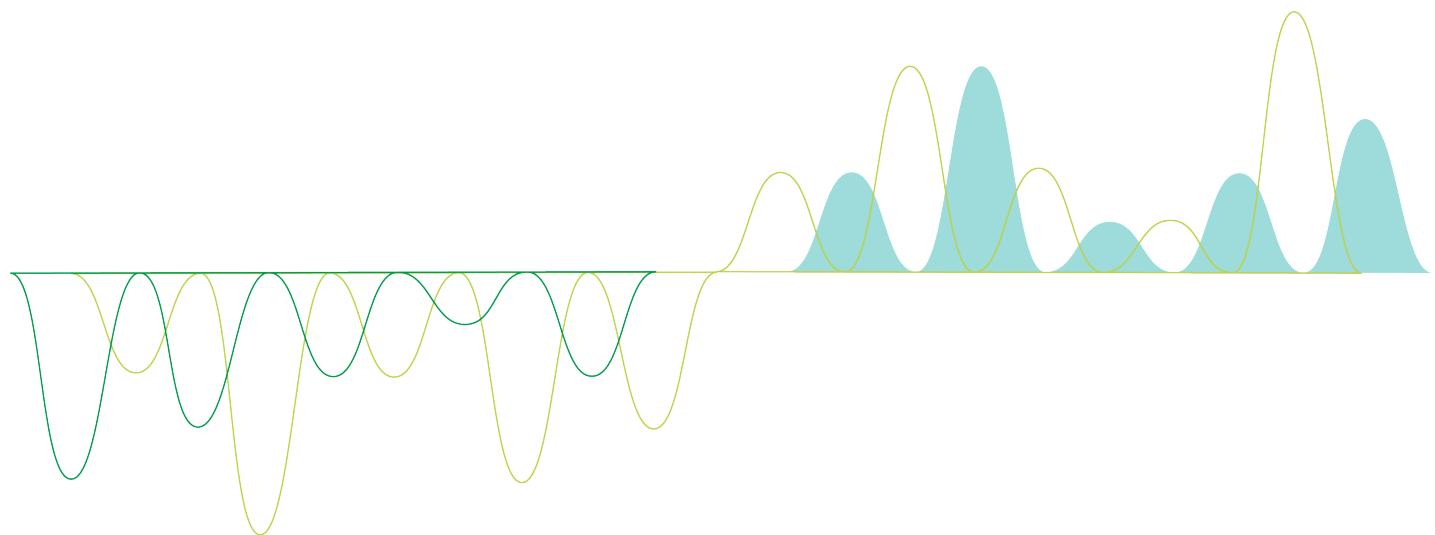
When you have associated the tables, click **Load data**.

Script syntax and chart functions

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 What is Qlik Sense?	16
1.1 What can you do in Qlik Sense?	16
1.2 How does Qlik Sense work?	16
The app model	16
The associative experience	16
Collaboration and mobility	16
1.3 How can you deploy Qlik Sense?	16
Qlik Sense Desktop	16
Qlik Sense Enterprise	17
1.4 How to administer and manage a Qlik Sense site	17
1.5 Extend Qlik Sense and adapt it for your own purposes	17
Building extensions and mashups	17
Building clients	17
Building server tools	17
Connecting to other data sources	17
2 Script syntax overview	18
2.1 Introduction to script syntax	18
2.2 What is Backus-Naur formalism?	18
2 Script statements and keywords	20
2.3 Script control statements	20
Script control statements overview	20
Call	22
Do..loop	23
End	24
Exit	24
Exit script	24
For..next	24
For each..next	26
If..then..elseif..else..end if	29
Next	30
Sub..end sub	30
Switch..case..default..end switch	31
To	32
2.4 Script prefixes	32
Script prefixes overview	32
Add	36
Buffer	38
Concatenate	39
Crosstable	44
First	54
Generic	56
Hierarchy	62
HierarchyBelongsTo	64
Inner	65
IntervalMatch	66
Join	69
Keep	79

Contents

Left	80
Mapping	81
Merge	83
NoConcatenate	87
Only	96
Outer	96
Partial reload	97
Replace	100
Right	102
Sample	103
Semantic	106
Unless	110
When	116
2.5 Script regular statements	122
Script regular statements overview	122
Alias	128
AutoNumber	129
Binary	132
Comment field	133
Comment table	134
Connect	134
Declare	136
Derive	138
Direct Query	139
Directory	144
Disconnect	145
Drop	146
Drop table	147
Execute	148
Field/Fields	149
FlushLog	149
Force	149
From	151
Load	151
Let	168
Loosen Table	168
Map	169
NullAsNull	170
NullAsValue	170
Qualify	171
Rem	172
Rename	173
Search	174
Section	175
Select	175
Set	178
Sleep	178
SQL	179

Contents

SQLColumns	179
SQLTables	180
SQLTypes	181
Star	182
Store	183
Table/Tables	185
Tag	185
Trace	186
Unmap	186
Unqualify	187
Untag	187
2.6 Working directory	188
Qlik Sense Desktop working directory	188
Qlik Sense working directory	188
2 Working with variables in the data load editor	189
2.7 Overview	189
2.8 Defining a variable	189
2.9 Deleting a variable	190
2.10 Loading a variable value as a field value	190
2.11 Variable calculation	190
2.12 System variables	191
System variables overview	191
CreateSearchIndexOnReload	194
HidePrefix	194
HideSuffix	194
Include	195
OpenUrlTimeout	196
StripComments	196
Verbatim	197
2.13 Value handling variables	197
Value handling variables overview	197
NullDisplay	198
NullInterpret	198
NullValue	198
OtherSymbol	198
2.14 Number interpretation variables	199
Currency formatting	199
Number formatting	199
Time formatting	200
BrokenWeeks	201
DateFormat	202
DayNames	208
DecimalSep	213
FirstWeekDay	215
LongDayNames	219
LongMonthNames	222
MoneyDecimalSep	226

Contents

MoneyFormat	230
MoneyThousandSep	234
MonthNames	238
NumericalAbbreviation	243
ReferenceDay	244
ThousandSep	249
TimeFormat	255
TimestampFormat	255
2.15 Direct Discovery variables	258
Direct Discovery system variables	258
Teradata query banding variables	259
Direct Discovery character variables	260
Direct Discovery number interpretation variables	261
2.16 Error variables	262
Error variables overview	262
ErrorMode	262
ScriptError	263
ScriptErrorCount	264
ScriptErrorList	264
2 Script expressions	265
3 Chart expressions	266
3.1 Defining the aggregation scope	266
3.2 Set analysis	268
Set expressions	269
Examples	270
Natural sets	270
Set identifiers	272
Set operators	273
Set modifiers	274
Inner and outer set expressions	295
Tutorial - Creating a set expression	297
Syntax for set expressions	307
3.3 General syntax for chart expressions	307
3.4 General syntax for aggregations	308
4 Operators	309
4.1 Bit operators	309
4.2 Logical operators	310
4.3 Numeric operators	310
4.4 Relational operators	311
4.5 String operators	312
&	313
like	313
5 Script and chart functions	314
5.1 Analytic connections for server-side extensions (SSE)	314
5.2 Aggregation functions	314
Using aggregation functions in a data load script	315

Contents

Using aggregation functions in chart expressions	315
How aggregations are calculated	315
Aggregation of key fields	315
Basic aggregation functions	316
Counter aggregation functions	338
Financial aggregation functions	355
Statistical aggregation functions	382
Statistical test functions	447
String aggregation functions	510
Synthetic dimension functions	522
Nested aggregations	525
5.3 Aggr - chart function	525
Examples: Chart expressions using Aggr	528
5.4 Color functions	531
Pre-defined color functions	533
ARGB	534
RGB	535
HSL	537
5.5 Conditional functions	537
Conditional functions overview	537
alt	538
class	539
coalesce	541
if	542
match	545
mixmatch	548
pick	551
wildmatch	552
5.6 Counter functions	555
Counter functions overview	555
autonumber	556
autonumberhash128	559
autonumberhash256	561
IterNo	563
RecNo	564
RowNo	565
RowNo - chart function	566
5.7 Date and time functions	568
Date and time functions overview	569
addmonths	577
addyears	587
age	594
converttocalctime	596
day	599
dayend	605
daylightsaving	613
dayname	613
daynumberofquarter	615

Contents

daynumberofyear	621
daystart	628
firstworkdate	635
GMT	637
hour	641
inday	644
indaytotime	652
inlunarweek	662
inlunarweektodate	674
inmonth	685
inmonths	693
inmonthstodate	706
inmonthtodate	719
inquarter	729
inquartertodate	742
inweek	754
inweektodate	770
inyear	784
inyeartodate	796
lastworkdate	809
localtime	818
lunarweekend	819
lunarweekname	831
lunarweekstart	844
makedate	855
maketime	861
makeweekdate	868
minute	877
month	882
monthend	888
monthname	898
monthsend	905
monthsname	918
monthsstart	931
monthstart	944
networkdays	953
now	963
quarterend	970
quartername	983
quarterstart	995
second	1006
setdateyear	1011
setdateyearmonth	1013
timezone	1015
today	1015
UTC	1020
week	1021
weekday	1037

Contents

weekend	1045
weekname	1058
weekstart	1072
weekyear	1084
year	1094
yearend	1100
yearname	1112
yearstart	1125
yeartodate	1137
5.8 Exponential and logarithmic functions	1152
5.9 Field functions	1153
Count functions	1153
Field and selection functions	1154
GetAlternativeCount - chart function	1154
GetCurrentSelections - chart function	1155
GetExcludedCount - chart function	1157
GetFieldSelections - chart function	1158
GetNotSelectedCount - chart function	1160
GetObjectDimension - chart function	1161
GetObjectField - chart function	1161
GetObjectMeasure - chart function	1162
GetPossibleCount - chart function	1163
GetSelectedCount - chart function	1164
5.10 File functions	1165
File functions overview	1165
Attribute	1167
ConnectionString	1174
FileBaseName	1175
FileDir	1175
FileExtension	1176
FileName	1176
FilePath	1176
FileSize	1177
FileTime	1177
GetFolderPath	1178
QvdCreateTime	1179
QvdFieldName	1180
QvdNoOfFields	1181
QvdNoOfRecords	1182
QvdTableName	1183
5.11 Financial functions	1184
Financial functions overview	1184
BlackAndSchole	1185
FV	1186
nPer	1187
Pmt	1188
PV	1189
Rate	1189

Contents

5.12 Formatting functions	1190
Formatting functions overview	1191
ApplyCodepage	1192
Date	1193
Dual	1194
Interval	1196
Money	1197
Num	1198
Time	1201
Timestamp	1202
5.13 General numeric functions	1203
General numeric functions overview	1203
Combination and permutation functions	1204
Modulo functions	1204
Parity functions	1205
Rounding functions	1205
BitCount	1205
Ceil	1206
Combin	1207
Div	1207
Even	1208
fabs	1208
Fact	1209
Floor	1209
Fmod	1210
Frac	1211
Mod	1212
Odd	1213
Permut	1213
Round	1214
Sign	1215
5.14 Geospatial functions	1216
Geospatial functions overview	1216
GeoAggrGeometry	1218
GeoBoundingBox	1219
GeoCountVertex	1219
GeoGetBoundingBox	1220
GeoGetPolygonCenter	1220
GeoInvProjectGeometry	1221
GeoMakePoint	1221
GeoProject	1222
GeoProjectGeometry	1223
GeoReduceGeometry	1223
5.15 Interpretation functions	1224
Interpretation functions overview	1225
Date#	1226
Interval#	1227
Money#	1227

Contents

Num#	1229
Text	1229
Time#	1230
Timestamp#	1231
5.16 Inter-record functions	1232
Row functions	1232
Column functions	1233
Field functions	1234
Pivot table functions	1234
Inter-record functions in the data load script	1235
Above - chart function	1235
Below - chart function	1240
Bottom - chart function	1243
Column - chart function	1248
Dimensionality - chart function	1250
Exists	1251
FieldIndex	1255
FieldValue	1256
FieldValueCount	1258
LookUp	1260
NoOfRows - chart function	1262
Peek	1264
Previous	1271
Top - chart function	1272
SecondaryDimensionality - chart function	1276
After - chart function	1276
Before - chart function	1277
First - chart function	1279
Last - chart function	1280
ColumnNo - chart function	1281
NoOfColumns - chart function	1281
5.17 Logical functions	1282
5.18 Mapping functions	1283
Mapping functions overview	1283
ApplyMap	1283
MapSubstring	1285
5.19 Mathematical functions	1287
5.20 NULL functions	1287
NULL functions overview	1288
EmptyIsNull	1288
IsNull	1288
NULL	1289
5.21 Range functions	1290
Basic range functions	1290
Counter range functions	1291
Statistical range functions	1292
Financial range functions	1292
RangeAvg	1293

Contents

RangeCorrel	1295
RangeCount	1297
RangeFractile	1300
RangeIRR	1302
RangeKurtosis	1303
RangeMax	1304
RangeMaxString	1306
RangeMin	1307
RangeMinString	1309
RangeMissingCount	1311
RangeMode	1312
RangeNPV	1314
RangeNullCount	1315
RangeNumericCount	1317
RangeOnly	1318
RangeSkew	1319
RangeStdev	1320
RangeSum	1322
RangeTextCount	1324
RangeXIRR	1325
RangeXNPV	1327
5.22 Relational functions	1329
Ranking functions	1329
Clustering functions	1330
Time series decomposition functions	1331
Rank - chart function	1332
HRank - chart function	1335
Optimizing with k-means: A real-world example	1337
KMeans2D - chart function	1346
KMeansND - chart function	1361
KMeansCentroid2D - chart function	1376
KMeansCentroidND - chart function	1377
STL_Trend - chart function	1378
STL_Seasonal - chart function	1380
STL_Residual - chart function	1382
Tutorial - Time series decomposition in Qlik Sense	1384
5.23 Statistical distribution functions	1388
Statistical distribution functions overview	1388
BetaDensity	1391
BetaDist	1391
BetaInv	1391
BinomDist	1392
BinomFrequency	1392
BinomInv	1393
ChiDensity	1393
ChiDist	1394
Chilnv	1394
FDensity	1395

Contents

FDist	1395
FInv	1396
GammaDensity	1397
GammaDist	1397
Gammaln	1398
NormDist	1398
NormInv	1399
PoissonDist	1400
PoissonFrequency	1400
PoissonInv	1400
TDensity	1401
TDist	1401
TInv	1402
5.24 String functions	1403
String functions overview	1403
Capitalize	1406
Chr	1407
Evaluate	1408
FindOneOf	1408
Hash128	1410
Hash160	1410
Hash256	1411
Index	1412
IsJson	1413
JsonGet	1414
JsonSet	1415
KeepChar	1416
Left	1417
Len	1418
LevenshteinDist	1419
Lower	1421
LTrim	1422
Mid	1423
Ord	1424
PurgeChar	1425
Repeat	1426
Replace	1427
Right	1427
RTrim	1428
SubField	1429
SubStringCount	1433
TextBetween	1433
Trim	1434
Upper	1435
5.25 System functions	1436
System functions overview	1436
EngineVersion	1439
InObject - chart function	1439

Contents

IsPartialReload	1443
ObjectId - chart function	1443
ProductVersion	1446
StateName - chart function	1447
5.26 Table functions	1447
Table functions overview	1447
FieldName	1449
FieldNumber	1450
NoOfFields	1450
NoOfRows	1451
5.27 Trigonometric and hyperbolic functions	1451
6 File system access restriction	1454
6.1 Security aspects when connecting to file based ODBC and OLE DB data connections	1454
6.2 Limitations in standard mode	1454
System variables	1454
Regular script statements	1456
Script control statements	1457
File functions	1457
System functions	1459
6.3 Disabling standard mode	1459
Qlik Sense	1460
Qlik Sense Desktop	1460
6 Chart level scripting	1461
6.4 Control statements	1461
Chart modifier control statements overview	1461
Call	1463
Do..loop	1464
End	1464
Exit	1464
Exit script	1464
For..next	1465
For each..next	1466
If..then..elseif..else..end if	1469
Next	1470
Sub..end sub	1470
Switch..case..default..end switch	1471
To	1472
6.5 Prefixes	1472
Chart modifier prefixes overview	1472
Add	1473
Replace	1473
6.6 Regular statements	1473
Chart modifier regular statements overview	1474
Load	1474
Let	1478
Set	1479
Put	1479

Contents

HCValue	1480
7 QlikView functions and statements not supported in Qlik Sense	1481
7.1 Script statements not supported in Qlik Sense	1481
7.2 Functions not supported in Qlik Sense	1481
7.3 Prefixes not supported in Qlik Sense	1481
8 Functions and statements not recommended in Qlik Sense	1482
8.1 Script statements not recommended in Qlik Sense	1482
8.2 Script statement parameters not recommended in Qlik Sense	1482
8.3 Functions not recommended in Qlik Sense	1483
ALL qualifier	1484

1 What is Qlik Sense?

Qlik Sense is a platform for data analysis. With Qlik Sense you can analyze data and make data discoveries on your own. You can share knowledge and analyze data in groups and across organizations. Qlik Sense lets you ask and answer your own questions and follow your own paths to insight. Qlik Sense enables you and your colleagues to reach decisions collaboratively.

1.1 What can you do in Qlik Sense?

Most Business Intelligence (BI) products can help you answer questions that are understood in advance. But what about your follow-up questions? The ones that come after someone reads your report or sees your visualization? With the Qlik Sense associative experience, you can answer question after question after question, moving along your own path to insight. With Qlik Sense you can explore your data freely, with just clicks, learning at each step along the way and coming up with next steps based on earlier findings.

1.2 How does Qlik Sense work?

Qlik Sense generates views of information on the fly for you. Qlik Sense does not require predefined and static reports or you being dependent on other users – you just click and learn. Every time you click, Qlik Sense instantly responds, updating every Qlik Sense visualization and view in the app with a newly calculated set of data and visualizations specific to your selections.

The app model

Instead of deploying and managing huge business applications, you can create your own Qlik Sense apps that you can reuse, modify and share with others. The app model helps you ask and answer the next question on your own, without having to go back to an expert for a new report or visualization.

The associative experience

Qlik Sense automatically manages all the relationships in the data and presents information to you using a **green/white/gray** metaphor. Selections are highlighted in green, associated data is represented in white, and excluded (unassociated) data appears in gray. This instant feedback enables you to think of new questions and continue to explore and discover.

Collaboration and mobility

Qlik Sense further enables you to collaborate with colleagues no matter when and where they are located. All Qlik Sense capabilities, including the associative experience and collaboration, are available on mobile devices. With Qlik Sense, you can ask and answer your questions and follow-up questions, with your colleagues, wherever you are.

1.3 How can you deploy Qlik Sense?

There are two versions of Qlik Sense to deploy, Qlik Sense Desktop and Qlik Sense Enterprise.

Qlik Sense Desktop

This is an easy-to-install single user version that is typically installed on a local computer.

Qlik Sense Enterprise

This version is used to deploy Qlik Sense sites. A site is a collection of one or more server machines connected to a common logical repository or central node.

1.4 How to administer and manage a Qlik Sense site

With the Qlik Management Console you can configure, manage and monitor Qlik Sense sites in an easy and intuitive way. You can manage licenses, access and security rules, configure nodes and data source connections and synchronize content and users among many other activities and resources.

1.5 Extend Qlik Sense and adapt it for your own purposes

Qlik Sense provides you with flexible APIs and SDKs to develop your own extensions and adapt and integrate Qlik Sense for different purposes, such as:

Building extensions and mashups

Here you can do web development using JavaScript to build extensions that are custom visualization in Qlik Sense apps, or you use a mashups APIs to build websites with Qlik Sense content.

Building clients

You can build clients in .NET and embed Qlik Sense objects in your own applications. You can also build native clients in any programming language that can handle WebSocket communication by using the Qlik Sense client protocol.

Building server tools

With service and user directory APIs you can build your own tool to administer and manage Qlik Sense sites.

Connecting to other data sources

Create Qlik Sense connectors to retrieve data from custom data sources.

2 Script syntax overview

2.1 Introduction to script syntax

In a script, the name of the data source, the names of the tables, and the names of the fields included in the logic are defined. Furthermore, the fields in the access rights definition are defined in the script. A script consists of a number of statements that are executed consecutively.

The Qlik Sense command line syntax and script syntax are described in a notation called Backus-Naur Formalism, or BNF code.

The first lines of code are already generated when a new Qlik Sense file is created. The default values of these number interpretation variables are derived from the regional settings of the OS.

The script consists of a number of script statements and keywords that are executed consecutively. All script statements must end with a semicolon, ";".

You can use expressions and functions in the **LOAD**-statements to transform the data that has been loaded.

For a table file with commas, tabs or semicolons as delimiters, a **LOAD**-statement may be used. By default a **LOAD**-statement will load all fields of the file.

General databases can be accessed through ODBC or OLE DBdatabase connectors. . Here standard SQL statements are used. The SQL syntax accepted differs between different ODBC drivers.

Additionally, you can access other data sources using custom connectors.

2.2 What is Backus-Naur formalism?

The Qlik Sense command line syntax and script syntax are described in a notation called Backus-Naur formalism, also known as BNF code.

The following table provides a list of symbols used in BNF code, with a description of how they are interpreted:

Symbols	
Symbol	Description
	Logical OR: the symbol on either side can be used.
()	Parentheses defining precedence: used for structuring the BNF syntax.
[]	Square brackets: enclosed items are optional.
{ }	Braces: enclosed items may be repeated zero or more times.
Symbol	A non-terminal syntactic category, that: can be divided further into other symbols. For example, compounds of the above, other non-terminal symbols, text strings, and so on.
::=	Marks the beginning of a block that defines a symbol.
LOAD	A terminal symbol consisting of a text string. Should be written as it is into the script.

All terminal symbols are printed in a **bold face** font. For example, "(" should be interpreted as a parenthesis defining precedence, whereas "(" should be interpreted as a character to be printed in the script.

Example:

The description of the alias statement is:

```
alias fieldname as aliasname { , fieldname as aliasname}
```

This should be interpreted as the text string "alias", followed by an arbitrary field name, followed by the text string "as", followed by an arbitrary alias name. Any number of additional combinations of "fieldname as alias" may be given, separated by commas.

The following statements are correct:

```
alias a as first;  
alias a as first, b as second;  
alias a as first, b as second, c as third;
```

The following statements are not correct:

```
alias a as first b as second;  
alias a as first { , b as second };
```

2 Script statements and keywords

The Qlik Sense script consists of a number of statements. A statement can be either a regular script statement or a script control statement. Certain statements can be preceded by prefixes.

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by a semicolon or the end-of-line.

Prefixes may be applied to applicable regular statements but never to control statements. The **when** and **unless** prefixes can however be used as suffixes to a few specific control statement clauses.

In the next subchapter, an alphabetical listing of all script statements, control statements and prefixes, are found.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

2.3 Script control statements

The Qlik Sense script consists of a number of statements. A statement can be either a regular script statement or a script control statement.

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by semicolon or end-of-line.

Prefixes are never applied to control statements, with the exceptions of the prefixes **when** and **unless** which may be used with a few specific control statements.

All script keywords can be typed with any combination of lower case and upper case characters.

Script control statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

```
Call name ( [ paramlist ] )
```

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

```
Do..loop [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop [ ( while | until ) condition ]
```

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

```
Exit script[ (when | unless) condition ]
```

For each ..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

```
For each..next var in list  
[statements]  
[exit for [ ( when | unless ) condition ]  
[statements]  
next [var]
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

```
For..next counter = expr1 to expr2 [ stepexpr3 ]  
[statements]  
[exit for [ ( when | unless ) condition ]  
[statements]  
Next [counter]
```

If..then

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.



Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

```
If..then..elseif..else..end if condition then  
[ statements ]  
{ elseif condition then  
[ statements ] }  
[ else  
[ statements ] ]  
end if
```

Sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

```
Sub..end sub name [ ( paramlist )] statements end sub
```

Switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

```
Switch..case..default..end switch expression {case valuelist [ statements ]}  
[default statements] end switch
```

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

Syntax:

```
Call name ( [ paramlist ] )
```

Arguments:

Arguments	
Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of the actual parameters to be sent to the subroutine. Each item in the list may be a field name, a variable or an arbitrary expression.

The subroutine called by a **call** statement must be defined by a **sub** encountered earlier during script execution.

Parameters are copied into the subroutine and, if the parameter in the **call** statement is a variable and not an expression, copied back out again upon exiting the subroutine.

Limitations:

- Since the **call** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.
- When you define a subroutine with **sub..end sub** inside a control statement, for example **if..then**, you can only call the subroutine from within the same control statement.

Example:

This example lists all Qlik related files in a folder and its subfolders, and stores file information in a table. It is assumed that you have created a data connection named Apps to the folder .

The DoDir subroutine is called with the reference to the folder, 'lib://Apps', as parameter. Inside the subroutine, there is a recursive call, **call Dodir (Dir)**, that makes the function look for files recursively in subfolders.

```

sub DoDir (Root)
    For Each Ext in 'qvw', 'qvo', 'qvs', 'qvt', 'qvd', 'qvc', 'qvf'
        For Each File in filelist (Root&'\*.' &Ext)
            LOAD
                '$(File)' as Name,
                Filesize( '$(File)' ) as Size,
                FileTime( '$(File)' ) as FileTime
            autogenerate 1;
        Next File
    Next Ext
    For Each Dir in dirlist (Root&'\*')
        call DoDir (Dir)
    Next Dir
End Sub

call DoDir ('lib://Apps')

```

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

Syntax:

```

Do [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop[ ( while | until ) condition ]

```



*Since the **do..loop** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**do**, **exit do** and **loop**) must not cross a line boundary.*

Arguments:

Arguments

Argument	Description
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
while / until	The while or until conditional clause must only appear once in any do..loop statement, i.e. either after do or after loop . Each condition is interpreted only the first time it is encountered but is evaluated for every time it encountered in the loop.
exit do	If an exit do clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the loop clause denoting the end of the loop. An exit do clause can be made conditional by the optional use of a when or unless suffix.

Example:

```
// LOAD files file1.csv..file9.csv
```

```
Set a=1;
Do while a<10
LOAD * from file$(a).csv;
Let a=a+1;
Loop
```

End

The **End** script keyword is used to close **If**, **Sub** and **Switch** clauses.

Exit

The **Exit** script keyword is part of the **Exit Script** statement, but can also be used to exit **Do**, **For** or **Sub** clauses.

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

Syntax:

```
Exit Script [ (when | unless) condition ]
```

Since the **exit script** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.

Arguments:

Arguments

Argument	Description
condition	A logical expression evaluating to True or False.
when / unless	An exit script statement can be made conditional by the optional use of when or unless clause.

Examples:

```
//Exit script
Exit Script;

//Exit script when a condition is fulfilled
Exit Script when a=1
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

Syntax:

```
For counter = expr1 to expr2 [ step expr3 ]
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
Next [counter]
```

The expressions *expr1*, *expr2* and *expr3* are only evaluated the first time the loop is entered. The value of the counter variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.



Since the **for..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for..to..step**, **exit for** and **next**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
counter	A variable name. If <i>counter</i> is specified after next it must be the same variable name as the one found after the corresponding for .
expr1	An expression which determines the first value of the <i>counter</i> variable for which the loop should be executed.
expr2	An expression which determines the last value of the <i>counter</i> variable for which the loop should be executed.
expr3	An expression which determines the value indicating the increment of the <i>counter</i> variable each time the loop has been executed.
condition	a logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.

Example 1: Loading a sequence of files

```
// LOAD files file1.csv..file9.csv
for a=1 to 9
    LOAD * from file$(a).csv;
next
```

Example 2: Loading a random number of files

In this example, we assume there are data files `x1.csv`, `x3.csv`, `x5.csv`, `x7.csv` and `x9.csv`. Loading is stopped at a random point using the `if rand()<0.5 then` condition.

```
for counter=1 to 9 step 2
    set filename=x$(counter).csv;
    if rand()<0.5 then
        exit for unless counter=1
    end if
    LOAD a,b from $(filename);
next
```

For each..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

Syntax:

Special syntax makes it possible to generate lists with file and directory names in the current directory.

```
for each var in list
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
next [var]
```

Arguments:

Arguments

Argument	Description
var	A script variable name which will acquire a new value from list for each loop execution. If var is specified after next it must be the same variable name as the one found after the corresponding for each .

The value of the **var** variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.



Since the **for each..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for each**, **exit for** and **next**) must not cross a line boundary.

Syntax:

```
list := item { , item }
item := constant | (expression) | filelist mask | dirlist mask |
fieldvaluelist mask
```

Arguments

Argument	Description
constant	Any number or string. Note that a string written directly in the script must be enclosed by single quotes. A string without single quotes will be interpreted as a variable, and the value of the variable will be used. Numbers do not need to be enclosed by single quotes.
expression	An arbitrary expression.
mask	A filename or folder name mask which may include any valid filename characters as well as the standard wildcard characters, * and ?. You can use absolute file paths or lib:// paths.
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
filelist mask	This syntax produces a comma separated list of all files in the current directory matching the filename mask.
<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> This argument supports only library connections in standard mode. </div>	
dirlist mask	This syntax produces a comma separated list of all folders in the current folder matching the folder name mask.
<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> This argument supports only library connections in standard mode. </div>	
fieldvaluelist mask	This syntax iterates through the values of a field already loaded into Qlik Sense.



The Qlik Web Storage Provider Connectors and other DataFiles connections do not support filter masks that use wildcard (*) and (?) characters.

Example 1: Loading a list of files

```
// LOAD the files 1.csv, 3.csv, 7.csv and xyz.csv
for each a in 1,3,7,'xyz'
    LOAD * from file$(a).csv;
next
```

Example 2: Creating a list of files on disk

This example loads a list of all Qlik Sense related files in a folder.

```
sub DoDir (Root)
    for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'

        for each File in filelist (Root&'/*.' &Ext)

            LOAD
                '$(File)' as Name,
                FileSize( '$(File)' ) as Size,
                FileTime( '$(File)' ) as FileTime
            autogenerate 1;

        next File

    next Ext
    for each Dir in dirlist (Root&'/*' )

        call DoDir (Dir)

    next Dir

end sub

call DoDir ('lib://dataFiles')
```

Example 3: Iterating through a the values of a field

This example iterates through the list of loaded values of FIELD and generates a new field, NEWFIELD. For each value of FIELD, two NEWFIELD records will be created.

```
load * inline [
FIELD
one
two
three
];

FOR Each a in FieldvalueList('FIELD')
LOAD '$(a)' & '-' & RecNo() as NEWFIELD AutoGenerate 2;
NEXT a
```

The resulting table looks like this:

Example table

NEWFIELD
one-1
one-2
two-1
two-2
three-1
three-2

If..then..elseif..else..end if

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.

Control statements are typically used to control the flow of the script execution. In a chart expression, use the **if** conditional function instead.

Syntax:

```
If condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

Arguments:

Arguments	
Argument	Description
condition	A logical expression which can be evaluated as True or False.
statements	Any group of one or more Qlik Sense script statements.

Example 1:

```
if a=1 then
  LOAD * from abc.csv;
  SQL SELECT e, f, g from tab1;
end if
```

Example 2:

```
if a=1 then; drop table xyz; end if;
```

Example 3:

```
if x>0 then
    LOAD * from pos.csv;
elseif x<0 then
    LOAD * from neg.csv;
else
    LOAD * from zero.txt;
end if
```

Next

The **Next** script keyword is used to close **For** loops.

Sub..end sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

Syntax:

```
Sub name [ ( paramlist )] statements end sub
```

Arguments are copied into the subroutine and, if the corresponding actual parameter in the **call** statement is a variable name, copied back out again upon exiting the subroutine.

If a subroutine has more formal parameters than actual parameters passed by a **call** statement, the extra parameters will be initialized to NULL and can be used as local variables within the subroutine.

Arguments:

Arguments

Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of variable names for the formal parameters of the subroutine. These can be used as any variable inside the subroutine.
statements	Any group of one or more Qlik Sense script statements.

Limitations:

- Since the **sub** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its two clauses (**sub** and **end sub**) must not cross a line boundary.
- When you define a subroutine with **sub..end sub** inside a control statement, for example **if..then**, you can only call the subroutine from within the same control statement.

Example 1:

```
Sub INCR (I,J)
I = I + 1
Exit Sub when I < 10
J = J + 1
End Sub
Call INCR (X,Y)
```

Example 2: - parameter transfer

```
Sub ParTrans (A,B,C)
A=A+1
B=B+1
C=C+1
End Sub
A=1
X=1
C=1
Call ParTrans (A, (X+1)*2)
```

The result of the above will be that locally, inside the subroutine, A will be initialized to 1, B will be initialized to 4 and C will be initialized to NULL.

When exiting the subroutine, the global variable A will get 2 as value (copied back from subroutine). The second actual parameter “(X+1)*2” will not be copied back since it is not a variable. Finally, the global variable C will not be affected by the subroutine call.

Switch..case..default..end switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

Syntax:

```
Switch expression {case valuelist [ statements ]} [default statements] end
switch
```



Since the **switch** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**switch**, **case**, **default** and **end switch**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
expression	An arbitrary expression.
valuelist	A comma separated list of values with which the value of expression will be compared. Execution of the script will continue with the statements in the first group encountered with a value in valuelist equal to the value in expression. Each value in valuelist may be an arbitrary expression. If no match is found in any case clause, the statements under the default clause, if specified, will be executed.
statements	Any group of one or more Qlik Sense script statements.

Example:

```
Switch I
Case 1
LOAD '$(I): CASE 1' as case autogenerate 1;
Case 2
LOAD '$(I): CASE 2' as case autogenerate 1;
Default
LOAD '$(I): DEFAULT' as case autogenerate 1;
End Switch
```

To

The **To** script keyword is used in several script statements.

2.4 Script prefixes

Prefixes may be applied to applicable regular statements but never to control statements. The **when** and **unless** prefixes can however be used as suffixes to a few specific control statement clauses.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Script prefixes overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Add

The **Add** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that it should add records to another table. It also specifies that this statement should be run in a partial reload. The **Add** prefix can also be used in a **Map** statement.

```
Add [only] [Concatenate[(tablename )]] (loadstatement | selectstatement)
Add [ Only ] mapstatement
```

Buffer

QVD files can be created and maintained automatically via the **buffer** prefix. This prefix can be used on most **LOAD** and **SELECT** statements in script. It indicates that QVD files are used to cache/buffer the result of the statement.

```
Buffer[ (option [ , option]) ] ( loadstatement | selectstatement )
option:= incremental | stale [after] amount [(days | hours)]
```

Concatenate

If two tables that are to be concatenated have different sets of fields, concatenation of two tables can still be forced with the **Concatenate** prefix.

```
Concatenate[ (tablename ) ] ( loadstatement | selectstatement )
```

Crosstable

The **crosstable** load prefix is used to transpose “cross table” or “pivot table” structured data. Data structured this way is commonly encountered when working with spreadsheet sources. The output and aim of the **crosstable** load prefix is to transpose such structures into a regular column-oriented table equivalent, as this structure is generally better suited for analysis in Qlik Sense.

```
Crosstable (attribute field name, data field name [ , n ] ) ( loadstatement | selectstatement )
```

First

The **First** prefix to a **LOAD** or **SELECT (SQL)** statement is used for loading a set maximum number of records from a data source table.

```
First n( loadstatement | selectstatement )
```

Generic

The **Generic** load prefix allows for conversion of entity–attribute–value modeled data (EAV) into a traditional, normalized relational table structure. EAV modeling is alternatively referred to as "generic data modeling" or "open schema".

```
Generic ( loadstatement | selectstatement )
```

Hierarchy

The **hierarchy** prefix is used to transform a parent-child hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName], [PathSource],
[PathName], [PathDelimiter], [Depth])(loadstatement | selectstatement)
```

HierarchBelongsTo

This prefix is used to transform a parent-child hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName,  
[DepthDiff]) (loadstatement | selectstatement)
```

Inner

The **join** and **keep** prefixes can be preceded by the prefix **inner**.

If used before **join** it specifies that an inner join should be used. The resulting table will thus only contain combinations of field values from the raw data tables where the linking field values are represented in both tables. If used before **keep**, it specifies that both raw data tables should be reduced to their common intersection before being stored in Qlik Sense.

```
.
```

```
Inner ( Join | Keep) [ (tablename) ] (loadstatement |selectstatement )
```

IntervalMatch

The **IntervalMatch** prefix is used to create a table matching discrete numeric values to one or more numeric intervals, and optionally matching the values of one or several additional keys.

```
IntervalMatch (matchfield) (loadstatement | selectstatement )  
IntervalMatch (matchfield, keyfield1 [ , keyfield2, ... keyfield5 ] )  
(loadstatement | selectstatement )
```

Join

The **join** prefix joins the loaded table with an existing named table or the last previously created data table.

```
[Inner | Outer | Left | Right] Join [ (tablename) ] ( loadstatement |  
selectstatement )
```

Keep

The **keep** prefix is similar to the **join** prefix. Just as the **join** prefix, it compares the loaded table with an existing named table or the last previously created data table, but instead of joining the loaded table with an existing table, it has the effect of reducing one or both of the two tables before they are stored in Qlik Sense, based on the intersection of table data. The comparison made is equivalent to a natural join made over all the common fields, i.e. the same way as in a corresponding join. However, the two tables are not joined and will be kept in Qlik Sense as two separately named tables.

```
(Inner | Left | Right) Keep [ (tablename) ] ( loadstatement | selectstatement )
```

Left

The **Join** and **Keep** prefixes can be preceded by the prefix **left**.

If used before **join** it specifies that a left join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the first table. If used before **keep**, it specifies that the second raw data table should be reduced to its common intersection with the first table, before being stored in Qlik Sense.

```
Left ( Join | Keep) [ (tablename) ] (loadstatement |selectstatement )
```

Mapping

The **mapping** prefix is used to create a mapping table that can be used to, for example, replacing field values and field names during script execution.

```
Mapping ( loadstatement | selectstatement )
```

Merge

The **Merge** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that the loaded table should be merged into another table. It also specifies that this statement should be run in a partial reload.

```
Merge [only] [(SequenceNoField [, SequenceNoVar])] On ListOfKeys [Concatenate  
[(TableName)]] (loadstatement | selectstatement)
```

NoConcatenate

The **NoConcatenate** prefix forces two loaded tables with identical field sets to be treated as two separate internal tables, when they would otherwise be automatically concatenated.

```
NoConcatenate( loadstatement | selectstatement )
```

Outer

The explicit **Join** prefix can be preceded by the prefix **Outer** to specify an outer join. In an outer join, all combinations between the two tables are generated. The resulting table will thus contain combinations of field values from the raw data tables where the linking field values are represented in one or both tables. The **Outer** keyword is optional and is the default join type used when a join prefix is not specified.

```
Outer Join [(tablename)] (loadstatement | selectstatement )
```

Partial reload

A full reload always starts by deleting all tables in the existing data model, and then runs the load script.

A *Partial reload* (page 97) will not do this. Instead it keeps all tables in the data model and then executes only **Load** and **Select** statements preceded by an **Add**, **Merge**, or **Replace** prefix. Other data tables are not affected by the command. The **only** argument denotes that the statement should be executed only during partial reloads, and should be disregarded during full reloads. The following table summarizes statement execution for partial and full reloads.

Replace

The **Replace** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that the loaded table should replace another table. It also specifies that this statement should be run in a partial reload. The **Replace** prefix can also be used in a **Map** statement.

```
Replace [only] [Concatenate[(tablename)]] (loadstatement | selectstatement)  
Replace [only] mapstatement
```

Right

The **Join** and **Keep** prefixes can be preceded by the prefix **right**.

2 Script statements and keywords

If used before **join** it specifies that a right join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the second table. If used before **keep**, it specifies that the first raw data table should be reduced to its common intersection with the second table, before being stored in Qlik Sense.

```
Right (Join | Keep) [(tablename)] (loadstatement | selectstatement )
```

Sample

The **sample** prefix to a **LOAD** or **SELECT** statement is used for loading a random sample of records from the data source.

```
Sample p ( loadstatement | selectstatement )
```

Semantic

Tables containing relations between records can be loaded through a **semantic** prefix. This can for example be self-references within a table, where one record points to another, such as parent, belongs to, or predecessor.

```
Semantic ( loadstatement | selectstatement )
```

Unless

The **unless** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be evaluated or not. It may be seen as a compact alternative to the full **if..end if** statement.

```
(Unless condition statement | exitstatement Unless condition )
```

When

The **when** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be executed or not. It may be seen as a compact alternative to the full **if..end if** statement.

```
( When condition statement | exitstatement when condition )
```

Add

The **Add** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that it should add records to another table. It also specifies that this statement should be run in a partial reload. The **Add** prefix can also be used in a **Map** statement.



For partial reload to work properly, the app must be opened with data before a partial reload is triggered.

Perform a partial reload using the **Reload** button. You can also use the Qlik Engine JSON API.

Syntax:

```
Add [only] [Concatenate[(tablename)]] (loadstatement | selectstatement)
```

```
Add [only] mapstatement
```

During a normal (non-partial) reload, the **Add LOAD** construction will work as a normal **LOAD** statement. Records will be generated and stored in a table.

2 Script statements and keywords

If the **Concatenate** prefix is used, or if there exists a table with the same set of fields, the records will be appended to the relevant existing table. Otherwise, the **Add LOAD** construction will create a new table.

A partial reload will do the same. The only difference is that the **Add LOAD** construction will never create a new table. There always exists a relevant table from the previous script execution to which the records should be appended.

No check for duplicates is performed. Therefore, a statement using the **Add** prefix will often include either a distinct qualifier or a where clause guarding duplicates.

The **Add Map...Using** statement causes mapping to take place also during partial script execution.

Arguments:

Arguments

Argument	Description
only	An optional qualifier denoting that the statement should be executed only during partial reloads. It should be disregarded during normal (non-partial) reloads.

Examples and results:

Example	Result
<pre>Tab1: LOAD Name, Number FROM Persons.csv; Add LOAD Name, Number FROM newPersons.csv;</pre>	<p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. Data from <i>NewPersons.csv</i> is then concatenated to the same Qlik Sense table.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> and appended to the Qlik Sense table Tab1. No check for duplicates is made.</p>
<pre>Tab1: SQL SELECT Name, Number FROM Persons.csv; Add LOAD Name, Number FROM NewPersons.csv where not exists(Name);</pre>	<p>A check for duplicates is made by means of looking if Name exists in the previously loaded table data.</p> <p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. Data from <i>NewPersons.csv</i> is then concatenated to the same Qlik Sense table.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> which is appended to the Qlik Sense table Tab1. A check for duplicates is made by means of seeing if Name exists in the previously loaded table data.</p>
<pre>Tab1: LOAD Name, Number FROM Persons.csv; Add Only LOAD Name, Number FROM NewPersons.csv where not exists(Name);</pre>	<p>During normal reload, data is loaded from <i>Persons.csv</i> and stored in the Qlik Sense table Tab1. The statement loading <i>NewPersons.csv</i> is disregarded.</p> <p>During partial reload, data is loaded from <i>NewPersons.csv</i> which is appended to the Qlik Sense table Tab1. A check for duplicates is made by means of seeing if Name exists in the previously loaded table data.</p>

Buffer

QVD files can be created and maintained automatically via the **buffer** prefix. This prefix can be used on most **LOAD** and **SELECT** statements in script. It indicates that QVD files are used to cache/buffer the result of the statement.

Syntax:

```
Buffer [(option [ , option])] ( loadstatement | selectstatement )
option ::= incremental | stale [after] amount [(days | hours)]
```

If no option is used, the QVD buffer created by the first execution of the script will be used indefinitely.

The buffer file is stored in the *Buffers* sub-folder, typically *C:\ProgramData\Qlik\Sense\Engine\Buffers* (server installation) or *C:\Users\{user}\Documents\Qlik\Sense\Buffers* (Qlik Sense Desktop).

The name of the QVD file is a calculated name, a 160-bit hexadecimal hash of the entire following **LOAD** or **SELECT** statement and other discriminating info. This means that the QVD buffer will be rendered invalid by any change in the following **LOAD** or **SELECT** statement.

QVD buffers will normally be removed when no longer referenced anywhere throughout a complete script execution in the app that created it or when the app that created it no longer exists.

Arguments:

Arguments

Argument	Description
incremental	<p>The incremental option enables the ability to read only part of an underlying file. Previous size of the file is stored in the XML header in the QVD file. This is particularly useful with log files. All records loaded at a previous occasion are read from the QVD file whereas the following new records are read from the original source and finally an updated QVD-file is created.</p> <p>The incremental option can only be used with LOAD statements and text files. Incremental load cannot be used where old data is changed or deleted.</p>
stale [after] amount [(days hours)]	<p>amount is a number specifying the time period. Decimals may be used. The unit is assumed to be days if omitted.</p> <p>The stale after option is typically used with DB sources where there is no simple timestamp on the original data. Instead you specify how old the QVD snapshot can be to be used. A stale after clause simply states a time period from the creation time of the QVD buffer after which it will no longer be considered valid. Before that time the QVD buffer will be used as source for data and after that the original data source will be used. The QVD buffer file will then automatically be updated and a new period starts.</p>

Limitations:

Numerous limitations exist, most notable is that there must be either a file **LOAD** or a **SELECT** statement at the core of any complex statement.

Example 1:

```
Buffer SELECT * from MyTable;
```

Example 2:

```
Buffer (stale after 7 days) SELECT * from MyTable;
```

Example 3:

```
Buffer (incremental) LOAD * from MyLog.log;
```

Concatenate

concatenate is a script load prefix that enables a dataset to be appended to an already existing in-memory table. It is often used to append different sets of transactional data to a single central fact table, or to build up common reference datasets of a specific type that originate from multiple sources. It is similar in functionality to a SQL UNION operator.

The resulting table from a concatenate operation will contain the original dataset with the new rows of data appended to the bottom of that table. The source and target tables may have different fields present. Where fields are different, the resulting table will be widened to represent the combined result of all fields present in both the source table and the target table.

Syntax:

```
Concatenate[ (tablename) ] ( loadstatement | selectstatement )
```

Arguments

Argument	Description
tablename	The name of an existing table. The named table will be the target of the Concatenate operation and any records of data loaded will be appended to that table. If the tablename parameter isn't used, the target table will be the last loaded table before this statement.
loadstatement/selectstatement	The loadstatement/selectstatement argument that follows the tablename argument will be concatenated to the specified table.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the **SET DateFormat** statement in your data load script. The default date formatting may

be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function example

Example	Result
Concatenate (Transactions) Load ... ;	The data loaded in the load statement below the concatenate prefix will be appended to the existing in-memory table named Transactions (assuming that a table named Transactions has been loaded prior to this point in the load script).

Example 1 – Appending multiple sets of data to a target table with Concatenate load prefix

Load script and results

Overview

In this example you will load two scripts in sequential order.

- The first load script contains an initial dataset with dates and amounts that is sent to a table named Transactions.
- The second load script contains:
 - A second dataset that is appended to the initial dataset by using the concatenate prefix. This dataset has an additional field, type, that is not in the initial dataset.
 - The concatenate prefix.

Open the data load editor and add the load script below to a new tab.

First load script

Transactions:

```
Load * Inline [
```

```
id, date, amount
3750, 08/30/2018, 23.56
3751, 09/07/2018, 556.31
3752, 09/16/2018, 5.75
3753, 09/22/2018, 125.00
3754, 09/22/2018, 484.21
3756, 09/22/2018, 59.18
3757, 09/23/2018, 177.42
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

First load script results table

id	date	amount
3750	08/30/2018	23.56
3751	09/07/2018	556.31
3752	09/16/2018	5.75
3753	09/22/2018	125.00
3754	09/22/2018	484.21
3756	09/22/2018	59.18
3757	09/23/2018	177.42

The table shows the initial dataset.

Second load script

Open the data load editor and add the load script below .

```
Concatenate(Technologies)
Load * Inline [
id, date, amount, type
3758, 10/01/2018, 164.27, Internal
3759, 10/03/2018, 384.00, External
3760, 10/06/2018, 25.82, Internal
3761, 10/09/2018, 312.00, Internal
3762, 10/15/2018, 4.56, Internal
3763, 10/16/2018, 90.24, Internal
3764, 10/18/2018, 19.32, External
];
```

Results

Load the data and go to the sheet. Create this field as a dimension:

- type

Second load script results table

id	date	amount	type
3750	08/30/2018	23.56	-
3751	09/07/2018	556.31	-
3752	09/16/2018	5.75	-
3753	09/22/2018	125.00	-
3754	09/22/2018	484.21	-
3756	09/22/2018	59.18	-
3757	09/23/2018	177.42	-
3758	10/01/2018	164.27	Internal
3759	10/03/2018	384.00	External
3760	10/06/2018	25.82	Internal
3761	10/09/2018	312.00	Internal
3762	10/15/2018	4.56	Internal
3763	10/16/2018	90.24	Internal
3764	10/18/2018	19.32	External

Note the null values in the type field for the first seven records loaded where type had not been defined.

Example 2 – Appending multiple sets of data to a target table using implicit concatenation

Load script and results

Overview

A typical use case for implicitly appending data is when you load several files of identically structured data and want to append them all to a target table.

For example, by using wildcards in file names with syntax such as:

```
myTable:  
Load * from [myFile_*.qvd] (qvd);
```

or in loops using constructs such as:

```
for each file in filelist('myFile_*.qvd')  
  
myTable:  
Load * from [$(file)] (qvd);  
  
next file
```



Implicit concatenation will take place between any two tables that are loaded with identically named fields, even if they aren't defined after one another in the script. This can lead to data being unintentionally appended to tables. If you don't want a secondary table with identical fields to be appended in this way, use the `NoConcatenate` load prefix. Renaming the table with an alternate table name tag is not sufficient to prevent implicit concatenation to occur. For more information, see [NoConcatenate \(page 87\)](#).

In this example you will load two scripts in sequential order.

- The first load script contains an initial dataset with four fields that is sent to a table named `Transactions`.
- The second load script contains a dataset with the same fields as the first dataset.

Open the data load editor and add the load script below to a new tab.

First load script

`Transactions:`

```
Load * Inline [
    id, date, amount, type
    3758, 10/01/2018, 164.27, Internal
    3759, 10/03/2018, 384.00, External
    3760, 10/06/2018, 25.82, Internal
    3761, 10/09/2018, 312.00, Internal
    3762, 10/15/2018, 4.56, Internal
    3763, 10/16/2018, 90.24, Internal
    3764, 10/18/2018, 19.32, External
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `amount`
- `type`

First load script results table

id	date	type	amount
3758	10/01/2018	Internal	164.27
3759	10/03/2018	External	384.00
3760	10/06/2018	Internal	25.82
3761	10/09/2018	Internal	312.00
3762	10/15/2018	Internal	4.56

id	date	type	amount
3763	10/16/2018	Internal	90.24
3764	10/18/2018	External	19.32

The table shows the initial dataset.

Second load script

Open the data load editor and add the load script below .

```
Load * Inline [
id, date, amount, type
3765, 11/03/2018, 129.40, Internal
3766, 11/05/2018, 638.50, External
];
```

Results

Load the data and go to the sheet.

Second load script results table

id	date	type	amount
3758	10/01/2018	Internal	164.27
3759	10/03/2018	External	384.00
3760	10/06/2018	Internal	25.82
3761	10/09/2018	Internal	312.00
3762	10/15/2018	Internal	4.56
3763	10/16/2018	Internal	90.24
3764	10/18/2018	External	19.32
3765	11/03/2018	Internal	129.40
3766	11/05/2018	External	638.50

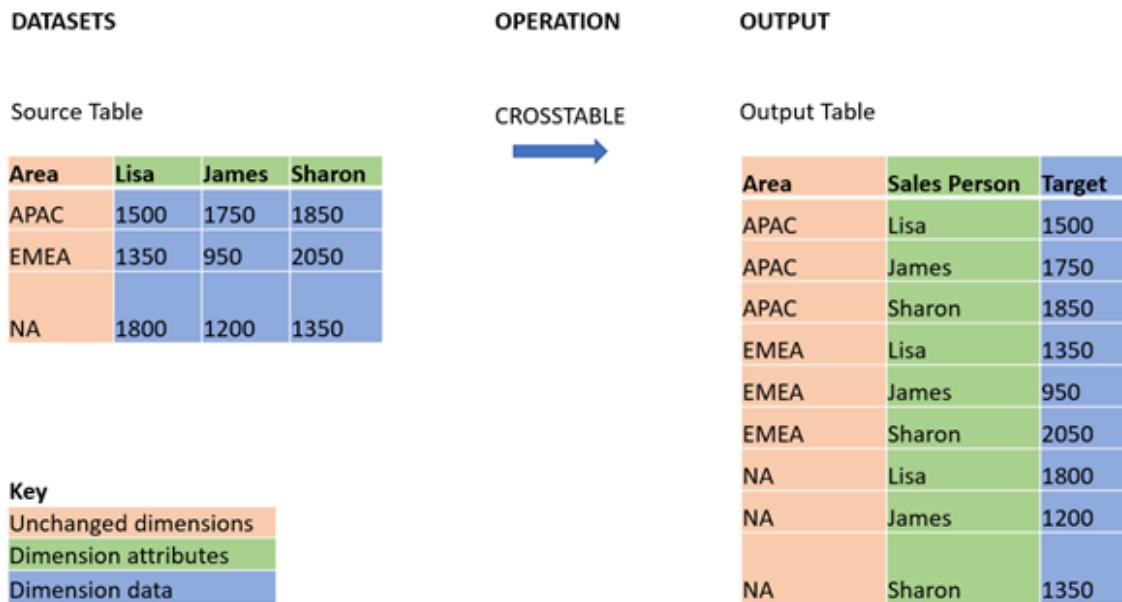
The second dataset was implicitly concatenated onto the initial dataset because they had identical fields.

Crosstable

The **crosstable** load prefix is used to transpose “cross table” or “pivot table” structured data. Data structured this way is commonly encountered when working with spreadsheet sources. The output and aim of the **crosstable** load prefix is to transpose such structures into a regular column-oriented table equivalent, as this structure is generally better suited for analysis in Qlik Sense.

2 Script statements and keywords

Example of data structured as a crosstable and its equivalent structure after a crosstable transformation



Syntax:

```
crosstable (attribute field name, data field name [ , n ] ) ( loadstatement |
selectstatement )
```

Arguments

Argument	Description
attribute field name	The desired output field name describing the horizontally oriented dimension that is to be transposed (the header row).
data field name	The desired output field name which describes the horizontally oriented data of the dimension that is to be transposed (the matrix of data values beneath the header row).
n	The number of qualifier fields, or unchanged dimensions, preceding the table to be transformed to generic form. The default value is 1.

This scripting function is related to the following functions:

Related functions

Function	Interaction
Generic (page 56)	A transformation load prefix which takes an entity-attribute-value structured data set and transforms it into a regular relational table structure, separating each attribute encountered into a new field or column of data.

Example 1 – Transforming pivoted sales data (simple)

Load scripts and results

Overview

Open the Data load editor and add the first load script below to a new tab.

The first load script contains a dataset to which the crosstable script prefix will be applied later, with the section applying crosstable commented out. This means that comment syntax was used to disable this section in the load script.

The second load script is the same as the first, but with the application of crosstable uncommented (enabled by removing the comment syntax). The scripts are shown this way to highlight the value of this scripting function in transforming data.

First load script (function not applied)

```
tmpData:  
//Crosstable (MonthText, Sales)  
Load * inline [  
Product, Jan 2021, Feb 2021, Mar 2021, Apr 2021, May 2021, Jun 2021  
A, 100, 98, 103, 63, 108, 82  
B, 284, 279, 297, 305, 294, 292  
C, 50, 53, 50, 54, 49, 51];  
  
//Final:  
//Load Product,  
//Date(Date#(MonthText,'MMM YYYY'), 'MMM YYYY') as Month,  
//Sales  
  
//Resident tmpData;  
  
//Drop Table tmpData;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Product
- Jan 2021
- Feb 2021
- Mar 2021
- Apr 2021
- May 2021
- Jun 2021

Results table

Product	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	Jun 2021
A	100	98	103	63	108	82
B	284	279	297	305	294	292
C	50	53	50	54	49	51

This script allows the creation of a crosstable with one column for each month and one row per product. In its current format, this data is not easy to analyze. It would be much better to have all numbers in one field and all months in another, in a three-column table. The next section explains how to do this transformation to the crosstable.

Second load script (function applied)

Uncomment the script by removing the //. The load script should look like this:

```
tmpData:
Crosstable (MonthText, Sales)
Load * inline [
Product, Jan 2021, Feb 2021, Mar 2021, Apr 2021, May 2021, Jun 2021
A, 100, 98, 103, 63, 108, 82
B, 284, 279, 297, 305, 294, 292
C, 50, 53, 50, 54, 49, 51];

Final:
Load Product,
Date(Date#(MonthText,'MMM YYYY'), 'MMM YYYY') as Month,
Sales

Resident tmpData;

Drop Table tmpData;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Product
- Month
- Sales

Results table

Product	Month	Sales
A	Jan 2021	100
A	Feb 2021	98
A	Mar 2021	103

Product	Month	Sales
A	Apr 2021	63
A	May 2021	108
A	Jun 2021	82
B	Jan 2021	284
B	Feb 2021	279
B	Mar 2021	297
B	Apr 2021	305
B	May 2021	294
B	Jun 2021	292
C	Jan 2021	50
C	Feb 2021	53
C	Mar 2021	50
C	Apr 2021	54
C	May 2021	49
C	Jun 2021	51

Once the script prefix has been applied, the crosstable is transformed into a straight table with one column for Month and another for sales. This improves the readability of the data.

Example 2 – Transforming pivoted sales target data into a vertical table structure (intermediate)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named Targets.
- The crosstable load prefix, which transposes the pivoted sales person names into a field of its own, labeled Sales Person.
- The associated sales target data, which is structured into a field called Target.

Load script

```
SalesTargets:  
CROSSTABLE([Sales Person],Target,1)
```

```
LOAD
*
INLINE [
Area, Lisa, James, Sharon
APAC, 1500, 1750, 1850
EMEA, 1350, 950, 2050
NA, 1800, 1200, 1350
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Area
- Sales Person

Add this measure:

```
=Sum(Target)
```

Results table

Area	Sales Person	=Sum(Target)
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

If you want to replicate the display of data as the pivoted input table, you can create an equivalent pivot table in a sheet.

Do the following:

1. Copy and paste the table you have just created into the sheet.
2. Drag the **Pivot table** chart object on top of the newly created table copy. Select **Convert**.
3. Click **✓ Done editing**.
4. Drag the **Sales Person** field from the vertical column shelf to the horizontal column shelf.

The following table shows the data in its initial table form, as it is displayed in Qlik Sense:

2 Script statements and keywords

Original results table, as shown in Qlik Sense

Area	Sales Person	=Sum(Target)
Totals	-	13800
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

The equivalent pivot table looks similar to the following, with the column for each sales person's name being contained within the larger row for Sales Person:

Equivalent pivot table with the Sales Person field pivoted horizontally

Area	James	Lisa	Sharon
APAC	1750	1500	1850
EMEA	950	1350	2050
NA	1350	1350	1350

Example of data displayed as a table and an equivalent pivot table with the Sales Person field pivoted horizontally

Area	Sales Person	Sum(Target)
Totals		13800
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

Area	Sales Person		
	James	Lisa	Sharon
APAC	1750	1500	1850
EMEA	950	1350	2050
NA	1200	1800	1350

Example 3 – Transforming pivoted sales and target data into a vertical table structure (advanced)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing sales and targets data, organized by area and month of the year. This is loaded into a table called `SalesAndTargets`.
- The `crosstable` load prefix. This is used to unpivot the `Month Year` dimension into a dedicated field, as well as to transpose the matrix of sales and target amounts into a dedicated field called `Amount`.
- A conversion of the `Month Year` field from text to a proper date, using the text-to-date conversion function `date#`. This date-converted `Month Year` field is joined back onto the `SalesAndTarget` table via a `Join` load prefix.

Load script

```
SalesAndTargets:  
CROSSTABLE(MonthYearAsText,Amount,2)  
LOAD  
    *  
INLINE [  
Area  Type   Jan-22  Feb-22  Mar-22  Apr-22  May-22  Jun-22  Jul-22  Aug-22  Sep-22  Oct-22  Nov-22  Dec-22  
APAC  Target  425     425     425     425     425     425     425     425     425     425     425     425     425  
APAC  Actual   435     434     397     404     458     447     413     458     385     421     448     425     397  
EMEA  Target  362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5   362.5  
EMEA  Actual  363.5   359.5   337.5   361.5   341.5   337.5   379.5   352.5   327.5   337.5   360.5   334.5  
NA    Target  375     375     375     375     375     375     375     375     375     375     375     375     375  
NA    Actual   378     415     363     356     403     343     401     365     393     340     360     375     375  
] (delimiter is '\t');  
  
tmp:  
LOAD DISTINCT MonthYearAsText,date#(MonthYearAsText,'MMM-YY') AS [Month Year]  
RESIDENT SalesAndTargets;  
  
JOIN (SalesAndTargets)  
LOAD * RESIDENT tmp;  
  
DROP TABLE tmp;  
DROP FIELD MonthYearAsText;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

2 Script statements and keywords

- Area
- Month Year

Create the following measure, with the label Actual:

```
=Sum({<Type={'Actual'}>} Amount)
```

Also create this measure, with the label Target:

```
=Sum({<Type={'Target'}>} Amount)
```

Results table (cropped)

Area	Month Year	Actual	Target
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

If you wish to replicate the display of data as the pivoted input table, you can create an equivalent pivot table in a sheet.

Do the following:

1. Copy and paste the table you have just created into the sheet.
2. Drag the **Pivot table** chart object on top of the newly created table copy. Select **Convert**.
3. Click **✓ Done editing**.
4. Drag the Month Year field from the vertical column shelf to the horizontal column shelf.
5. Drag the values item from the horizontal column shelf to the vertical column shelf.

The following table shows the data in its initial table form, as it is displayed in Qlik Sense:

2 Script statements and keywords

Original results table (cropped), as shown in Qlik Sense

Area	Month Year	Actual	Target
Totals	-	13812	13950
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

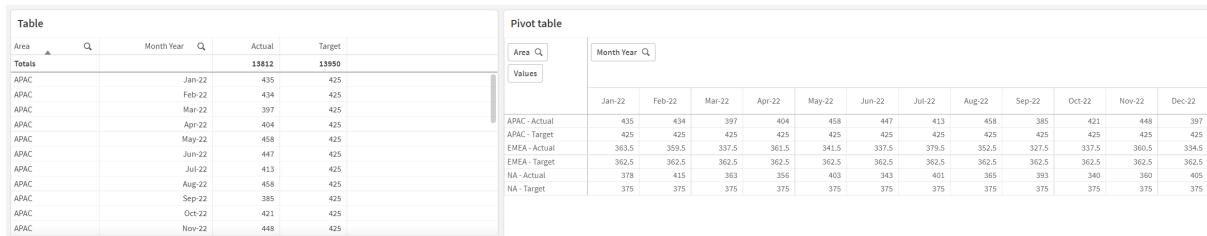
The equivalent pivot table looks similar to the following, with the column for each individual month of the year being contained within the larger row for Month Year:

Equivalent pivot table (cropped) with the Month Year field pivoted horizontally

Area (Value s)	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
APAC - Actual	435	434	397	404	458	447	413	458	385	421	448	397
APAC - Target	425	425	425	425	425	425	425	425	425	425	425	425
EMEA - Actual	363.5	359.5	337.5	361.5	341.5	337.5	379.5	352.5	327.5	337.5	360.5	334.5
EMEA - Target	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5
NA - Actual	378	415	363	356	403	343	401	365	393	340	360	405

Area (Value s)	Jan- 22	Feb- 22	Mar- 22	Apr- 22	May- 22	Jun- 22	Jul- 22	Aug- 22	Sep- 22	Oct- 22	Nov- 22	Dec- 22
NA - Target	375	375	375	375	375	375	375	375	375	375	375	375

Example of data displayed as a table and an equivalent pivot table with the Month Year field pivoted horizontally



The screenshot displays two views of the same dataset. On the left is a 'Table' view with columns for Area, Month Year, Actual, and Target. On the right is a 'Pivot table' view with a 'Values' section. The pivot table has 'Area' and 'Month Year' as dimensions, resulting in a grid where each row represents an area and each column represents a month from Jan-22 to Dec-22.

First

The **First** prefix to a LOAD or SELECT (SQL) statement is used for loading a set maximum number of records from a data source table. A typical use case for using the **First** prefix is when you want to retrieve a small subset of records from a large and/or slow data load step. As soon as the defined “n” number of records has been loaded, the load step terminates prematurely, and the rest of the script execution continues as normal.

Syntax:

```
First n ( loadstatement | selectstatement )
```

Arguments

Argument	Description
n	An arbitrary expression that evaluates to an integer indicating the maximum number of records to be read. n can also be enclosed in parentheses: (n).
loadstatement selectstatement	The load statement/select statement that follows the n argument will define the specified table that must be loaded with the set maximum number of records.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the SET DateFormat statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use

Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples	
Example	Result
FIRST 10 LOAD * from abc.csv;	This example will retrieve the first ten lines from an excel file.
FIRST (1) SQL SELECT * from Orders;	This example will retrieve the first selected line from the orders dataset.

Example – Load the first five rows

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates from the first two weeks of 2020.
- The First variable that instructs the application to only load the first five records.

Load script

```
Sales:  
FIRST 5  
LOAD  
*  
Inline [  
date,sales  
01/01/2020,6000  
01/02/2020,3000  
01/03/2020,6000  
01/04/2020,8000  
01/05/2020,5000  
01/06/2020,7000  
01/07/2020,3000  
01/08/2020,5000  
01/09/2020,9000  
01/10/2020,5000  
01/11/2020,7000  
01/12/2020,7000  
01/13/2020,7000  
01/14/2020,7000  
];
```

Results

Load the data and open a sheet. Create a new table and add date as a field and sum(sales) as a measure.

Results table

Date	sum(sales)
01/01/2020	6000
01/02/2020	3000
01/03/2020	6000
01/04/2020	8000
01/05/2020	5000

The script only loads the first five records of the sales table.

Generic

The **Generic** load prefix allows for conversion of entity–attribute–value modeled data (EAV) into a traditional, normalized relational table structure. EAV modeling is alternatively referred to as "generic data modeling" or "open schema".

Example of EAV modeled data and an equivalent denormalized relational table



Product ID	Attribute	Value
13	Status	Discontinued
13	Colour	Brown
20	Colour	White
13	Size	13-15
20	Size	16-18

Product ID	Status	Colour	Size
13	Discontinued	Brown	13-15
20		White	16-18

Example of EAV modeled data and an equivalent set of normalized relational tables



Product ID	Attribute	Value
13	Status	Discontinued
13	Colour	Brown
20	Colour	White
13	Size	13-15
20	Size	16-18

Product ID	Status
13	Discontinued

Product ID	Colour
13	Brown
20	White

Product ID	Size
13	13-15
20	16-18

2 Script statements and keywords

While it is technically possible to load and analyze EAV modeled data in Qlik, it is often easier to work with an equivalent traditional relational data structure.

Syntax:

```
Generic( loadstatement | selectstatement )
```

These topics may help you work with this function:

Related topics

Topic	Description
Crosstable (page 44)	The crosstable load prefix transforms data that is horizontally-oriented into vertically-oriented data. From a purely functional perspective, it performs the opposite transformation to the Generic load prefix, although the prefixes typically serve entirely different use cases.
Generic databases in Manage data	EAV structured data models are further described here.

Example 1 – Transforming EAV structured data with the Generic load prefix

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains a dataset which is loaded into a table named `Transactions`. The dataset includes a date field. The default `MonthNames` definition is used.

Load script

```
Products:  
Generic  
Load * inline [  
Product ID, Attribute, Value  
13, Status, Discontinued  
13, Color, Brown  
20, Color, White  
13, Size, 13-15  
20, Size, 16-18  
2, Status, Discontinued  
5, Color, Brown  
2, Color, White  
44, Color, Brown  
45, Size, 16-18  
45, Color, Brown  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: color.

Add this measure:

```
=Count([Product ID])
```

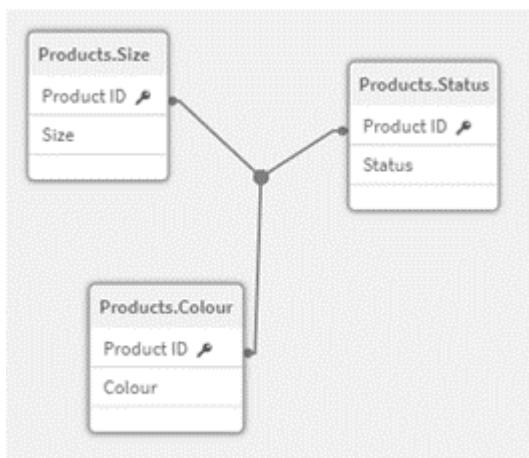
Now you can inspect the number of products by color.

Results table

Color	=Count([Product ID])
Brown	4
White	2

Note the shape of the data model, where each attribute has been broken out into a separate table named according to the original target table tag Product. Each table has the attribute as a suffix. One example of this is Product.color. The resulting Product Attribute output records are associated by the Product ID.

Data model viewer representation of the results



Resulting table of records: Products.Status

Product ID	Status
13	Discontinued
2	Discontinued

Resulting table of records: Products.Size

Product ID	Size
13	13-15
20	16-18
45	16-18

Resulting table of records: Products.Color

Product ID	Color
13	Brown
5	Brown
44	Brown
45	Brown
20	White
2	White

Example 2 – Analyzing EAV structured data without the Generic load prefix

Load script and chart expression

Overview

This example shows how to analyze EAV structured data in its original form.

Open the Data load editor and add the load script below to a new tab.

The load script contains a dataset which is loaded into a table named `Products` in an EAV structure.

In this example, we are still counting products by color attribute. In order to analyze data structured in this way, you will need to apply expression-level filtering of products carrying the Attribute value `color`.

Furthermore, individual attributes are not available to select as dimensions or fields, making it harder to determine how to build effective visualizations.

Load script

```
Products:
Load * Inline
[
Product ID, Attribute, value
13, Status, Discontinued
13, Color, Brown
20, Color, White
```

```
13, Size, 13-15
20, Size, 16-18
2, Status, Discontinued
5, Color, Brown
2, Color, white
44, Color, Brown
45, Size, 16-18
45, Color, Brown
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: value.

Create the following measure:

```
=Count({<Attribute={'Color'}>} [Product ID])
```

Now you can inspect the number of products by color.

Resulting table of records: Products.Status

Value	=Count({<Attribute={'Color'}>} [Product ID])
Brown	4
White	2

Example 3 – Denormalizing the resulting output tables from a Generic load (advanced)

Load script and chart expression

Overview

In this example, we show how the normalised data structure produced by the Generic load prefix can be denormalised back into a consolidated Product dimension table. This is an advanced modeling technique which can be employed as part of data model performance tuning.

Open the Data load editor and add the load script below to a new tab.

Load script

Products:

```
Generic
Load * inline [
Product ID, Attribute, value
13, Status, Discontinued
13, Color, Brown
20, Color, white
13, Size, 13-15
20, Size, 16-18
```

2 Script statements and keywords

```
2, Status, Discontinued  
5, Color, Brown  
2, Color, White  
44, Color, Brown  
45, Size, 16-18  
45, Color, Brown  
];  
  
RENAME TABLE Products.Color TO Products;  
  
OUTER JOIN (Products)  
LOAD * RESIDENT Products.Size;  
  
OUTER JOIN (Products)  
LOAD * RESIDENT Products.Status;  
DROP TABLES Products.Size,Products.Status;
```

Results

Open the Data model viewer and note the shape of the resulting data model. Only one denormalized table is present. It is a combination of the three intermediary output tables: Products.Size, Products.Status, and Products.Color.

Resulting
internal data
model

Products
Product ID
Status
Color
Size

Resulting table of records: Products

Product ID	Status	Color	Size
13	Discontinued	Brown	13-15
20	-	White	16-18
2	Discontinued	White	-
5	-	Brown	-
44	-	Brown	-
45	-	Brown	16-18

Load the data and open a sheet. Create a new table and add this field as a dimension: color.

Add this measure:

```
=Count([Product ID])
```

Results table	
Color	=Count([Product ID])
Brown	4
White	2

Hierarchy

The **hierarchy** prefix is used to transform a parent-child hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

The prefix creates an expanded nodes table, which normally has the same number of records as the input table, but in addition each level in the hierarchy is stored in a separate field. The path field can be used in a tree structure.

Syntax:

```
Hierarchy (NodeID, ParentID, NodeName, [ParentName, [PathSource, [PathName,  
[PathDelimiter, Depth]]]]) (loadstatement | selectstatement)
```

The input table must be an adjacent nodes table. Adjacent nodes tables are tables where each record corresponds to a node and has a field that contains a reference to the parent node. In such a table the node is stored on one record only but the node can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

The prefix creates an expanded nodes table, which normally has the same number of records as the input table, but in addition each level in the hierarchy is stored in a separate field. The path field can be used in a tree structure.

Usually the input table has exactly one record per node and in such a case the output table will contain the same number of records. However, sometimes there are nodes with multiple parents, i.e. one node is represented by several records in the input table. If so, the output table may have more records than the input table.

All nodes with a parent id not found in the node id column (including nodes with missing parent id) will be considered as roots. Also, only nodes with a connection to a root node - direct or indirect - will be loaded, thus avoiding circular references.

Additional fields containing the name of the parent node, the path of the node and the depth of the node can be created.

Arguments:

Arguments

Argument	Description
NodeID	The name of the field that contains the node id. This field must exist in the input table.
ParentID	The name of the field that contains the node id of the parent node. This field must exist in the input table.
NodeName	The name of the field that contains the name of the node. This field must exist in the input table.
ParentName	A string used to name the new ParentName field. If omitted, this field will not be created.
ParentSource	The name of the field that contains the name of the node used to build the node path. Optional parameter. If omitted, NodeName will be used.
PathName	A string used to name the new Path field, which contains the path from the root to the node. Optional parameter. If omitted, this field will not be created.
PathDelimiter	A string used as delimiter in the new Path field. Optional parameter. If omitted, '/' will be used.
Depth	A string used to name the new Depth field, which contains the depth of the node in the hierarchy. Optional parameter. If omitted, this field will not be created.

Example:

```
Hierarchy(NodeID, ParentID, NodeName, ParentName, NodeName, PathName, '\', Depth) LOAD *
inline [
NodeID, ParentID, NodeName
1, 4, London
2, 3, Munich
3, 5, Germany
4, 5, UK
5, , Europe
];
```

NodeID	ParentID	NodeName	NodeName	NodeName	NodeName	ParentName	PathName	Depth
1	4	London	Europe	UK	London	UK	Europe\UK\London	3
2	3	Munich	Europe	Germany	Munich	Germany	Europe\Germany\Munich	3
3	5	Germany	Europe	Germany	-	Europe	Europe\Germany	2
4	5	UK	Europe	UK	-	Europe	Europe\UK	2
5		Europe	Europe	-	-	-	Europe	1

HierarchyBelongsTo

This prefix is used to transform a parent-child hierarchy table to a table that is useful in a Qlik Sense data model. It can be put in front of a **LOAD** or a **SELECT** statement and will use the result of the loading statement as input for a table transformation.

The prefix creates a table containing all ancestor-child relations of the hierarchy. The ancestor fields can then be used to select entire trees in the hierarchy. The output table in most cases contains several records per node.

Syntax:

```
HierarchyBelongsTo (NodeID, ParentID, NodeName, AncestorID, AncestorName,  
[DepthDiff]) (loadstatement | selectstatement)
```

The input table must be an adjacent nodes table. Adjacent nodes tables are tables where each record corresponds to a node and has a field that contains a reference to the parent node. In such a table the node is stored on one record only but the node can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

The prefix creates a table containing all ancestor-child relations of the hierarchy. The ancestor fields can then be used to select entire trees in the hierarchy. The output table in most cases contains several records per node.

An additional field containing the depth difference of the nodes can be created.

Arguments:

Arguments

Argument	Description
NodeID	The name of the field that contains the node id. This field must exist in the input table.
ParentID	The name of the field that contains the node id of the parent node. This field must exist in the input table.
NodeName	The name of the field that contains the name of the node. This field must exist in the input table.
AncestorID	A string used to name the new ancestor id field, which contains the id of the ancestor node.
AncestorName	A string used to name the new ancestor field, which contains the name of the ancestor node.
DepthDiff	A string used to name the new DepthDiff field, which contains the depth of the node in the hierarchy relative the ancestor node. Optional parameter. If omitted, this field will not be created.

Example:

```
HierarchyBelongsTo (NodeID, AncestorID, NodeName, AncestorID, AncestorName, DepthDiff) LOAD *
inline [
NodeID, AncestorID, NodeName
1, 4, London
2, 3, Munich
3, 5, Germany
4, 5, UK
5, , Europe
];
```

Results

NodeID	AncestorID	NodeName	AncestorName	DepthDiff
1	1	London	London	0
1	4	London	UK	1
1	5	London	Europe	2
2	2	Munich	Munich	0
2	3	Munich	Germany	1
2	5	Munich	Europe	2
3	3	Germany	Germany	0
3	5	Germany	Europe	1
4	4	UK	UK	0
4	5	UK	Europe	1
5	5	Europe	Europe	0

Inner

The **join** and **keep** prefixes can be preceded by the prefix **inner**. If used before **join** it specifies that an inner join should be used. The resulting table will thus only contain combinations of field values from the raw data tables where the linking field values are represented in both tables. If used before **keep**, it specifies that both raw data tables should be reduced to their common intersection before being stored in Qlik Sense.

Syntax:

```
Inner ( Join | Keep) [ (tablename) ] (loadstatement | selectstatement )
```

Arguments:

Arguments	
Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Table1:

```
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];
```

Table2:

```
Inner Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

Result

Resulting table

Column1	Column2	Column3
A	B	C
1	aa	xx

Explanation

This example demonstrates the Inner Join output where only values present in both the first (left) and the second (right) tables are joined.

IntervalMatch

The **IntervalMatch** prefix is used to create a table matching discrete numeric values to one or more numeric intervals, and optionally matching the values of one or several additional keys.

Syntax:

```
IntervalMatch (matchfield) (loadstatement | selectstatement )
IntervalMatch (matchfield, keyfield1 [ , keyfield2, ... keyfield5 ] )
(loadstatement | selectstatement )
```

The **IntervalMatch** prefix must be placed before a **LOAD** or a **SELECT** statement that loads the intervals. The field containing the discrete data points (Time in the example below) and additional keys must already have been loaded into Qlik Sense before the statement with the **IntervalMatch** prefix. The prefix does not by itself read this field from the database table. The prefix transforms the loaded table of intervals and keys to a table that contains an additional column: the discrete numeric data points. It also expands the number of records so that the new table has one record per possible combination of discrete data point, interval and value of the key field(s).

The intervals may be overlapping and the discrete values will be linked to all matching intervals.

When the IntervalMatch prefix is extended with key fields, it is used to create a table matching discrete numeric values to one or more numeric intervals, while at the same time matching the values of one or several additional keys.

In order to avoid undefined interval limits being disregarded, it may be necessary to allow NULL values to map to other fields that constitute the lower or upper limits to the interval. This can be handled by the **NullAsValue** statement or by an explicit test that replaces NULL values with a numeric value well before or after any of the discrete numeric data points.

Arguments:

Arguments

Argument	Description
matchfield	The field containing the discrete numeric values to be linked to intervals.
keyfield	Fields that contain the additional attributes that are to be matched in the transformation.
loadstatement orselectstatement	Must result in a table, where the first field contains the lower limit of each interval, the second field contains the upper limit of each interval, and in the case of using key matching, the third and any subsequent fields contain the keyfield(s) present in the IntervalMatch statement. The intervals are always closed, i.e. the end points are included in the interval. Non-numeric limits render the interval to be disregarded (undefined).

Example 1:

In the two tables below, the first one lists a number of discrete events and the second one defines the start and end times for the production of different orders. By means of the **IntervalMatch** prefix it is possible to logically connect the two tables in order to find out e.g. which orders were affected by disturbances and which orders were processed by which shifts.

```
EventLog:
LOAD * Inline [
Time, Event, Comment
00:00, 0, Start of shift 1
01:18, 1, Line stop
02:23, 2, Line restart 50%
```

```

04:15, 3, Line speed 100%
08:00, 4, Start of shift 2
11:43, 5, End of production
];

OrderLog:
LOAD * INLINE [
Start, End, Order
01:00, 03:35, A
02:30, 07:58, B
03:04, 10:27, C
07:23, 11:43, D
];

//Link the field Time to the time intervals defined by the fields Start and End.
Inner Join IntervalMatch ( Time )
LOAD Start, End
Resident OrderLog;

```

The table **OrderLog** contains now an additional column: *Time*. The number of records is also expanded.

Table with additional column

Time	Start	End	Order
00:00	-	-	-
01:18	01:00	03:35	A
02:23	01:00	03:35	A
04:15	02:30	07:58	B
04:15	03:04	10:27	C
08:00	03:04	10:27	C
08:00	07:23	11:43	D
11:43	07:23	11:43	D

Example 2: (using keyfield)

Same example than above, adding *ProductionLine* as a key field.

```

EventLog:
LOAD * Inline [
Time, Event, Comment, ProductionLine
00:00, 0, Start of shift 1, P1
01:00, 0, Start of shift 1, P2
01:18, 1, Line stop, P1
02:23, 2, Line restart 50%, P1
04:15, 3, Line speed 100%, P1
08:00, 4, Start of shift 2, P1
09:00, 4, Start of shift 2, P2
11:43, 5, End of production, P1
11:43, 5, End of production, P2
];

```

```

OrderLog:
LOAD * INLINE [
Start, End, Order, ProductionLine
01:00, 03:35, A, P1
02:30, 07:58, B, P1
03:04, 10:27, C, P1
07:23, 11:43, D, P2
];
//Link the field Time to the time intervals defined by the fields Start and End and match the
values
// to the key ProductionLine.
Inner Join
IntervalMatch ( Time, ProductionLine )
LOAD Start, End, ProductionLine
Resident OrderLog;

```

A table box could now be created as below:

Tablebox example

ProductionLine	Time	Event	Comment	Order	Start	End
P1	00:00	0	Start of shift 1	-	-	-
P2	01:00	0	Start of shift 1	-	-	-
P1	01:18	1	Line stop	A	01:00	03:35
P1	02:23	2	Line restart 50%	A	01:00	03:35
P1	04:15	3	Line speed 100%	B	02:30	07:58
P1	04:15	3	Line speed 100%	C	03:04	10:27
P1	08:00	4	Start of shift 2	C	03:04	10:27
P2	09:00	4	Start of shift 2	D	07:23	11:43
P1	11:43	5	End of production	-	-	-
P2	11:43	5	End of production	D	07:23	11:43

Join

The **join** prefix joins the loaded table with an existing named table or the last previously created data table.

The effect of joining data is to extend the target table by an additional set of fields or attributes, namely ones not already present in the target table. Any common field names between the source data set and the target table are used to work out how to associate the new incoming records. This is commonly referred to as a “natural join”. A Qlik join operation can lead to the resulting target table having more or fewer records than it started with, depending on the uniqueness of the join association and the type of join employed.

There are four types of joins:

Left join

Left joins are the most common join type. For example, if you have a transaction data set and would like to combine it with a reference data set, you would typically use a `Left Join`. You would load the transaction table first, then load the reference data set while joining it via a `Left Join` prefix onto the already loaded transaction table. A `Left Join` would keep all transactions as-is and add on the supplementary reference data fields where a match is found.

Inner join

When you have two data sets where you only care about any results where there is a matching association, consider using an `Inner Join`. This will eliminate all records from both the source data loaded and the target table if no match is found. As a result, this may leave your target table with fewer records than before the join operation took place.

Outer join

When you need to keep both the target records and all of the incoming records, use an `outer Join`. Where no match is found, each set of records is still kept while the fields from the opposite side of the join will remain unpopulated (null).

If the `type` keyword is omitted, the default join type is an outer join.

Right join

This join type keeps all the records about to be loaded, while reducing the records in the table targeted by the join to only those records where there is an association match in the incoming records. This is a niche join type that is sometimes used as a means of trimming down an already pre-loaded table of records to a required subset.

2 Script statements and keywords

Example results sets from different types of join operations

DATASETS		OPERATION	OUTPUT	
Target Table		LEFT JOIN →	Trade ID	Asset Class
Trade ID	Asset Class		101533	Fixed Income LSE
101533	Fixed Income		606601	Commodities
606601	Commodities			
Incoming Dataset		INNER JOIN →	Trade ID	Asset Class
Trade ID	Exchange		101533	Fixed Income LSE
101533	LSE		606601	Commodities
79052	Hong Kong		79052	Hong Kong
Outer Join		OUTER JOIN →	Trade ID	Asset Class
Trade ID	Exchange		101533	Fixed Income LSE
101533	LSE		79052	Hong Kong
79052	Hong Kong			
Right Join		RIGHT JOIN →	Trade ID	Asset Class
Trade ID	Exchange		101533	Fixed Income LSE
79052	Hong Kong			



If there are no field names in common between the source and target of a join operation, the join will result in a cartesian product of all rows – this is called a “cross join”.

Example result set from a “cross join” operation

DATASETS		OPERATION	OUTPUT		
Target Table		JOIN (any type) →	Trade ID	Base Currency	Amount
Trade ID	Base Currency		101533	EUR	1250
101533	EUR		606601	EUR	1650
Incoming Dataset			Trade ID	Base Currency	Amount
Target Currency	Rate		101533	EUR	1250
USD	1.08		101533	EUR	1250
GBP	0.84		606601	EUR	1650
			606601	EUR	1650

Syntax:

```
[inner | outer | left | right ]Join [ (tablename) ]( loadstatement |
selectstatement )
```

Arguments	
Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

These topics may help you work with this function:

Related topics

Topic	Description
Combining tables with Join and Keep <i>in Manage data</i>	This topic provides further explanation of the concepts of “joining” and “keeping” data sets.
<i>Keep (page 79)</i>	The keep load prefix is similar to the <code>join</code> prefix, but it does not combine the source and target datasets. Instead, it trims each dataset according to the type of operation adopted (inner, outer, left, or right).

Example 1 - Left join: Enriching a target table with a reference data set

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing change records, which is loaded into a table named changes. It includes a Status ID key field.
- A second dataset representing change statuses, which is loaded and combined with the original change records by joining it with a left `join` load prefix.

This left join ensures that the change records remain intact while adding on status attributes where a match in the incoming status records is found based on a common Status ID.

Load script

Changes:

```
Load * inline [
Change ID      Status ID      Scheduled Start Date    Scheduled End Date      Business Impact
10030  4          19/01/2022    23/02/2022        None
10015  3          04/01/2022    15/02/2022        Low
10103  1          02/04/2022    29/05/2022        Medium
10185  2          23/06/2022    08/09/2022        None
10323  1          08/11/2022    26/11/2022        High
10326  2          11/11/2022    05/12/2022        None
```

```
10138 2      07/05/2022    03/08/2022    None
10031 3      20/01/2022    25/03/2022    Low
10040 1      29/01/2022    22/04/2022    None
10134 1      03/05/2022    08/07/2022    Low
10334 2      19/11/2022    06/02/2023    Low
10220 2      28/07/2022    06/09/2022    None
10264 1      10/09/2022    17/10/2022    Medium
10116 1      15/04/2022    24/04/2022    None
10187 2      25/06/2022    24/08/2022    Low
] (delimiter is '\t');
```

```
Status:
Left Join (Changes)
Load * inline [
Status ID      Status Sub Status
1      Open     Not Started
2      Open     Started
3      Closed   Completed
4      Closed   Cancelled
5      Closed   Obsolete
] (delimiter is '\t');
```

Results

Open the Data model viewer and note the shape of the data model. Only one denormalized table is present. It is a combination of all the original change records, with the matching status attributes joined onto each change record.

Resulting internal data
model

Changes
Change ID
Status ID
Scheduled Start Date
Scheduled End Date
Business Impact
Status
Sub Status

If you expand the preview window in the Data model viewer, you will see a portion of this full result set organized into a table:

2 Script statements and keywords

Preview of Changes table in the Data model viewer

Change ID	Status ID	Scheduled Start Date	Scheduled End Date	Business Impact	Status	Sub Status
10030	4	19/01/2022	23/02/2022	None	Closed	Cancelled
10031	3	20/01/2022	25/03/2022	Low	Closed	Completed
10015	3	04/01/2022	15/02/2022	Low	Closed	Completed
10103	1	02/04/2022	29/05/2022	Medium	Open	Not Started
10116	1	15/04/2022	24/04/2022	None	Open	Not Started
10134	1	03/05/2022	08/07/2022	Low	Open	Not Started
10264	1	10/09/2022	17/10/2022	Medium	Open	Not Started
10040	1	29/01/2022	22/04/2022	None	Open	Not Started
10323	1	08/11/2022	26/11/2022	High	Open	Not Started
10187	2	25/06/2022	24/08/2022	Low	Open	Started
10185	2	23/06/2022	08/09/2022	None	Open	Started
10220	2	28/07/2022	06/09/2022	None	Open	Started
10326	2	11/11/2022	05/12/2022	None	Open	Started
10138	2	07/05/2022	03/08/2022	None	Open	Started
10334	2	19/11/2022	06/02/2023	Low	Open	Started

Since the fifth row in the Status table (Status ID: '5', Status: 'Closed', Sub Status: 'Obsolete') does not correspond to any of the records in the Changes table, the information in this row does not appear in the result set above.

Return to the Data load editor. Load the data and open a sheet. Create a new table and add this field as a dimension: status.

Add this measure:

```
=Count([change ID])
```

Now you can inspect the number of Changes by Status.

Results table

Status	=Count([Change ID])
Open	12
Closed	3

Example 2 – Inner join: Combining matching records only

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing change records, which is loaded into a table named changes.
- A second dataset representing change records originating from the source system JIRA. This is loaded and combined with the original records by joining it with an `Inner Join` load prefix.

This `Inner Join` ensures that only the five change records which are found in both datasets are kept.

Load script

Changes:

```
Load * inline [
Change ID      Status ID      Scheduled Start Date      Scheduled End Date      Business Impact
10030  4          19/01/2022    23/02/2022      None
10015  3          04/01/2022    15/02/2022      Low
10103  1          02/04/2022    29/05/2022      Medium
10185  2          23/06/2022    08/09/2022      None
10323  1          08/11/2022    26/11/2022      High
10326  2          11/11/2022    05/12/2022      None
10138  2          07/05/2022    03/08/2022      None
10031  3          20/01/2022    25/03/2022      Low
10040  1          29/01/2022    22/04/2022      None
10134  1          03/05/2022    08/07/2022      Low
10334  2          19/11/2022    06/02/2023      Low
10220  2          28/07/2022    06/09/2022      None
10264  1          10/09/2022    17/10/2022      Medium
10116  1          15/04/2022    24/04/2022      None
10187  2          25/06/2022    24/08/2022      Low
] (delimiter is '\t');
```

JIRA_changes:

```
Inner Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
[
Ticket ID      Source System
10000  JIRA
10030  JIRA
10323  JIRA
10134  JIRA
10334  JIRA
10220  JIRA
```

```
20000 TFS
] (delimiter is '\t');
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Source System
- Change ID
- Business Impact

Now you can inspect the five resulting records. The resultant table from an `Inner Join` will only include records with matching information in both datasets.

Results table

Source System	Change ID	Business Impact
JIRA	10030	None
JIRA	10134	Low
JIRA	10220	None
JIRA	10323	High
JIRA	10334	Low

Example 3 – Outer join: Combining overlapping record sets

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing change records, which is loaded into a table named `changes`.
- A second dataset representing change records originating from the source system `JIRA`, which is loaded and combined with the original records by joining it with an `outer Join` load prefix.

This ensures that all the overlapping change records from both datasets are kept.

Load script

```
// 8 Change records

Changes:
Load * inline [
    Change ID      Status ID      Scheduled Start Date      Scheduled End Date      Business Impact
```

2 Script statements and keywords

```
10030 4      19/01/2022    23/02/2022    None
10015 3      04/01/2022    15/02/2022    Low
10138 2      07/05/2022    03/08/2022    None
10031 3      20/01/2022    25/03/2022    Low
10040 1      29/01/2022    22/04/2022    None
10134 1      03/05/2022    08/07/2022    Low
10334 2      19/11/2022    06/02/2023    Low
10220 2      28/07/2022    06/09/2022    None
] (delimiter is '\t');

// 6 Change records

JIRA_changes:
Outer Join (Changes)
Load
[Ticket ID] AS [Change ID],
[Source System]
inline
[
Ticket ID      Source System
10030  JIRA
10323  JIRA
10134  JIRA
10334  JIRA
10220  JIRA
10597  JIRA
] (delimiter is '\t');
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Source System
- Change ID
- Business Impact

Now you can inspect the 10 resulting records.

Results table

Source System	Change ID	Business Impact
JIRA	10030	None
JIRA	10134	Low
JIRA	10220	None
JIRA	10323	-
JIRA	10334	Low
JIRA	10597	-

Source System	Change ID	Business Impact
-	10015	Low
-	10031	Low
-	10040	None
-	10138	None

Example 4 – Right join: Trimming down a target table by a secondary master dataset

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing change records, which is loaded into a table named Changes.
- A second dataset representing change records originating from the source system Teamwork. This is loaded and combined with the original records by joining it with a `Right Join` load prefix.

This ensures that only Teamwork change records are kept, while not losing any Teamwork records if the target table does not have a matching change ID.

Load script

Changes:

```
Load * inline [
Change ID      Status ID      Scheduled Start Date      Scheduled End Date      Business Impact
10030 4          19/01/2022    23/02/2022      None
10015 3          04/01/2022    15/02/2022      Low
10103 1          02/04/2022    29/05/2022      Medium
10185 2          23/06/2022    08/09/2022      None
10323 1          08/11/2022    26/11/2022      High
10326 2          11/11/2022    05/12/2022      None
10138 2          07/05/2022    03/08/2022      None
10031 3          20/01/2022    25/03/2022      Low
10040 1          29/01/2022    22/04/2022      None
10134 1          03/05/2022    08/07/2022      Low
10334 2          19/11/2022    06/02/2023      Low
10220 2          28/07/2022    06/09/2022      None
10264 1          10/09/2022    17/10/2022      Medium
10116 1          15/04/2022    24/04/2022      None
10187 2          25/06/2022    24/08/2022      Low
] (delimiter is '\t');
```

Teamwork_changes:

```
Right Join (Changes)
Load
```

```
[Ticket ID] AS [Change ID],  
[Source System]  
inline  
[  
Ticket ID      Source System  
10040  Teamwork  
10015  Teamwork  
10103  Teamwork  
10031  Teamwork  
50231  Teamwork  
] (delimiter is '\t');
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Source System
- Change ID
- Business Impact

Now you can inspect the five resulting records.

Results table

Source System	Change ID	Business Impact
Teamwork	10015	Low
Teamwork	10031	Low
Teamwork	10040	None
Teamwork	10103	Medium
Teamwork	50231	-

Keep

The **keep** prefix is similar to the **join** prefix. Just as the **join** prefix, it compares the loaded table with an existing named table or the last previously created data table, but instead of joining the loaded table with an existing table, it has the effect of reducing one or both of the two tables before they are stored in Qlik Sense, based on the intersection of table data. The comparison made is equivalent to a natural join made over all the common fields, i.e. the same way as in a corresponding join. However, the two tables are not joined and will be kept in Qlik Sense as two separately named tables.

Syntax:

```
(inner | left | right) keep [(tablename ) ]( loadstatement | selectstatement )
```

The **keep** prefix must be preceded by one of the prefixes **inner**, **left** or **right**.

The explicit **join** prefix in Qlik Sense script language performs a full join of the two tables. The result is one table. In many cases such joins will result in very large tables. One of the main features of Qlik Sense is its ability to make associations between multiple tables instead of joining them, which greatly reduces memory usage, increases processing speed and offers enormous flexibility. Explicit joins should therefore generally be avoided in Qlik Sense scripts. The keep functionality was designed to reduce the number of cases where explicit joins needs to be used.

Arguments:

Arguments

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example:

```
Inner Keep LOAD * from abc.csv;
Left Keep SELECT * from table1;
tab1:
LOAD * from file1.csv;
tab2:
LOAD * from file2.csv;
...
Left Keep (tab1) LOAD * from file3.csv;
```

Left

The **Join** and **Keep** prefixes can be preceded by the prefix **left**.

If used before **join** it specifies that a left join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the first table. If used before **keep**, it specifies that the second raw data table should be reduced to its common intersection with the first table, before being stored in Qlik Sense.



Were you looking for the string function by the same name? See: [Left \(page 1417\)](#)

Syntax:

```
Left ( Join | Keep ) [ (tablename) ] (loadstatement | selectstatement)
```

Arguments:

Arguments

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Table1:

```
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];
```

Table2:

```
Left Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

Result

Resulting table

Column1	Column2	Column3
A	B	C
1	aa	xx
2	cc	-
3	ee	-

Explanation

This example demonstrates the Left Join output where only values present in the first (left) table are joined.

Mapping

The **mapping** prefix is used to create a mapping table that can be used to, for example, replacing field values and field names during script execution.

Syntax:

```
Mapping( loadstatement | selectstatement )
```

The **mapping** prefix can be put in front of a **LOAD** or a **SELECT** statement and will store the result of the loading statement as a mapping table. Mapping provides an efficient way to substituting field values during script execution, e.g. replacing US, U.S. or America with USA. A mapping table consists of two columns, the first containing comparison values and the second containing the desired mapping values. Mapping tables are stored temporarily in memory and dropped automatically after script execution.

2 Script statements and keywords

The content of the mapping table can be accessed using e.g. the **Map ... Using** statement, the **Rename Field** statement, the **Applymap()** function or the **Mapsubstring()** function.

Example:

In this example we load a list of salespersons with a country code representing their country of residence. We use a table mapping a country code to a country to replace the country code with the country name. Only three countries are defined in the mapping table, other country codes are mapped to 'Rest of the world'.

```
// Load mapping table of country codes:  
map1:  
mapping LOAD *  
Inline [  
CCode, Country  
Sw, Sweden  
Dk, Denmark  
No, Norway  
]  
// Load list of salesmen, mapping country code to country  
// If the country code is not in the mapping table, put Rest of the world  
Salespersons:  
LOAD *,  
ApplyMap('map1', CCode,'Rest of the world') As Country  
Inline [  
CCode, Salesperson  
Sw, John  
Sw, Mary  
Sw, Per  
Dk, Preben  
Dk, olle  
No, ole  
Sf, Risttu] ;  
// We don't need the ccode anymore  
Drop Field 'CCode';
```

The resulting table looks like this:

Mapping table

Salesperson	Country
John	Sweden
Mary	Sweden
Per	Sweden
Preben	Denmark
olle	Denmark
Ole	Norway
Risttu	Rest of the world

Merge

The **Merge** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that the loaded table should be merged into another table. It also specifies that this statement should be run in a partial reload.

The typical use case is when you load a change log and want to use this to apply inserts, updates, and deletes to an existing table.



For partial reload to work properly, the app must be opened with data before a partial reload is triggered.

Perform a partial reload using the **Reload** button. You can also use the Qlik Engine JSON API.

Syntax:

```
Merge [only] [(SequenceNoField [, SequenceNoVar])] On ListOfKeys [Concatenate [(TableName)]] (loadstatement | selectstatement)
```

Arguments:

Arguments	
Argument	Description
only	An optional qualifier denoting that the statement should be executed only during partial reloads. The statement is disregarded during normal (non-partial) reloads.
SequenceNoField	The name of the field containing a timestamp or a sequence number that defines the order of the operations.
SequenceNoVar	The name of the variable that gets assigned the maximum value for SequenceNoField of the table being merged.
ListOfKeys	A comma separated list of field names specifying the primary key.
Operation	The first field of the load statement must contain the operation as a text string: 'Insert', 'Update', or 'Delete'. 'i', 'u' and 'd' are also accepted.

General functionality

During a normal (non-partial) reload, the **Merge LOAD** construction works as a normal **Load** statement but with the additional functionality of removing older obsolete records and records marked for deletion. The first field of the **Load** statement must hold information about the operation: Insert, Update, or Delete.

For each loaded record, the record identifier is compared with previously loaded records, and only the latest record (according to the sequence number) will be kept. If the latest record is marked with Delete, none will be kept.

Target table

Which table to modify is determined by the set of fields. If a table with the same set of fields (except the first field; the operation) already exists, this will be the relevant table to modify. Alternatively, a **Concatenate** prefix can be used to specify the table. If the target table is not determined, the result of the **Merge LOAD** construction is stored in a new table.

If the Concatenate prefix is used, the resulting table has a set of fields corresponding to the union of the existing table and the input to the merge. Hence, the target table may get more fields than the change log that is used as input to the merge.

A partial reload does the same as a full reload. One difference is that a partial reload rarely creates a new table. Unless you have used the **Only** clause, a target table with the same set of fields from the previous script execution always exists.

Sequence number

If the loaded change log is an accumulated log, that is, it contains changes that already have been loaded, the parameter SequenceNoVar can be used in a **Where** clause to limit the amount of input data. The **Merge LOAD** could then be made to only load records where the field SequenceNoField is greater than SequenceNoVar. Upon completion, the **Merge LOAD** assigns a new value to the SequenceNoVar with the maximum value seen in the SequenceNoField field.

Operations

The **Merge LOAD** can have fewer fields than the target table. The different operations treat missing fields differently:

Insert: Fields missing in the **Merge LOAD**, but existing in the target table, get a NULL in the target table.

Delete: Missing fields do not affect the result. The relevant records are deleted anyway.

Update: Fields listed in the **Merge LOAD** are updated in the target table. Missing fields are not changed. This means that the two following statements are not identical:

- Merge on Key Concatenate Load 'U' as Operation, Key, F1, Null() as F2 From ...;
- Merge on Key Concatenate Load 'U' as Operation, Key, F1 From ...;

The first statement updates the listed records and changes F2 to NULL. The second does not change F2, but instead, leaves the values in the target table.

Examples

Example 1: Simple merge with specified table

In this example, an inline table named Persons is loaded with three rows. **Merge** then changes the table as follows:

- Adds the row, *Mary, 4* .
- Deletes the row, *Steven, 3*.
- Assigns the number 5 to *Jake* .

2 Script statements and keywords

The *LastChangeDate* variable is set to the maximum value in the *ChangeDate* column after **Merge** is executed.

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
Set DateFormat='D/M/YYYY';
Persons:
Load * inline [
Name, Number
Jake, 3
Jill, 2
Steven, 3
];

Merge (ChangeDate, LastChangeDate) on Name Concatenate(Persons)
LOAD * inline [
Operation, ChangeDate,     Name,      Number
Insert,    1/1/2021,       Mary,      4
Delete,    1/1/2021,       Steven,
Update,    2/1/2021,       Jake,      5
];
```

Result

Prior to the **Merge Load**, the resulting table appears as follows:

Resulting table

Name	Number
Jake	3
Jill	2
Steven	3

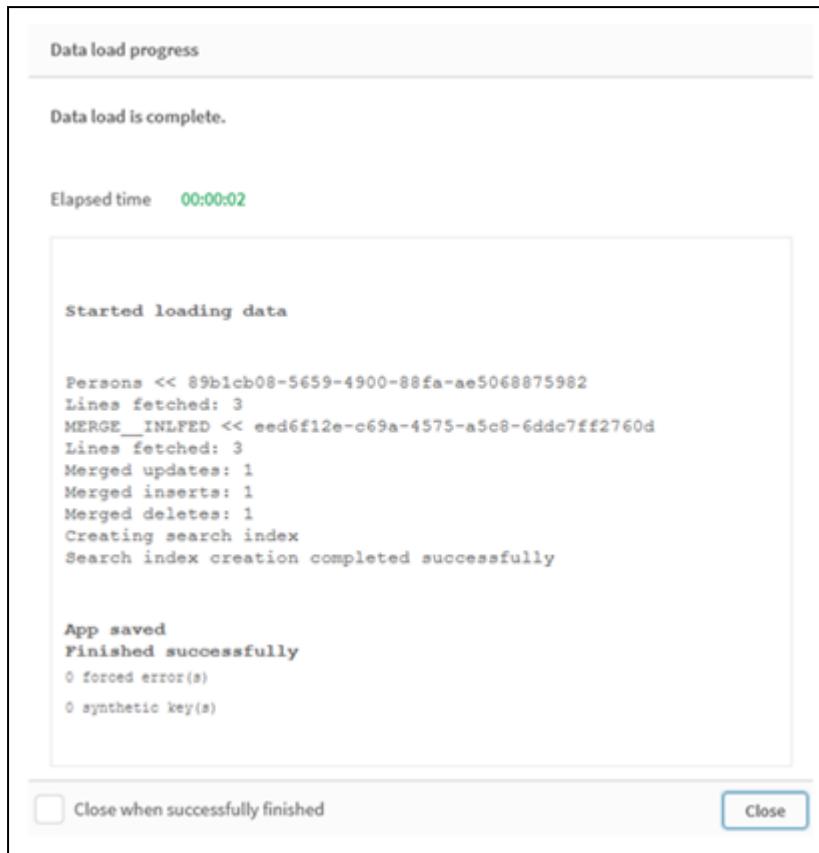
Following the **Merge Load**, the table appears as follows:

Resulting table

ChangeDate	Name	Number
2/1/2021	Jake	5
-	Jill	2
1/1/2021	Mary	4

When the data is loaded, the **Data load progress** dialog box shows the operations that are performed:

Data load progress dialog box



Example 2: Data load script with missing fields

In this example, the same data as above is loaded, but now with an ID for each person.

Merge changes the table as follows:

- Adds the row, *Mary*, 4.
- Deletes the row, *Steven*, 3.
- Assigns the number 5 to *Jake*.
- Assigns the number 6 to *Jill*.

Load script

Here we use two **Merge Load** statements, one for ‘Insert’ and ‘Delete’, and a second one for the ‘Update’.

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
Set DateFormat='D/M/YYYY';
Persons:
Load * Inline [
PersonID, Name, Number
1, Jake, 3
2, Jill, 2
3, Steven, 3
];
```

```
Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons)
Load * Inline [
Operation, ChangeDate, PersonID, Name, Number
Insert, 1/1/2021, 4, Mary, 4
Delete, 1/1/2021, 3, Steven,
];
```

```
Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons)
Load * Inline [
Operation, ChangeDate, PersonID, Number
Update, 2/1/2021, 1, 5
Update, 3/1/2021, 2, 6
];
```

Result

Following the **Merge Load** statements, the table appears as follows:

Resulting table

PersonID	ChangeDate	Name	Number
1	2/1/2021	Jake	5
2	3/1/2021	Jill	6
4	1/1/2021	Mary	4

Note that the second **Merge** statement does not include the field **Name**, and as a consequence, the names have not been changed.

Example 3: Data load script - Partial reload using a Where-clause with ChangeDate

In the following example, the **Only** argument specifies that the **Merge** command is only executed during a partial reload. Updates are filtered based on the previously captured LastChangeDate. After **Merge** is finished, LastChangeDate variable is assigned the maximum value of the ChangeDate column processed during the merge.

Load script

```
Merge Only (ChangeDate, LastChangeDate) on Name Concatenate(Persons)
LOAD Operation, ChangeDate, Name, Number
from [lib://ChangeFilesFolder/BulkChangesInPersonsTable.csv] (txt)
where ChangeDate >= $(LastChangeDate);
```

NoConcatenate

The **NoConcatenate** prefix forces two loaded tables with identical field sets to be treated as two separate internal tables, when they would otherwise be automatically concatenated.

Syntax:

```
NoConcatenate( loadstatement | selectstatement )
```

By default, if a table is loaded that contains an identical number of fields and matching field names to a table loaded earlier in the script, Qlik Sense will auto concatenate these two tables. This will happen even if the second table is named differently.

However, if the script prefix NoConcatenate is included before the load statement or select statement of the second table, then these two tables will be loaded separately.

A typical use case for NoConcatenate is when you may need to create a temporary copy of a table to perform some temporary transformations on that copy, while retaining a copy of the original data. NoConcatenate ensures that you can make that copy without implicitly adding it back onto the source table.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function example

Example	Result
<code>Source: LOAD A,B from file1.csv; CopyOfSource: NoConcatenate LOAD A,B resident Source;</code>	A table with A and B as measures is loaded. A second table with the same fields is loaded separately by using the NoConcatenate variable.

Example 1 – Implicit concatenation

Load script and results

Overview

In this example, you will add two load scripts in sequential order.

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- An initial dataset with dates and amounts that is sent to a table named `Transactions`.

First load script

`Transactions:`

```
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
```

```
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

First results table

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

Second load script

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A second dataset with identical fields is sent to a table named sales.

```
Sales:
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];
```

Results

Load the data and go to the table.

Second results table

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

When the script runs, the `sales` table is implicitly concatenated onto the existing `Transactions` table due to the two datasets sharing an identical number of fields, with identical field names. This happens despite the second table name tag attempting to name the result set ‘`Sales`’.

You can see that the Sales dataset is implicitly concatenated by looking at the **Data load progress** log.

2 Script statements and keywords

Data load progress log showing Transactions data being implicitly concatenated.

Data load progress

Data load is complete.

Elapsed time 00:00:01

```
Started loading data

Transactions << a4c3e539-0aa6-48f8-9e46-70238963eeb6
Lines fetched: 7
Transactions << 0213f0e3-a623-4820-8a86-b43faacf2395
Lines fetched: 14
Creating search index
Search index creation completed successfully

App saved
Finished successfully
0 forced error(s)
0 synthetic key(s)
```

Close when successfully finished Close

Example 2 – Use case scenario

Load script and results

Overview

In this use case scenario you have:

- A transactions dataset with:
 - id
 - date
 - amount (in GBP)
- A currency table with:
 - Conversion rates for USD to GBP
- A second transactions dataset with:
 - id

- date
- amount (in USD)

You will load five scripts in sequential order.

- The first load script contains an initial dataset with dates and amounts in GBP that is sent to a table named `Transactions`.
- The second load script contains:
 - A second dataset with dates and amounts in USD that is sent to a table named `Transactions_in_USD`.
 - The `noconcatenate` prefix which is placed before the load statement of the `Transactions_in_USD` dataset to prevent implicit concatenation.
- The third load script contains the `join` prefix which will be used to create a currency exchange rate between GBP and USD in the `Transactions_in_USD` table.
- The fourth load script contains the `concatenate` prefix which will add the `Transactions_in_USD` to the initial `Transactions` table.
- The fifth load script contains the `drop table` statement which will remove the `Transactions_in_USD` table once its data has been concatenated to the `Transactions` table.

First load script

`Transactions:`

```
Load * Inline [
id, date, amount
1, 12/30/2018, 23.56
2, 12/07/2018, 556.31
3, 12/16/2018, 5.75
4, 12/22/2018, 125.00
5, 12/22/2018, 484.21
6, 12/22/2018, 59.18
7, 12/23/2018, 177.42
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `amount`

First load script results

id	date	amount
1	12/30/2018	23.56
2	12/07/2018	556.31

id	date	amount
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42

The table shows the initial dataset with amounts in GBP.

Second load script

Transactions_in_USD:

NoConcatenate

```
Load * Inline [
    id, date, amount
    8, 01/01/2019, 164.27
    9, 01/03/2019, 384.00
    10, 01/06/2019, 25.82
    11, 01/09/2019, 312.00
    12, 01/15/2019, 4.56
    13, 01/16/2019, 90.24
    14, 01/18/2019, 19.32
];
```

Results

Load the data and go to the table.

Second load script results

id	date	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	164.27
9	01/03/2019	384.00
10	01/06/2019	25.82
11	01/09/2019	312.00

id	date	amount
12	01/15/2019	4.56
13	01/16/2019	90.24
14	01/18/2019	19.32

You will see that the second dataset from the `Transactions_in_USD` table has been added.

Third load script

This load script joins a currency exchange rate from USD to GBP to the `Transactions_in_USD` table.

```
Join (Transactions_in_USD)
Load * Inline [
rate
0.7
];
```

Results

Load the data and go to the Data model viewer. Select the `Transactions_in_USD` table and you will see that every existing record has a 'rate' field value of 0.7.

Fourth load script

Using resident load, this load script will concatenate the `Transactions_in_USD` table to the `Transactions` table after converting the amounts into USD.

```
Concatenate (Transactions)
LOAD
id,
date,
amount * rate as amount
Resident Transactions_in_USD;
```

Results

Load the data and go to the table. You will see new entries with amounts in GBP from lines eight to fourteen.

Fourth load script results

id	date	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21

id	date	amount
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	114.989
8	01/01/2019	164.27
9	01/03/2019	268.80
9	01/03/2019	384.00
10	01/06/2019	18.074
10	01/06/2019	25.82
11	01/09/2019	218.40
11	01/09/2019	312.00
12	01/15/2019	3.192
12	01/15/2019	4.56
13	01/16/2019	63.168
13	01/16/2019	90.24
14	01/18/2019	13.524
14	01/18/2019	19.32

Fifth load script

This load script will drop the duplicate entries from the fourth load script results table, leaving only entries with amounts in GBP.

```
drop tables Transactions_in_USD;
```

Results

Load the data and go to the table.

Fifth load script results

id	date	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18

id	date	amount
7	12/23/2018	177.42
8	01/01/2019	114.989
9	01/03/2019	268.80
10	01/06/2019	18.074
11	01/09/2019	218.40
12	01/15/2019	3.192
13	01/16/2019	63.168
14	01/18/2019	13.524

After loading the fifth load script, the results table shows all fourteen transactions that existed in both transaction datasets; however, transactions 8-14 have had their amounts converted to GBP.

If we remove the `NoConcatenate` prefix that was used before the `Transactions_in_USD` in the second load script, the script will fail with the error: "Table 'Transactions_in_usd' not found". This is because the `Transactions_in_USD` table would have been auto concatenated onto the original `Transactions` table.

Only

The **Only** script keyword is used as an aggregation function, or as part of the syntax in partial reload prefixes **Add**, **Replace**, and **Merge**.

Outer

The explicit **Join** prefix can be preceded by the prefix **Outer** to specify an outer join. In an outer join, all combinations between the two tables are generated. The resulting table will thus contain combinations of field values from the raw data tables where the linking field values are represented in one or both tables. The **Outer** keyword is optional and is the default join type used when a join prefix is not specified.

Syntax:

```
Outer Join [ (tablename) ] (loadstatement |selectstatement )
```

Arguments:

Arguments

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Table1:

```
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];
```

Table2:

```
Outer Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

Resulting table

Column1	Column2	Column3
A	B	C
1	aa	xx
2	cc	-
3	ee	-
4	-	yy

Explanation

In this example, the two tables, Table1 and Table2, are merged into a single table labeled Table1. In cases like this, the **outer** prefix is often used to join several tables into a single table to perform aggregations over the values of a single table.

Partial reload

A full reload always starts by deleting all tables in the existing data model, and then runs the load script.

A partial reload will not do this. Instead it keeps all tables in the data model and then executes only **Load** and **Select** statements preceded by an **Add**, **Merge**, or **Replace** prefix. Other data tables are not affected by the command. The **only** argument denotes that the statement should be executed only during partial reloads, and should be disregarded during full reloads. The following table summarizes statement execution for partial and full reloads.

Statement	Full reload	Partial reload
Load ...	Statement will run	Statement will not run
Add/Replace/Merge Load ...	Statement will run	Statement will run
Add/Replace/Merge Only Load ...	Statement will not run	Statement will run

Partial reloads have several benefits compared to full reloads:

- Faster, because only data recently changed needs to be loaded. With large data sets the difference is significant.
- Less memory is consumed, because less data is loaded.
- More reliable, because queries to source data run faster, reducing the risk of network problems.



For partial reload to work properly, the app must be opened with data before a partial reload is triggered.

Perform a partial reload using the **Reload** button. You can also use the Qlik Engine JSON API.

Limitations

A partial reload will fail if there are commands with references to tables that existed during full reload, but not during partial reload.

Example

Example commands

```
LEFT JOIN(<Table_removed_after_full_reload>)
CONCATENATE(<Table_removed_after_full_reload>)
```

Where <Table_removed_after_full_reload> is a table that existed in full reload, but not in partial reload.

Workaround

As a workaround you can surround the command with following if-statement:

```
IF NOT IsPartialReload() THEN ... ENDIF.
```

A partial reload can remove values from the data. However, this will not be reflected in the list of distinct values, which is a table maintained internally. So, after a partial reload, the list will contain all distinct values that have existed in the field since the last full reload, which may be more than what currently exists after the partial reload. This affects the output of the FieldValueCount() and the FieldValue() functions. The FieldValueCount() could potentially return a number greater than the current number of field values.

Example

Example 1

Load script

Add the example script to your app and do a partial reload. To see the result, add the fields listed in the results column to a sheet in your app.

T1:

```
Add only Load distinct recno()+10 as Num autogenerate 10;
```

Result

Resulting table

Num	Count(Num)
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1

Explanation

The statement is only executed during a partial reload. If the "distinct" prefix is omitted, the count of the **Num** field will increase with each subsequent partial reload.

Example 2

Load script

Add the example script to your app. Do a full reload and view the result. Next, do a partial reload and view the result. To see the results, add the fields listed in the results column to a sheet in your app.

T1:

```
Load recno() as ID, recno() as value autogenerate 10;
```

T1:

```
Replace only Load recno() as ID, repeat(recno(),3) as value autogenerate 10;
```

Result

Output table after full reload

ID	Value
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Output table after partial reload

ID	Value
1	111
2	222
3	333
4	444
5	555
6	666
7	777
8	888
9	999
10	101010

Explanation

The first table is loaded during a full reload and the second table simply replaces the first table during a partial reload.

Replace

The **Replace** script keyword is used as a string function, or as a prefix in partial reload.

Replace

The **Replace** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that the loaded table should replace another table. It also specifies that this statement should be run in a partial reload. The **Replace** prefix can also be used in a **Map** statement.



For partial reload to work properly, the app must be opened with data before a partial reload is triggered.

Perform a partial reload using the **Reload** button. You can also use the Qlik Engine JSON API.

Syntax:

```
Replace [only] [Concatenate[(tablename)]] (loadstatement | selectstatement)
```

```
Replace [only] mapstatement
```

During a normal (non-partial) reload, the **Replace LOAD** construction will work as a normal **LOAD** statement but be preceded by a **Drop Table**. First the old table will be dropped, then records will be generated and stored as a new table.

If the **Concatenate** prefix is used, or if there exists a table with the same set of fields, this will be the relevant table to drop. Otherwise, there is no table to drop and the **Replace LOAD** construction will be identical to a normal **LOAD**.

A partial reload will do the same. The only difference is that there is always a table from the previous script execution to drop. The **Replace LOAD** construction will always first drop the old table, then create a new one.

The **Replace Map...Using** statement causes mapping to take place also during partial script execution.

Arguments:

Arguments

Argument	Description
only	An optional qualifier denoting that the statement should be executed only during partial reloads. It should be disregarded during normal (non-partial) reloads.

Examples and results:

Example	Result
Tab1: Replace LOAD * from File1.csv;	During both normal and partial reload, the Qlik Sense table Tab1 is initially dropped. Thereafter new data is loaded from File1.csv and stored in Tab1.

Example	Result
Tab1: Replace only LOAD * from File1.csv;	During normal reload, this statement is disregarded. During partial reload, any Qlik Sense table previously named Tab1 is initially dropped. Thereafter new data is loaded from File1.csv and stored in Tab1.
Tab1: LOAD a,b,c from File1.csv; Replace LOAD a,b,c from File2.csv;	During normal reload, the file File1.csv is first read into the Qlik Sense table Tab1, but then immediately dropped and replaced by new data loaded from File2.csv. All data from File1.csv is lost. During partial reload, the entire Qlik Sense table Tab1 is initially dropped. Thereafter it is replaced by new data loaded from File2.csv.
Tab1: LOAD a,b,c from File1.csv; Replace only LOAD a,b,c from File2.csv;	During normal reload, data is loaded from File1.csv and stored in the Qlik Sense table Tab1. File2.csv is disregarded. During partial reload, the entire Qlik Sense table Tab1 is initially dropped. Thereafter it is replaced by new data loaded from File2.csv. All data from File1.csv is lost.

Right

The **Join** and **Keep** prefixes can be preceded by the prefix **right**.

If used before **join** it specifies that a right join should be used. The resulting table will only contain combinations of field values from the raw data tables where the linking field values are represented in the second table. If used before **keep**, it specifies that the first raw data table should be reduced to its common intersection with the second table, before being stored in Qlik Sense.



Were you looking for the string function by the same name? See: *Right* (page 1427)

Syntax:

```
Right (Join | Keep) [ (tablename) ] (loadstatement | selectstatement )
```

Arguments:

Arguments

Argument	Description
tablename	The named table to be compared to the loaded table.
loadstatement or selectstatement	The LOAD or SELECT statement for the loaded table.

Example

Load script

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
Table1:
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];
```

```
Table2:
Right Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

Result

Resulting table

Column1	Column2	Column3
A	B	C
1	aa	xx
4	-	yy

Explanation

This example demonstrates the Right Join output where only values present in the second (right) table are joined.

Sample

The **sample** prefix to a **LOAD** or **SELECT** statement is used for loading a random sample of records from the data source.

Syntax:

```
Sample p ( loadstatement | selectstatement )
```

The expression that is evaluated does not define the percentage of records from the dataset that will be loaded into the Qlik Sense application, but the probability of each record that is read to be loaded into the application. In other words, specifying a value $p = 0.5$ does not mean that 50% of the total number of records will be loaded, but instead that for each record there will be a 50% chance that it is loaded into the Qlik Sense application.

Arguments

Argument	Description
p	An arbitrary expression which evaluates to a number larger than 0 and lower or equal to 1. The number indicates the probability for a given record to be read. All records will be read but only some of them will be loaded into Qlik Sense.

When to use it

Sample is useful when you would like to sample data coming from a large table, to understand the nature of data, distribution or field contents. As it brings a subset of data, the data loads are faster, allowing faster testing of scripts. Unlike First, the Sample function brings data from the whole table, instead of being limited to the first few rows. This can provide a more accurate representation of the data in some cases.

The following examples show two possible uses of the Sample script prefix:

```
Sample 0.15 SQL SELECT * from Longtable;  
Sample(0.15) LOAD * from Longtab.csv;
```

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the SET DateFormat statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Sample from an inline table

Load script and results

Overview

In this example, the script loads a sample set of data from a dataset containing seven records into a table named Transactions from an inline table.

Load script

```
Transactions:  
SAMPLE 0.3  
LOAD  
*  
InLine [  
id, date, amount  
1, 08/30/2018, 23.56  
2, 09/07/2018, 556.31  
3, 09/16/2018, 5.75  
4, 09/22/2018, 125.00  
5, 09/22/2018, 484.21  
6, 09/22/2018, 59.18  
7, 09/23/2018, 177.42  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- amount

Add the following the measure:

```
=sum(amount)8
```

Results table

id	date	=Sum(amount)
2	09/07/2018	556.31
4	09/22/2018	125
1	08/30/2018	23.56
3	09/16/2018	5.75

In the iteration of the load used in this example, all seven records were read, but only four records were loaded into the data table. Any re-run load could result in a different number, and a different set of records being loaded into the application.

Example 2 – Sample from an autogenerated table

Load script and results

Overview

In this example, using Autogenerate, a dataset of 100 records is created with the fields date, id, and amount. However, the Sample prefix is used, with a value of 0.1.

Load script

```
SampleData:  
Sample 0.1  
LOAD  
RecNo() AS id,  
MakeDate(2013, Ceil(Rand() * 12), Ceil(Rand() * 29)) as date,  
Rand() * 1000 AS amount  
  
Autogenerate(100);
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- amount

Add the following the measure:

Results table

id	date	=Sum(amount)
48	9/28/2013	763
20	5/15/2013	752
19	11/8/2013	657
25	3/24/2013	522
27	8/23/2013	389
81	6/1/2013	53
100	8/15/2013	17

In the iteration of the load used in this example, seven records were loaded from the created dataset. Once again, any re-run load could result in a different number, and a different set of records being loaded into the application.

Semantic

The `semantic` load prefix creates a special type of field that can be used in Qlik Sense to connect and manage relational data, such as tree structures, self-referencing parent-child structured data and/or data that can be described as a graph.

Note that the `semantic` load can function similarly to the *Hierarchy* (page 62) and *HierarchyBelongsTo* (page 64) prefixes. All three prefixes can be used as building blocks in effective front-end solutions for traversing relational data.

Syntax:

```
Semantic( loadstatement | selectstatement )
```

A semantic load expects an input that is exactly three or four fields wide with a strict definition of what each ordered field represents, as shown in the table below:

Semantic load fields

Field name	Field description
1st Field:	This tag is a representation of the first of two objects between which there is a relationship.
2nd Field:	This tag will be used to describe the “forward” relationship between the first and second object. If the first object is a child and the second object is a parent, you can create a relationship tab that states “parent” or “parent of” as if you are following the relationship from

Field name	Field description
	child to parent.
3rd Field:	This tag is a representation of the second of two objects between which there is a relationship.
4th Field:	This field is optional. This tag describes the “backward” or “inverse” relationship between the first and second object. If the first object is a child and the second object is a parent, a relationship tab could state “child” or “child of” as if you are following the relationship from parent to child. If you do not add a fourth field, then the second field tag will be used to describe the relationship in either direction. In that case, an arrow symbol is automatically added as part of the tag.

The following code is an example of the semantic prefix.

```
Semantic
Load
Object,
'Parent' AS Relationship,
NeighbouringObject AS Object,
'Child' AS Relationship
from graphdata.csv;
```



It is allowed and typical practice to label the third field the same as the first field. This creates a self-referencing lookup, so that you can follow object(s) to the related object(s) one relationship step away at a time. If the 3rd field does not carry the same name, then the end result will be a simple lookup from an object(s) to its direct relational neighbor(s) one step away only, which is an output of little practical use.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Related functions

Functions	Interaction
<code>Hierarchy</code> (page 62)	The Hierarchy load prefix is used to divide and organize nodes in parent-child and other graph-like data structures and transform them into tables.

Functions	Interaction
<i>HierarchyBelongsTo</i> (page 64)	The HierarchyBelongsTo load prefix is used to locate and organize the ancestors of parent-child and other graph-like data structures and transform them into tables.

Example - Creating a special field for connecting relationships using the semantic prefix

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset representing geography relation records which is loaded into a table named GeographyTree.
 - Each entry has an ID at the beginning of the line and a ParentID at the end of the line.
- The semantic prefix which will add one special behavior field labeled, Relation.

Load script

```
GeographyTree:  
LOAD  
    ID,  
    Geography,  
    if(ParentID=' ',null(),ParentID) AS ParentID  
  
INLINE [  
ID,Geography,ParentID  
1,world  
2,Europe,1  
3,Asia,1  
4,North America,1  
5,South America,1  
6,UK,2  
7,Germany,2  
8,Sweden,2  
9,South Korea,3  
10,North Korea,3  
11,China,3  
12,London,6  
13,Birmingham,6  
];  
  
SemanticTable:  
Semantic Load  
    ID as ID,  
    'Parent' as Relation,  
    ParentID as ID,
```

```
'Child' as Relation  
resident GeographyTree;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Id
- Geography

Then, create a filter pane with Relation as a dimension. Click **Done editing**.

Results table

Id	Geography
1	World
2	Europe
3	Asia
4	North America
5	South America
6	UK
7	Germany
8	Sweden
9	South Korea
10	North Korea
11	China
12	London
13	Birmingham

Filter pane

Relation

Child

Parent

Click **Europe** from the Geography dimension in the table and click **Child** from the Relation dimension in the filter pane. Note the expected result in the table:

Results table showing
"children" of Europe

Id Geography

6	UK
7	Germany
8	Sweden

Clicking **Child** again will show places that are "children" of the UK, one step further down.

Results table showing
"children" of UK

Id Geography

12	London
13	Birmingham

Unless

The **unless** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be evaluated or not. It may be seen as a compact alternative to the full **if..end if** statement.

Syntax:

```
(Unless condition statement | exitstatement Unless condition )
```

The **statement** or the **exitstatement** will only be executed if **condition** is evaluated to False.

The **unless** prefix may be used on statements which already have one or several other statements, including additional **when** or **unless** prefixes.

Arguments

Argument	Description
condition	A logical expression evaluating to True or False.
statement	Any Qlik Sense script statement except control statements.
exitstatement	An exit for , exit do or exit sub clause or an exit script statement.

When to use it

The **unless** statement returns a Boolean result. Typically, this type of function will be used as a condition when the user would like to conditionally load or exclude parts of the script.

The following lines show three examples of how the **unless** function may be used:

```
exit script unless A=1;
unless A=1 LOAD * from myfile.csv;
unless A=1 when B=2 drop table Tab1;
```

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Unless prefix

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The creation of variable A, which is given a value of 1.
- A dataset which is loaded into a table named Transactions, unless the variable A = 2.

Load script

```
LET A = 1;  
  
UNLESS A = 2  
  
Transactions:  
LOAD  
*  
Inline [  
id, date, amount  
1, 08/30/2018, 23.56  
2, 09/07/2018, 556.31  
3, 09/16/2018, 5.75  
4, 09/22/2018, 125.00  
5, 09/22/2018, 484.21  
6, 09/22/2018, 59.18  
7, 09/23/2018, 177.42  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

Results table

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

Because the variable A is assigned the value of 1 at the start of the script, the condition following the unless prefix is evaluated, returning a result of FALSE. As a result, the script continues to run the Load statement. In the results table, all the records from the Transactions table can be seen.

If this variable value is set to equal to 2, no data will be loaded into the data model.

Example 2 – Unless suffix

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script starts by loading an initial dataset into a table named Transactions. The script is then terminated unless there are less than 10 records in the Transactions table.

If this condition does not result in a termination of the script, a further set of transactions is concatenated into the Transactions table and this process is repeated.

Load script

Transactions:

```
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
```

2 Script statements and keywords

```
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];

exit script unless NoOfRows('Transactions') < 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];

exit script unless NoOfRows('Transactions') < 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
15, 10/01/2018, 164.27
16, 10/03/2018, 384.00
17, 10/06/2018, 25.82
18, 10/09/2018, 312.00
19, 10/15/2018, 4.56
20, 10/16/2018, 90.24
21, 10/18/2018, 19.32
];

exit script unless NoOfRows('Transactions') < 10 ;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

Results table

id	date	amount
1	08/30/2018	23.56

id	date	amount
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

There are seven records in each of the three datasets of the load script.

The first dataset (with transaction id 1 through 7) is loaded into the application. The `unless` condition evaluates whether there are less than 10 rows in the `Transactions` table. This evaluates to `TRUE`, and therefore the second dataset (with transaction id 8 through 14) is loaded into the application. The second `unless` condition evaluates if there are less than 10 records in the `Transactions` table. This evaluates to `FALSE`, and so the script terminates.

Example 3 – Multiple Unless prefixes

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

In this example, a dataset containing one transaction is created as a table called `Transactions`. A 'for' loop is then triggered, in which two nested unless statements evaluate:

1. Unless there are more than 100 records in the `Transactions` table
2. Unless the number of records in the `Transactions` table is a multiple of 6

If these conditions are `FALSE`, a further seven records are generated and concatenated onto the existing `Transactions` table. This process is repeated until one of the two transactions returns a value of `TRUE`.

Load script

```
Transactions:  
Load  
    0 as id  
Autogenerate 1;  
  
For i = 1 to 100  
    unless NoOfRows('Transactions') > 100 unless mod(NoOfRows('Transactions'),6) = 0  
        Concatenate  
        Load  
        if(isnull(Peek(id)),1,peek(id)+1) as id  
            Autogenerate 7;  
    next i
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:id.

Results table

id
0
1
2
3
4
5
+30 more rows

The nested unless statements that occur in the 'for' loop evaluate the following:

1. Are there more than 100 rows in the Transactions table?
2. Is the total number of records in the Transactions table a multiple of 6?

Whenever both unless statements return a value of FALSE, a further seven records are generated and concatenated onto the existing Transactions table.

These statements return a value of FALSE five times, at which point there are a total of 36 rows of data in the Transactions table.

After this, the second unless statement returns a value of TRUE, and therefore the load statement following this will no longer be executed.

When

The **when** prefix and suffix is used for creating a conditional clause which determines whether a statement or exit clause should be executed or not. It may be seen as a compact alternative to the full **if..end if** statement.

Syntax:

```
(when condition statement | exitstatement when condition )
```

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

The **statement** or the **exitstatement** will only be executed if condition is evaluated to TRUE.

The when prefix may be used on statements which already have one or several other statements, including additional when or unless prefixes.

When to use it

The when statement returns a Boolean result. Typically, this type of function will be used as a condition when the user would like to load or exclude parts of a script.

Arguments

Argument	Description
condition	A logical expression evaluating to TRUE or FALSE
statement	Any Qlik Sense script statement except control statements.
exitstatement	An exit for , exit do or exit sub clause or an exit script statement.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the **SET DateFormat** statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
exit script when A=1;	When the statement A=1 is evaluated to be TRUE, the script will stop.
when A=1 LOAD * from myfile.csv;	When the statement A=1 is evaluated to be TRUE, the myfile.csv will be loaded.
when A=1 unless B=2 drop table Tab1;	When the statement A=1 is evaluated to be TRUE, and if B=2 is evaluated to be FALSE, than the Tab1 table will be dropped.

Example 1 – When prefix

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset with dates and amounts that is sent to a table named ‘Transactions’.
- The Let statement which states that the variable A is created and has the value of 1.
- The when condition which provides the condition that if A equals 1, then the script will continue to load.

Load script

```
LET A = 1;

WHEN A = 1

Transactions:
LOAD
*
Inline [
id, date, amount
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

Results table

id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

Because the variable A is assigned the value of 1 at the start of the script, the condition following the when prefix is evaluated and returns a result of TRUE. Because it returns a TRUE result, the script continues to run the load statement. All the records from the results table can be seen.

If this variable value was set to any value not equal to 1, no data would be loaded into the data model.

Example 2 – When suffix

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Three datasets with dates and amounts that are sent to a table named ‘Transactions’.
 - The first dataset contains transactions 1-7.
 - The second dataset contains transactions 8-14.
 - The third dataset contains transactions 15-21.
- A when condition which determines whether the ‘Transactions’ table contains more than ten rows. If any of the when statements are evaluated to be TRUE, the load script will stop. This condition is placed at the end of each of the three datasets.

Load script

Transactions:

```
LOAD
*
Inline [
id, date, amount
```

```
1, 08/30/2018, 23.56
2, 09/07/2018, 556.31
3, 09/16/2018, 5.75
4, 09/22/2018, 125.00
5, 09/22/2018, 484.21
6, 09/22/2018, 59.18
7, 09/23/2018, 177.42
];

exit script when NoOfRows('Transactions') > 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
8, 10/01/2018, 164.27
9, 10/03/2018, 384.00
10, 10/06/2018, 25.82
11, 10/09/2018, 312.00
12, 10/15/2018, 4.56
13, 10/16/2018, 90.24
14, 10/18/2018, 19.32
];
;

exit script when NoOfRows('Transactions') > 10 ;

Concatenate
LOAD
*
Inline [
id, date, amount
15, 10/01/2018, 164.27
16, 10/03/2018, 384.00
17, 10/06/2018, 25.82
18, 10/09/2018, 312.00
19, 10/15/2018, 4.56
20, 10/16/2018, 90.24
21, 10/18/2018, 19.32
];
;

exit script when NoOfRows('Transactions') > 10 ;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- amount

Results table		
id	date	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

There are seven transactions in each of the three datasets. The first dataset contains transaction 1-7 and is loaded into the application. The when condition following this load statement is evaluated as FALSE because there are less than ten rows in the 'Transactions' table. The load script continues to the next dataset.

The second dataset contains transaction 8-14 and is loaded into the application. The second when condition evaluates as TRUE because there are more than ten rows in the 'Transactions' table. Therefore, the script terminates.

Example 3 – Multiple When prefixes

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a single transaction is created as a table called 'Transactions'.
- A For loop which is triggered contains two nested when conditions which evaluate whether:
 1. There are less than 100 records in the 'Transactions' table.
 2. The number of records in the 'Transactions' table is not a multiple of 6.

Load script

```
Transactions:  
Load  
    0 as id  
Autogenerate 1;  
  
For i = 1 to 100  
    when NoofRows('Transactions') < 100 when mod(NoofRows('Transactions'),6) <> 0  
        Concatenate  
        Load  
            if(isnull(Peek(id)),1,peek(id)+1) as id  
        Autogenerate 7;  
    next i
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- id

The results table only shows the first five transaction IDs but the load script creates 36 rows and then terminates once the when condition is fulfilled.

Results table

id
0
1
2
3
4
5
+30 more rows

The nested when conditions in the For loop evaluate the following questions:

- Are there less than 100 rows in the 'Transactions' table?
- Is the total number of records in the 'Transactions' table not a multiple of six?

Whenever both when conditions return a value of TRUE, a further seven records are generated and concatenated onto the existing 'Transactions' table.

The when conditions return a TRUE value five times. At that point there are a total of 36 rows of data in the 'Transactions' table.

When 36 rows of data are created in the 'Transactions' table, the second when statement returns a value of FALSE and therefore the load statement following this will no longer be executed.

2.5 Script regular statements

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Script regular statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Alias

The **alias** statement is used for setting an alias according to which a field will be renamed whenever it occurs in the script that follows.

```
Aliasfieldname as aliasname {,fieldname as aliasname}
```

Autonumber

This statement creates a unique integer value for each distinct evaluated value in a field encountered during the script execution.

```
AutoNumber fields [Using namespace] ]
```

Binary

The **binary** statement is used for loading the data from another QlikView document, including section access data.

```
Binary [path] filename
```

comment

Provides a way of displaying the field comments (metadata) from databases and spreadsheets. Field names not present in the app will be ignored. If multiple occurrences of a field name are found, the last value is used.

```
Comment field *fieldlist using mapname  
Comment field fieldname with comment
```

comment table

Provides a way of displaying the table comments (metadata) from databases or spreadsheets.

```
Comment table tablelist using mapname  
Comment table tablename with comment
```

Connect



This functionality is not available in Qlik Sense SaaS.

The **CONNECT** statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.

```
ODBC Connect TO connect-string [ ( access_info ) ]
OLEDB CONNECT TO connect-string [ ( access_info ) ]
CUSTOM CONNECT TO connect-string [ ( access_info ) ]
LIB CONNECT TO connection
```

Declare

The **Declare** statement is used to create field definitions, where you can define relations between fields or functions. A set of field definitions can be used to automatically generate derived fields, which can be used as dimensions. For example, you can create a calendar definition, and use that to generate related dimensions, such as year, month, week and day, from a date field.

```
definition_name:
Declare [Field[s]] Definition [Tagged tag_list ]
[Parameters parameter_list ]
Fields field_list
[Groups group_list ]

<definition name>:
Declare [Field][s] Definition
Using <existing_definition>
[With <parameter_assignment> ]
```

Derive

The **Derive** statement is used to generate derived fields based on a field definition created with a **Declare** statement. You can either specify which data fields to derive fields for, or derive them explicitly or implicitly based on field tags.

```
Derive [Field[s]] From [Field[s]] field_list Using definition
Derive [Field[s]] From Explicit [Tag[s]] (tag_list) Using definition
Derive [Field[s]] From Implicit [Tag[s]] Using definition
```

Direct Query

The **DIRECT QUERY** statement allows you to access tables through an ODBC or OLE DB connection using the Direct Discovery function.

```
Direct Query [path]
```

Directory

The **Directory** statement defines which directory to look in for data files in subsequent **LOAD** statements, until a new **Directory** statement is made.

```
Directory [path]
```

Disconnect

The **Disconnect** statement terminates the current ODBC/OLE DB/Custom connection. This statement is optional.

```
Disconnect
```

drop field

One or several Qlik Sense fields can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop field** statement. The "distinct" property of a table is removed after a **drop field** statement.



Both **drop field** and **drop fields** are allowed forms with no difference in effect. If no table is specified, the field will be dropped from all tables where it occurs.

```
Drop field fieldname [, fieldname2 ...] [from tablename1 [, tablename2 ...]]  
drop fields fieldname [, fieldname2 ...] [from tablename1 [, tablename2 ...]]
```

drop table

One or several Qlik Sense internal tables can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop table** statement.



The forms **drop table** and **drop tables** are both accepted.

```
Drop table tablename [, tablename2 ...]  
drop tables[ tablename [, tablename2 ...]]
```

Execute

The **Execute** statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.

```
Execute cmdline
```

FlushLog

The **FlushLog** statement forces Qlik Sense to write the content of the script buffer to the script log file.

```
FlushLog
```

Force

The **force** statement forces Qlik Sense to interpret field names and field values of subsequent **LOAD** and **SELECT** statements as written with only upper case letters, with only lower case letters, as always capitalized or as they appear (mixed). This statement makes it possible to associate field values from tables made according to different conventions.

```
Force ( capitalization | case upper | case lower | case mixed )
```

LOAD

The **LOAD** statement loads fields from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically. It is also possible to load data from analytic connections.

```
Load [ distinct ] *fieldlist  
[( from file [ format-spec ] |
```

```
from_field fieldassource [format-spec]
inline data [ format-spec ] |
resident table-label |
autogenerate size )
[ where criterion | while criterion ]
[ group_by groupbyfieldlist ]
[order_by orderbyfieldlist ]
[extension pluginname.functionname(tabledescription)]
```

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '=' at script run time before it is assigned to the variable.

```
Let varianlename=expression
```

Loosen Table

One or more Qlik Sense internal data tables can be explicitly declared loosely coupled during script execution by using a **Loosen Table** statement. When a table is loosely coupled, all associations between field values in the table are removed. A similar effect could be achieved by loading each field of the loosely coupled table as independent, unconnected tables. Loosely coupled can be useful during testing to temporarily isolate different parts of the data structure. A loosely coupled table can be identified in the table viewer by the dotted lines. The use of one or more **Loosen Table** statements in the script will make Qlik Sense disregard any setting of tables as loosely coupled made before the script execution.

```
tablename [ , tablename2 ...]
Loosen Tables tablename [ , tablename2 ...]
```

Map ... using

The **map ... using** statement is used for mapping a certain field value or expression to the values of a specific mapping table. The mapping table is created through the **Mapping** statement.

```
Map *fieldlist Using mapname
```

NullAsNull

The **NullAsNull** statement turns off the conversion of NULL values to string values previously set by a **NullAsValue** statement.

```
NullAsNull *fieldlist
```

NullAsValue

The **NullAsValue** statement specifies for which fields that NULL should be converted to a value.

```
NullAsValue *fieldlist
```

Qualify

The **Qualify** statement is used for switching on the qualification of field names, i.e. field names will get the table name as a prefix.

```
Qualify *fieldlist
```

Rem

The **rem** statement is used for inserting remarks, or comments, into the script, or to temporarily deactivate script statements without removing them.

```
Rem string
```

Rename Field

This script function renames one or more existing Qlik Sense field(s) after they have been loaded.

```
Rename field (using mapname | oldname to newname{ , oldname to newname })
```

```
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

Rename Table

This script function renames one or more existing Qlik Sense internal table(s) after they have been loaded.

```
Rename table (using mapname | oldname to newname{ , oldname to newname })
```

```
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

Section

With the **section** statement, it is possible to define whether the subsequent **LOAD** and **SELECT** statements should be considered as data or as a definition of the access rights.

```
Section (access | application)
```

Select

The selection of fields from an ODBC data source or OLE DB provider is made through standard SQL **SELECT** statements. However, whether the **SELECT** statements are accepted depends on the ODBC driver or OLE DB provider used.

```
Select [all | distinct | distinctrow | top n [percent] ] *fieldlist
```

```
From tablelist
```

```
[Where criterion ]
```

```
[Group by fieldlist [having criterion ] ]
```

```
[Order by fieldlist [asc | desc] ]
```

```
[ (Inner | Left | Right | Full)Join tablename on fieldref = fieldref ]
```

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

```
Set variablename=string
```

Sleep

The **sleep** statement pauses script execution for a specified time.

```
Sleep n
```

SQL

The **SQL** statement allows you to send an arbitrary SQL command through an ODBC or OLE DB connection.

```
SQL sql_command
```

SQLColumns

The **sqlcolumns** statement returns a set of fields describing the columns of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLColumns
```

SQLTables

The **sqltables** statement returns a set of fields describing the tables of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLTables
```

SQLTypes

The **sqltypes** statement returns a set of fields describing the types of an ODBC or OLE DB data source, to which a **connect** has been made.

```
SQLTypes
```

Star

The string used for representing the set of all the values of a field in the database can be set through the **star** statement. It affects the subsequent **LOAD** and **SELECT** statements.

```
Star is [ string ]
```

Store

The **Store** statement creates a QVD, or text file.

```
Store [ *fieldlist from] table into filename [ format-spec ];
```

Tag

This script statement provides a way to assign tags to one or more fields or tables. If an attempt to tag a field or table not present in the app is made, the tagging will be ignored. If conflicting occurrences of a field or tag name are found, the last value is used.

```
Tag[field|fields] fieldlist with tagname  
Tag [field|fields] fieldlist using mapname  
Tag table tablelist with tagname
```

Trace

The **trace** statement writes a string to the **Script Execution Progress** window and to the script log file, when used. It is very useful for debugging purposes. Using \$-expansions of variables that are calculated prior to the **trace** statement, you can customize the message.

```
Trace string
```

Unmap

The **Unmap** statement disables field value mapping specified by a previous **Map ... Using** statement for subsequently loaded fields.

```
Unmap *fieldlist
```

Unqualify

The **Unqualify** statement is used for switching off the qualification of field names that has been previously switched on by the **Qualify** statement.

```
Unqualify *fieldlist
```

Untag

This script statement provides a way to remove tags from fields or tables. If an attempt to untag a field or table not present in the app is made, the untagging will be ignored.

```
Untag[field|fields] fieldlist with tagname  
Tag [field|fields] fieldlist using mapname  
Tag table tablelist with tagname
```

Alias

The **alias** statement is used for setting an alias according to which a field will be renamed whenever it occurs in the script that follows.

Syntax:

```
alias fieldname as aliasname {,fieldname as aliasname}
```

Arguments:

Arguments

Argument	Description
fieldname	The name of the field in your source data
aliasname	An alias name you want to use instead

Examples and results:

Example	Result
Alias ID_N as NameID;	
Alias A as Name, B as Number, C as Date;	The name changes defined through this statement are used on all subsequent SELECT and LOAD statements. A new alias can be defined for a field name by a new alias statement at any subsequent position in the script.

AutoNumber

This statement creates a unique integer value for each distinct evaluated value in a field encountered during the script execution.

You can also use the *autonumber* (page 556) function inside a **LOAD** statement, but this has some limitations when you want to use an optimized load. You can create an optimized load by loading the data from a **QVD** file first, and then using the **AutoNumber** statement to convert values to symbol keys.

Syntax:

```
AutoNumber *fieldlist [Using namespace] ]
```

Arguments:

Arguments	
Argument	Description
*fieldlist	A comma-separated list of the fields where the values should be replaced by a unique integer value. You can use wildcard characters ? and * in the field names to include all fields with matching names. You can also use * to include all fields. You need to quote field names when wildcards are used.
namespace	Using namespace is optional. You can use this option if you want to create a namespace, where identical values in different fields share the same key. If you do not use this option, all fields will have a separate key index.

Limitations:

When you have several **LOAD** statements in the script, you need to place the **AutoNumber** statement after the final **LOAD** statement.

Example - script with AutoNumber

Script example

In this example, the data is first loaded without the **AutoNumber** statement. The **AutoNumber** statement is then added to show the effect.

Data used in the example

Load the following data as an inline load in the data load editor to create the script example below. Leave the **AutoNumber** statement commented out for now.

```
Regionsales:  
LOAD *,  
Region &'|'& Year &'|'& Month as KeyToOtherTable  
INLINE  
[ Region, Year, Month, Sales
```

2 Script statements and keywords

```
North, 2014, May, 245
North, 2014, May, 347
North, 2014, June, 127
South, 2014, June, 645
South, 2013, May, 367
South, 2013, May, 221
];

Budget:
LOAD Budget,
Region &'|& Year &'|& Month as KeyToOtherTable
INLINE
[Region, Year, Month, Budget
North, 2014, May, 200
North, 2014, May, 350
North, 2014, June, 150
South, 2014, June, 500
South, 2013, May, 300
South, 2013, May, 200
];
//AutoNumber KeyToOtherTable;
```

Create visualizations

Create two table visualizations in a Qlik Sense sheet. Add **KeyToOtherTable**, **Region**, **Year**, **Month**, and **Sales** as dimensions to the first table. Add **KeyToOtherTable**, **Region**, **Year**, **Month**, and **Budget** as dimensions to the second table.

Result

RegionSales table

KeyToOtherTable	Region	Year	Month	Sales
North 2014 June	North	2014	June	127
North 2014 May	North	2014	May	245
North 2014 May	North	2014	May	347
South 2013 May	South	2013	May	221
South 2013 May	South	2013	May	367
South 2014 June	South	2014	June	645

Budget table

KeyToOtherTable	Region	Year	Month	Budget
North 2014 June	North	2014	June	150

2 Script statements and keywords

KeyToOtherTable	Region	Year	Month	Budget
North 2014 May	North	2014	May	200
North 2014 May	North	2014	May	350
South 2013 May	South	2013	May	200
South 2013 May	South	2013	May	300
South 2014 June	South	2014	June	500

Explanation

The example shows a composite field **KeyToOtherTable** that links the two tables. **AutoNumber** is not used. Note the length of the **KeyToOtherTable** values.

Add AutoNumber statement

Uncomment the **AutoNumber** statement in the load script.

```
AutoNumber KeyToOtherTable;
```

Result

RegionSales table

KeyToOtherTable	Region	Year	Month	Sales
1	North	2014	June	127
1	North	2014	May	245
2	North	2014	May	347
3	South	2013	May	221
4	South	2013	May	367
4	South	2014	June	645

Budget table

KeyToOtherTable	Region	Year	Month	Budget
1	North	2014	June	150
1	North	2014	May	200
2	North	2014	May	350
3	South	2013	May	200
4	South	2013	May	300
4	South	2014	June	500

Explanation

The **KeyToOtherTable** field values have been replaced with unique integer values and, as a result, the length of the field values has been reduced, thus conserving memory. The key fields in both tables are affected by **AutoNumber** and the tables remain linked. The example is brief for demonstration purposes, but would be meaningful with a table containing a large number of rows.

Binary

The **binary** statement is used for loading the data from another Qlik Sense app or QlikView document, including section access data. Other elements of the app are not included, for example, sheets, stories, visualizations, master items or variables.

Only one **binary** statement is allowed in the script. The **binary** statement must be the first statement of the script, even before the SET statements usually located at the beginning of the script.

Syntax:

```
binary [path] filename
```

Arguments:

Arguments

Argument	Description
path	<p>The path to the file which should be a reference to a folder data connection. This is required if the file is not located in the Qlik Sense working directory.</p> <p>Example: <code>'lib://Table Files/'</code></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">• absolute <p>Example: <code>c:\data\</code></p> <ul style="list-style-type: none">• relative to the app containing this script line. <p>Example: <code>data\</code></p>
filename	The name of the file, including the file extension .qvw or .qvf.

Limitations:

You cannot use **binary** to load data from an app on the same Qlik Sense Enterprise deployment by referring to the app ID. You can only load from a .qvff file.

Examples

String	Description
Binary lib://DataFolder/customer.qvw;	In this example, the file must be located in the Folder data connection. This may be, for example, a folder that your administrator creates on the Qlik Sense server. Click Create new connection in the data load editor and then select Folder under File locations .
Binary customer.qvf;	In this example, the file must be located in the Qlik Sense working directory.
Binary c:\qv\customer.qvw;	This example using an absolute file path will only work in legacy scripting mode.

Comment field

Provides a way of displaying the field comments (metadata) from databases and spreadsheets. Field names not present in the app will be ignored. If multiple occurrences of a field name are found, the last value is used.

Syntax:

```
comment [fields] *fieldlist using mapname
comment [field] fieldname with comment
```

The map table used should have two columns, the first containing field names and the second the comments.

Arguments:

Arguments

Argument	Description
*fieldlist	A comma separated list of the fields to be commented. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.
mapname	The name of a mapping table previously read in a mapping LOAD or mapping SELECT statement.
fieldname	The name of the field that should be commented.
comment	The comment that should be added to the field.

Example 1:

```
commentmap:
mapping LOAD * inline [
a,b
Alpha,This field contains text values
Num,This field contains numeric values
];
```

```
comment fields using commentmap;
```

Example 2:

```
comment field Alpha with AFieldContainingCharacters;
comment field Num with '*A field containing numbers';
comment Gamma with 'Mickey Mouse field';
```

Comment table

Provides a way of displaying the table comments (metadata) from databases or spreadsheets.

Table names not present in the app are ignored. If multiple occurrences of a table name are found, the last value is used. The keyword can be used to read comments from a data source.

Syntax:

```
comment [tables] tablelist using mapname
comment [table] tablename with comment
```

Arguments:

Arguments

Argument	Description
tablelist	(table[,table])
mapname	The name of a mapping table previously read in a mapping LOAD or mapping SELECT statement.
tablename	The name of the table that should be commented.
comment	The comment that should be added to the table.

Example 1:

Commentmap:

```
mapping LOAD * inline [
a,b
Main,This is the fact table
Currencies, Currency helper table
];
comment tables using Commentmap;
```

Example 2:

```
comment table Main with 'Main fact table';
```

Connect

The **CONNECT** statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.



This functionality is not available in Qlik Sense SaaS.



This statement supports only folder data connections in standard mode.

Syntax:

```
ODBC CONNECT TO connect-string
OLEDB CONNECT TO connect-string
CUSTOM CONNECT TO connect-string
LIB CONNECT TO connection
```

Arguments:

Arguments

Argument	Description
connect-string	<p>connect-string ::= datasourcename { ; conn-spec-item }</p> <p>The connection string is the data source name and an optional list of one or more connection specification items. If the data source name contains blanks, or if any connection specification items are listed, the connection string must be enclosed by quotation marks.</p> <p>datasourcename must be a defined ODBC data source or a string that defines an OLE DB provider.</p> <p>conn-spec-item ::=DBQ=database_specifier DriverID=driver_specifier UID=userid PWD=password</p> <p>The possible connection specification items may differ between different databases. For some databases, also other items than the above are possible. For OLE DB, some of the connection specific items are mandatory and not optional.</p>
connection	The name of a data connection stored in the data load editor.

If the **ODBC** is placed before **CONNECT**, the ODBC interface will be used; else, OLE DB will be used.

Using **LIC CONNECT TO** connects to a database using a stored data connection that was created in the data load editor.

Example 1:

```
ODBC CONNECT TO 'Sales
DBQ=C:\Program Files\Access\Samples\Sales.mdb';
The data source defined through this statement is used by subsequent Select (SQL) statements, until a new
CONNECT statement is made.
```

Example 2:

```
LIB CONNECT TO 'DataConnection';
```

Connect32

This statement is used the same way as the **CONNECT** statement, but forces a 64-bit system to use a 32-bit ODBC/OLE DB provider. Not applicable for custom connect.

Connect64

This statement is used the same way as the as the **CONNECT** statement, but forces use of a 64-bit provider. Not applicable for custom connect.

Declare

The **Declare** statement is used to create field definitions, where you can define relations between fields or functions. A set of field definitions can be used to automatically generate derived fields, which can be used as dimensions. For example, you can create a calendar definition, and use that to generate related dimensions, such as year, month, week and day, from a date field.

You can use **Declare** to either set up a new field definition, or to create a field definition based on an already existing definition.

Setting up a new field definition

Syntax:

```
definition_name:  
Declare [Field[s]] Definition [Tagged tag_list ]  
[Parameters parameter_list ]  
Fields field_list
```

Arguments:

Argument	Description
definition_name	Name of the field definition, ended with a colon.  <i>Do not use autoCalendar as name for field definitions, as this name is reserved for auto-generated calendar templates.</i> Example: Calendar:

Argument	Description
tag_list	<p>A comma separated list of tags to apply to fields derived from the field definition. Applying tags is optional, but if you do not apply tags that are used to specify sort order, such as \$date, \$numeric or \$text, the derived field will be sorted by load order as default.</p> <p>Example:</p> <pre>'\$date' Thank you for bringing this to our attention, and apologies for the inconvenience.</pre>
parameter_list	<p>A comma separated list of parameters. A parameter is defined in the form name=value and is assigned a start value, which can be overridden when a field definition is re-used. Optional.</p> <p>Example:</p> <pre>first_month_of_year = 1</pre>
field_list	<p>A comma separated list of fields to generate when the field definition is used. A field is defined in the form <expression> As field_name tagged tag. Use \$1 to reference the data field from which the derived fields should be generated.</p> <p>Example:</p> <pre>Year(\$1) As Year tagged ('\$numeric')</pre>

Example:

Calendar:

```
DECLARE FIELD DEFINITION TAGGED '$date'
    Parameters
        first_month_of_year = 1
    Fields
        Year($1) As Year Tagged ('$numeric'),
        Month($1) as Month Tagged ('$numeric'),
        Date($1) as Date Tagged ('$date'),
        Week($1) as Week Tagged ('$numeric'),
        Weekday($1) as Weekday Tagged ('$numeric'),
        DayNumberofYear($1, first_month_of_year) as DayNumberofYear Tagged ('$numeric')
;
```

The calendar is now defined, and you can apply it to the date fields that have been loaded, in this case OrderDate and ShippingDate, using a **Derive** clause.

Re-using an existing field definition

Syntax:

```
<definition name>:
Declare [Field][s] Definition
Using <existing_definition>
[With <parameter_assignment> ]
```

Arguments:

Argument	Description
definition_name	<p>Name of the field definition, ended with a colon.</p> <p>Example:</p> <pre>MyCalendar:</pre>
existing_definition	<p>The field definition to re-use when creating the new field definition. The new field definition will function the same way as the definition it is based on, with the exception if you use parameter_assignment to change a value used in the field expressions.</p> <p>Example:</p> <pre>Using Calendar</pre>
parameter_assignment	<p>A comma separated list of parameter assignments. A parameter assignment is defined in the form name=value and overrides the parameter value that is set in the base field definition. Optional.</p> <p>Example:</p> <pre>first_month_of_year = 4</pre>

Example:

In this example we re-use the calendar definition that was created in the previous example. In this case we want to use a fiscal year that starts in April. This is achieved by assigning the value 4 to the first_month_of_year parameter, which will affect the DayNumberOfYear field that is defined.

The example assumes that you use the sample data and field definition from the previous example.

```
MyCalendar:
DECLARE FIELD DEFINITION USING Calendar WITH first_month_of_year=4;

DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING MyCalendar;
```

When you have reloaded the data script, the generated fields are available in the sheet editor, with names OrderDate.MyCalendar.* and ShippingDate.MyCalendar.*.

Derive

The **Derive** statement is used to generate derived fields based on a field definition created with a **Declare** statement. You can either specify which data fields to derive fields for, or derive them explicitly or implicitly based on field tags.

Syntax:

```
Derive [Field[s]] From [Field[s]] field_list Using definition
Derive [Field[s]] From Explicit [Tag[s]] tag_list Using definition
```

```
Derive [Field[s]] From Implicit [Tag[s]] Using definition
```

Arguments:

Arguments	
Argument	Description
definition	Name of the field definition to use when deriving fields. Example: Calendar
field_list	A comma separated list of data fields from which the derived fields should be generated, based on the field definition. The data fields should be fields you have already loaded in the script. Example: OrderDate, ShippingDate
tag_list	A comma separated list of tags. Derived fields will be generated for all data fields with any of the listed tags. The list of tags should be enclosed by round brackets. Example: ('\$date', '\$timestamp')

Examples:

- Derive fields for specific data fields.
In this case we specify the OrderDate and ShippingDate fields.
DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING Calendar;
- Derive fields for all fields with a specific tag.
In this case we derive fields based on Calendar for all fields with a \$date tag.
DERIVE FIELDS FROM EXPLICIT TAGS ('\$date') USING Calendar;
- Derive fields for all fields with the field definition tag.
In this case we derive fields for all data fields with the same tag as the Calendar field definition, which in this case is \$date.
DERIVE FIELDS FROM IMPLICIT TAG USING Calendar;

Direct Query

The **DIRECT QUERY** statement allows you to access tables through an ODBC or OLE DB connection using the Direct Discovery function.

Syntax:

```
DIRECT QUERY DIMENSION fieldlist [MEASURE fieldlist] [DETAIL fieldlist] FROM
tablelist
[WHERE where_clause]
```

The **DIMENSION**, **MEASURE**, and **DETAIL** keywords can be used in any order.

The **DIMENSION** and **FROM** keyword clauses are required on all **DIRECT QUERY** statements. The **FROM** keyword must appear after the **DIMENSION** keyword.

2 Script statements and keywords

The fields specified directly after the **DIMENSION** keyword are loaded in memory and can be used to create associations between in-memory and Direct Discovery data.



*The **DIRECT QUERY** statement cannot contain **DISTINCT** or **GROUP BY** clauses.*

Using the **MEASURE** keyword you can define fields that Qlik Sense is aware of on a “meta level”. The actual data of a measure field resides only in the database during the data load process, and is retrieved on an ad hoc basis driven by the chart expressions that are used in a visualization.

Typically, fields with discrete values that will be used as dimensions should be loaded with the **DIMENSION** keyword, whereas numbers that will be used in aggregations only should be selected with the **MEASURE** keyword.

DETAIL fields provide information or details, like comment fields, that a user may want to display in a drill-to-details table box. **DETAIL** fields cannot be used in chart expressions.

By design, the **DIRECT QUERY** statement is data-source neutral for data sources that support SQL. For that reason, the same **DIRECT QUERY** statement can be used for different SQL databases without change. Direct Discovery generates database-appropriate queries as needed.

Native data-source syntax can be used when the user knows the database to be queried and wants to exploit database-specific extensions to SQL. Native data-source syntax is supported:

- As field expressions in **DIMENSION** and **MEASURE** clauses
- As the content of the **WHERE** clause

Examples:

```
DIRECT QUERY  
  DIMENSION Dim1, Dim2  
  MEASURE  
    NATIVE ('X % Y') AS X_MOD_Y  
  
FROM TableName  
DIRECT QUERY  
  DIMENSION Dim1, Dim2  
  MEASURE X, Y  
  FROM TableName  
  WHERE NATIVE ('EMAIL MATCHES "\*.EDU"')
```



The following terms are used as keywords and so cannot be used as column or field names without being quoted: and, as, detach, detail, dimension, distinct, from, in, is, like, measure, native, not, or, where

Arguments:

Argument	Description
fieldlist	A comma-separated list of field specifications, <i>fieldname</i> {, <i>fieldname</i> }. A field specification can be a field name, in which case the same name is used for the database column name and the Qlik Sense field name. Or a field specification can be a "field alias," in which case a database expression or column name is given a Qlik Sense field name.
tablelist	A list of the names of tables or views in the database from which data will be loaded. Typically, it will be views that contain a JOIN performed on the database.
where_clause	The full syntax of database WHERE clauses is not defined here, but most SQL "relational expressions" are allowed, including the use of function calls, the LIKE operator for strings, IS NULL and IS NOT NULL , and IN. BETWEEN is not included. NOT is a unary operator, as opposed to a modifier on certain keywords. Examples: <code>WHERE x > 100 AND "Region Code" IN ('south', 'west')</code> <code>WHERE Code IS NOT NULL and Code LIKE '%prospect'</code> <code>WHERE NOT X in (1,2,3)</code> The last example can not be written as: <code>WHERE X NOT in (1,2,3)</code>

Example:

In this example, a database table called TableName, containing fields Dim1, Dim2, Num1, Num2 and Num3, is used. Dim1 and Dim2 will be loaded into the Qlik Sense dataset.

```
DIRECT QUERY DIMENSTION Dim1, Dim2 MEASURE Num1, Num2, Num3 FROM TableName ;
```

Dim1 and Dim2 will be available for use as dimensions. Num1, Num2 and Num3 will be available for aggregations. Dim1 and Dim2 are also available for aggregations. The type of aggregations for which Dim1 and Dim2 can be used depends on their data types. For example, in many cases **DIMENSION** fields contain string data such as names or account numbers. Those fields cannot be summed, but they can be counted: `count(Dim1)`.



DIRECT QUERY statements are written directly in the script editor. To simplify construction of **DIRECT QUERY** statements, you can generate a **SELECT** statement from a data connection, and then edit the generated script to change it into a **DIRECT QUERY** statement.

For example, the **SELECT** statement:

```
SQL SELECT  
SalesOrderID,  
RevisionNumber,  
OrderDate,  
SubTotal,  
TaxAmt  
FROM MyDB.Sales.SalesOrderHeader;
```

could be changed to the following **DIRECT QUERY** statement:

```
DIRECT QUERY  
DIMENSION  
SalesorderID,  
RevisionNumber  
  
MEASURE  
SubTotal,  
TaxAmt  
  
DETAIL  
OrderDate  
  
FROM MyDB.Sales.SalesOrderHeader;
```

Direct Discovery field lists

A field list is a comma-separated list of field specifications, *fieldname* {, *fieldname*}. A field specification can be a field name, in which case the same name is used for the database column name and the field name. Or a field specification can be a field alias, in which case a database expression or column name is given a Qlik Sense field name.

Field names can be either simple names or quoted names. A simple name begins with an alphabetic Unicode character and is followed by any combination of alphabetic or numeric characters or underscores. Quoted names begin with a double quotation mark and contain any sequence of characters. If a quoted name contains double quotation marks, those quotation marks are represented using two adjacent double quotation marks.

Qlik Sense field names are case-sensitive. Database field names may or may not be case-sensitive, depending on the database. A Direct Discovery query preserves the case of all field identifiers and aliases. In the following example, the alias "MyState" is used internally to store the data from the database column "STATEID".

```
DIRECT QUERY Dimension STATEID as MyState Measure AMOUNT from SALES_TABLE;
```

This differs from the result of an **SQL Select** statement with an alias. If the alias is not explicitly quoted, the result contains the default case of column returned by the target database. In the following example, the **SQL Select** statement to an Oracle database creates "MYSTATE," with all upper case letters, as the internal Qlik Sense alias even though the alias is specified as mixed case. The **SQL Select** statement uses the column name returned by the database, which in the case of Oracle is all upper case.

```
SQL Select STATEID as MyState, STATENAME from STATE_TABLE;
```

To avoid this behavior, use the LOAD statement to specify the alias.

```
Load STATEID as MyState, STATENAME;  
SQL Select STATEID, STATEMENT from STATE_TABLE;
```

In this example, the "STATEID" column is stored internally byQlik Sense as "MyState".

Most database scalar expressions are allowed as field specifications. Function calls can also be used in field specifications. Expressions can contain constants that are boolean, numeric, or strings contained in single quotation marks (embedded single quotation marks are represented by adjacent single quotation marks).

Examples:

```
DIRECT QUERY
```

```
DIMENSION
```

```
SalesOrderID, RevisionNumber
```

```
MEASURE
```

```
SubTotal AS "Sub Total"
```

```
FROM Adventureworks.Sales.SalesOrderHeader;
```

```
DIRECT QUERY
```

```
DIMENSION
```

```
"SalesOrderID" AS "Sales Order ID"
```

```
MEASURE
```

```
SubTotal,TaxAmt,(SubTotal-TaxAmt) AS "Net Total"
```

```
FROM Adventureworks.Sales.SalesOrderHeader;
```

```
DIRECT QUERY
```

```
DIMENSION
```

```
(2*Radius*3.14159) AS Circumference,
```

```
Molecules/6.02e23 AS Moles
```

```
MEASURE  
  
    Num1 AS numA  
  
    FROM TableName;  
  
DIRECT QUERY  
DIMENSION  
    concat(region, 'code') AS region_code  
MEASURE  
    Num1 AS NumA  
FROM TableName;
```

Direct Discovery does not support using aggregations in **LOAD** statements. If aggregations are used, the results are unpredictable. A **LOAD** statement such as the following should not be used:

```
DIRECT QUERY DIMENSION stateid, SUM(amount*7) AS MultiFirst MEASURE amount FROM sales_table;  
The SUM should not be in the LOAD statement.
```

Direct Discovery also does not support Qlik Sense functions in **Direct Query** statements. For example, the following specification for a **DIMENSION** field results in a failure when the "Mth" field is used as a dimension in a visualization:

```
month(ModifiedDate) as Mth
```

Directory

The **Directory** statement defines which directory to look in for data files in subsequent **LOAD** statements, until a new **Directory** statement is made.

Syntax:

```
Directory [path]
```

If the **Directory** statement is issued without a **path** or left out, Qlik Sense will look in the Qlik Sense working directory.

Arguments:

Arguments

Argument	Description
path	<p>A text that can be interpreted as the path to the data file.</p> <p>The path is the path to the file, either:</p> <ul style="list-style-type: none"> • absolute <p>Example: <code>c:\data</code></p> <ul style="list-style-type: none"> • relative to the Qlik Sense app working directory. <p>Example: <code>data</code></p> <ul style="list-style-type: none"> • URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <code>http://wwwqlik.com</code></p>

Examples:

```
DIRECTORY C:\userfiles\data; // OR -> DIRECTORY data\

LOAD * FROM
[data1.csv] // ONLY THE FILE NAME CAN BE SPECIFIED HERE (WITHOUT THE FULL PATH)
(ansi, txt, delimiter is ',', embedded labels);

LOAD * FROM
[data2.txt] // ONLY THE FILE NAME CAN BE SPECIFIED HERE UNTIL A NEW DIRECTORY STATEMENT IS
MADE
(ansi, txt, delimiter is '\t', embedded labels);
```

Disconnect

The **Disconnect** statement terminates the current ODBC/OLE DB/Custom connection. This statement is optional.

Syntax:

```
Disconnect
```

The connection will be automatically terminated when a new **connect** statement is executed or when the script execution is finished.

Example:

```
Disconnect;
```

Drop

The **Drop** script keyword can be used to drop tables or fields from the database.

Drop field

One or several Qlik Sense fields can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop field** statement. The "distinct" property of a table is removed after a **drop field** statement.



*Both **drop field** and **drop fields** are allowed forms with no difference in effect. If no table is specified, the field will be dropped from all tables where it occurs.*

Syntax:

```
Drop field fieldname { , fieldname2 ...} [from tablename1 { , tablename2 ...}]  
Drop fields fieldname { , fieldname2 ...} [from tablename1 { , tablename2 ...}]
```

Examples:

```
Drop field A;  
Drop fields A,B;  
Drop field A from X;  
Drop fields A,B from X,Y;
```

Drop table

One or several Qlik Sense internal tables can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop table** statement.

Syntax:

```
drop table tablename { , tablename2 ...}  
drop tables tablename { , tablename2 ...}
```



*The forms **drop table** and **drop tables** are both accepted.*

The following items will be lost as a result of this:

- The actual table(s).
- All fields which are not part of remaining tables.
- Field values in remaining fields, which came exclusively from the dropped table(s).

Examples and results:

Example	Result
drop table orders, salesmen, T456a;	This line results in three tables being dropped from memory.
<pre>Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;</pre>	Once the table <i>Tab2</i> is created, the table <i>Tab1</i> is dropped.

Drop table

One or several Qlik Sense internal tables can be dropped from the data model, and thus from memory, at any time during script execution, by means of a **drop table** statement.

Syntax:

```
drop table tablename {, tablename2 ...}
drop tables tablename {, tablename2 ...}
```



The forms **drop table** and **drop tables** are both accepted.

The following items will be lost as a result of this:

- The actual table(s).
- All fields which are not part of remaining tables.
- Field values in remaining fields, which came exclusively from the dropped table(s).

Examples and results:

Example	Result
drop table orders, salesmen, T456a;	This line results in three tables being dropped from memory.

Example	Result
<pre>Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;</pre>	Once the table <i>Tab2</i> is created, the table <i>Tab1</i> is dropped.

Execute

The **Execute** statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.



This functionality is not available in Qlik Sense SaaS.



This statement is not supported in standard mode.

Syntax:

```
execute commandline
```

Arguments:

Arguments

Argument	Description
<i>commandline</i>	A text that can be interpreted by the operating system as a command line. You can refer to an absolute file path or a lib:// folder path.

If you want to use **Execute** the following conditions need to be met:

- You must run in legacy mode (applicable for Qlik Sense and Qlik Sense Desktop).
- You need to set *OverrideScriptSecurity* to 1 in *Settings.ini* (applicable for Qlik Sense). *Settings.ini* is located in *C:\ProgramData\Qlik\Sense\Engine* and is generally an empty file.



*If you set *OverrideScriptSecurity* to enable **Execute**, any user can execute files on the server. For example, a user can attach an executable file to an app, and then execute the file in the data load script.*

Do the following:

1. Make a copy of *Settings.ini* and open it in a text editor.
2. Check that the file includes *[Settings 7]* in the first line.
3. Insert a new line and type *OverrideScriptSecurity=1*.
4. Insert an empty line at the end of the file.
5. Save the file.
6. Substitute *Settings.ini* with your edited file.
7. Restart Qlik Sense Engine Service (QES).



If Qlik Sense is running as a service, some commands may not behave as expected.

Example:

```
Execute C:\Program Files\Office12\Excel.exe;  
Execute lib://win\notepad.exe // win is a folder connection referring to c:\windows
```

Field/Fields

The **Field** and **Fields** script keywords are used in **Declare**, **Derive**, **Drop**, **Comment**, **Rename** and **Tag/Untag** statements.

FlushLog

The **FlushLog** statement forces Qlik Sense to write the content of the script buffer to the script log file.

Syntax:

```
FlushLog
```

The content of the buffer is written to the log file. This command can be useful for debugging purposes, as you will receive data that otherwise may have been lost in a failed script execution.

Example:

```
FlushLog;
```

Force

The **force** statement forces Qlik Sense to interpret field names and field values of subsequent **LOAD** and **SELECT** statements as written with only upper case letters, with only lower case letters, as always capitalized or as they appear (mixed). This statement makes it possible to associate field values from tables made according to different conventions.

Syntax:

```
Force ( capitalization | case upper | case lower | case mixed )
```

2 Script statements and keywords

If nothing is specified, force case mixed is assumed. The force statement is valid until a new force statement is made.

The **force** statement has no effect in the access section: all field values loaded are case insensitive.

Examples and results

Example	Result
This example shows how to force capitalization. FORCE Capitalization; Capitalization: LOAD * Inline [ab cd eF GH];	The Capitalization table contains the following values: Ab Cd Ef Gh All values are capitalized.
This example shows how to force case upper. FORCE Case Upper; CaseUpper: LOAD * Inline [ab cd eF GH];	The CaseUpper table contains the following values: AB CD EF GH All values are upper case.
This example shows how to force case lower. FORCE Case Lower; CaseLower: LOAD * Inline [ab cd eF GH];	The CaseLower table contains the following values: ab cd ef gh All values are lower case.
This example shows how to force case mixed. FORCE Case Mixed; CaseMixed: LOAD * Inline [ab cd eF GH];	The CaseMixed table contains the following values: ab cd eF Gh All values are as they appear in the script.

See also:

From

The **From** script keyword is used in **Load** statements to refer to a file, and in **Select** statements to refer to a database table or view.

Load

The **LOAD** statement loads fields from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically. It is also possible to load data from analytic connections.

Syntax:

```
LOAD [ distinct ] fieldlist
[ ( from file [ format-spec ] |
from_field fieldassource [ format-spec ] |
inline data [ format-spec ] |
resident table-label |
autogenerate size ) | extension pluginname.functionname([script]
tabledescription) ]
[ where criterion | while criterion ]
[ group by groupbyfieldlist ]
[order by orderbyfieldlist ]
```

Arguments:

Arguments

Argument	Description
distinct	You can use distinct as a predicate if you only want to load unique records. If there are duplicate records, the first instance will be loaded. If you are using preceding loads, you need to place distinct in the first load statement, as distinct only affects the destination table.

Argument	Description
fieldlist	<p><i>fieldlist ::= (* field {, * field})</i> A list of the fields to be loaded. Using * as a field list indicates all fields in the table.</p> <p><i>field ::= (fieldref expression) [as aliasname]</i> The field definition must always contain a literal, a reference to an existing field, or an expression.</p> <p><i>fieldref ::= (fieldname @fieldnumber @startpos:endpos [I U R B T])</i> <i>fieldname</i> is a text that is identical to a field name in the table. Note that the field name must be enclosed by straight double quotation marks or square brackets if it contains e.g. spaces. Sometimes field names are not explicitly available. Then a different notation is used:</p> <p>@<i>fieldnumber</i> represents the field number in a delimited table file. It must be a positive integer preceded by "@". The numbering is always made from 1 and up to the number of fields.</p> <p>@<i>startpos:endpos</i> represents the start and end positions of a field in a file with fixed length records. The positions must both be positive integers. The two numbers must be preceded by "@" and separated by a colon. The numbering is always made from 1 and up to the number of positions. In the last field, n is used as end position.</p> <ul style="list-style-type: none"> • If @<i>startpos:endpos</i> is immediately followed by the characters I or U, the bytes read will be interpreted as a binary signed (I) or unsigned (U) integer (Intel byte order). The number of positions read must be 1, 2 or 4. • If @<i>startpos:endpos</i> is immediately followed by the character R, the bytes read will be interpreted as a binary real number (IEEE 32-bit or 64 bit floating point). The number of positions read must be 4 or 8. • If @<i>startpos:endpos</i> is immediately followed by the character B, the bytes read will be interpreted as a BCD (Binary Coded Decimal) numbers according to the COMP-3 standard. Any number of bytes may be specified. <p><i>expression</i> can be a numeric function or a string function based on one or several other fields in the same table. For further information, see the syntax of expressions.</p> <p>as is used for assigning a new name to the field.</p>

Argument	Description
from	<p>from is used if data should be loaded from a file using a folder or a web file data connection.</p> <p><i>file ::= [path] filename</i></p> <p>Example: 'lib://Table Files'</p> <p>If the path is omitted, Qlik Sense searches for the file in the directory specified by the Directory statement. If there is no Directory statement, Qlik Sense searches in the working directory, <code>C:\Users\{user}\Documents\Qlik\Sense\Apps</code>.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>In a Qlik Sense server installation, the working directory is specified in Qlik Sense Repository Service, by default it is C:\ProgramData\Qlik\Sense\Apps.</i> </div> <p>The <i>filename</i> may contain the standard DOS wildcard characters (* and ?). This will cause all the matching files in the specified directory to be loaded.</p> <p><i>format-spec ::= (fspec-item {, fspec-item})</i></p> <p>The format specification consists of a list of several format specification items, within brackets.</p> <p>Legacy scripting mode</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> • absolute <p>Example: c:\data\</p> <ul style="list-style-type: none"> • relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none"> • URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://wwwqlik.com</p>
from_field	<p>from_field is used if data should be loaded from a previously loaded field.</p> <p><i>fieldassource::=(tablename, fieldname)</i></p> <p>The field is the name of the previously loaded <i>tablename</i> and <i>fieldname</i>.</p> <p><i>format-spec ::= (fspec-item {, fspec-item})</i></p> <p>The format specification consists of a list of several format specification items, within brackets.</p>

2 Script statements and keywords

Argument	Description
inline	<p>inline is used if data should be typed within the script, and not loaded from a file.</p> <p><i>data ::= [text]</i></p> <p>Data entered through an inline clause must be enclosed by double quotation marks or by square brackets. The text between these is interpreted in the same way as the content of a file. Hence, where you would insert a new line in a text file, you should also do it in the text of an inline clause, i.e. by pressing the Enter key when typing the script. The number of columns are defined by the first line.</p> <p><i>format-spec ::= (fspec-item {, fspec-item})</i></p> <p>The format specification consists of a list of several format specification items, within brackets.</p>
resident	<p>resident is used if data should be loaded from a previously loaded table.</p> <p><i>table label</i> is a label preceding the LOAD or SELECT statement(s) that created the original table. The label should be given with a colon at the end.</p>
autogenerate	<p>autogenerate is used if data should be automatically generated by Qlik Sense.</p> <p><i>size ::= number</i></p> <p><i>Number</i> is an integer indicating the number of records to be generated.</p> <p>The field list must not contain expressions which require data from an external data source or a previously loaded table, unless you refer to a single field value in a previously loaded table with the Peek function.</p>

Argument	Description
extension	<p>You can load data from analytic connections. You need to use the extension clause to call a function defined in the server-side extension (SSE) plugin, or evaluate a script.</p> <p>You can send a single table to the SSE plugin, and a single data table is returned. If the plugin does not specify the names of the fields that are returned, the fields will be named Field1, Field2, and so on.</p> <pre>Extension pluginname.functionname(tabledescription);</pre> <ul style="list-style-type: none"> • Loading data using a function in an SSE plugin <i>tabledescription ::= (table { ,tablefield})</i> If you do not state table fields, the fields will be used in load order. • Loading data by evaluating a script in an SSE plugin <i>tabledescription ::= (script, table { ,tablefield})</i> <p>Data type handling in the table field definition</p> <p>Data types are automatically detected in analytic connections. If the data has no numeric values and at least one non-NULL text string, the field is considered as text. In any other case it is considered as numeric.</p> <p>You can force the data type by wrapping a field name with String() or Mixed().</p> <ul style="list-style-type: none"> • String() forces the field to be text. If the field is numeric, the text part of the dual value is extracted, there is no conversion performed. • Mixed() forces the field to be dual. <p>String() or Mixed() cannot be used outside extension table field definitions, and you cannot use other Qlik Sense functions in a table field definition.</p> <p>More about analytic connections</p> <p>You need to configure analytic connections before you can use them.</p>
where	where is a clause used for stating whether a record should be included in the selection or not. The selection is included if <i>criterion</i> is True. <i>criterion</i> is a logical expression.
while	while is a clause used for stating whether a record should be repeatedly read. The same record is read as long as <i>criterion</i> is True. In order to be useful, a while clause must typically include the IterNo() function. <i>criterion</i> is a logical expression.

Argument	Description
group by	<p>group by is a clause used for defining over which fields the data should be aggregated (grouped). The aggregation fields should be included in some way in the expressions loaded. No other fields than the aggregation fields may be used outside aggregation functions in the loaded expressions.</p> <p><code>groupbyfieldlist ::= (fieldname {,fieldname})</code></p>
order by	<p>order by is a clause used for sorting the records of a resident table before they are processed by the load statement. The resident table can be sorted by one or more fields in ascending or descending order. The sorting is made primarily by numeric value and secondarily by national collation order. This clause may only be used when the data source is a resident table.</p> <p>The ordering fields specify which field the resident table is sorted by. The field can be specified by its name or by its number in the resident table (the first field is number 1).</p> <p><code>orderbyfieldlist ::= fieldname [sortorder] {, fieldname [sortorder] }</code></p> <p><code>sortorder</code> is either <code>asc</code> for ascending or <code>desc</code> for descending. If no <code>sortorder</code> is specified, <code>asc</code> is assumed.</p> <p><code>fieldname</code>, <code>path</code>, <code>filename</code> and <code>aliasname</code> are text strings representing what the respective names imply. Any field in the source table can be used as <code>fieldname</code>. However, fields created through the <code>as</code> clause (<code>aliasname</code>) are out of scope and cannot be used inside the same load statement.</p>

If no source of data is given by means of a **from**, **inline**, **resident**, **from_field**, **extension** or **autogenerate** clause, data will be loaded from the result of the immediately succeeding **SELECT** or **LOAD** statement. The succeeding statement should not have a prefix.

Examples:

Loading different file formats

Load a delimited data file with default options:

```
LOAD * from data1.csv;
```

Load a delimited data file from a library connection (DataFiles):

```
LOAD * from 'lib://DataFiles/data1.csv';
```

Load all delimited data files from a library connection (DataFiles):

```
LOAD * from 'lib://DataFiles/*.csv';
```

Load a delimited file, specifying comma as delimiter and with embedded labels:

```
LOAD * from 'c:\userfiles\data1.csv' (ansi, txt, delimiter is ',', embedded labels);
```

Load a delimited file specifying tab as delimiter and with embedded labels:

2 Script statements and keywords

```
LOAD * from 'c:\userfiles\data2.txt' (ansi, txt, delimiter is '\t', embedded labels);
```

Load a dif file with embedded headers:

```
LOAD * from file2.dif (ansi, dif, embedded labels);
```

Load three fields from a fixed record file without headers:

```
LOAD @1:2 as ID, @3:25 as Name, @57:80 as City from data4.fix (ansi, fix, no labels, header is 0, record is 80);
```

Load a QVX file, specifying an absolute path:

```
LOAD * from C:\qdssamples\xyz.qvx (qvx);
```

Loading web files

Load from the default URL set in the web file data connection:

```
LOAD * from [lib://MywebFile];
```

Load from a specific URL, and override the URL set in the web file data connection:

```
LOAD * from [lib://MyWebFile] (URL is 'http://localhost:8000/foo.bar');
```

Load from a specific URL set in a variable using dollar-sign expansion:

```
SET dynamicURL = 'http://localhost/foo.bar';
LOAD * from [lib://MywebFile] (URL is '$(dynamicURL)');
```

Selecting certain fields, renaming and calculating fields

Load only three specific fields from a delimited file:

```
LOAD FirstName, LastName, Number from data1.csv;
```

Rename first field as A and second field as B when loading a file without labels:

```
LOAD @1 as A, @2 as B from data3.txt (ansi, txt, delimiter is '\t', no labels);
```

Load Name as a concatenation of FirstName, a space character, and LastName:

```
LOAD FirstName&' '&LastName as Name from data1.csv;
```

Load Quantity, Price and Value (the product of Quantity and Price):

```
LOAD Quantity, Price, Quantity*Price as value from data1.csv;
```

Selecting certain records

Load only unique records, duplicate records will be discarded:

```
LOAD distinct FirstName, LastName, Number from data1.csv;
```

Load only records where the field Litres has a value above zero:

```
LOAD * from Consumption.csv where Litres>0;
```

Loading data not on file and auto-generated data

Load a table with inline data, two fields named CatID and Category:

```
LOAD * Inline  
[CatID, Category  
0,Regular  
1,Occasional  
2,Permanent];
```

Load a table with inline data, three fields named UserID, Password and Access:

```
LOAD * Inline [UserID, Password, Access  
A, ABC456, User  
B, VIP789, Admin];
```

Load a table with 10 000 rows. Field A will contain the number of the read record (1,2,3,4,5...) and field B will contain a random number between 0 and 1:

```
LOAD RecNo() as A, rand() as B autogenerate(10000);
```



The parenthesis after autogenerate is allowed but not required.

Loading data from a previously loaded table

First we load a delimited table file and name it tab1:

```
tab1:  
SELECT A,B,C,D from 'lib://DataFiles/data1.csv';
```

Load fields from the already loaded tab1 table as tab2:

```
tab2:  
LOAD A,B,month(C),A*B+D as E resident tab1;
```

Load fields from already loaded table tab1 but only records where A is larger than B:

```
tab3:  
LOAD A,A+B+C resident tab1 where A>B;
```

Load fields from already loaded table tab1 ordered by A:

```
LOAD A,B*C as E resident tab1 order by A;
```

Load fields from already loaded table tab1, ordered by the first field, then the second field:

```
LOAD A,B*C as E resident tab1 order by 1,2;
```

Load fields from already loaded table tab1 ordered by C descending, then B in ascending order, and then the first field in descending order:

```
LOAD A,B*C as E resident tab1 order by C desc, B asc, 1 desc;
```

Loading data from previously loaded fields

Load field Types from previously loaded table Characters as A:

2 Script statements and keywords

```
LOAD A from_field (characters, Types);
```

Loading data from a succeeding table (preceding load)

Load A, B and calculated fields X and Y from Table1 that is loaded in succeeding **SELECT** statement:

```
LOAD A, B, if(C>0,'positive','negative') as X, weekday(D) as Y;  
SELECT A,B,C,D from Table1;
```

Grouping data

Load fields grouped (aggregated) by ArtNo:

```
LOAD ArtNo, round(Sum(TransAmount),0.05) as ArtNoTotal from table.csv group by ArtNo;
```

Load fields grouped (aggregated) by Week and ArtNo:

```
LOAD Week, ArtNo, round(Avg(TransAmount),0.05) as WeekArtNoAverages from table.csv group by  
Week, ArtNo;
```

Reading one record repeatedly

In this example we have a input file Grades.csv containing the grades for each student condensed in one field:

```
Student,Grades  
Mike,5234  
John,3345  
Pete,1234  
Paul,3352
```

The grades, in a 1-5 scale, represent subjects Math, English, Science and History. We can separate the grades into separate values by reading each record several times with a **while** clause, using the **IterNo()** function as a counter. In each read, the grade is extracted with the **Mid** function and stored in Grade, and the subject is selected using the **Pick** function and stored in Subject. The final **while** clause contains the test to check if all grades have been read (four per student in this case), which means next student record should be read.

```
MyTab:  
LOAD Student,  
mid(Grades,IterNo(),1) as Grade,  
pick(IterNo(), 'Math', 'English', 'Science', 'History') as Subject from Grades.csv  
while ISNum(mid(Grades,IterNo(),1));
```

The result is a table containing this data:

Student	Subject	Grade
John	English	3
John	History	5
John	Math	3
John	Science	4
Mike	English	2
Mike	History	4
Mike	Math	5
Mike	Science	3
Paul	English	3
Paul	History	2
Paul	Math	3
Paul	Science	5
Pete	English	2
Pete	History	4
Pete	Math	1
Pete	Science	3

Loading from analytic connections

The following sample data is used.

Values:

```
Load
Rand() as A,
Rand() as B,
Rand() as C
AutoGenerate(50);
```

Loading data using a function

In these examples, we assume that we have an analytic connection plugin named *P* that contains a custom function *Calculate(Parameter1, Parameter2)*. The function returns the table *Results* that contains the fields *Field1* and *Field2*.

```
Load * Extension P.Calculate( values{A, C} );
```

Load all fields that are returned when sending the fields A and C to the function.

```
Load Field1 Extension P.Calculate( values{A, C} );
```

Load only the Field1 field when sending the fields A and C to the function.

```
Load * Extension P.Calculate( values );
```

Load all fields that are returned when sending the fields A and B to the function. As fields are not specified, A and B are used as they are the first in order in the table.

```
Load * Extension P.Calculate( values {C, C});
```

Load all fields that are returned when sending the field C to both parameters of the function.

```
Load * Extension P.Calculate( values {string(A), Mixed(B)});
```

Load all fields that are returned when sending the field A forced as a string and B forced as a numeric to the function.

Loading data by evaluating a script

Load A as A_echo, B as B_echo Extension R.ScriptEval('q;', values{A, B});
Load the table returned by the script q when sending the values of A and B.

Load * Extension R.ScriptEval('\$(My_R_Script)', values{A, B});
Load the table returned by the script stored in the My_R_Script variable when sending the values of A and B.

Load * Extension R.ScriptEval('\$(My_R_Script)', values{B as D, *});
Load the table returned by the script stored in the My_R_Script variable when sending the values of B renamed to D, A and C. Using * sends the remaining unreferenced fields.



The file extension of DataFiles connections is case sensitive. For example: .qvd.

Format specification items

Each format specification item defines a certain property of the table file:

```
fspec-item ::= [ ansi | oem | mac | UTF-8 | Unicode | txt | fix | dif | biff | ooxml | html | xml | kml | qvd | qvx delimiter is char | no eof | embedded labels | explicit labels | no labels | table is [tablename] | header is n | header is line | header is n lines | comment is string | record is n | record is line | record is n lines | no quotes | msq | URL is string | userAgent is string]
```

Character set

Character set is a file specifier for the **LOAD** statement that defines the character set used in the file.

The **ansi**, **oem** and **mac** specifiers were used in QlikView and will still work. However, they will not be generated when creating the **LOAD** statement with Qlik Sense.

Syntax:

```
utf8 | unicode | ansi | oem | mac | codepage is
```

Arguments:

Arguments

Argument	Description
utf8	UTF-8 character set
unicode	Unicode character set
ansi	Windows, codepage 1252
oem	DOS, OS/2, AS400 and others
mac	Codepage 10000
codepage is	With the codepage specifier, it is possible to use any Windows codepage as <i>N</i> .

Limitations:

Conversion from the **oem** character set is not implemented for macOS. If nothing is specified, codepage 1252 is assumed under Windows.

Example:

```
LOAD * from a.txt (utf8, txt, delimiter is ',', embedded labels)
LOAD * from a.txt (unicode, txt, delimiter is ',', embedded labels)
LOAD * from a.txt (codepage is 10000, txt, delimiter is ',', no labels)
```

See also:

- ❑ [Load \(page 151\)](#)

Table format

The table format is a file specifier for the **LOAD** statement that defines the file type. If nothing is specified, a *.txt* file is assumed.

Table format types

Type	Description
txt	In a delimited text file the columns in the table are separated by a delimiter character.
fix	In a fixed record file, each field is exactly a certain number of characters. Typically, many fixed record length files contain records separated by a linefeed, but there are more advanced options to specify record size in bytes or to span over more than one line with Record is .
	<p> <i>If the data contains multi-byte characters, field breaks can become misaligned as the format is based on a fixed length in bytes.</i></p>
dif	In a <i>.dif</i> file, (Data Interchange Format) a special format for defining the table is used.
biff	Qlik Sense can also interpret data in standard Excel files by means of the <i>biff</i> format (Binary Interchange File Format).
ooxml	Excel 2007 and later versions use the <i>ooxml .xlsx</i> format.
html	If the table is part of an html page or file, <i>html</i> should be used.
xml	<i>xml</i> (Extensible Markup Language) is a common markup language that is used to represent data structures in a textual format.
qvd	The format <i>qvd</i> is the proprietary QVD files format, exported from a Qlik Sense app.
qvx	<i>qvx</i> is a file/stream format for high performance output to Qlik Sense.

Delimiter is

For delimited table files, an arbitrary delimiter can be specified through the **delimiter is** specifier. This specifier is relevant only for delimited .txt files.

Syntax:

```
delimiter is char
```

Arguments:

Arguments

Argument	Description
char	Specifies a single character from the 127 ASCII characters.

Additionally, the following values can be used:

Optional values

Value	Description
'\t'	representing a tab sign, with or without quotation marks.
'\'	representing a backslash (\) character.
'spaces'	representing all combinations of one or more spaces. Non-printable characters with an ASCII-value below 32, with the exception of CR and LF, will be interpreted as spaces.

If nothing is specified, **delimiter is ','** is assumed.

Example:

```
LOAD * from a.txt (utf8, txt, delimiter is ',', embedded labels);
```

See also:

 [Load \(page 151\)](#)

No eof

The **no eof** specifier is used to disregard end-of-file character when loading delimited .txt files.

Syntax:

```
no eof
```

If the **no eof** specifier is used, characters with code point 26, which otherwise denotes end-of-file, are disregarded and can be part of a field value.

It is relevant only for delimited text files.

Example:

```
LOAD * from a.txt (txt, utf8, embedded labels, delimiter is ' ', no eof);
```

See also:

 [Load \(page 151\)](#)

Labels

Labels is a file specifier for the **LOAD** statement that defines where in a file the field names can be found.

Syntax:

```
embedded labels|explicit labels|no labels
```

The field names can be found in different places of the file. If the first record contains the field names, **embedded labels** should be used. If there are no field names to be found, **no labels** should be used. In *dif* files, a separate header section with explicit field names is sometimes used. In such a case, **explicit labels** should be used. If nothing is specified, **embedded labels** is assumed, also for *dif* files.

Example 1:

```
LOAD * from a.txt (unicode, txt, delimiter is ',', embedded labels
```

Example 2:

```
LOAD * from a.txt (codePage is 1252, txt, delimiter is ',', no labels)
```

See also:

 [Load \(page 151\)](#)

Header is

Specifies the header size in table files. An arbitrary header length can be specified through the **header is** specifier. A header is a text section not used by Qlik Sense.

Syntax:

```
header is n
header is line
header is n lines
```

The header length can be given in bytes (**header is n**), or in lines (**header is line** or **header is n lines**). **n** must be a positive integer, representing the header length. If not specified, **header is 0** is assumed. The **header is** specifier is only relevant for table files.

Example:

This is an example of a data source table containing a header text line that should not be interpreted as data by Qlik Sense.

```
*Header line  
Col1,Col2  
a,B  
c,D
```

Using the **header is 1 lines** specifier, the first line will not be loaded as data. In the example, the **embedded labels** specifier tells Qlik Sense to interpret the first non-excluded line as containing field labels.

```
LOAD Col1, Col2  
FROM 'lib://files/header.txt'  
(txt, embedded labels, delimiter is ',', msq, header is 1 lines);
```

The result is a table with two fields, Col1 and Col2.

See also:

 [Load \(page 151\)](#)

Record is

For fixed record length files, the record length must be specified through the **record is** specifier.

Syntax:

```
Record is n  
Record is line  
Record is n lines
```

Arguments:

Arguments

Argument	Description
n	Specifies the record length in bytes.
line	Specifies the record length as one line.
n lines	Specifies the record length in lines where n is a positive integer representing the record length.

Limitations:

The **record is** specifier is only relevant for **fix** files.

See also:

 [Load \(page 151\)](#)

Quotes

Quotes is a file specifier for the **LOAD** statement that defines whether quotes can be used and the precedence between quotes and separators. For text files only.

Syntax:

```
no quotes  
msq
```

If the specifier is omitted, standard quoting is used, that is, the quotes " " or ' ' can be used, but only if they are the first and last non blank character of a field value.

Arguments:

Arguments	
Argument	Description
no quotes	Used if quotation marks are not to be accepted in a text file.
msq	<p>Used to specify modern style quoting, allowing multi-line content in fields. Fields containing end-of-line characters must be enclosed within double quotes.</p> <p>One limitation of the msq option is that single double-quote ("") characters appearing as first or last character in field content will be interpreted as start or end of multi-line content, which may lead to unpredicted results in the data set loaded. In this case you should use standard quoting instead, omitting the specifier.</p>

XML

This script specifier is used when loading xml files. Valid options for the **XML** specifier are listed in syntax.



You cannot load DTD files in Qlik Sense.

Syntax:

```
xmllsimple
```

See also:

[Load \(page 151\)](#)

KML

This script specifier is used when loading KML files to use in a map visualization.

Syntax:

```
kml
```

The KML file can represent either area data (for example, countries or regions) represented by polygons, line data (for example tracks or roads), or point data (for example, cities or places) represented by points in the form [long, lat].

URL is

This script specifier is used to set the URL of a web file data connection when loading a web file.

Syntax:

```
URL is string
```

Arguments:

Arguments

Argument	Description
string	Specifies the URL of the file to load. This will override the URL set in the web file connection that is used.

Limitations:

The **URL is** specifier is only relevant for web files. You need to use an existing web file data connection.

See also:

 [Load \(page 151\)](#)

userAgent is

This script specifier is used to set the browser user agent when loading a web file.

Syntax:

```
userAgent is string
```

Arguments:

Arguments

Argument	Description
string	Specifies the browser user agent string. This will override the default browser user agent "Mozilla/5.0".

Limitations:

The **userAgent is** specifier is only relevant for web files.

See also:

 [Load \(page 151\)](#)

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '=' at script run time before it is assigned to the variable.

Syntax:

```
Let variablename=expression
```

Examples and results:

Example	Result
<pre>Set x=3+4; Let y=3+4; z=\$(y)+1;</pre>	<p><code>\$(x)</code> will be evaluated as ' 3+4 '</p> <p><code>\$(y)</code> will be evaluated as ' 7 '</p> <p><code>\$(z)</code> will be evaluated as ' 8 '</p> <p>Note the difference between the Set and Let statements. The Set statement assigns the string '3+4' to the variable, whereas the Let statement evaluates the string and assigns 7 to the variable.</p>
<code>Let T=now();</code>	<code>\$(T)</code> will be given the value of the current time.

Loosen Table

One or more Qlik Sense internal data tables can be explicitly declared loosely coupled during script execution by using a **Loosen Table** statement. When a table is loosely coupled, all associations between field values in the table are removed. A similar effect could be achieved by loading each field of the loosely coupled table as independent, unconnected tables. Loosely coupled can be useful during testing to temporarily isolate different parts of the data structure. A loosely coupled table can be identified in the table viewer by the dotted lines. The use of one or more **Loosen Table** statements in the script will make Qlik Sense disregard any setting of tables as loosely coupled made before the script execution.

Syntax:

```
Loosen Tabletablename [ , tablename2 ...]
Loosen Tablestablename [ , tablename2 ...]
```

Either syntax: **Loosen Table** or **Loosen Tables** can be used.



Should Qlik Sense find circular references in the data structure which cannot be broken by tables declared loosely coupled interactively or explicitly in the script, one or more additional tables will be forced loosely coupled until no circular references remain. When this happens, the **Loop Warning** dialog, gives a warning.

Example:

Tab1:
 SELECT * from Trans;
 Loosen Table Tab1;

Map

The **map ... using** statement is used for mapping a certain field value or expression to the values of a specific mapping table. The mapping table is created through the **Mapping** statement.

Syntax:

```
Map fieldlist Using mapname
```

The automatic mapping is done for fields loaded after the **Map ... Using** statement until the end of the script or until an **Unmap** statement is encountered.

The mapping is done last in the chain of events leading up to the field being stored in the internal table in Qlik Sense. This means that mapping is not done every time a field name is encountered as part of an expression, but rather when the value is stored under the field name in the internal table. If mapping on the expression level is required, the **Applymap()** function has to be used instead.

Arguments:

Arguments

Argument	Description
<i>fieldlist</i>	A comma separated list of the fields that should be mapped from this point in the script. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.
<i>mapname</i>	The name of a mapping table previously read in a mapping load or mapping select statement.

Examples and results:

Example	Result
Map Country Using Cmap;	Enables mapping of the field Country using the map Cmap.
Map A, B, C Using X;	Enables mapping of the fields A, B and C using the map X.
Map * Using GenMap;	Enables mapping of all fields using GenMap.

NullAsNull

The **NullAsNull** statement turns off the conversion of NULL values to string values previously set by a **NullAsValue** statement.

Syntax:

```
NullAsNull *fieldlist
```

The **NullAsValue** statement operates as a switch and can be turned on or off several times in the script, using either a **NullAsValue** or a **NullAsNull** statement.

Arguments:

Arguments

Argument	Description
*fieldlist	A comma separated list of the fields for which NullAsNull should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

```
NullAsNull A,B;  
LOAD A,B from x.csv;
```

NullAsValue

The **NullAsValue** statement specifies for which fields that NULL should be converted to a value.

Syntax:

```
NullAsValue *fieldlist
```

By default, Qlik Sense considers NULL values to be missing or undefined entities. However, certain database contexts imply that NULL values are to be considered as special values rather than simply missing values. The fact that NULL values are normally not allowed to link to other NULL values can be suspended by means of the **NullAsValue** statement.

The **NullAsValue** statement operates as a switch and will operate on subsequent loading statements. It can be switched off again by means of the **NullAsNull** statement.

Arguments:

Arguments

Argument	Description
*fieldlist	A comma separated list of the fields for which NullAsValue should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

```
NullAsValue A,B;
Set Nullvalue = 'NULL';
LOAD A,B from x.csv;
```

Qualify

The **Qualify** statement is used for switching on the qualification of field names, i.e. field names will get the table name as a prefix.

Syntax:

```
Qualify *fieldlist
```

The automatic join between fields with the same name in different tables can be suspended by means of the **qualify** statement, which qualifies the field name with its table name. If qualified, the field name(s) will be renamed when found in a table. The new name will be in the form of *tablename.fieldname*. *Tablename* is equivalent to the label of the current table, or, if no label exists, to the name appearing after **from** in **LOAD** and **SELECT** statements.

The qualification will be made for all fields loaded after the **qualify** statement.

Qualification is always turned off by default at the beginning of script execution. Qualification of a field name can be activated at any time using a **qualify** statement. Qualification can be turned off at any time using an **Unqualify** statement.



*The **qualify** statement should not be used in conjunction with partial reload.*

Arguments:

Arguments

Argument	Description
*fieldlist	A comma separated list of the fields for which qualification should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example 1:

```
Qualify B;  
LOAD A,B from x.csv;  
LOAD A,B from y.csv;
```

The two tables **x.csv** and **y.csv** are associated only through **A**. Three fields will result: A, x.B, y.B.

Example 2:

In an unfamiliar database, it is often useful to start out by making sure that only one or a few fields are associated, as illustrated in this example:

```
qualify *;  
unqualify TransID;  
SQL SELECT * from tab1;  
SQL SELECT * from tab2;  
SQL SELECT * from tab3;
```

Only **TransID** will be used for associations between the tables *tab1*, *tab2* and *tab3*.

Rem

The **rem** statement is used for inserting remarks, or comments, into the script, or to temporarily deactivate script statements without removing them.

Syntax:

```
Rem string
```

Everything between the **rem** and the next semicolon ; is considered to be a comment.

There are two alternative methods available for making comments in the script:

1. It is possible to create a comment anywhere in the script - except between two quotes - by placing the section in question between /* and */.
2. When typing // in the script, all text that follows to the right on the same row becomes a comment.
(Note the exception //: that may be used as part of an Internet address.)

Arguments:

Arguments

Argument	Description
string	An arbitrary text.

Example:

```
Rem ** This is a comment **;  
/* This is also a comment */  
// This is a comment as well
```

Rename

The **Rename** script keyword can be used to rename tables or fields that are already loaded.

Rename field

This script function renames one or more existing Qlik Sense field(s) after they have been loaded.



It is not recommended to name a variable identically to a field or a function in Qlik Sense.

Either syntax: **rename field** or **rename fields** can be used.

Syntax:

```
Rename Field (using mapname | oldname to newname{ , oldname to newname })
Rename Fields (using mapname | oldname to newname{ , oldname to newname })
```

Arguments:

Argument	Description
mapname	The name of a previously loaded mapping table containing one or more pairs of old and new field names.
oldname	The old field name.
newname	The new field name.

Limitations:

You cannot rename two fields to having the same name.

Example 1:

```
Rename Field xAZ0007 to Sales;
```

Example 2:

```
FieldMap:
Mapping SQL SELECT oldnames, newnames from datadictionary;
Rename Fields using FieldMap;
```

Rename table

This script function renames one or more existing Qlik Sense internal table(s) after they have been loaded.

Either syntax: **rename table** or **rename tables** can be used.

Syntax:

```
Rename Table (using mapname | oldname to newname{ , oldname to newname })
Rename Tables (using mapname | oldname to newname{ , oldname to newname })
```

Arguments:

Arguments

Argument	Description
mapname	The name of a previously loaded mapping table containing one or more pairs of old and new table names.
oldname	The old table name.
newname	The new table name.

Limitations:

Two differently named tables cannot be renamed to having the same name. The script will generate an error if you try to rename a table to the same name as an existing table.

Example 1:

Tab1:
SELECT * from Trans;
Rename Table Tab1 to Xyz;

Example 2:

TabMap:
Mapping LOAD oldnames, newnames from tabnames.csv;
Rename Tables using TabMap;

Search

The **Search** statement is used for including or excluding fields in smart search.

Syntax:

```
Search Include *fieldlist
Search Exclude *fieldlist
```

You can use several Search statements to refine your selection of fields to include. The statements are evaluated from top to bottom.

Arguments:

Arguments	
Argument	Description
*fieldlist	A comma separated list of the fields to include or exclude from searches in smart search. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Example:

Search examples	
Statement	Description
Search Include *;	Include all fields in searches in smart search.
Search Exclude [*ID];	Exclude all fields ending with ID from searches in smart search.
Search Exclude '*ID';	Exclude all fields ending with ID from searches in smart search.
Search Include ProductID;	Include the field ProductID in searches in smart search.

The combined result of these three statements, in this sequence, is that all fields ending with ID except ProductID are excluded from searches in smart search.

Section

With the **section** statement, it is possible to define whether the subsequent **LOAD** and **SELECT** statements should be considered as data or as a definition of the access rights.

Syntax:

```
Section (access | application)
```

If nothing is specified, **section application** is assumed. The **section** definition is valid until a new **section** statement is made.

Example:

```
Section access;
Section application;
```

Select

The selection of fields from an ODBC data source or OLE DB provider is made through standard SQL **SELECT** statements. However, whether the **SELECT** statements are accepted depends on the ODBC driver or OLE DB provider used. Use of the **SELECT** statement requires an open data connection to the source.

Syntax:

```
Select [all | distinct | distinctrow | top n [percent] ] fieldlist
From tablelist
[where criterion ]
[group by fieldlist [having criterion ] ]
[order by fieldlist [asc | desc] ]
[ (Inner | Left | Right | Full) join tablename on fieldref = fieldref ]
```

Furthermore, several **SELECT** statements can sometimes be concatenated into one through the use of a **union** operator:

```
selectstatement Union selectstatement
```

The **SELECT** statement is interpreted by the ODBC driver or OLE DB provider, so deviations from the general SQL syntax might occur depending on the capabilities of the ODBC drivers or OLE DB provider, for example.:

- **as** is sometimes not allowed, i.e. *aliasname* must follow immediately after *fieldname*.
- **as** is sometimes compulsory if an *aliasname* is used.
- **distinct, as, where, group by, order by, or union** is sometimes not supported.
- The ODBC driver sometimes does not accept all the different quotation marks listed above.



*This is not a complete description of the SQL **SELECT** statement! E.g. **SELECT** statements can be nested, several joins can be made in one **SELECT** statement, the number of functions allowed in expressions is sometimes very large, etc.*

Arguments:

Arguments

Argument	Description
distinct	distinct is a predicate used if duplicate combinations of values in the selected fields only should be loaded once.
distinctrow	distinctrow is a predicate used if duplicate records in the source table only should be loaded once.

Argument	Description
fieldlist	<p>fieldlist ::= (* field) {, field}</p> <p>A list of the fields to be selected. Using * as field list indicates all fields in the table.</p> <p>fieldlist ::= field {, field}</p> <p>A list of one or more fields, separated by commas.</p> <p>field ::= (fieldref expression) [as aliasname]</p> <p>The expression can e.g. be a numeric or string function based on one or several other fields. Some of the operators and functions usually accepted are: +, -, *, /, & (string concatenation), sum(fieldname), count(fieldname), avg(fieldname)(average), month(fieldname), etc. See the documentation of the ODBC driver for more information.</p> <p>fieldref ::= [tablename.] fieldname</p> <p>The tablename and the fieldname are text strings identical to what they imply. They must be enclosed by straight double quotation marks if they contain e.g. spaces.</p> <p>The as clause is used for assigning a new name to the field.</p>
from	<p>tablelist ::= table {, table}</p> <p>The list of tables that the fields are to be selected from.</p> <p>table ::= tablename [[as] aliasname]</p> <p>The tablename may or may not be put within quotes.</p>
where	<p>where is a clause used for stating whether a record should be included in the selection or not.</p> <p>criterion is a logical expression that can sometimes be very complex. Some of the operators accepted are: numeric operators and functions, =, <> or #(not equal), >, >=, <, <=, and, or, not, exists, some, all, in and also new SELECT statements. See the documentation of the ODBC driver or OLE DB providerfor more information.</p>
group by	<p>group by is a clause used for aggregating (group) several records into one. Within one group, for a certain field, all the records must either have the same value, or the field can only be used from within an expression, e.g. as a sum or an average. The expression based on one or several fields is defined in the expression of the field symbol.</p>
having	<p>having is a clause used for qualifying groups in a similar manner to how the where clause is used for qualifying records.</p>
order by	<p>order by is a clause used for stating the sort order of the resulting table of the SELECT statement.</p>
join	<p>join is a qualifier stating if several tables are to be joined together into one. Field names and table names must be put within quotes if they contain blank spaces or letters from the national character sets. When the script is automatically generated by Qlik Sense, the quotation mark used is the one preferred by the ODBC driver or OLE DB provider specified in the data source definition of the data source in the Connect statement.</p>

Example 1:

```
SELECT * FROM `Categories`;
```

Example 2:

```
SELECT `Category ID`, `Category Name` FROM `Categories`;
```

Example 3:

```
SELECT `Order ID`, `Product ID`,  
`Unit Price` * Quantity * (1-Disc) as NetSales  
FROM `Order Details`;
```

Example 4:

```
SELECT `Order Details`.`Order ID`,  
Sum(`Order Details`.`Unit Price` * `Order Details`.Quantity) as `Result`  
FROM `Order Details`, Orders  
where Orders.`Order ID` = `Order Details`.`Order ID`  
group by `Order Details`.`Order ID`;
```

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

Syntax:

```
Set variablename=string
```

Example 1:

```
Set FileToUse=Data1.csv;
```

Example 2:

```
Set Constant="My string";
```

Example 3:

```
Set BudgetYear=2012;
```

Sleep

The **sleep** statement pauses script execution for a specified time.

Syntax:

```
Sleep n
```

Arguments:

Argument	Description
n	Stated in milliseconds, where <i>n</i> is a positive integer no larger than 3600000 (i.e. 1 hour). The value may be an expression.

Example 1:

```
sleep 10000;
```

Example 2:

```
sleep t*1000;
```

SQL

The **SQL** statement allows you to send an arbitrary SQL command through an ODBC or OLE DB connection.

Syntax:

```
SQL sql_command
```

Sending SQL statements which update the database will return an error if Qlik Sense has opened the ODBC connection in read-only mode.

The syntax:

```
SQL SELECT * from tab1;
```

is allowed, and is the preferred syntax for **SELECT**, for reasons of consistency. The SQL prefix will, however, remain optional for **SELECT** statements.

Arguments:

Argument	Description
sql_command	A valid SQL command.

Example 1:

```
SQL leave;
```

Example 2:

```
SQL Execute <storedProc>;
```

SQLColumns

The **sqlcolumns** statement returns a set of fields describing the columns of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

```
SQLcolumns
```

The fields can be combined with the fields generated by the **sqltables** and **sqltypes** commands in order to give a good overview of a given database. The twelve standard fields are:

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

COLUMN_NAME

DATA_TYPE

TYPE_NAME

PRECISION

LENGTH

SCALE

RADIX

NULLABLE

REMARKS

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\Datasrc\QWT.mbd';  
SQLcolumns;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

SQLTables

The **sqltables** statement returns a set of fields describing the tables of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

SQLTables

The fields can be combined with the fields generated by the **sqlcolumns** and **sqltypes** commands in order to give a good overview of a given database. The five standard fields are:

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

TABLE_TYPE

REMARKS

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\Datasrc\QWT.mbd';  
SQLTables;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

SQLTypes

The **sqltypes** statement returns a set of fields describing the types of an ODBC or OLE DB data source, to which a **connect** has been made.

Syntax:

SQLTypes

The fields can be combined with the fields generated by the **sqlcolumns** and **sqltables** commands in order to give a good overview of a given database. The fifteen standard fields are:

```
TYPE_NAME  
DATA_TYPE  
PRECISION  
LITERAL_PREFIX  
LITERAL_SUFFIX  
CREATE_PARAMS  
NULLABLE  
CASE_SENSITIVE  
SEARCHABLE  
UNSIGNED_ATTRIBUTE  
MONEY  
AUTO_INCREMENT  
LOCAL_TYPE_NAME  
MINIMUM_SCALE  
MAXIMUM_SCALE
```

For a detailed description of these fields, see an ODBC reference handbook.

Example:

```
Connect to 'MS Access 7.0 Database; DBQ=C:\Course3\datasrc\QWT.mbd';  
SQLTypes;
```



Some ODBC drivers may not support this command. Some ODBC drivers may produce additional fields.

Star

The string used for representing the set of all the values of a field in the database can be set through the **star** statement. It affects the subsequent **LOAD** and **SELECT** statements.

Syntax:

```
Star is[ string ]
```

Arguments:

Arguments	
Argument	Description
string	An arbitrary text. Note that the string must be enclosed by quotation marks if it contains blanks. If nothing is specified, star is; is assumed, i.e. there is no star symbol available unless explicitly specified. This definition is valid until a new star statement is made.

The **Star is** statement is not recommended for use in the data part of the script (under **Section Application**) if section access is used. The star character is however fully supported for the protected fields in the **Section Access** part of the script. In this case you do not need to use the explicit **Star is** statement since this is always implicit in section access.

Limitations

- You cannot use the star character with key fields; that is, fields that link tables.
- You cannot use the star character with any fields affected by the **Unqualify** statement as this can affect fields that link tables.
- You cannot use the star character with non-logical tables, for example, info-load tables or mapping-load tables.
- When the star character is used in a reducing field (a field that links to the data) in section access , it represents the values listed in this field in section access. It does not represent other values that may exist in the data but are not listed in section access.
- You cannot use the star character with fields affected by any form of data reduction outside the **Section Access** area.

Example

The example below is an extract of a data load script featuring section access.

```
Star is *;

Section Access;
LOAD * INLINE [
ACCESS, USERID, OMIT
ADMIN, ADMIN,
USER, USER1, SALES
USER, USER2, WAREHOUSE
USER, USER3, EMPLOYEES
USER, USER4, SALES
USER, USER4, WAREHOUSE
USER, USER5, *
];

Section Application;
LOAD * INLINE [
SALES, WAREHOUSE, EMPLOYEES, ORDERS
1, 2, 3, 4
];
```

The following applies:

- The *Star* sign is `*`.
- The user `ADMIN` sees all fields. Nothing is omitted.
- The user `USER1` is not able to see the field `SALES`.
- The user `USER2` is not able to see the field `WAREHOUSE`.
- The user `USER3` cannot see the field `EMPLOYEES`.
- The user `USER4` is added twice to the solution to OMIT two fields for this user, `SALES` and `WAREHOUSE`.
- The `USER5` has a `**` added which means that all listed fields in OMIT are unavailable, that is, user `USER5` cannot see the fields `SALES`, `WAREHOUSE` and `EMPLOYEES` but this user can see the field `ORDERS`.

Store

The **Store** statement creates a QVD, or text file.

Syntax:

```
Store [ fieldlist from] table into filename [ format-spec ];
```

The statement will create an explicitly named QVD, or text file.

The statement can only export fields from one data table. If fields from several tables are to be exported, an explicit join must be made previously in the script to create the data table that should be exported.

The text values are exported to the CSV file in UTF-8 format. A delimiter can be specified, see **LOAD**. The **store** statement to a CSV file does not support BIFF export.

Arguments:

Store command arguments

Argument	Description
<code>fieldlist ::= (* field) { , field }</code>	<p>A list of the fields to be selected. Using * as field list indicates all fields.</p> <p><code>field ::= fieldname [as aliasname]</code></p> <p><code>fieldname</code> is a text that is identical to a field name in <code>table</code>. (Note that the field name must be enclosed by straight double quotation marks or square brackets if it contains spaces or other non-standard characters.)</p> <p><code>aliasname</code> is an alternate name for the field to be used in the resulting QVD or CSV file.</p>
<code>table</code>	A script label representing an already loaded table to be used as source for data.
<code>filename</code>	<p>The name of the target file including a valid path to an existing folder data connection.</p> <p>Example: 'lib://Table Files/target.qvd'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: c:\data\sales.qvd</p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: data\sales.qvd</p> <p>If the path is omitted, Qlik Sense stores the file in the directory specified by the Directory statement. If there is no Directory statement, Qlik Sense stores the file in the working directory, <code>C:\Users\{user}\Documents\Qlik\Sense\Apps</code>.</p>
<code>format-spec ::= (txt qvd)</code>	<p>You can set the format specification to either of these file formats. If the format specification is omitted, qvd is assumed.</p> <ul style="list-style-type: none"> txt for text files. qvd for qvd files.

Examples:

```
Store mytable into xyz.qvd (qvd);
Store * from mytable into 'lib://FolderConnection/myfile.qvd';
Store Name, RegNo from mytable into xyz.qvd;
Store Name as a, RegNo as b from mytable into 'lib://FolderConnection/myfile.qvd';
Store mytable into myfile.txt (txt);
Store * from mytable into 'lib://FolderConnection/myfile.qvd';
```



The file extension of DataFiles connections is case sensitive. For example: .qvd.

Table/Tables

The **Table** and **Tables** script keywords are used in **Drop**, **Comment** and **Rename** statements, as well as a format specifier in **Load** statements.

Tag

This script statement provides a way to assign tags to one or more fields or tables. If an attempt to tag a field or table not present in the app is made, the tagging will be ignored. If conflicting occurrences of a field or tag name are found, the last value is used.

Syntax:

```
Tag [field|fields] fieldlist with tagname
```

```
Tag [field|fields] fieldlist using mapname
```

```
Tag table tablelist with tagname
```

Arguments

Argument	Description
fieldlist	One or several fields that should be tagged, in a comma separated list.
mapname	The name of a mapping table previously loaded in a mapping Load or mapping Select statement.
tablelist	A comma separated list of the tables that should be tagged.
tagname	The name of the tag that should be applied to the field.

Example 1:

```
tagmap:
mapping LOAD * inline [
a,b
Alpha,MyTag
Num,MyTag
];
tag fields using tagmap;
```

Example 2:

```
tag field Alpha with 'MyTag2';
```

Trace

The **trace** statement writes a string to the **Script Execution Progress** window and to the script log file, when used. It is very useful for debugging purposes. Using \$-expansions of variables that are calculated prior to the **trace** statement, you can customize the message.

Syntax:

```
Trace string
```

Example 1:

The following statement can be used right after the Load statement that loads the 'Main' table.

```
Trace Main table loaded;
```

This will display the text 'Main table loaded' in the script execution dialog and in the log file.

Example 2:

The following statements can be used right after the Load statement that loads the 'Main' table.

```
Let MyMessage = NoOfRows('Main') & ' rows in Main table';
```

```
Trace ${MyMessage};
```

This will display a text showing the number of rows in the script execution dialog and in the log file, for example, '265,391 rows in Main table' .

Unmap

The **Unmap** statement disables field value mapping specified by a previous **Map ... Using** statement for subsequently loaded fields.

Syntax:

```
Unmap *fieldlist
```

Arguments:

Arguments

Argument	Description
*fieldlist	a comma separated list of the fields that should no longer be mapped from this point in the script. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used.

Examples and results:

Example	Result
Unmap Country;	Disables mapping of field Country.
Unmap A, B, C;	Disables mapping of fields A, B and C.
Unmap * ;	Disables mapping of all fields.

Unqualify

The **Unqualify** statement is used for switching off the qualification of field names that has been previously switched on by the **Qualify** statement.

Syntax:

```
Unqualify *fieldlist
```

Arguments:

Arguments

Argument	Description
*fieldlist	A comma separated list of the fields for which qualification should be turned on. Using * as field list indicates all fields. The wildcard characters * and ? are allowed in field names. Quoting of field names may be necessary when wildcards are used. Refer to the documentation for the Qualify statement for further information.

Example 1:

In an unfamiliar database, it is often useful to start out by making sure that only one or a few fields are associated, as illustrated in this example:

```
qualify *;
unqualify TransID;
SQL SELECT * from tab1;
SQL SELECT * from tab2;
SQL SELECT * from tab3;
```

First, qualification is turned on for all fields.

Then qualification is turned off for **TransID**.

Only **TransID** will be used for associations between the tables *tab1*, *tab2* and *tab3*. All other fields will be qualified with the table name.

Untag

This script statement provides a way to remove tags from fields or tables. If an attempt to untag a field or table not present in the app is made, the untagging will be ignored.

Syntax:

```
Untag [field|fields] fieldlist with tagname
```

```
Untag [field|fields] fieldlist using mapname
```

```
Untag table tablelist with tagname
```

Arguments:

Arguments

Argument	Description
fieldlist	One or several fields which tags should be removed, in a comma separated list.
mapname	The name of a mapping table previously loaded in a mapping LOAD or mapping SELECT statement.
tablelist	A comma separated list of the tables that should be untagged.
tagname	The name of the tag that should be removed from the field.

Example 1:

```
tagmap:  
mapping LOAD * inline [  
a,b  
Alpha,MyTag  
Num,MyTag  
];  
Untag fields using tagmap;
```

Example 2:

```
Untag field Alpha with MyTag2;
```

2.6 Working directory

If you are referencing a file in a script statement and the path is omitted, Qlik Sense searches for the file in the following order:

1. The directory specified by a **Directory** statement (only supported in legacy scripting mode).
2. If there is no **Directory** statement, Qlik Sense searches in the working directory.

Qlik Sense Desktop working directory

In Qlik Sense Desktop, the working directory is *C:\Users\{user}\Documents\Qlik\Sense\Apps*.

Qlik Sense working directory

In a Qlik Sense server installation, the working directory is specified in Qlik Sense Repository Service, by default it is *C:\ProgramData\Qlik\Sense\Apps*. See the Qlik Management Console help for more information.

2 Working with variables in the data load editor

A variable in Qlik Sense is a container storing a static value or a calculation, for example a numeric or alphanumeric value. When you use the variable in the app, any change made to the variable is applied everywhere the variable is used. You can define variables in the variables overview, or in the script using the data load editor. You set the value of a variable using **Let** or **Set** statements in the data load script.



You can also work with the Qlik Sense variables from the variables overview when editing a sheet.

2.7 Overview

If the first character of a variable value is an equals sign '=' Qlik Sense will try to evaluate the value as a formula (Qlik Sense expression) and then display or return the result rather than the actual formula text.

When used, the variable is substituted by its value. Variables can be used in the script for dollar sign expansion and in various control statements. This is very useful if the same string is repeated many times in the script, for example, a path.

Some special system variables will be set by Qlik Sense at the start of the script execution regardless of their previous values.

2.8 Defining a variable

Variables provide the ability to store static values or the result of a calculation. When defining a variable, use the following syntax:

```
set variablename = string
```

or

```
let variable = expression
```

The **Set** statement is used for string assignment. It assigns the text to the right of the equal sign to the variable. The **Let** statement evaluates an expression to the right of the equal sign at script run time and assigns the result of the expression to the variable.

Variables are case sensitive.



It is not recommended to name a variable identically to a field or a function in Qlik Sense.

Examples:

```
set x = 3 + 4; // the variable will get the string '3 + 4' as the value.
```

```
let x = 3 + 4; // returns 7 as the value.
```

```
set x = Today(); // returns 'Today()' as the value.  
let x = Today(); // returns today's date as the value, for example, '9/27/2021'.
```

2.9 Deleting a variable

If you remove a variable from the script and reload the data, the variable stays in the app. If you want to fully remove the variable from the app, you must also delete the variable from the variables dialog.

2.10 Loading a variable value as a field value

If you want to load a variable value as a field value in a **LOAD** statement and the result of the dollar expansion is text rather than numeric or an expression then you need to enclose the expanded variable in single quotes.

Example:

This example loads the system variable containing the list of script errors to a table. You can note that the expansion of ScriptErrorCount in the **If** clause does not require quotes, while the expansion of ScriptErrorList requires quotes.

```
IF $(ScriptErrorCount) >= 1 THEN  
    LOAD '$(ScriptErrorList)' AS Error AutoGenerate 1;  
END IF
```

2.11 Variable calculation

There are several ways to use variables with calculated values in Qlik Sense, and the result depends on how you define it and how you call it in an expression.

In this example, we load some inline data:

```
LOAD * INLINE [  
    Dim, Sales  
    A, 150  
    A, 200  
    B, 240  
    B, 230  
    C, 410  
    C, 330  
];
```

Let's define two variables:

```
Let vsales = 'Sum(Sales)' ;  
Let vsales2 = '=Sum(Sales)' ;
```

In the second variable, we add an equal sign before the expression. This will cause the variable to be calculated before it is expanded and the expression is evaluated.

If you use the vSales variable as it is, for example in a measure, the result will be the string Sum(Sales), that is, no calculation is performed.

2 Working with variables in the data load editor

If you add a dollar-sign expansion and call `$(vSales)` in the expression, the variable is expanded, and the sum of Sales is displayed.

Finally, if you call `$(vSales2)`, the variable will be calculated before it is expanded. This means that the result displayed is the total sum of Sales. The difference between using `=$(vSales)` and `=$(vSales2)` as measure expressions is seen in this chart showing the results:

Results		
Dim	<code>\$(vSales)</code>	<code>\$(vSales2)</code>
A	350	1560
B	470	1560
C	740	1560

As you can see, `$(vSales)` results in the partial sum for a dimension value, while `$(vSales2)` results in the total sum.

The following script variables are available:

- *Error variables (page 262)*
- *Number interpretation variables (page 199)*
- *System variables (page 191)*
- *Value handling variables (page 197)*

2.12 System variables

System variables, some of which are system-defined, provide information about the system and the Qlik Sense app.

System variables overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

CreateSearchIndexOnReload

This variable defines if search index files should be created during data reload.

CreateSearchIndexOnReload

Floppy

Returns the drive letter of the first floppy drive found, normally `a:`. This is a system-defined variable.

Floppy



This variable is not supported in standard mode.

2 Working with variables in the data load editor

CD

Returns the drive letter of the first CD-ROM drive found. If no CD-ROM is found, then c: is returned. This is a system-defined variable.

CD



This variable is not supported in standard mode.

HidePrefix

All field names beginning with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

HidePrefix

HideSuffix

All field names ending with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

HideSuffix

Include

The **Include/Must_Include** variable specifies a file that contains text that should be included in the script and evaluated as script code. It is not used to add data. You can store parts of your script code in a separate text file and reuse it in several apps. This is a user-defined variable.

```
$ (Include=filename)  
$ (Must_Include=filename)
```

OpenUrlTimeout

This variable defines the timeout in seconds that Qlik Sense should respect when getting data from URL sources (e.g. HTML pages). If omitted, the timeout is about 20 minutes.

OpenUrlTimeout

QvPath

Returns the browse string to the Qlik Sense executable. This is a system-defined variable.

QvPath



This variable is not supported in standard mode.

QvRoot

Returns the root directory of the Qlik Sense executable. This is a system-defined variable.

QvRoot



This variable is not supported in standard mode.

2 Working with variables in the data load editor

QvWorkPath

Returns the browse string to the current Qlik Sense app. This is a system-defined variable.

QvWorkPath



This variable is not supported in standard mode.

QvWorkRoot

Returns the root directory of the current Qlik Sense app. This is a system-defined variable.

QvWorkRoot



This variable is not supported in standard mode.

StripComments

If this variable is set to 0, stripping of /*..*/ and // comments in the script will be inhibited. If this variable is not defined, stripping of comments will always be performed.

StripComments

Verbatim

Normally all field values are automatically stripped of leading and trailing blanks (ASCII 32) before being loaded into the Qlik Sense database. Setting this variable to 1 suspends the stripping of blanks. Tab (ASCII 9) and hard space (ANSI 160) characters are never stripped.

Verbatim

WinPath

Returns the browse string to Windows. This is a system-defined variable.

WinPath



This variable is not supported in standard mode.

WinRoot

Returns the root directory of Windows. This is a system-defined variable.

WinRoot



This variable is not supported in standard mode.

CollationLocale

Specifies which locale to use for sort order and search matching. The value is the culture name of a locale, for example 'en-US'.This is a system-defined variable.

CollationLocale

CreateSearchIndexOnReload

This variable defines if search index files should be created during data reload.

Syntax:

CreateSearchIndexOnReload

You can define if search index files should be created during data reload, or if they should be created after the first search request of the user. The benefit of creating search index files during data reload is that you avoid the waiting time experienced by the first user making a search. This needs to be weighed against the longer data reload time required by search index creation.

If this variable is omitted, search index files will not be created during data reload.



For session apps, search index files will not be created during data reload, regardless of the setting of this variable.

Example 1: Create search index fields during data reload

```
set CreateSearchIndexOnReload=1;
```

Example 2: Create search index fields after first search request

```
set CreateSearchIndexOnReload=0;
```

HidePrefix

All field names beginning with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

Syntax:

HidePrefix

Example:

```
set HidePrefix='_';
```

If this statement is used, the field names beginning with an underscore will not be shown in the field name lists when the system fields are hidden.

HideSuffix

All field names ending with this text string will be hidden in the same manner as the system fields. This is a user-defined variable.

Syntax:

HideSuffix

Example:

```
set HideSuffix='%';
```

If this statement is used, the field names ending with a percentage sign will not be shown in the field name lists when the system fields are hidden.

Include

The **Include/Must_Include** variable specifies a file that contains text that should be included in the script and evaluated as script code. It is not used to add data. You can store parts of your script code in a separate text file and reuse it in several apps. This is a user-defined variable.



This variable supports only folder data connections in standard mode.

Syntax:

```
$ (Include=filename)
```

```
$ (Must_Include=filename)
```

There are two versions of the variable:

- **Include** does not generate an error if the file cannot be found, it will fail silently.
- **Must_Include** generates an error if the file cannot be found.

If you don't specify a path, the filename will be relative to the Qlik Sense app working directory. You can also specify an absolute file path, or a path to a lib:// folder connection. Do not put a space character before or after the equal sign.



*The construction **set Include =filename** is not applicable.*

Examples:

```
$ (Include=abc.txt);
```

```
$ (Must_Include=lib://DataFiles/abc.txt);
```

Limitations

Limited cross-compatibility between UTF-8 encoded files under Windows versus Linux.

It is optional to use UTF-8 with BOM (Byte Order Mark). BOM can interfere with the use of UTF-8 in software that does not expect non-ASCII bytes at the start of a file, but that could otherwise handle the text stream.

- Windows systems use BOM in UTF-8 to identify that a file is UTF-8 encoded, despite the fact that there is no ambiguity in the byte storage.

2 Working with variables in the data load editor

- Unix/Linux use UTF-8 for Unicode, but does not use the BOM as this interferes with the syntax for command files.

This has some implications for Qlik Sense.

- In Windows any file that begins with an UTF-8 BOM is considered a UTF-8 script file. Otherwise ANSI encoding is assumed.
- In Linux, the system default 8 bit code page is UTF-8. This is why the UTF-8 works although it does not contain a BOM.

As a result, portability cannot be guaranteed. It is not always possible to create a file on Windows that can be interpreted by Linux and vice versa. There is no cross compatibility between the two systems regarding UTF-8 encoded files due to different handling of the BOM.

OpenUrlTimeout

This variable defines the timeout in seconds that Qlik Sense should respect when getting data from URL sources (e.g. HTML pages). If omitted, the timeout is about 20 minutes.

Syntax:

```
OpenUrlTimeout
```

Example:

```
set OpenUrlTimeout=10;
```

StripComments

If this variable is set to 0, stripping of /*..*/ and // comments in the script will be inhibited. If this variable is not defined, stripping of comments will always be performed.

Syntax:

```
StripComments
```

Certain database drivers use /*..*/ as optimization hints in **SELECT** statements. If this is the case, the comments should not be stripped before sending the **SELECT** statement to the database driver.



It is recommended that this variable be reset to 1 immediately after the statement(s) where it is needed.

Example:

```
set StripComments=0;
SQL SELECT * /* <optimization directive> */ FROM Table ;
set StripComments=1;
```

Verbatim

Normally all field values are automatically stripped of leading and trailing blanks (ASCII 32) before being loaded into the Qlik Sense database. Setting this variable to 1 suspends the stripping of blanks. Tab (ASCII 9) and hard space (ANSI 160) characters are never stripped.

Syntax:

Verbatim

Example:

```
set verbatim = 1;
```

2.13 Value handling variables

This section describes variables that are used for handling NULL and other values.

Value handling variables overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

NullDisplay

The defined symbol will substitute all NULL values from ODBC, and connectors, on the lowest level of data. This is a user-defined variable.

NullDisplay

NullInterpret

The defined symbol will be interpreted as NULL when it occurs in a text file, Excel file or an inline statement. This is a user-defined variable.

NullInterpret

NullValue

If the **NullAsValue** statement is used, the defined symbol will substitute all NULL values in the **NullAsValue** specified fields with the specified string.

NullValue

OtherSymbol

Defines a symbol to be treated as 'all other values' before a **LOAD/SELECT** statement. This is a user-defined variable.

OtherSymbol

NullDisplay

The defined symbol will substitute all NULL values from ODBC, and connectors, on the lowest level of data. This is a user-defined variable.

Syntax:

```
NullDisplay
```

Example:

```
set NullDisplay='<NULL>';
```

NullInterpret

The defined symbol will be interpreted as NULL when it occurs in a text file, Excel file or an inline statement. This is a user-defined variable.

Syntax:

```
NullInterpret
```

Examples:

```
set NullInterpret=' ';
```

```
set NullInterpret =;
```

will not return NULL values for blank values in Excel, but it will for a CSV text file.

```
set NullInterpret ='';
```

will return NULL values for blank values in Excel.

NullValue

If the **NullAsValue** statement is used, the defined symbol will substitute all NULL values in the **NullAsValue** specified fields with the specified string.

Syntax:

```
NullValue
```

Example:

```
NullAsvalue Field1, Field2;
```

```
set Nullvalue='<NULL>';
```

OtherSymbol

Defines a symbol to be treated as 'all other values' before a **LOAD/SELECT** statement. This is a user-defined variable.

Syntax:

```
OtherSymbol
```

Example:

```
set Othersymbol='+';  
LOAD * inline  
[X, Y  
a, a  
b, b];  
LOAD * inline  
[X, Z  
a, a  
+, c];  
The field value Y='b' will now link to Z='c' through the other symbol.
```

2.14 Number interpretation variables

Number interpretation variables are system defined. The variables are included at the top of the load script and apply number formatting settings at the time of the script execution. They can be deleted, edited, or duplicated.

Number interpretation variables are automatically generated according to the current regional settings of the operating system when a new app is created. In Qlik Sense Desktop, this is according to the settings of the computer operating system. In Qlik Sense, it is according to the operating system of the server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Currency formatting

MoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency set by your regional settings.

MoneyDecimalSep

The symbol defined replaces the currency symbol set by your regional settings.

MoneyFormat

The thousands separator defined replaces the digit grouping symbol for currency set by your regional settings.

MoneyThousandSep

Number formatting

DecimalSep

The decimal separator defined replaces the decimal symbol set by your regional settings.

DecimalSep

ThousandSep

The thousands separator defined replaces the digit grouping symbol of the operating system (regional settings).

ThousandSep

NumericalAbbreviation

The numerical abbreviation sets which abbreviation to use for scale prefixes of numerals, for example M for mega or a million (10^6), and μ for micro (10^{-6}).

NumericalAbbreviation

Time formatting

DateFormat

This environment variable defines the date format used as the default in the app. The format is used both to interpret and format dates. If the variable is not defined, the date format of the regional settings of the operating system will be fetched when the script runs.

DateFormat

TimeFormat

The format defined replaces the time format of the operating system (regional settings).

TimeFormat

TimestampFormat

The format defined replaces the date and time formats of the operating system (regional settings).

TimestampFormat

MonthNames

The format defined replaces the month names convention of the regional settings.

MonthNames

LongMonthNames

The format defined replaces the long month names convention in the regional settings.

LongMonthNames

DayNames

The format defined replaces the weekday names convention set by your regional settings.

DayNames

LongDayNames

The format defined replaces the long weekday names convention in the regional settings.

LongDayNames

FirstWeekDay

Integer that defines which day to use as the first day of the week.

FirstWeekDay

BrokenWeeks

This setting defines if weeks are broken or not.

BrokenWeeks

ReferenceDay

The setting defines which day in January to set as reference day to define week 1.

ReferenceDay

FirstMonthOfYear

The setting defines which month to use as first month of the year, which can be used to define financial years that use a monthly offset, for example starting April 1.



This setting is currently unused but reserved for future use.

Valid settings are 1 (January) to 12 (December). Default setting is 1.

Syntax:

FirstMonthOfYear

Example:

```
Set FirstMonthOfYear=4; //Sets the year to start in April
```

BrokenWeeks

This setting defines if weeks are broken or not.

Syntax:

BrokenWeeks

In Qlik Sense, the regional settings are fetched when the app is created, and the corresponding settings are stored in the script as environment variables.

A North American app developer often gets `Set Brokenweeks=1;` in the script, corresponding to broken weeks. A European app developer often gets `Set Brokenweeks=0;` in the script, corresponding to unbroken weeks.

Unbroken weeks means that:

- In some years, week 1 starts in December, and in other years, the last week of previous year continues into January.
- According to ISO 8601, week 1 always has at least 4 days in January. In Qlik Sense, this can be configured using the `ReferenceDay` variable.

Broken weeks means that:

2 Working with variables in the data load editor

- The last week of the year never continues into January.
- Week 1 starts on January 1 and is, in most cases, not a full week.

The following values can be used:

- 0 (=use unbroken weeks)
- 1 (= use broken weeks)

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Examples:

If you want ISO settings for weeks and week numbers, make sure to have the following in the script:

```
Set FirstWeekDay=0;  
Set BrokenWeeks=0; // (use unbroken weeks)  
Set ReferenceDay=4;
```

If you want US settings, make sure to have the following in the script:

```
Set FirstWeekDay=6;  
Set BrokenWeeks=1; // (use broken weeks)  
Set ReferenceDay=1;
```

DateFormat

This environment variable defines the date format used as the default in the app and by date returning functions like `date()` and `date#()`. The format is used to interpret and format dates. If the variable is not defined, the date format set by your regional settings is fetched when the script runs.

Syntax:

DateFormat

DateFormat Function examples

Example	Result
<code>Set DateFormat='M/D/YY'; // (US format)</code>	This use of the <code>DateFormat</code> function defines the date as the US format, month/day/year.

Example	Result
<pre>Set DateFormat='DD/MM/YY'; // (UK date format)</pre>	This use of the <code>DateFormat</code> function defines the date as the UK format, day/month/year.
<pre>Set DateFormat='YYYY/MM/DD'; // (ISO date format)</pre>	This use of the <code>DateFormat</code> function defines the date as the ISO format, year/month/day.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – System variables default

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates.
- The `DateFormat` function, which will use the US date format.

In this example, a dataset is loaded into a table named 'Transactions'. It includes a date field. The US `DateFormat` definition is used. This pattern will be used for implicit text to date conversion when the text dates are loaded.

Load script

```
Set DateFormat='MM/DD/YYYY';
```

Transactions:

```
LOAD  
date,  
month(date) as month,  
id,  
amount  
INLINE  
[  
date,id,amount
```

```
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- month

Create this measure:

```
=sum(amount)
```

Results table

date	month	=sum(amount)
01/01/2022	Jan	1000
02/01/2022	Feb	2123
03/01/2022	Mar	4124
04/01/2022	Apr	2431

The `DateFormat` definition `MM/DD/YYYY` is used for implicit conversion of text to dates, which is why the `date` field is properly interpreted as a date. The same format is used to display the date, as shown in the results table.

Example 2 – Change system variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the previous example.
- The `DateFormat` function, which will use the '`DD/MM/YYYY`' format.

Load script

```
SET DateFormat='DD/MM/YYYY';  
Transactions:  
LOAD  
date,  
month(date) as month,  
id,  
amount
```

2 Working with variables in the data load editor

```
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- month

Create this measure:

```
=sum(amount)
```

Results table

date	month	=sum(amount)
01/01/2022	Jan	1000
02/01/2022	Jan	2123
03/01/2022	Jan	4124
04/01/2022	Jan	2431

Because the `DateFormat` definition was set to ‘DD/MM/YYYY’, you can see that the two digits after the first “/” symbol have been interpreted as the month, resulting in all records being from the month of January.

Example 3 – Date interpretation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset with dates in numerical format.
- The `DateFormat` variable, which will use the ‘DD/MM/YYYY’ format.
- The `date()` variable.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

Load

2 Working with variables in the data load editor

```
date(numerical_date),
month(date(numerical_date)) as month,
id,
amount
Inline
[
numerical_date,id,amount
43254,1,1000
43255,2,2123
43256,3,4124
43258,4,2431
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- month

Create this measure:

```
=sum(amount)
```

Results table

date	month	=sum(amount)
06/03/2022	Jun	1000
06/04/2022	Jun	2123
06/05/2022	Jun	4124
06/07/2022	Jun	2431

In the load script, you use the `date()` function to convert the numerical date into a date format. Because you do not provide a specified format as a second argument in the function, the `DateFormat` is used. This results in the date field using the format 'MM/DD/YYYY'.

Example 4 – Foreign date formatting

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates.
- The `DateFormat` variable, which uses the 'DD/MM/YYYY' format but is uncommented by forward slashes.

Load script

```
// SET DateFormat='DD/MM/YYYY';
```

Transactions:

```
Load  
date,  
month(date) as month,  
id,  
amount  
Inline  
[  
    date,id,amount  
    22-05-2022,1,1000  
    23-05-2022,2,2123  
    24-05-2022,3,4124  
    25-05-2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- month

Create this measure:

```
=sum(amount)
```

Results table		
date	month	=sum(amount)
22-05-2022	-	1000
23-05-2022	-	2123
24-05-2022	-	4124
25-05-2022	-	2431

In the initial load script, the `DateFormat` being used is the default ‘MM/DD/YYYY’. Because the `date` field in the transactions dataset is not in this format, the field is not interpreted as a date. This is shown in the results table where the `month` field values are null.

You can verify the interpreted data types in the Data model viewer by inspecting the `date` field’s “Tags” properties:

2 Working with variables in the data load editor

Preview of the *Transactions* table. Note the “Tags” for the *date* field indicating that the textual input data has not been implicitly converted to a date/timestamp.

date		Transactions			
		date	month	id	amount
Density	100%	22-05-2022	-	1	1000
Subset ratio	100%	23-05-2022	-	2	2123
Has duplicates	false	24-05-2022	-	3	4124
Total distinct values	4	25-05-2022	-	4	2431
Present distinct values	4				
Non-null values	4				
Tags	\$ascii \$text				

This can be solved by enabling the *DateFormat* system variable:

```
// SET DateFormat='DD/MM/YYYY';
```

Remove the double forward slashes and reload the data.

Preview of the *Transactions* table. Note the “Tags” for the *date* field indicating that the textual input data has been implicitly converted to a date/timestamp.

date		Transactions			
		date	month	id	amount
Density	100%	22-05-2022	May	1	1000
Subset ratio	100%	23-05-2022	May	2	2123
Has duplicates	false	24-05-2022	May	3	4124
Total distinct values	4	25-05-2022	May	4	2431
Present distinct values	4				
Non-null values	4				
Tags	\$numeric \$integer \$timestamp \$date				

DayNames

The format defined replaces the weekday names convention set by your regional settings.

Syntax:

DayNames

When modifying the variable, a semicolon ; is required to separate the individual values.

DayName Function examples

Function example

```
Set  
DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

Result definition

This use of the DayNames function defines day names in their abbreviated form.

Function example

```
Set DayNames='M;Tu;W;Th;F;Sa;Su';
```

Result definition

This use of the DayNames function defines day names by their first letters.

The DayNames function is often used in combination with the following functions:

Function	Related functions
	Interaction
<i>weekday</i> (page 1037)	Script function to return DayNames as field values .
<i>Date</i> (page 1193)	Script function to return DayNames as field values.
<i>LongDayNames</i> (page 219)	Long form values of DayNames.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - System variables default

Load script and results

Overview

In this example, the dates in the dataset are set in the MM/DD/YYYY format.

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset with dates, which will be loaded into a table named, `Transactions`.
- A date field.
- The default `DayNames` definition.

Load script

```
SET DayNames='Mon;Tue;wed;Thu;Fri;Sat;Sun';
```

`Transactions:`

```
LOAD
```

2 Working with variables in the data load editor

```
date,  
weekDay(date) as dayname,  
id,  
amount  
INLINE  
[  
date,id,amount  
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- dayname

Create this measure:

```
sum(amount)
```

Results table

date	dayname	sum(amount)
01/01/2022	Sat	1000
02/01/2022	Tue	2123
03/01/2022	Tue	4124
04/01/2022	Fri	2431

In the load script, the `weekday` function is used with the `date` field as the provided argument. In the results table, the output of this `weekday` function displays the days of the week in the format of the `DayNames` definition.

Example 2 - Change system variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab. The same dataset and scenario from the first example are used.

However, at the start of the script, the `DayNames` definition is modified to use the abbreviated days of the week in Afrikaans.

Load script

```
SET DayNames='Ma;Di;Wo;Do;Vr;Sa;So';
```

Transactions:

```
Load  
date,  
weekDay(date) as dayname,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- dayname

Create this measure:

```
sum(amount)
```

Results table

date	dayname	sum(amount)
01/01/2022	Sa	1000
02/01/2022	Di	2123
03/01/2022	Di	4124
04/01/2022	Vr	2431

In the results table, the output of this `weekday` function displays the days of the week in the format of the `DayNames` definition.

It is important to remember that if the language for the `DayNames` is modified like it has been in this example, the `LongDayNames` would still contain the days of the week in English. This would need to be modified as well if both variables are used in the application.

Example 3 – Date function

Load script and results

2 Working with variables in the data load editor

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset with dates, which will be loaded into a table named, `Transactions`.
- A date field.
- The default `DayNames` definition.

Load script

```
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

`Transactions`:

```
Load  
date,  
Date(date,'www') as dayname,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `dayname`

Create this measure:

```
sum(amount)
```

Results table		
date	dayname	sum(amount)
01/01/2022	Sat	1000
02/01/2022	Tue	2123
03/01/2022	Tue	4124
04/01/2022	Fri	2431

The default `DayNames` definition is used. In the load script, the `Date` function is used with the `date` field as the first argument. The second argument is `www`. This formatting converts the result into the values stored in the `DayNames` definition. This is displayed in the output of the results table.

DecimalSep

The decimal separator defined replaces the decimal symbol set by your regional settings.

Qlik Sense automatically interprets text as numbers whenever a recognizable number pattern is encountered. The `Thousandsep` and `Decimalsep` system variables determine the makeup of the patterns applied when parsing text as numbers. The `Thousandsep` and `Decimalsep` variables set the default number format pattern when visualizing numeric content in front-end charts and tables. That is, it directly impacts the **Number formatting** options for any front end expression.

Assuming a thousand separator of comma ‘,’ and a decimal separator of ‘.’, these are examples of patterns that would be implicitly converted to numeric equivalent values:

0,000.00

0000.00

0,000

These are examples of patterns that would remain unchanged as text; that is, not converted to numeric:

0.000,00

0,00

Syntax:

`Decimalsep`

Function examples

Example	Result
<code>Set Decimalsep='.';</code>	Sets ‘.’ as the decimal separator.
<code>Set Decimalsep=',';</code>	Sets ‘,’ as the decimal separator.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example – Effect of setting number separator variables on different input data

Load script and results

2 Working with variables in the data load editor

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of sums and dates with the sums set in different format patterns.
- A table named `Transactions`.
- The `DecimalSep` variable which is set to `'.'`.
- The `ThousandSep` variable which is set to `,`.
- The `delimiter` variable that is set as the `'|'` character to separate the different fields in a line.

Load script

```
Set Thousandsep=',';  
Set DecimalSep='.';
```

`Transactions:`

```
Load date,  
id,  
amount as amount  
InLine  
[  
date|id|amount  
01/01/2022|1|1.000-45  
01/02/2022|2|23.344  
01/03/2022|3|4124,35  
01/04/2022|4|2431.36  
01/05/2022|5|4,787  
01/06/2022|6|2431.84  
01/07/2022|7|4132.5246  
01/08/2022|8|3554.284  
01/09/2022|9|3.756,178  
01/10/2022|10|3,454.356  
] (delimiter is '|');
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: `amount`.

Create this measure:

```
=sum(amount)
```

Results table

Amount	=Sum(amount)	
Totals		20814.7086
1.000-45		

Amount	=Sum(amount)
3.756,178	
4124,35	
	23.344
	2431.36
	2431.84
	3,454.356
	3554.284
	4132.5246
	4,787
	23.344
	2431.36
	2431.84
	3454.356
	3554.284
	4132.5246
	4787

Any value not interpreted as number remains as text and is aligned to the left by default. Any successfully converted values are aligned to the right, retaining the original input format.

The expression column shows the numeric equivalent, which is by default formatted with only a decimal separator ‘.’. This can be overridden with the **Number formatting** drop down setting in the expression configuration.

FirstWeekDay

Integer that defines which day to use as the first day of the week.

Syntax:

FirstWeekDay

Monday is the first day of the week according to ISO 8601, the international standard for the representation of dates and times. Monday is also used as the first day of the week in a number of countries, for example on the UK, France, Germany and Sweden.

But in other countries, like in the United States and Canada, Sunday is considered to be the start of the week.

In Qlik Sense, the regional settings are fetched when the app is created, and the corresponding settings are stored in the script as environment variables.

A North American app developer often gets set `Firstweekday=6`; in the script, corresponding to Sunday. A European app developer often gets set `FirstWeekDay=0`; in the script, corresponding to Monday.

Values that can be set for

FirstWeekDay

Value	Day
0	Monday
1	Tuesday
2	Wednesday

Value	Day
3	Thursday
4	Friday
5	Saturday
6	Sunday

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Examples:

If you want ISO settings for weeks and week numbers, make sure to have the following in the script:

```
Set FirstWeekDay=0; // Monday as first week day  
Set BrokenWeeks=0;  
Set ReferenceDay=4;
```

If you want US settings, make sure to have the following in the script:

```
Set FirstWeekDay=6; // Sunday as first week day  
Set BrokenWeeks=1;  
Set ReferenceDay=1;
```

Example 1 – Using default value (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

In this example, the load script uses the default Qlik Sense system variable value, `FirstWeekDay=6`. This data contains data for the first 14 days in 2020.

Load script

```
// Example 1: Load Script using the default value of FirstWeekDay=6, i.e. Sunday
```

2 Working with variables in the data load editor

```
SET FirstWeekDay = 6;

Sales:
LOAD
    date,
    sales,
    week(date) as week,
    weekday(date) as weekday
INLINE [
date,sales
01/01/2021,6000
01/02/2021,3000
01/03/2021,6000
01/04/2021,8000
01/05/2021,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week
- weekday

Results table

Date	week	weekday
01/01/2021	1	Wed
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat
01/05/2021	2	Sun
01/06/2020	2	Mon
01/07/2020	2	Tue
01/08/2020	2	Wed
01/09/2020	2	Thu

Date	week	weekday
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	3	Sun
01/13/2020	3	Mon
01/14/2020	3	Tue

Because the default settings are being used, the `FirstWeekday` system variable is set to 6. In the results table, each new week can be seen beginning on Sunday (the 5th and 12th of January).

Example 2 – Changing the FirstWeekDay variable (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

In this example, the data contains the first 14 days in 2020. At the start of the script, we set the `FirstWeekday` variable to 3.

Load script

```
// Example 2: Load Script setting the value of FirstWeekDay=3, i.e. Thursday

SET FirstWeekDay = 3;

Sales:
LOAD
    date,
    sales,
    week(date) as week,
    weekday(date) as weekday
INLINE [
date,sales
01/01/2021,6000
01/02/2021,3000
01/03/2021,6000
01/04/2021,8000
01/05/2021,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
```

];

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week
- weekday

Results table

Date	week	weekday
01/01/2021	52	Wed
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat
01/05/2021	1	Sun
01/06/2020	1	Mon
01/07/2020	1	Tue
01/08/2020	1	Wed
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	2	Sun
01/13/2020	2	Mon
01/14/2020	2	Tue

Because the `FirstWeekDay` system variable is set to 3, the first day of each week will be a Thursday. In the results table, each new week can be seen beginning on Thursday (the 2nd and 9th of January).

LongDayNames

The format defined replaces the long weekday names convention in the regional settings.

Syntax:

LongDayNames

The following example of the `LongDayNames` function defines day names in full:

```
Set LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';  
When modifying the variable, a semicolon ; is required to separate the individual values.
```

2 Working with variables in the data load editor

The `LongDayNames` function can be used in combination with the `Date` (page 1193) function which returns `DayNames` as field values.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - System variable default

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset with dates, which will be loaded into a table named, `Transactions`.
- A date field.
- The default `LongDayNames` definition.

Load script

```
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';

Transactions:
LOAD
date,
Date(date,'www') as dayname,
id,
amount
INLINE
[
date,id,amount
01/01/2022,1,1000
02/01/2022,2,2123
03/01/2022,3,4124
04/01/2022,4,2431
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- dayname

Create this measure:

```
=sum(amount)
```

Results table

date	dayname	=sum(amount)
01/01/2022	Saturday	1000
02/01/2022	Tuesday	2123
03/01/2022	Tuesday	4124
04/01/2022	Friday	2431

In the load script, to create a field called, dayname, the Date function is used with the date field as the first argument. The second argument in the function is the formatting `www`.

Using this formatting converts the values from the first argument into the corresponding full day name that is set in the variable `LongDayNames`. In the results table, the field values of our created field `dayname` display this.

Example 2 – Change system variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The same dataset and scenario from the first example are used. However, at the start of the script, the `LongDayNames` definition is modified to use the days of the week in Spanish.

Load Script

```
SET LongDayNames='Lunes;Martes;Miércoles;Jueves;Viernes;Sábado;Domingo';
```

Transactions:

```
LOAD  
date,  
Date(date,'www') as dayname,  
id,  
amount  
INLINE  
[  
date,id,amount  
01/01/2022,1,1000
```

2 Working with variables in the data load editor

```
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- dayname

Create this measure:

```
=sum(amount)
```

Results table		
date	dayname	=sum(amount)
01/01/2022	Sábado	1000
02/01/2022	Martes	2123
03/01/2022	Martes	4124
04/01/2022	Viernes	2431

In the load script, the `LongDayNames` variable is modified to list the days of the week in Spanish.

Then, you create a field called, `dayname`, which is the `date` function used with the `date` field as the first argument.

The second argument in the function is the formatting `www`. By using this formatting Qlik Sense converts the values from the first argument into the corresponding full day name set in the variable `LongDayNames`.

In the results table, the field values of our created field `dayname` displays the days of the week written in Spanish and in full.

LongMonthNames

The format defined replaces the long month names convention in the regional settings.

Syntax:

```
LongMonthNames
```

When modifying the variable, the ; needs to be used to separate the individual values.

The following example of the `LongMonthNames` function defines month names in full:

```
Set  
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;December';
```

The `LongMonthNames` function is often used in combination with the following functions:

Related functions	
Function	Interaction
<i>Date</i> (page 1193)	Script function to return DayNamesas field values.
<i>LongDayNames</i> (page 219)	Long form values of DayNames.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - System variables default

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates that is loaded into a table named `Transactions`.
- A date field.
- The default `LongMonthNames` definition.

Load script

```
SET
LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;December';

Transactions:
Load
date,
Date(date,'MMMM') as monthname,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,1000.45
01/02/2022,2,2123.34
```

```
01/03/2022,3,4124.35  
01/04/2022,4,2431.36  
01/05/2022,5,4787.78  
01/06/2022,6,2431.84  
01/07/2022,7,2854.83  
01/08/2022,8,3554.28  
01/09/2022,9,3756.17  
01/10/2022,10,3454.35  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- monthname

Create this measure:

```
=sum(amount)
```

Results table		
date	monthname	sum(amount)
01/01/2022	January	1000.45
01/02/2022	January	2123.34
01/03/2022	January	4124.35
01/04/2022	January	2431.36
01/05/2022	January	4787.78
01/06/2022	January	2431.84
01/07/2022	January	2854.83
01/08/2022	January	3554.28
01/09/2022	January	3756.17
01/10/2022	January	3454.35

The default `LongMonthNames` definition is used. In the load script, to create a field called, `month`, the `date` function is used with the `date` field as the first argument. The second argument in the function is the formatting `MMMM`.

Using this formatting Qlik Sense converts the values from the first argument into the corresponding full month name set in the variable `LongMonthNames`. In the results table, the field values of our created field `month` display this.

Example 2 - Change system variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates that is loaded into a table named `Transactions`.
- A date field.
- The `LongMonthNames` variable that is modified to use the abbreviated days of the week in Spanish.

Load script

```
SET  
LongMonthNames='Enero;Febrero;Marzo;Abril;Mayo;Junio;Julio;Agosto;Septiembre;Octubre;Noviembre;  
Diciembre';
```

`Transactions`:

```
LOAD  
date,  
Date(date,'MMMM') as monthname,  
id,  
amount  
INLINE  
[  
date,id,amount  
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add `sum(amount)` as a measure and these fields as dimensions:

- `date`
- `monthname`

Create this measure:

```
=sum(amount)
```

Results table

date	monthname	sum(amount)
01/01/2022	Enero	1000.45
01/02/2022	Enero	2123.34
01/03/2022	Enero	4124.35

date	monthname	sum(amount)
01/04/2022	Enero	2431.36
01/05/2022	Enero	4787.78
01/06/2022	Enero	2431.84
01/07/2022	Enero	2854.83
01/08/2022	Enero	3554.28
01/09/2022	Enero	3756.17
01/10/2022	Enero	3454.35

In the load script, the `LongMonthNames` variable is modified to list the months of the year in Spanish. Then, to create a field called, `monthname`, the `date` function is used with the `date` field as the first argument. The second argument in the function is the formatting `MMMM`.

Using this formatting Qlik Sense converts the values from the first argument into the corresponding full month name set in the variable `LongMonthNames`. In the results table, the field values of our created field `monthname` display the month name written in Spanish.

MoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency set by your regional settings.



By default, Qlik Sense displays numbers and text differently in table charts. Numbers are right-aligned, and text is left-aligned. This makes it easy to find text-to-number conversion issues. Any tables on this page that show Qlik Sense results will use this formatting.

Syntax:

MoneyDecimalSep

Qlik Sense applications will interpret text fields that conform to this formatting as monetary values. The text field must contain the currency symbol that is defined in the `MoneyFormat` system variable. `MoneyDecimalSep` is particularly helpful when handling data sources received from multiple different regional settings.

The following example shows a possible use of the `MoneyDecimalSep` system variable:

```
Set MoneyDecimalSep='.';
```

This function is often used together with the following functions:

Related functions

Function	Interaction
<code>MoneyFormat</code>	In instances of text field interpretation, the <code>MoneyFormat</code> symbol will be used as part of the interpretation. For Number Formatting, the <code>MoneyFormat</code> formatting will be used by Qlik Sense in Chart Objects.

Function	Interaction
MoneyThousandSep	In instances of text field interpretation, the MoneyThousandSep function must also be adhered to.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - MoneyDecimalSep dot (.) notation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named `Transactions`.
- Provided data that has its monetary field in text format with a dot '.' used as the decimal separator. Each record is also prefixed by a '\$' symbol, except for the last record, which is prefixed by a '£' symbol.

Keep in mind that the `MoneyFormat` system variable defines dollar '\$' as the default currency.

Load script

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='###0.00;-###0.00';
```

`Transactions:`

```
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$14.41'
```

2 Working with variables in the data load editor

```
01/02/2022,2,'$2,814.32'  
01/03/2022,3,'$249.36'  
01/04/2022,4,'$24.37'  
01/05/2022,5,'$7.54'  
01/06/2022,6,'$243.63'  
01/07/2022,7,'$545.36'  
01/08/2022,8,'$3.55'  
01/09/2022,9,'$3.436'  
01/10/2022,10,'£345.66'  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:amount.

Add the following measures:

- `=isNum(amount)`
- `=sum(amount)`

Review the results below, demonstrating the correct interpretation of all dollar '\$' values only.

Results table

amount	=isNum(amount)	=Sum(amount)
Totals	0	\$3905.98
£345.66	0	\$0.00
\$3.436	-1	\$3.44
\$3.55	-1	\$3.55
\$7.54	-1	\$7.54
\$14.41	-1	\$14.41
\$24.37	-1	\$24.37
243.63	-1	\$243.63
\$249.36	-1	\$249.36
\$545.36	-1	\$545.36
\$2,814.32	-1	\$2814.32

The results table above shows how the amount field has been interpreted correctly for all dollar (\$) prefixed values, whilst the pound (£) prefixed amount has not been converted to a monetary value.

Example 2 - MoneyDecimalSep comma (,) notation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named `Transactions`.
- Provided data that has its monetary field in text format with a comma ',' used as the decimal separator. Each record is also prefixed by a '\$' symbol, except for the last record, which erroneously uses the dot decimal separator '.'.

Keep in mind that the `MoneyFormat` system variable defines dollar '\$' as the default currency.

Load script

```
SET MoneyThousandSep='.';
SET MoneyDecimalSep=',';
SET MoneyFormat='###0.00;-$##0.00';
```

`Transactions:`

```
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$14,41'
01/02/2022,2,'$2.814,32'
01/03/2022,3,'$249,36'
01/04/2022,4,'$24,37'
01/05/2022,5,'$7,54'
01/06/2022,6,'$243,63'
01/07/2022,7,'$545,36'
01/08/2022,8,'$3,55'
01/09/2022,9,'$3,436'
01/10/2022,10,'$345.66'
];
```

Results

Paragraph text for Results.

Load the data and open a sheet. Create a new table and add this field as a dimension:`amount`.

Add the following measures:

- `=isNum(amount)`
- `=sum(amount)`

Review the results below, demonstrating the correct interpretation of all values, except for the amount in which the decimal separator uses dot '.' notation. In that case, a comma should have been used instead.

Results table

amount	=isNum(amount)	=Sum(amount)
Totals	0	\$3905.98
\$345.66	0	\$0.00
\$3,436	-1	\$3.44
\$3,55	-1	\$3.55
\$7,54	-1	\$7.54
\$14,41	-1	\$14.41
\$24,37	-1	\$24.37
\$243,63	-1	\$243.63
\$249,36	-1	\$249.36
\$545,36	-1	\$545.36
\$2.814,32	-1	\$2814.32

MoneyFormat

This system variable defines the format pattern used by Qlik for automatic translation of text to number where the number is prefixed by a monetary symbol. It also defines how measures whose Number Formatting properties are set to ‘Money’ will be displayed in chart objects.

The symbol defined as part of the format pattern in the `MoneyFormat` system variable replaces the currency symbol set by your regional settings.



By default, Qlik Sense displays numbers and text differently in table charts. Numbers are right-aligned, and text is left-aligned. This makes it easy to find text-to-number conversion issues. Any tables on this page that show Qlik Sense results will use this formatting.

Syntax:

MoneyFormat

```
Set MoneyFormat='$ #,##0.00; ($ #,##0.00)';
```

This formatting will be displayed in chart objects when a numerical field’s `Number Formatting` property is set to `Money`. Further, when numerical text fields are interpreted by Qlik Sense, if the currency symbol of the text field matches that of the symbol defined in the `MoneyFormat` variable, Qlik Sense will interpret this field as a monetary value.

2 Working with variables in the data load editor

This function is often used together with the following functions:

Related functions	
Function	Interaction
<i>MoneyDecimalSep</i> (page 226)	For Number Formatting, <i>MoneyDecimalSep</i> will be used in field formatting of objects.
<i>MoneyThousandSep</i> (page 234)	For Number Formatting, <i>MoneyThousandSep</i> will be used in field formatting of objects.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - MoneyFormat

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains a dataset which is loaded into a table named `Transactions`. The default `MoneyFormat` variable definition is used.

Load script

```
SET MoneyThousandSep=',';  
SET MoneyDecimalSep='.';  
SET MoneyFormat='###0.00;-$##0.00';
```

```
Transactions:  
Load  
date,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,$1000000441
```

2 Working with variables in the data load editor

```
01/02/2022,2,$21237492432  
01/03/2022,3,$249475336  
01/04/2022,4,$24313369837  
01/05/2022,5,$7873578754  
01/06/2022,6,$24313884663  
01/07/2022,7,$545883436  
01/08/2022,8,$35545828255  
01/09/2022,9,$37565817436  
01/10/2022,10,$3454343566  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- amount

Add this measure:

```
=Sum(amount)
```

Under **Number formatting**, select **Money** to configure sum(amount) as a monetary value.

Results table

date	Amount	=Sum(amount)
Totals		\$165099674156.00
01/01/2022	\$10000000441	\$10000000441.00
01/02/2022	\$21237492432	\$21237492432.00
01/03/2022	\$249475336	\$249475336.00
01/04/2022	\$24313369837	\$24313369837.00
01/05/2022	\$7873578754	\$7873578754.00
01/06/2022	\$24313884663	\$24313884663.00
01/07/2022	\$545883436	\$545883436.00
01/08/2022	\$35545828255	\$35545828255.00
01/09/2022	\$37565817436	\$37565817436.00
01/10/2022	\$3454343566	\$3454343566.00

The default MoneyFormat definition is used. This looks as follows: \$###0.00;-\$###0.00. In the results table, the format of the amount field displays the currency symbol and the decimal point and decimal places have been included.

Example 2 - MoneyFormat with thousands separator and mixed input formats

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A mixed-input format dataset, which is loaded into a table named `Transactions` with thousands separators and decimal separators interspersed.
- A modification of the `MoneyFormat` definition is modified to include a comma as the thousands separator.
- One of the rows of data erroneously delimited with thousands separator commas in the wrong places.
Note how this amount is left as text and not interpretable as a number.

Load script

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat = '$#,##0.00;-$#,##0.00';
```

`Transactions`:

```
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$10,000,000,441.45'
01/02/2022,2,'$212,3749,24,32.23'
01/03/2022,3,$249475336.45
01/04/2022,4,$24,313,369,837
01/05/2022,5,$7873578754
01/06/2022,6,$24313884663
01/07/2022,7,$545883436
01/08/2022,8,$35545828255
01/09/2022,9,$37565817436
01/10/2022,10,$3454343566
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `amount`

Add this measure:

2 Working with variables in the data load editor

=Sum(amount)

Under **Number formatting**, select **Money** to configure sum(amount) as a monetary value.

Results table

date	Amount	=Sum(amount)
Totals		\$119,548,811,911.90
01/01/2022	\$10,000,000,441.45	\$10,000,000,441.45
01/02/2022	\$212,3749,24,32.23	\$0.00
01/03/2022	\$249475336.45	\$249,475,336.45
01/04/2022	\$24	\$24.00
01/05/2022	\$7873578754	\$7,873,578,754.00
01/06/2022	\$24313884663	\$24,313,884,663.00
01/07/2022	\$545883436	\$545,883,436.00
01/08/2022	\$35545828255	\$35,545,828,255.00
01/09/2022	\$37565817436	\$37,565,817,436.00
01/10/2022	\$3454343566	\$3,454,343,566.00

At the start of the script, the **MoneyFormat** system variable is modified to include a comma as a thousands separator. In the Qlik Sense table, the formatting can be seen to include this separator. Furthermore, the row with the erroneous separator has not been interpreted correctly and remains as text. This is why it does not contribute towards the summation of the amount.

MoneyThousandSep

The thousands separator defined replaces the digit grouping symbol for currency set by your regional settings.



By default, Qlik Sense displays numbers and text differently in table charts. Numbers are right-aligned, and text is left-aligned. This makes it easy to find text-to-number conversion issues. Any tables on this page that show Qlik Sense results will use this formatting.

Syntax:

MoneyThousandSep

Qlik Sense applications will interpret text fields that conform to this formatting as monetary values. The text field must contain the currency symbol that is defined in the **MoneyFormat** system variable. **MoneyThousandSep** is particularly helpful when handling data sources received from multiple different regional settings.

The following example shows a possible use of the **MoneyThousandSep** system variable:

```
Set MoneyDecimalSep=',';
```

This function is often used together with the following functions:

Related functions

Function	Interaction
MoneyFormat	In instances of text field interpretation, the MoneyFormat symbol will be used as part of the interpretation. For Number Formatting, the MoneyFormat formatting will be used by Qlik Sense in chart objects.
MoneyDecimalSep	In instances of text field interpretation, the MoneyDecimalSep function must also be adhered to.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - MoneyThousandSep comma (,) notation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named `Transactions`.
- Provided data that has its monetary field in text format with a comma used as the thousands separator. Each record is also prefixed by a '\$' symbol.

Keep in mind that the `MoneyFormat` system variable defines dollar '\$' as the default currency.

Load script

```
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='###0.00;-$##0.00';
```

Transactions:

```
Load
date,
id,
amount
```

2 Working with variables in the data load editor

```
Inline
[
date,id,amount
01/01/2022,1,'$10,000,000,441'
01/02/2022,2,'$21,237,492,432'
01/03/2022,3,'$249,475,336'
01/04/2022,4,'$24,313,369,837'
01/05/2022,5,'$7,873,578,754'
01/06/2022,6,'$24,313,884,663'
01/07/2022,7,'$545,883,436'
01/08/2022,8,'$35,545,828,255'
01/09/2022,9,'$37,565,817,436'
01/10/2022,10,'$3.454.343.566'
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:amount.

Add the following measures:

- `isNum(amount)`
- `sum(amount)`

Review the results below. The table demonstrates the correct interpretation of all values using comma ',' notation as the thousands separator.

The amount field has been interpreted correctly for all values, with the exception of one value which used a dot '.' as the thousands separator.

Results table

amount	=isNum(amount)	=Sum(amount)
Totals	0	\$161645330590.00
\$3.454.343.566	0	\$0.00
\$249,475,336	-1	\$249475336.00
\$545,883,436	-1	\$545883436.00
\$7,873,578,754	-1	\$7873578754.00
\$10,000,000,441	-1	\$10000000441.00
\$21,237,492,432	-1	\$21237492432.00
\$24,313,369,837	-1	\$24313369837.00
\$24,313,884,663	-1	\$24313884663.00
\$35,545,828,255	-1	\$35545828255.00
\$37,565,817,436	-1	\$37565817436.00

Example 2 - MoneyThousandSep dot(.) notation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named Transactions.
- Provided data that has its monetary field in text format with a dot '.' used as the thousands separator.
Each record is also prefixed by a '\$' symbol.

Keep in mind that the MoneyFormat system variable defines dollar '\$' as the default currency.

Load script

```
SET MoneyThousandSep='.';  
SET MoneyDecimalSep=',';  
SET MoneyFormat='$##0.00;-$##0.00';
```

Transactions:

```
Load  
date,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,'$10.000.000.441'  
01/02/2022,2,'$21.237.492.432'  
01/03/2022,3,'$249.475.336'  
01/04/2022,4,'$24.313.369.837'  
01/05/2022,5,'$7.873.578.754'  
01/06/2022,6,'$24.313.884.663'  
01/07/2022,7,'$545.883.436'  
01/08/2022,8,'$35.545.828.255'  
01/09/2022,9,'$37.565.817.436'  
01/10/2022,10,'$3,454,343,566'  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:amount.

Add the following measures:

- `isNum(amount)`
- `sum(amount)`

2 Working with variables in the data load editor

Review the results below, demonstrating the correct interpretation of all values using dot '.' notation as the thousand separator.

The amount field has been interpreted correctly for all values, with the exception of one value which used a comma ',' as the thousands separator.

Results table

amount	=isNum(amount)	=Sum(amount)
Totals	0	\$161645330590.00
\$3,545,343,566	0	\$0.00
\$249.475.336	-1	\$249475336.00
\$545.883.436	-1	545883436.00
\$7.873.578.754	-1	\$7873578754.00
\$10.000.000.441	-1	\$10000000441.00
\$21.237.492.432	-1	\$21237492432.00
\$24.313.884.663	-1	\$24313884663.00
\$24.313.884.663	-1	\$24313884663.00
\$35.545.828.255	-1	\$35545828255.00
\$37.565.817.436	-1	\$37565817436.00

MonthNames

The format defined replaces the month names convention of the regional settings.

Syntax:

MonthNames

When modifying the variable, the ; needs to be used to separate the individual values.

Function examples

Example

```
Set MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

Results

This use of the MonthNames function defines month names in English and their abbreviated form.

```
Set
```

```
MonthNames='Enero;Feb;Marzo;Abr;Mayo;Jun;Jul;Agosto;Set;Oct;Nov;Dic';
```

This use of the MonthNames

function defines month names in Spanish and their abbreviated form.

2 Working with variables in the data load editor

The MonthNames function can be used in combination with the following functions:

Related functions	
Function	Interaction
<i>month</i> (page 882)	Script function to return values defined in MonthNames as field values
<i>Date</i> (page 1193)	Script function to return values defined in MonthNames as field values based on a formatting argument provided
<i>LongMonthNames</i> (page 222)	Long form values of MonthNames

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – System variables default

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates that is loaded into a table named `Transactions`.
- A date field.
- The default `MonthNames` definition.

Load script

```
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

`Transactions`:

```
LOAD  
date,  
Month(date) as monthname,  
id,  
amount  
INLINE
```

2 Working with variables in the data load editor

```
[  
date,id,amount  
01/01/2022,1,1000.45  
01/02/2022,2,2123.34  
01/03/2022,3,4124.35  
01/04/2022,4,2431.36  
01/05/2022,5,4787.78  
01/06/2022,6,2431.84  
01/07/2022,7,2854.83  
01/08/2022,8,3554.28  
01/09/2022,9,3756.17  
01/10/2022,10,3454.35  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- monthname

Create this measure:

```
=sum(amount)
```

Results table		
date	monthname	sum(amount)
01/01/2022	Jan	1000.45
01/02/2022	Jan	2123.34
01/03/2022	Jan	4124.35
01/04/2022	Jan	2431.36
01/05/2022	Jan	4787.78
01/06/2022	Jan	2431.84
01/07/2022	Jan	2854.83
01/08/2022	Jan	3554.28
01/09/2022	Jan	3756.17
01/10/2022	Jan	3454.35

The default MonthNames definition is used. In the load script, the Month function is used with the date field as the provided argument.

In the results table, the output of this Month function displays the months of the year in the format of the MonthNames definition.

Example 2 - Change system variable

Load script and results

2 Working with variables in the data load editor

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates that is loaded into a table named `Transactions`.
- A date field.
- The `MonthNames` variable that is modified to use the abbreviated months in Spanish.

Load script

```
Set MonthNames='Enero;Feb;Marzo;Abr;Mayo;Jun;Jul;Agosto;Set;Oct;Nov;Dic';
```

`Transactions`:

```
LOAD  
date,  
month(date) as month,  
id,  
amount  
INLINE  
[  
date,id,amount  
01/01/2022,1,1000  
02/01/2022,2,2123  
03/01/2022,3,4124  
04/01/2022,4,2431  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `monthname`

Create this measure:

```
=sum(amount)
```

Results table		
date	monthname	sum(amount)
01/01/2022	Enero	1000.45
01/02/2022	Enero	2123.34
01/03/2022	Enero	4124.35
01/04/2022	Enero	2431.36
01/05/2022	Enero	4787.78

date	monthname	sum(amount)
01/06/2022	Enero	2431.84
01/07/2022	Enero	2854.83
01/08/2022	Enero	3554.28
01/09/2022	Enero	3756.17
01/10/2022	Enero	3454.35

In the load script, first the `MonthNames` variable is modified to list the months of the year abbreviated in Spanish. The `Month` function is used with the `date` field as the provided argument.

In the results table, the output of this `Month` function displays the months of the year in the format of the `MonthNames` definition.

It is important to remember that if the language for the `MonthNames` variable is modified like it has been in this example, the `LongMonthNames` variable would still contain the months of the year in English. The `LongMonthNames` variable would have to be modified if both variables are used in the application.

Example 3 – Date function

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates that is loaded into a table named `Transactions`.
- A `date` field.
- The default `MonthNames` definition.

Load script

```
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

`Transactions`:

```
LOAD  
date,  
Month(date, 'MMM') as monthname,  
id,  
amount  
INLINE  
[  
date,id,amount  
01/01/2022,1,1000.45  
01/02/2022,2,2123.34  
01/03/2022,3,4124.35  
01/04/2022,4,2431.36  
01/05/2022,5,4787.78
```

2 Working with variables in the data load editor

```
01/06/2022,6,2431.84  
01/07/2022,7,2854.83  
01/08/2022,8,3554.28  
01/09/2022,9,3756.17  
01/10/2022,10,3454.35  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- monthname

Create this measure:

```
=sum(amount)
```

Results table		
date	monthname	sum(amount)
01/01/2022	Jan	1000.45
01/02/2022	Jan	2123.34
01/03/2022	Jan	4124.35
01/04/2022	Jan	2431.36
01/05/2022	Jan	4787.78
01/06/2022	Jan	2431.84
01/07/2022	Jan	2854.83
01/08/2022	Jan	3554.28
01/09/2022	Jan	3756.17
01/10/2022	Jan	3454.35

The default MonthNames definition is used. In the load script, the date function is used with the date field as the first argument. The second argument is MMM.

Using this formatting Qlik Sense converts the values from the first argument into the corresponding month name set in the variable MonthNames. In the results table, the field values of our created field month display this.

NumericalAbbreviation

The numerical abbreviation sets which abbreviation to use for scale prefixes of numerals, for example M for mega or a million (10^6), and μ for micro (10^{-6}).

Syntax:

```
NumericalAbbreviation
```

2 Working with variables in the data load editor

You set the `NumericalAbbreviation` variable to a string containing a list of abbreviation definition pairs, delimited by semi colon. Each abbreviation definition pair should contain the scale (the exponent in decimal base) and the abbreviation separated by a colon, for example, `6:M` for a million.

The default setting is '`3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y`'.

Examples:

This setting will change the prefix for a thousand to t and the prefix for a billion to B. This would be useful for financial applications where you would expect abbreviations like t\$, M\$, and B\$.

```
Set NumericalAbbreviation='3:t;6:M;9:B;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
```

ReferenceDay

The setting defines which day in January to set as reference day to define week 1. In other words, this setting prescribes how many days in week 1 must be dates within January.

Syntax:

ReferenceDay

`ReferenceDay` sets how many days are included in the first week of the year. `ReferenceDay` can be set to any value between 1 and 7. Any value outside of the 1-7 range is interpreted as the midpoint of the week (4), which is equivalent to `ReferenceDay` being set to 4.

If you do not select a value for the `ReferenceDay` setting, then the default value will show `ReferenceDay=0` which will be interpreted as the midpoint of the week (4), as seen in the `ReferenceDay` values table below.

The `ReferenceDay` function is often used in combination with the following functions:

Related functions

Variable	Interaction
<code>BrokenWeeks</code> (page 201)	If the Qlik Sense app is operating with unbroken weeks, the <code>ReferenceDay</code> variable setting will be enforced. However, if broken weeks are being used, week 1 will begin on January 1 and terminate in conjunction with the <code>FirstWeekDay</code> variable setting and ignore the <code>ReferenceDay</code> flag.
<code>FirstWeekDay</code> (page 215)	Integer that defines which day to use as the first day of the week.

Qlik Sense allows the following values to be set for `ReferenceDay`:

ReferenceDay values

Value	Reference day
0 (default)	January 4
1	January 1

Value	Reference day
2	January 2
3	January 3
4	January 4
5	January 5
6	January 6
7	January 7

In the following example the `ReferenceDay = 3` defines January 3 as the reference day:

```
SET ReferenceDay=3; // (set January 3 as the reference day)
```

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Examples:

If you want ISO settings for weeks and week numbers, make sure to have the following in the script:

```
Set FirstWeekDay=0;  
Set BrokenWeeks=0;  
Set ReferenceDay=4; // Jan 4th is always in week 1
```

If you want US settings, make sure to have the following in the script:

```
Set FirstWeekDay=6;  
Set BrokenWeeks=1;  
Set ReferenceDay=1; // Jan 1st is always in week 1
```

Example 1 - Load script using the default value; ReferenceDay=0

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

2 Working with variables in the data load editor

- The ReferenceDay variable that is set to 0.
- The BrokenWeeks variable that is set to 0 which forces the app to use unbroken weeks.
- A dataset of dates from the end of 2019 to the start of 2020.

Load script

```
SET BrokenWeeks = 0;
SET ReferenceDay = 0;

Sales:
LOAD
date,
sales,
week(date) as week,
weekday(date) as weekday
Inline [
date,sales
12/27/2019,5000
12/28/2019,6000
12/29/2019,7000
12/30/2019,4000
12/31/2019,3000
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week
- weekday

Results table

date	week	weekday
12/27/2019	52	Fri
12/28/2019	52	Sat
12/29/2019	1	Sun

date	week	weekday
12/30/2019	1	Mon
12/31/2019	1	Tue
01/01/2020	1	Wed
01/02/2020	1	Thu
01/03/2020	1	Fri
01/04/2020	1	Sat
01/05/2020	2	Sun
01/06/2020	2	Mon
01/07/2020	2	Tue
01/08/2020	2	Wed
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat

Week 52 concludes on Saturday, December 28. Because ReferenceDay requires January 4 to be included in week 1, week 1 therefore begins on December 29 and concludes on Saturday, January 4.

Example - ReferenceDay variable set to 5

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The ReferenceDay variable that is set to 5.
- The BrokenWeeks variable that is set to 0 which forces the app to use unbroken weeks.
- A dataset of dates from the end of 2019 to the start of 2020.

Load script

```
SET BrokenWeeks = 0;  
SET ReferenceDay = 5;
```

```
Sales:  
LOAD  
date,  
sales,  
week(date) as week,  
weekday(date) as weekday  
Inline [
```

```
date,sales
12/27/2019,5000
12/28/2019,6000
12/29/2019,7000
12/30/2019,4000
12/31/2019,3000
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week
- weekday

Results table

date	week	weekday
12/27/2019	52	Fri
12/28/2019	52	Sat
12/29/2019	53	Sun
12/30/2019	53	Mon
12/31/2019	53	Tue
01/01/2020	53	Wed
01/02/2020	53	Thu
01/03/2020	53	Fri
01/04/2020	53	Sat
01/05/2020	1	Sun
01/06/2020	1	Mon
01/07/2020	1	Tue
01/08/2020	1	Wed
01/09/2020	1	Thu

date	week	weekday
01/10/2020	1	Fri
01/11/2020	1	Sat

Week 52 concludes on Saturday, December 28. The `brokenweeks` variable forces the app to use unbroken weeks. The reference day value of 5 requires January 5 to be included in week 1.

However, this is eight days after the conclusion of week 52 of the previous year. Therefore, week 53 begins on December 29 and concludes on January 4. Week 1 begins on Sunday, January 5.

ThousandSep

The thousands separator defined replaces the digit grouping symbol of the operating system (regional settings).

Syntax:

ThousandSep

Qlik Sense object using the `ThousandSep` variable (with thousands separator)

`max(amount)`

47,873,578,754.00

Qlik Sense apps interpret text fields that conform to this formatting as numbers. This formatting will be displayed in chart objects when a numerical field's **Number formatting** property is set to **Number**.

`ThousandSep` is helpful when handling data sources received from multiple regional settings.



*If the `ThousandSep` variable is modified after objects have already been created and formatted in the application, the user will need to re-format each relevant field by de-selecting and then re-selecting the **Number formatting** property **Number**.*

The following examples show possible uses of the `ThousandSep` system variable:

```
Set ThousandSep=','; // (for example, seven billion will be displayed as: 7,000,000,000)
```

```
Set ThousandSep=' '; // (for example, seven billion will be displayed as: 7 000 000 000)
```

These topics may help you work with this function:

Related topics

Topic	Description
<i>DecimalSep</i> (page 213)	In instances of text field interpretation, the decimal separator settings, as provided by this function, must also be respected. For number formatting, DecimalSep will be used by Qlik Sense where necessary.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - Default system variables

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named `Transactions`.
- Use of the default `ThousandsSep` variable definition.

Load script

```
Transactions:  
Load  
date,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,10000000441  
01/02/2022,2,21237492432  
01/03/2022,3,41249475336  
01/04/2022,4,24313369837  
01/05/2022,5,47873578754  
01/06/2022,6,24313884663  
01/07/2022,7,28545883436
```

2 Working with variables in the data load editor

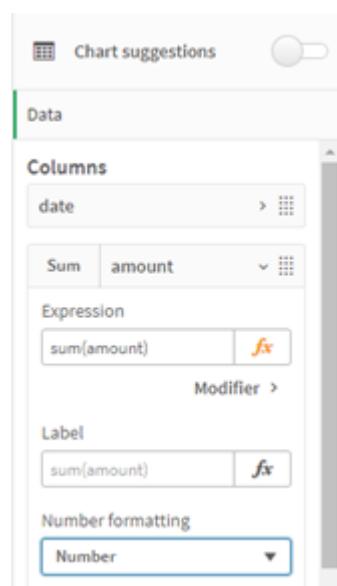
```
01/08/2022,8,35545828255  
01/09/2022,9,37565817436  
01/10/2022,10,3454343566  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add this field as a dimension:date.
2. Add the following measure:
 $=\text{sum}(\text{amount})$
3. In the properties panel, under **Data**, select the measure.
4. Under **Number formatting**, select **Number**.

Adjusting number formatting for a chart measure



Results table

date	=sum(amount)
01/01/2022	10,000,000,441.00
01/02/2022	21,237,492,432.00
01/03/2022	41,249,475,336.00
01/04/2022	24,313,369,837.00
01/05/2022	47,873,578,754.00
01/06/2022	24,313,884,663.00
01/07/2022	28,545,883,436.00

date	=sum(amount)
01/08/2022	35,545,828,255.00
01/09/2022	37,565,817,436.00
01/10/2022	3,454,343,566.00

In this example, the default `ThousandSep` definition, which is set to comma format (','), is used. In the results table, the format of the amount field displays a comma between thousand groupings.

Example 2 - Changing system variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example, which is loaded into a table named `Transactions`.
- Modification of the `ThousandSep` definition, at the start of the script, to display a '*' character as the thousands separator. This is an extreme example, and is used solely to demonstrate the functionality of the variable.

The modification used in this example is extreme and not commonly used, but is shown here to demonstrate the functionality of the variable.

Load script

```
SET ThousandSep='*';  
  
Transactions:  
Load  
date,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,10000000441  
01/02/2022,2,21237492432  
01/03/2022,3,41249475336  
01/04/2022,4,24313369837  
01/05/2022,5,47873578754  
01/06/2022,6,24313884663  
01/07/2022,7,28545883436  
01/08/2022,8,35545828255  
01/09/2022,9,37565817436  
01/10/2022,10,3454343566  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add this field as a dimension: date.
2. Add the following measure:
 $=\text{sum}(\text{amount})$
3. In the properties panel, under **Data**, select the measure.
4. Under **Number formatting**, select **Custom**.

Results table

date	=sum(amount)
01/01/2022	10*000*000*441.00
01/02/2022	21*237*492*432.00
01/03/2022	41*249*475*336.00
01/04/2022	24*313*369*837.00
01/05/2022	47*873*578*754.00
01/06/2022	24*313*884*663.00
01/07/2022	28*545*883*436.00
01/08/2022	35*545*828*255.00
01/09/2022	37*565*817*436.00
01/10/2022	3*454*343*566.00

At the start of the script, the ThousandSep system variable is modified to a '*'. In the results table, the format of the amount field can be seen to display a '**' between thousand grouping.

Example 3 - Text interpretation

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table named `Transactions`.
- Data which has its numerical field in text format, with a comma used as the thousands separator.
- Use of the default `ThousandSep` system variable.

Load script

```
Transactions:  
Load  
date,  
id,  
amount  
Inline  
[  
date,id,amount  
01/01/2022,1,'10,000,000,441'  
01/02/2022,2,'21,492,432'  
01/03/2022,3,'4,249,475,336'  
01/04/2022,4,'24,313,369,837'  
01/05/2022,5,'4,873,578,754'  
01/06/2022,6,'313,884,663'  
01/07/2022,7,'2,545,883,436'  
01/08/2022,8,'545,828,255'  
01/09/2022,9,'37,565,817,436'  
01/10/2022,10,'3,454,343,566'  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add this field as a dimension:date.
2. Add the following measure:
`=sum(amount)`
3. In the properties panel, under **Data**, select the measure.
4. Under **Number formatting**, select **Number**.
5. Add the following measure to evaluate whether or not the amount field is a numerical value:
`=isnum(amount)`

Results table

date	=sum(amount)	=isnum(amount)
01/01/2022	10,000,000,441.00	-1
01/02/2022	21,492,432.00	-1
01/03/2022	4,249,475,336.00	-1
01/04/2022	24,313,369,837.00	-1
01/05/2022	4,873,578,754.00	-1
01/06/2022	313,884,663.00	-1
01/07/2022	2,545,883,436.00	-1
01/08/2022	545,828,255.00	-1

2 Working with variables in the data load editor

date	=sum(amount)	=isnum(amount)
01/09/2022	37,565,817,436.00	-1
01/10/2022	3*454*343*566.00	-1

Once the data is loaded, we can see that Qlik Sense has interpreted the amount field as a numerical value, due to the data conforming to the Thousandsep variable. This is demonstrated by the `isnum()` function, which evaluates each entry to -1, or TRUE.



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

TimeFormat

The format defined replaces the time format of the operating system (regional settings).

Syntax:

`TimeFormat`

Example:

```
Set TimeFormat='hh:mm:ss';
```

TimestampFormat

The format defined replaces the date and time formats of the operating system (regional settings).

Syntax:

`TimestampFormat`

Example:

The following examples use `1983-12-14T13:15:30Z` as timestamp data to show the results of different **SET TimestampFormat** statements. The date format used is **YYYYMMDD** and the time format is **h:mm:ss TT**. The date format is specified in the **SET DateFormat** statement and the time format is specified in the **SET TimeFormat** statement, at the top of the data load script.

Results

Example	Result
<code>SET TimestampFormat='YYYYMMDD' ;</code>	19831214
<code>SET TimestampFormat='M/D/YY hh:mm:ss[.fff]' ;</code>	12/14/83 13:15:30
<code>SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff]' ;</code>	14/12/1983 13:15:30
<code>SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff] TT' ;</code>	14/12/1983 1:15:30 PM
<code>SET TimestampFormat='YYYY-MM-DD hh:mm:ss[.fff] TT' ;</code>	1983-12-14 01:15:30

Examples: Load script

Example: Load script

In the first load script `SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT'` is used. In the second load script the timestamp format is changed to `SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'`. The different results show how the **SET TimeFormat** statement works with different time data formats.

The table below shows the data set that is used in the load scripts that follow. The second column of the table shows the format of each timestamp in the data set. The first five timestamps follow ISO 8601 rules but the sixth does not.

Data set

Table showing the time data used and the format for each timestamp in the data set.

transaction_timestamp	time data format
2018-08-30	YYYY-MM-DD
20180830T193614.857	YYYYMMDDhhmmss.sss
20180830T193614.857+0200	YYYYMMDDhhmmss.sss±hhmm
2018-09-16T12:30-02:00	YYYY-MM-DDhh:mm±hh:mm
2018-09-16T13:15:30Z	YYYY-MM-DDhh:mmZ
9/30/18 19:36:14	M/D/YY hh:mm:ss

In the **Data load editor**, create a new section, and then add the example script and run it. Then add, at least, the fields listed in the results column to a sheet in your app to see the result.

Load script

```
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
SET DateFormat='YYYYMMDD';
SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*,
Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp
;

Load * Inline [
transaction_id, transaction_timestamp, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 2018-08-30, 12423.56, 23, 0,2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange
```

2 Working with variables in the data load editor

```
3752, 20180830T193614.857+0200, 15.75, 1, 0.22, 5646471, s, blue
3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red
3755, 9/30/18 19:36:14, -59.18, 2, 0.333333333333333, 2038593, M, Blue
];
```

Results

Qlik Sense table showing results of the `TimestampFormat` interpretation variable being used in the load script. The last timestamp in the data set does not return a correct date.

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	-

The next load script uses the same data set. However, it uses `SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'` to match the non-ISO 8601 format of the sixth timestamp.

In the **Data load editor**, replace the previous example script with the one below and run it. Then add, at least, the fields listed in the results column to a sheet in your app to see the result.

Load script

```
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
SET DateFormat='YYYYMMDD';
SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]';

Transactions:
Load
*,
Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp
;

Load * Inline [
transaction_id, transaction_timestamp, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 2018-08-30, 12423.56, 23, 0, 2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180830T193614.857+0200, 15.75, 1, 0.22, 5646471, s, blue
3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red
```

```
3755, 9/30/18 19:36:14, -59.18, 2, 0.33333333333333, 2038593, M, Blue
];
```

Results

Qlik Sense table showing results of the `TimestampFormat` interpretation variable being used in the load script.

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	2018-09-16 19:36:14

2.15 Direct Discovery variables

Direct Discovery system variables

DirectCacheSeconds

You can set a caching limit to the Direct Discovery query results for visualizations. Once this time limit is reached, Qlik Sense clears the cache when new Direct Discovery queries are made. Qlik Sense queries the source data for the selections and creates the cache again for the designated time limit. The result for each combination of selections is cached independently. That is, the cache is refreshed for each selection independently, so one selection refreshes the cache only for the fields selected, and a second selection refreshes cache for its relevant fields. If the second selection includes fields that were refreshed in the first selection, they are not updated in cache again if the caching limit has not been reached.

The Direct Discovery cache does not apply to **Table** visualizations. Table selections query the data source every time.

The limit value must be set in seconds. The default cache limit is 1800 seconds (30 minutes).

The value used for **DirectCacheSeconds** is the value set at the time the **DIRECT QUERY** statement is executed. The value cannot be changed at runtime.

Example:

```
SET DirectCacheSeconds=1800;
```

DirectConnectionMax

You can do asynchronous, parallel calls to the database by using the connection pooling capability. The load script syntax to set up the pooling capability is as follows:

```
SET DirectConnectionMax=10;
```

The numeric setting specifies the maximum number of database connections the Direct Discovery code should use while updating a sheet. The default setting is 1.



This variable should be used with caution. Setting it to greater than 1 is known to cause problems when connecting to Microsoft SQL Server.

DirectUnicodeStrings

Direct Discovery can support the selection of extended Unicode data by using the SQL standard format for extended character string literals (N'<extended string>') as required by some databases (notably SQL Server). The use of this syntax can be enabled for Direct Discovery with the script variable **DirectUnicodeStrings**.

Setting this variable to 'true' will enable the use of the ANSI standard wide character marker "N" in front of the string literals. Not all databases support this standard. The default setting is 'false'.

DirectDistinctSupport

When a **DIMENSION** field value is selected in a Qlik Sense object, a query is generated for the source database. When the query requires grouping, Direct Discovery uses the **DISTINCT** keyword to select only unique values. Some databases, however, require the **GROUP BY** keyword. Set **DirectDistinctSupport** to 'false' to generate **GROUP BY** instead of **DISTINCT** in queries for unique values.

```
SET DirectDistinctSupport='false';
```

If DirectDistinctSupport is set to true, then **DISTINCT** is used. If it is not set, the default behavior is to use **DISTINCT**.

DirectEnableSubquery

In high cardinality multi-table scenarios, it is possible to generate sub queries in the SQL query instead of generating a large IN clause. This is activated by setting **DirectEnableSubquery** to 'true'. The default value is 'false'.



*When **DirectEnableSubquery** is enabled, you cannot load tables that are not in Direct Discovery mode.*

```
SET DirectEnableSubquery='true';
```

Teradata query banding variables

Teradata query banding is a function that enables enterprise applications to collaborate with the underlying Teradata database in order to provide for better accounting, prioritization, and workload management. Using query banding you can wrap metadata, such as user credentials, around a query.

Two variables are available, both are strings that are evaluated and sent to the database.

SQLSessionPrefix

This string is sent when a connection to the database is created.

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & chr(39) & 'who=' & osuser() & ';' & chr(39) & 'FOR SESSION';'
```

2 Working with variables in the data load editor

If **OSuser()** for example returns *WA\sbt*, this will be evaluated to `SET QUERY_BAND = 'who=WA\sbt;' FOR SESSION;`, which is sent to the database when the connection is created.

SQLQueryPrefix

This string is sent for each single query.

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'Who=' & osuser() & ';' & Chr(39) & '
FOR TRANSACTION;';
```

Direct Discovery character variables

DirectFieldColumnDelimiter

You can set the character used as the field delimiter in **Direct Query** statements for databases that require a character other than comma as the field delimiter. The specified character must be surrounded by single quotation marks in the **SET** statement.

```
SET DirectFieldColumnDelimiter= '|'
```

DirectStringQuoteChar

You can specify a character to use to quote strings in a generated query. The default is a single quotation mark. The specified character must be surrounded by single quotation marks in the **SET** statement.

```
SET DirectStringQuoteChar= ''';
```

DirectIdentifierQuoteStyle

You can specify that non-ANSI quoting of identifiers be used in generated queries. At this time, the only non-ANSI quoting available is GoogleBQ. The default is ANSI. Uppercase, lowercase, and mixed case can be used (ANSI, ansi, Ansi).

```
SET DirectIdentifierQuoteStyle="GoogleBQ";
```

For example, ANSI quoting is used in the following **SELECT** statement:

```
SELECT [Quarter] FROM [qvTest].[sales] GROUP BY [Quarter]
```

When **DirectIdentifierQuoteStyle** is set to "GoogleBQ", the **SELECT** statement would use quoting as follows:

```
SELECT [Quarter] FROM [qvTest.sales] GROUP BY [Quarter]
```

DirectIdentifierQuoteChar

You can specify a character to control the quoting of identifiers in a generated query. This can be set to either one character (such as a double quotation mark) or two (such as a pair of square brackets). The default is a double quotation mark.

```
SET DirectIdentifierQuoteChar='[]';
SET DirectIdentifierQuoteChar='``';
SET DirectIdentifierQuoteChar=' ';
SET DirectIdentifierQuoteChar='''';
```

DirectTableBoxListThreshold

When Direct Discovery fields are used in a **Table** visualization, a threshold is set to limit the number of rows displayed. The default threshold is 1000 records. The default threshold setting can be changed by setting the **DirectTableBoxListThreshold** variable in the load script. For example:

2 Working with variables in the data load editor

```
SET DirectTableBoxListThreshold=5000;
```

The threshold setting applies only to **Table** visualizations that contain Direct Discovery fields. **Table** visualizations that contain only in-memory fields are not limited by the **DirectTableBoxListThreshold** setting.

No fields are displayed in the **Table** visualization until the selection has fewer records than the threshold limit.

Direct Discovery number interpretation variables

DirectMoneyDecimalSep

The decimal separator defined replaces the decimal symbol for currency in the SQL statement generated to load data using Direct Discovery. This character must match the character used in **DirectMoneyFormat**.

Default value is '.'.

Example:

```
Set DirectMoneyDecimalSep='.';
```

DirectMoneyFormat

The symbol defined replaces the currency format in the SQL statement generated to load data using Direct Discovery. The currency symbol for the thousands separator should not be included.

Default value is '#.0000'.

Example:

```
Set DirectMoneyFormat='#.0000';
```

DirectTimeFormat

The time format defined replaces the time format in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectTimeFormat='hh:mm:ss';
```

DirectDateFormat

The date format defined replaces the date format in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectDateFormat='MM/DD/YYYY';
```

DirectTimeStampFormat

The format defined replaces the date and time format in the SQL statement generated in the SQL statement generated to load data using Direct Discovery.

Example:

```
Set DirectTimestampFormat='M/D/YY hh:mm:ss[.fff]';
```

2.16 Error variables

The values of all error variables will exist after the script execution. The first variable, `ErrorMode`, is input from the user, and the last three are output from Qlik Sense with information on errors in the script.

Error variables overview

Each variable is described further after the overview. You can also click the variable name in the syntax to immediately access the details for that specific variable.

Refer to the Qlik Sense online help for further details about the variables.

ErrorMode

This error variable determines what action is to be taken by Qlik Sense when an error is encountered during script execution.

ErrorMode

ScriptError

This error variable returns the error code of the last executed script statement.

ScriptError

ScriptErrorCount

This error variable returns the total number of statements that have caused errors during the current script execution. This variable is always reset to 0 at the start of script execution.

ScriptErrorCount

ScriptErrorList

This error variable will contain a concatenated list of all script errors that have occurred during the last script execution. Each error is separated by a line feed.

ScriptErrorList

ErrorMode

This error variable determines what action is to be taken by Qlik Sense when an error is encountered during script execution.

Syntax:

ErrorMode

Arguments:

Arguments

Argument	Description
ErrorMode=1	The default setting. The script execution will halt and the user will be prompted for action (non-batch mode).
ErrorMode =0	Qlik Sense will simply ignore the failure and continue script execution at the next script statement.
ErrorMode =2	Qlik Sense will trigger an "Execution of script failed..." error message immediately on failure, without prompting the user for action beforehand.

Example:

```
set ErrorMode=0;
```

ScriptError

This error variable returns the error code of the last executed script statement.

Syntax:

ScriptError

This variable will be reset to 0 after each successfully executed script statement. If an error occurs it will be set to an internal Qlik Sense error code. Error codes are dual values with a numeric and a text component. The following error codes exist:

Script error codes

Error code	Description
0	No error. Dual value text is empty.
1	General error.
2	Syntax error.
3	General ODBC error.
4	General OLE DB error.
5	General custom database error.
6	General XML error.
7	General HTML error.

Error code	Description
8	File not found.
9	Database not found.
10	Table not found.
11	Field not found.
12	File has wrong format.
16	Semantic error.

Example:

```
set ErrorMode=0;
LOAD * from abc.qvf;
if ScriptError=8 then
exit script;
//no file;
end if
```

ScriptErrorCount

This error variable returns the total number of statements that have caused errors during the current script execution. This variable is always reset to 0 at the start of script execution.

Syntax:

```
ScriptErrorCount
```

ScriptErrorList

This error variable will contain a concatenated list of all script errors that have occurred during the last script execution. Each error is separated by a line feed.

Syntax:

```
ScriptErrorList
```

2 Script expressions

Expressions can be used in both **LOAD** statements and **SELECT** statements. The syntax and functions described here apply to the **LOAD** statement, and not to the **SELECT** statement, since the latter is interpreted by the ODBC driver and not by Qlik Sense. However, most ODBC drivers are often capable of interpreting a number of the functions described below.

Expressions consist of functions, fields and operators, combined in a syntax.

All expressions in a Qlik Sense script return a number and/or a string, whichever is appropriate. Logical functions and operators return 0 for False and -1 for True. Number to string conversions and vice versa are implicit. Logical operators and functions interpret 0 as False and all else as True.

The general syntax for an expression is:

General syntax

Expression	Fields	Operator
expression ::= (constant)	constant	
expression ::= (constant)	fieldref	
expression ::= (constant)	operator1 expression	
expression ::= (constant)	expression operator2 expression	
expression ::= (constant)	function	
expression ::= (constant)	(expression))

where:

- **constant** is a string (a text, a date or a time) enclosed by single straight quotation marks, or a number. Constants are written with no thousands separator and with a decimal point as the decimal separator.
- **fieldref** is a field name of the loaded table.
- **operator1** is a unary operator (working on one expression, the one to the right).
- **operator2** is a binary operator (working on two expressions, one on each side).
- **function ::= functionname(parameters)**
- **parameters ::= expression { , expression }**

The number and types of parameters are not arbitrary. They depend on the function used.

Expressions and functions can thus be nested freely, and as long as the expression returns an interpretable value, Qlik Sense will not give any error messages.

3 Chart expressions

A chart (visualization) expression is a combination of functions, fields, and mathematical operators (+ * / =), and other measures. Expressions are used to process data in the app in order to produce a result that can be seen in a visualization. They are not limited to use in measures. You can build visualizations that are more dynamic and powerful, with expressions for titles, subtitles, footnotes, and even dimensions.

This means, for example, that instead of the title of a visualization being static text, it can be made from an expression whose result changes depending on the selections made.



For detailed reference regarding script functions and chart functions, see the Script syntax and chart functions.

3.1 Defining the aggregation scope

There are usually two factors that together determine which records are used to define the value of aggregation in an expression. When working in visualizations, these factors are:

- Dimensional value (of the aggregation in a chart expression)
- Selections

Together, these factors define the scope of the aggregation. You may come across situations where you want your calculation to disregard the selection, the dimension or both. In chart functions, you can achieve this by using the TOTAL qualifier, set analysis, or a combination of the two.

Aggregation: Method and description

Method	Description
TOTAL qualifier	<p>Using the total qualifier inside your aggregation function disregards the dimensional value. The aggregation will be performed on all possible field values.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets. These field names should be a subset of the chart dimension variables. In this case, the calculation is made disregarding all chart dimension variables except those listed, that is, one value is returned for each combination of field values in the listed dimension fields. Also, fields that are not currently a dimension in a chart may be included in the list. This may be useful in the case of group dimensions, where the dimension fields are not fixed. Listing all of the variables in the group causes the function to work when the drill-down level changes.</p>
Set analysis	Using set analysis inside your aggregation overrides the selection. The aggregation will be performed on all values split across the dimensions.

Method	Description
TOTAL qualifier and set analysis	Using the TOTAL qualifier and set analysis inside your aggregation overrides the selection and disregards the dimensions.
ALL qualifier	Using the ALL qualifier inside your aggregation disregards the selection and the dimensions. The equivalent can be achieved with the {1} set analysis statement and the TOTAL qualifier: <code>=sum(All Sales)</code> <code>=sum({1} Total Sales)</code>

Example: TOTAL qualifier

The following example shows how TOTAL can be used to calculate a relative share. Assuming that Q2 has been selected, using TOTAL calculates the sum of all values disregarding the dimensions.

Example: Total qualifier

Year	Quarter	Sum(Amount)	Sum(TOTAL Amount)	Sum(Amount)/Sum(TOTAL Amount)
		3000	3000	100%
2012	Q2	1700	3000	56,7%
2013	Q2	1300	3000	43,3%



To show the numbers as a percentage, in the properties panel, for the measure you want to show as a percentage value, under **Number formatting**, select **Number**, and from **Formatting**, choose **Simple** and one of the % formats.

Example: Set analysis

The following example shows how set analysis can be used to make a comparison between data sets before any selection was made. Assuming that Q2 has been selected, using set analysis with the set definition {1} calculates the sum of all values disregarding any selections but split by the dimensions.

Example: Set analysis

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
		3000	10800	27,8%
2012	Q1	0	1100	0%
2012	Q3	0	1400	0%
2012	Q4	0	1800	0%
2012	Q2	1700	1700	100%

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
2013	Q1	0	1000	0%
2013	Q3	0	1100	0%
2013	Q4	0	1400	0%
2013	Q2	1300	1300	100%

Example: TOTAL qualifier and set analysis

The following example shows how set analysis and the TOTAL qualifier can be combined to make a comparison between data sets before any selection was made and across all dimensions. Assuming that Q2 has been selected, using set analysis with the set definition {1} and the TOTAL qualifier calculates the sum of all values disregarding any selections and disregarding the dimensions.

Example: TOTAL qualifier and set analysis

Year	Quarter	Sum (Amount)	Sum({1} TOTAL Amount)	Sum(Amount)/Sum({1} TOTAL Amount)
		3000	10800	27,8%
2012	Q2	1700	10800	15,7%
2013	Q2	1300	10800	12%

Data used in examples:

AggregationScope:

```
LOAD * inline [
Year Quarter Amount
2012 Q1 1100
2012 Q2 1700
2012 Q3 1400
2012 Q4 1800
2013 Q1 1000
2013 Q2 1300
2013 Q3 1100
2013 Q4 1400] (delimiter is '');
```

3.2 Set analysis

When you make a selection in an app, you define a subset of records in the data. Aggregation functions, such as `sum()`, `Max()`, `Min()`, `Avg()`, and `count()` are calculated based on this subset.

In other words, your selection defines the scope of the aggregation; it defines the set of records on which calculations are made.

Set analysis offers a way of defining a scope that is different from the set of records defined by the current selection. This new scope can also be regarded as an alternative selection.

This can be useful if you want to compare the current selection with a particular value, for example last year's value or the global market share.

Set expressions

Set expressions can be used inside and outside aggregation functions, and are enclosed in curly brackets.

Example: Inner set expression

```
Sum( {$<Year={2021}>} Sales )
```

Example: Outer set expression

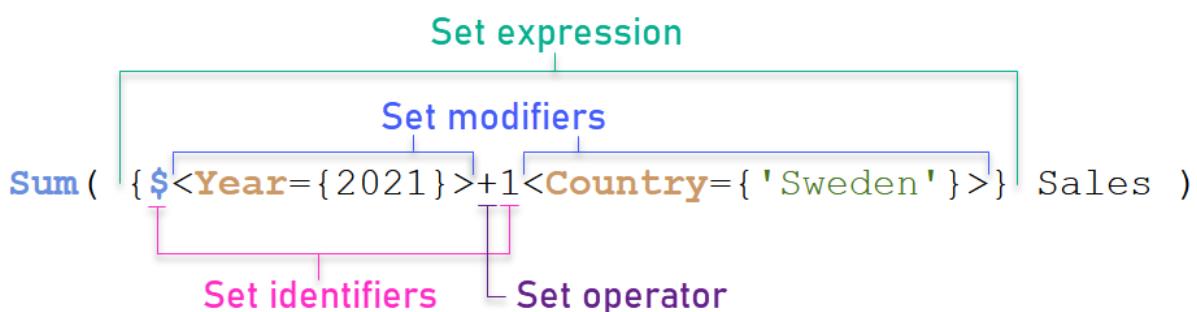
```
{<Year={2021}>} Sum(Sales) / Count(DISTINCT Customer)
```

A set expression consists of a combination of the following elements:

- **Identifiers.** A set identifier represents a selection, defined elsewhere. It also represents a specific set of records in the data. It could be the current selection, a selection from a bookmark, or a selection from an alternate state. A simple set expression consists of a single identifier, such as the dollar sign, \${}, which means all records in the current selection.
Examples: \$, 1, BookMark1, State2
- **Operators.** A set operator can be used to create unions, differences or intersections between different set identifiers. This way, you can create a subset or a superset of the selections defined by the set identifiers.
Examples: +, -, *, /
- **Modifiers.** A set modifier can be added to the set identifier to change its selection. A modifier can also be used on its own and will then modify the default identifier. A modifier must be enclosed in angle brackets <...>.
Examples: <Year={2020}>, <Supplier={ACME}>

The elements are combined to form set expressions.

Elements in a set expression



The set expression above, for example, is built from the aggregation sum(sales).

The first operand returns sales for the year 2021 for the current selection, which is indicated by the \$ set identifier and the modifier containing the selection of year 2021. The second operand returns sales for Sweden, and ignores the current selection, which is indicated by the 1 set identifier.

Finally, the expression returns a set consisting of the records that belongs to any of the two set operands, as indicated by the + set operator.

Examples

Examples that combine the set expression elements above are available in the following topics:

Natural sets

Usually, a set expression represents both a set of records in the data model, and a selection that defines this subset of data. In this case, the set is called a natural set.

Set identifiers, with or without set modifiers, always represent natural sets.

However, a set expression using set operators also represents a subset of the records, but can generally still not be described using a selection of field values. Such an expression is a non-natural set.

For example, the set given by {1-\$} cannot always be defined by a selection. It is therefore not a natural set. This can be shown by loading the following data, adding it to a table, and then making selections using filter panes.

```
Load * Inline  
[Dim1, Dim2, Number  
A, X, 1  
A, Y, 1  
B, X, 1  
B, Y, 1];
```

By making selections for Dim1 and Dim2, you get the view shown in the following table.

Table with natural and non-natural sets

Dim1	Dim2	Sum({\$} Number)	Sum({1-\$} Number)
Totals		1	3
A	X	1	0
A	Y	0	1
B	X	0	1
B	Y	0	1

The set expression in the first measure uses a natural set: it corresponds to the selection that is made {\$}.

The second measure is different. It uses {1-\$}. It is not possible to make a selection that corresponds to this set, so it is a non-natural set.

This distinction has a number of consequences:

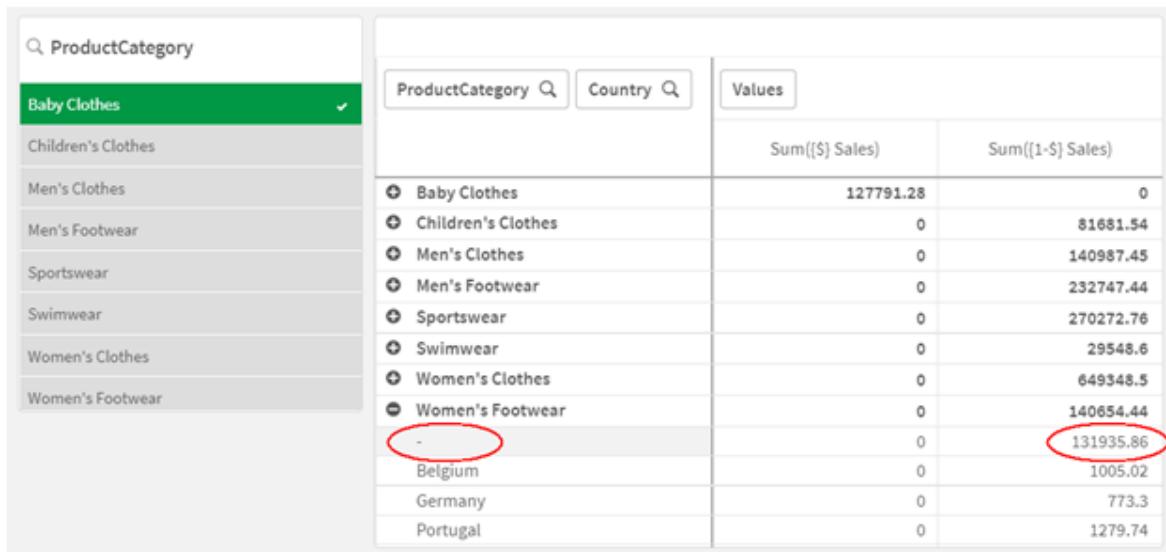
- Set modifiers can only be applied to set identifiers. They cannot be applied to an arbitrary set expression. For example, it is not possible to use a set expression such as:

{ (BM01 * BM02) <Field={x,y}> }

Here, the normal (round) brackets imply that the intersection between BM01 and BM02 should be evaluated before the set modifier is applied. The reason is that there is no element set that can be modified.

- You cannot use non-natural sets inside the P() and E() element functions. These functions return an element set, but it is not possible to deduce the element set from a non-natural set.
- A measure using a non-natural set cannot always be attributed to the right dimensional value if the data model has many tables. For example, in the following chart, some excluded sales numbers are attributed to the correct Country, whereas others have NULL as Country.

Chart with non-natural set



Whether or not the assignment is made correctly depends on the data model. In this case, the number cannot be assigned if it pertains to a country that is excluded by the selection.

Identifier	Description
1	Represents the full set of all the records in the application, irrespective of any selections made.
\$	Represents the records of the current selection. The set expression {\$} is thus the equivalent to not stating a set expression.
\$1	Represents the previous selection. \$2 represents the previous selection-but-one, and so on.
\$_1	Represents the next (forward) selection. \$_2 represents the next selection-but-one, and so on.
BM01	You can use any bookmark ID or bookmark name.
MyAltState	You can reference the selections made in an alternate state by its state name.

Example	Result
sum ({1} Sales)	Returns total sales for the app, disregarding selections but not the dimension.
sum ({\\$} Sales)	Returns the sales for the current selection, that is, the same as sum(Sales).
sum ({\\$1} Sales)	Returns the sales for the previous selection.
sum ({BM01} Sales)	Returns the sales for the bookmark named BM01.

Example	Result
sum({\$<OrderDate = DeliveryDate>} Sales)	Returns the sales for the current selection where OrderDate = DeliveryDate.
sum({1<Region = {US}>} Sales)	Returns the sales for region US, disregarding the current selection.
sum({\$<Region = >} Sales)	Returns the sales for the selection, but with the selection in Region removed.
sum({<Region = >} Sales)	Returns the same as the example above. When the set to modify is omitted, \$ is assumed.
sum({\$<Year={2000}, Region= {"U*"}>} Sales)	Returns the sales for the current selection, but with new selections both in Year and in Region.

Set identifiers

A set identifier represents a set of records in the data; either all the data or a subset of the data. It is the set of records defined by a selection. It could be the current selection, all data (no selection), a selection from a bookmark, or a selection from an alternate state.

In the example `sum({$<Year = {2009}>} Sales)`, the identifier is the dollar sign: \$. This represents the current selection. It also represents all the possible records. This set can then be altered by the modifier part of the set expression: the selection 2009 in Year is added.

In a more complex set expression, two identifiers can be used together with an operator to form a union, a difference, or an intersection of the two record sets.

The following table shows some common identifiers.

Examples with common identifiers

Identifier	Description
1	Represents the full set of all the records in the application, irrespective of any selections made.
\$	Represents the records of the current selection in the default state. The set expression {\$} is thus usually the equivalent to not stating a set expression.
\$1	Represents the previous selection in the default state. \$2 represents the previous selection-but-one, and so on.

Identifier	Description
<code>\$_1</code>	Represents the next (forward) selection. <code>\$_2</code> represents the next selection-but-one, and so on.
<code>BM01</code>	You can use any bookmark ID or bookmark name.
<code>AltState</code>	You can reference an alternate state by its state name.
<code>AltState::BM01</code>	A bookmark contains the selections of all states, and you can reference a specific bookmark by qualifying the bookmark name.

The following table shows examples with different identifiers.

Examples with different identifiers

Example	Result
<code>Sum ({1} Sales)</code>	Returns total sales for the app, disregarding selections but not the dimension.
<code>Sum ({\$} Sales)</code>	Returns the sales for the current selection, that is, the same as <code>Sum(Sales)</code> .
<code>Sum ({\$1} Sales)</code>	Returns the sales for the previous selection.
<code>Sum ({BM01} Sales)</code>	Returns the sales for the bookmark named <code>BM01</code> .

Set operators

Set operators are used to include, exclude, or intersect data sets. All operators use sets as operands and return a set as result.

You can use set operators in two different situations:

- To perform a set operation on set identifiers, representing sets of records in data.
- To perform a set operation on the element sets, on the field values, or inside a set modifier.

The following table shows the operators that can be used in set expressions.

Operators

Operator	Description
<code>+</code>	Union. This binary operation returns a set consisting of the records or elements that belong to any of the two set operands.
<code>-</code>	Exclusion. This binary operation returns a set consisting of the records or elements that belong to the first but not the other of the two set operands. Also, when used as a unary operator, it returns the complement set.
<code>*</code>	Intersection. This binary operation returns a set consisting of the records or elements that belong to both set operands.
<code>/</code>	Symmetric difference (XOR). This binary operation returns a set consisting of the records or elements that belong to either, but not both set operands.

The following table shows examples with operators.

Examples with operators	
Example	Result
Sum ({1-\$} Sales)	Returns sales for everything excluded by current selection.
Sum ({\$*BM01} Sales)	Returns sales for the intersection between the selection and bookmark BM01.
Sum ({-(\$+BM01)} Sales)	Returns sales excluded by the selection and bookmark BM01.
Sum ({\$<Year= {2009}>+1<Country= {'Sweden'}>} Sales)	Returns sales for the year 2009 associated with the current selections and add the full set of data associated with the country Sweden across all years.
Sum ({\$<Country={"S*"}+ {"*Land"}>} Sales)	Returns the sales for countries that begin with S or end with Land.

Set modifiers

Set expressions are used to define the scope of a calculation. The central part of the set expression is the set modifier that specifies a selection. This is used to modify the user selection, or the selection in the set identifier, and the result defines a new scope for the calculation.

The set modifier consists of one or more field names, each followed by a selection that should be made on the field. The modifier is enclosed by angled brackets: < >

For example:

- Sum ({\$<Year = {2015}>} Sales)
- Count ({1<Country = {Germany}>} distinct OrderID)
- Sum ({\$<Year = {2015}, Country = {Germany}>} Sales)

Element sets

An element set can be defined using the following:

- A list of values
- A search
- A reference to another field
- A set function

If the element set definition is omitted, the set modifier will clear any selection in this field. For example:

```
Sum( {$<Year = >} Sales )
```

Examples: Chart expressions for set modifiers based on element sets

Examples - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
MyTable:  
Load * Inline [  
Country, Year, Sales  
Argentina, 2014, 66295.03  
Argentina, 2015, 140037.89  
Austria, 2014, 54166.09  
Austria, 2015, 182739.87  
Belgium, 2014, 182766.87  
Belgium, 2015, 178042.33  
Brazil, 2014, 174492.67  
Brazil, 2015, 2104.22  
Canada, 2014, 101801.33  
Canada, 2015, 40288.25  
Denmark, 2014, 45273.25  
Denmark, 2015, 106938.41  
Finland, 2014, 107565.55  
Finland, 2015, 30583.44  
France, 2014, 115644.26  
France, 2015, 30696.98  
Germany, 2014, 8775.18  
Germany, 2015, 77185.68  
];
```

Chart expressions

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers based on element sets

Country	Sum(Sales)	Sum({1<Country= {Belgium}>} Sales)	Sum({1<Country= {"*A*"}>} Sales)	Sum({1<Country= {"A*"}>} Sales)	Sum({1<Year= \${=Max(Year)}>} Sales)
Totals	1645397.3	360809.2	1284588.1	443238.88	788617.07
Argentina	206332.92	0	206332.92	206332.92	140037.89
Austria	236905.96	0	236905.96	236905.96	182739.87
Belgium	360809.2	360809.2	0	0	178042.33
Brazil	176596.89	0	176596.89	0	2104.22
Canada	142089.58	0	142089.58	0	40288.25

Country	Sum(Sales)	Sum({1<Country={Belgium}>} Sales)	Sum({1<Country={"*A*"}>} Sales)	Sum({1<Country={"A*"}>} Sales)	Sum({1<Year={\$=Max(Year)}>} Sales)
Denmark	152211.66	0	152211.66	0	106938.41
Finland	138148.99	0	138148.99	0	30583.44
France	146341.24	0	146341.24	0	30696.98
Germany	85960.86	0	85960.86	0	77185.68

Explanation

- Dimensions:
 - Country
- Measures:
 - Sum(Sales)

Sum sales with no set expression.
 - Sum({1<Country={Belgium}>}Sales)

Select Belgium, and then sum corresponding sales.
 - Sum({1<Country={"*A*"}>}Sales)

Select all countries that have an A, and then sum corresponding sales.
 - Sum({1<Country={"A*"}>}Sales)

Select all countries that begin with an A, and then sum corresponding sales.
 - Sum({1<Year={\$=Max(Year)}>}Sales)

Calculate the Max(Year), which is 2015, and then sum corresponding sales.

Set modifiers based on element sets

My new sheet						
Country	Q	Sum (Sales)	Sum({1<Country={Belgium}>} Sales)	Sum({1<Country={"*A*"}>} Sales)	Sum({1<Country={"A*"}>} Sales)	Sum({1<Year={\$=Max(Year)}>} Sales)
Totals		1645397.3	360809.2	1284588.1	443238.88	788617.07
Argentina		206332.92	0	206332.92	206332.92	140037.89
Austria		236905.96	0	236905.96	236905.96	182739.87
Belgium		360809.2	360809.2	0	0	178042.33
Brazil		176596.89	0	176596.89	0	2104.22
Canada		142089.58	0	142089.58	0	40288.25
Denmark		152211.66	0	152211.66	0	106938.41
Finland		138148.99	0	138148.99	0	30583.44
France		146341.24	0	146341.24	0	30696.98
Germany		85960.86	0	85960.86	0	77185.68

Listed values

The most common example of an element set is one that is based on a list of field values enclosed in curly brackets. For example:

- `{$<Country = {Canada, Germany, Singapore}>}`
- `{$<Year = {2015, 2016}>}`

The inner curly brackets define the element set. The individual values are separated by commas.

Quotes and case sensitivity

If the values contain blanks or special characters, the values need to be quoted. Single quotes will be a literal, case-sensitive match with a single field value. Double quotes imply a case-insensitive match with one or several field values. For example:

- `<Country = {'New Zealand'}>`
Matches New Zealand only.
- `<Country = {"New Zealand"}>`
Matches New Zealand, NEW ZEALAND, and new zealand.

Dates must be enclosed in quotes and use the date format of the field in question. For example:

- `<ISO_Date = {'2021-12-31'}>`
- `<US_Date = {'12/31/2021'}>`
- `<UK_Date = {'31/12/2021'}>`

Double quotes can be substituted by square brackets or by grave accents.

Searches

Element sets can also be created through searches. For example:

- `<Country = {"C*"}>`
- `<Ingredient = {"*garlic*"}>`
- `<Year = {">2015"}>`
- `<Date = {">12/31/2015"}>`

Wildcards can be used in a text searches: An asterisk (*) represents any number of characters, and a question mark (?) represents a single character. Relational operators can be used to define numeric searches.

You should always use double quotes for searches. Searches are case-insensitive.

Dollar expansions

Dollar expansions are needed if you want to use a calculation inside your element set. For example, if you want to look at the last possible year only, you can use:

```
<Year = {$(=Max(Year))}>
```

Selected values in other fields

Modifiers can be based on the selected values of another field. For example:

```
<orderDate = DeliveryDate>
```

This modifier will take the selected values from `DeliveryDate` and apply those as a selection on `OrderDate`. If there are many distinct values – more than a couple of hundred – then this operation is CPU intensive and should be avoided.

Element set functions

The element set can also be based on the set functions `P()` (possible values) and `E()` (excluded values).

For example, if you want to select countries where the product cap has been sold, you can use:

```
<Country = P({1<Product={Cap}>} Country)>
```

Similarly, if you want to pick out the countries where the product cap has not been sold, you can use:

```
<Country = E({1<Product={Cap}>} Country)>
```

Set modifiers with searches

You can create element sets through searches with set modifiers.

For example:

- `<Country = {"C*"}>`
- `<Year = {">2015"}>`
- `<Ingredient = {"*garlic*"}>`

Searches should always be enclosed in double quotes, square brackets or grave accents. You can use a list with a mixture of literal strings (single quotes) and searches (double quotes). For example:

```
<Product = {'Nut', "*Bolt", Washer}>
```

Text searches

Wildcards and other symbols can be used in text searches:

- An asterisk (*) will represent any number of characters.
- A question mark (?) will represent a single character.
- A circumflex accent (^) will mark the beginning of a word.

For example:

- `<Country = {"C*", "*land"}>`
Match all countries beginning with a c or ending with land.
- `<Country = {"^z*"}>`
This will match all countries that have a word beginning with z, such as New Zealand.

Numeric searches

You can make numeric searches using these relational operators: `>`, `>=`, `<`, `<=`

A numeric search always begins with one of these operators. For example:

- `<Year = {">2015"}>`
Match 2016 and subsequent years.
- `<Date = {">=1/1/2015<1/1/2016"}>`
Match all dates during 2015. Note the syntax for describing a time range between two dates. The date format needs to match the date format of the field in question.

Expression searches

You can use expression searches to make more advanced searches. An aggregation is then evaluated for each field value in the search field. All values for which the search expression returns true are selected.

An expression search always begins with an equals sign: =

For example:

```
<Customer = {"=Sum(Sales)>1000"}>
```

This will return all customers with a sales value greater than 1000. sum(sales) is calculated on the current selection. This means that if you have a selection in another field, such as the Product field, you will get the customers that fulfilled the sales condition for the selected products only.

If you want the condition to be independent of the selection, you need to use set analysis inside the search string. For example:

```
<Customer = {"=Sum({1} Sales)>1000"}>
```

The expressions after the equals sign will be interpreted as a boolean value. This means that if it evaluates to something else, any non-zero number will be interpreted as true, while zero and strings are interpreted as false.

Quotes

Use quotation marks when the search strings contain blanks or special characters. Single quotes imply a literal, case-sensitive match with a single field value. Double quotes imply a case insensitive search that potentially matches multiple field values.

For example:

- `<Country = {'New Zealand'}>`
Match New Zealand only.
- `<Country = {"New Zealand"}>`
Match New Zealand, NEW ZEALAND, and new zealand

Double quotes can be substituted by square brackets or by grave accents.



In previous versions of Qlik Sense, there was no distinction between single quotes and double quotes, and all quoted strings were treated as searches. To maintain backward compatibility, apps created with older versions of Qlik Sense will continue to work as they did in previous versions. Apps created with Qlik Sense November 2017 or later will respect the difference between the two types of quotes.

Examples: Chart expressions for set modifiers with searches

Examples - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
MyTable:  
Load  
Year(Date) as Year,  
Date#(Date,'YYYY-MM-DD') as ISO_Date,  
Date(Date#(Date,'YYYY-MM-DD'), 'M/D/YYYY') as US_Date,  
Country, Product, Amount  
Inline  
[Date, Country, Product, Amount  
2018-02-20, Canada, Washer, 6  
2018-07-08, Germany, Anchor bolt, 10  
2018-07-14, Germany, Anchor bolt, 3  
2018-08-31, France, Nut, 2  
2018-09-02, Czech Republic, Bolt, 1  
2019-02-11, Czech Republic, Bolt, 3  
2019-07-31, Czech Republic, Washer, 6  
2020-03-13, France, Anchor bolt, 1  
2020-07-12, Canada, Anchor bolt, 8  
2020-09-16, France, Washer, 1];
```

Example 1: Chart expressions with text searches

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers with text searches

Country	Sum (Amount)	Sum({<Country= {"C*"}>} Amount)	Sum({<Country= {"*^R*"}>} Amount)	Sum({<Product= {"*bolt*"}>} Amount)
Totals	41	24	10	26
Canada	14	14	0	8
Czech Republic	10	10	10	4
France	4	0	0	1
Germany	13	0	0	13

Explanation

- Dimensions:
 - Country

- Measures:
 - `Sum(Amount)`
Sum Amount with no set expression.
 - `Sum({<Country={"C*"}>}Amount)`
Sum Amount for all countries that start with c, such as Canada and czech Republic.
 - `Sum({<Country={"*^R*"}>}Amount)`
Sum Amount for all countries that have a word that starts with R, such as Czech Republic.
 - `Sum({<Product={"*bolt*"}>}Amount)`
Sum Amount for all products that contain the string bolt, such as Bolt and Anchor bolt.

Set modifiers with text searches

My new sheet					
Country	Q	Sum (Amount)	Sum({<Country={"C*"}>} Amount)	Sum({<Country={"*^R*"}>} Amount)	Sum({<Product={"*bolt*"}>} Amount)
Totals		41	24	10	26
Canada		14	14	0	8
Czech Republic		10	10	10	4
France		4	0	0	1
Germany		13	0	0	13

Example 2: Chart expressions with numeric searches

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers with numeric searches

Country	Sum (Amount)	<code>Sum({<Year={">2019"}>} Amount)</code>	<code>Sum({<ISO_Date={">=2019-07-01"}>} Amount)</code>	<code>Sum({<US_Date={">=4/1/2018<=12/31/2018"}>} Amount)</code>
Totals	41	10	16	16
Canada	14	8	8	0
Czech Republic	10	0	6	1
France	4	2	2	2
Germany	13	0	0	13

Explanation

- Dimensions:
 - Country
- Measures:

- `Sum(Amount)`
Sum Amount with no set expression.
- `Sum({<Year={">2019"}>}Amount)`
Sum Amount for all years after 2019.
- `Sum({<ISO_Date={">=2019-07-01"}>}Amount)`
Sum Amount for all dates on or after 2019-07-01. The format of the date in the search must match the format of the field.
- `Sum({<US_Date={">=4/1/2018<=12/31/2018"}>}Amount)`
Sum Amount for all dates from 4/1/2018 to 12/31/2018, including the start and end dates. The format of the dates in the search must match the format of the field.

Set modifiers with numeric searches

My new sheet					
Country	Q	Sum (Amount)	Sum({<Year=">2019"} Amount)	Sum({<ISO_Date=">=2019-07-01"} Amount)	Sum({<US_Date=">=4/1/2018<=12 /31/2018"}>}Amount)
Totals		41	10	16	16
Canada		14	8	8	0
Czech Republic		10	0	6	1
France		4	2	2	2
Germany		13	0	0	13

Example 3: Chart expressions with expression searches

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers with expression searches

Country	Sum (Amount)	<code>Sum({<Country= {"=Sum (Amount)>10"}>} Amount)</code>	<code>Sum({<Country= {"=Count(distinct Product)=1"}>} Amount)</code>	<code>Sum({<Product= {"=Count (Amount)>3"}>} Amount)</code>
Totals	41	27	13	22
Canada	14	14	0	8
Czech Republic	10	0	0	0
France	4	0	0	1
Germany	13	13	13	13

Explanation

- Dimensions:
 - Country

- Measures:

- `Sum(Amount)`
Sum Amount with no set expression.
- `Sum({<Country={"=Sum(Amount)>10"}>}Amount)`
Sum Amount for all countries that have an aggregated sum of Amount greater than 10.
- `Sum({<Country={"=Count(distinct Product)=1"}>}Amount)`
Sum Amount for all countries that are associated with exactly one distinct product.
- `Sum({<Product={"=Count(Amount)>3"}>}Amount)`
Sum Amount for all countries that have more than three transactions in the data.

Set modifiers with expression searches

My new sheet

Country	Q	Sum (Amount)	Sum({<Country={"=Sum(Amount)>10"}>} Amount)	Sum({<Country={"=Count(distinct Product)=1"}>} Amount)	Sum({<Product={"=Count(Amount)>3"}>} Amount)
Totals		41	27	13	22
Canada		14	14	0	8
Czech Republic		10	0	0	0
France		4	0	0	1
Germany		13	13	13	13

Examples	Results
<code>sum(\${-1<Product = {"*Internal*", "*Domestic*"}>} Sales)</code>	Returns the sales for current selection, excluding transactions pertaining to products with the string 'Internal' or 'Domestic' in the product name.
<code>sum(\${<Customer = {"=Sum({1<Year = {2007}>} Sales) > 1000000"}>} Sales)</code>	Returns the sales for current selection, but with a new selection in the 'Customer' field: only customers who during 2007 had a total sales of more than 1000000.

Set modifiers with dollar-sign expansions

Dollar-sign expansions are constructs that are calculated before the expression is parsed and evaluated. The result is then injected into the expression instead of the `$(...)`. The calculation of the expression is then made using the result of the dollar expansion.

The expression editor shows a dollar expansion preview so that you can verify what your dollar-sign expansion evaluates to.

Dollar-sign expansion preview in expression editor

The screenshot shows the 'Edit expression' dialog. In the main area, there is a code editor with the following content:

```
1 Sum({<US_Date={">=$(=AddYears(Max(US_Date), -1))}>}Amount)
```

Below the code editor, a preview window displays the expanded expression:

Sum({<US_Date={">=9/16/2019"}>}Amount)

Use dollar-sign expansions when you want to use a calculation inside your element set.

For example, if you want to look at the last possible year only, you can use the following construction:

```
<Year = {$(=Max(Year))}>
```

Max(Year) is calculated first, and the result would be injected in the expression instead of the \${...}.

The result after the dollar expansion will be an expression such as the following:

```
<Year = {2021}>
```

The expression inside the dollar expansion is calculated based on the current selection. This means that if you have a selection in another field, the result of the expression will be affected.

If you want the calculation to be independent of the selection, use set analysis inside the dollar expansion. For example:

```
<Year = {$(=Max({1} Year))}>
```

Strings

When you want the dollar expansion to result in a string, normal quoting rules apply. For example:

```
<Country = {'$(=FirstSortedValue(Country, Date)}'>
```

The result after the dollar expansion will be an expression such as the following:

```
<Country = {'New Zealand'}>
```

You will get a syntax error if you do not use the quotation marks.

Numbers

When you want the dollar expansion to result in a number, ensure that the expansion gets the same formatting as the field. This means that you sometimes need to wrap the expression in a formatting function.

For example:

```
<Amount = {$(=Num(Max(Amount), '###0.00'))}>
```

The result after the dollar expansion will be an expression such as the following:

```
<Amount = {12362.00}>
```

Use a hash to force the expansion to always use decimal point and no thousand separator . For example:

```
<Amount = ${#=Max(Amount)}>
```

Dates

When you want the dollar expansion to result in a date, ensure that the expansion has the correct formatting. This means that you sometimes need to wrap the expression in a formatting function.

For example:

```
<Date = {'$(=Date(Max(Date)))'}>
```

The result after the dollar expansion will be an expression such as the following:

```
<Date = {'12/31/2015'}>
```

Just as with strings, you need to use the correct quotes.

A common use case is that you want your calculation to be limited to the last month (or year). Then you can use a numeric search in combination with the AddMonths() function.

For example:

```
<Date = {">=$(=AddMonths(Today(), -1))"}>
```

The result after the dollar expansion will be an expression such as the following:

```
<Date = {">=9/31/2021"}>
```

This will pick out all events that have occurred the last month.

Example: Chart expressions for set modifiers with dollar-sign expansions

Example - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
Let vToday = Today();
MyTable:
Load
Year(Date) as Year,
Date#(Date, 'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date, 'YYYY-MM-DD'), 'M/D/YYYY') as US_Date,
Country, Product, Amount
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
```

```
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2021-10-15, France, washer, 1];
```

Chart expressions with dollar-sign expansions

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers with dollar-sign expansions

Country	Sum (Amount)	Sum({<US_Date= {\$(vToday)}>} Amount)	Sum({<ISO_Date= {"\$(=Date(Min(ISO_ Date),'YYYY-MM-DD'))"}>} Amount)	Sum({<US_Date= ">=\$(=AddYears(Max (US_Date),-1))"}>} Amount)
Totals	41	1	6	1
Canada	14	0	6	0
Czech Republic	10	0	0	0
France	4	1	0	1
Germany	13	0	0	0

Explanation

- Dimensions:
 - Country
- Measures:
 - Sum(Amount)
Sum Amount with no set expression.
 - Sum({<US_Date={\$vToday}>}Amount)
Sum Amount for all records where the us_date is the same as in the variable vToday.
 - Sum({<ISO_Date={"\$(=Date(Min(ISO_Date),'YYYY-MM-DD'))"}>}Amount)
Sum Amount for all records where the iso_date is the same as the first (smallest) possible iso_date. The Date() function is needed to ensure that the format of the date matches that of the field.
 - Sum({<US_Date=">=\$(=AddYears(Max(US_Date),-1))"}>}Amount)
Sum Amount for all records that have a us_date after or on the date a year before the latest (largest) possible us_date. The AddYears() function will return a date in the format specified by the variable dateFormat, and this needs to match the format of the field us_date.

3 Chart expressions

Set modifiers with dollar-sign expansions

My new sheet					
Country	Q	Sum (Amount)	Sum({<US_Date='[\$(vToday)']>} Amount)	Sum({<ISO_Date= ["\$(=Date(Min(ISO_Date),'YYYY-MM-DD'))"]>} Amount)	Sum({<US_Date= [">=\$(=AddYears(Max(US_Date),-1))"]>} Amount)
Totals		41	1	6	1
Canada		14	0	6	0
Czech Republic		10	0	0	0
France		4	1	0	1
Germany		13	0	0	0

Examples	Results
sum({\$<Year = \${#vLastYear}>} Sales)	Returns the sales for the previous year in relation to current selection. Here, a variable vLastYear containing the relevant year is used in a dollar-sign expansion.
sum({\$<Year = \${#Only(Year)-1}>} Sales)	Returns the sales for the previous year in relation to current selection. Here, a dollar-sign expansion is used to calculate previous year.

Set modifiers with set operators

Set operators are used to include, exclude, or intersect different element sets. They combine the different methods to define element sets.

The operators are the same as those used for set identifiers.

Operators

Operator	Description
+	Union. This binary operation returns a set consisting of the records or elements that belong to any of the two set operands.
-	Exclusion. This binary operation returns a set consisting of the records or elements that belong to the first but not the other of the two set operands. Also, when used as a unary operator, it returns the complement set.
*	Intersection. This binary operation returns a set consisting of the records or elements that belong to both set operands.
/	Symmetric difference (XOR). This binary operation returns a set consisting of the records or elements that belong to either, but not both set operands.

For example, the following two modifiers define the same set of field values:

- <Year = {1997, "20*"}>
- <Year = {1997} + {"20*"}>

Both expressions select 1997 and the years that begin with 20. In other words, this is the union of the two conditions.

Set operators also allow for more complex definitions. For example:

```
<Year = {1997, "20*"} - {2000}>
```

This expression will select the same years as those above, but in addition exclude year 2000.

.

Examples: Chart expressions for set modifiers with set operators

Examples - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
MyTable:  
Load  
Year(Date) as Year,  
Date#(Date, 'YYYY-MM-DD') as ISO_Date,  
Date(Date#, 'YYYY-MM-DD'), 'M/D/YYYY') as US_Date,  
Country, Product, Amount  
Inline  
[Date, Country, Product, Amount  
2018-02-20, Canada, Washer, 6  
2018-07-08, Germany, Anchor bolt, 10  
2018-07-14, Germany, Anchor bolt, 3  
2018-08-31, France, Nut, 2  
2018-09-02, Czech Republic, Bolt, 1  
2019-02-11, Czech Republic, Bolt, 3  
2019-07-31, Czech Republic, Washer, 6  
2020-03-13, France, Anchor bolt, 1  
2020-07-12, Canada, Anchor bolt, 8  
2020-09-16, France, Washer, 1];
```

Chart expressions

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers with set operators

Country	Sum (Amount)	Sum({<Year= "2018"->2020>} Amount)	Sum({<Country=Germany>} Amount)	Sum({<Country=Germany>+P{<Product=Nut>}Country>} Amount)
Totals	41	9	28	17
Canada	14	0	14	0

Country	Sum (Amount)	Sum({<Year= "2018"-{2020}>} Amount)	Sum({<Country=-{Germany}>} Amount)	Sum({<Country={Germany}+P({<Product={Nut}>}Country)>} Amount)
Totals	41	9	28	17
Czech Republic	10	9	10	0
France	4	0	4	4
Germany	13	0	0	13

Explanation

- Dimensions:
 - Country
- Measures:
 - Sum(Amount)

Sum Amount with no set expression.
 - Sum({<Year={">2018"}-{2020}>}Amount)

Sum Amount for all years after 2018, except 2020.
 - Sum({<Country=-{Germany}>}Amount)

Sum Amount for all countries except Germany. Note the unary exclusion operator.
 - Sum({<Country={Germany}+P({<Product={Nut}>}Country)>}Amount)

Sum Amount for Germany and all countries associated with the product Nut.

Set modifiers with set operators

My new sheet				
Country	Q	Sum (Amount)	Sum({<Year="2018"-{2020}>} Amount)	Sum({<Country=-{Germany}>} Amount)
Totals		41	9	28
Canada		14	0	14
Czech Republic		10	9	10
France		4	0	4
Germany		13	0	0

Examples	Results
sum(\${<Product = Product + {OurProduct1} - {OurProduct2}>} Sales)	Returns the sales for the current selection, but with the product “OurProduct1” added to the list of selected products and “OurProduct2” removed from the list of selected products.

Examples	Results
<code>sum({\$<Year = Year + {"20*",1997} - {2000} } >} Sales)</code>	Returns the sales for the current selection but with additional selections in the field "Year": 1997 and all that begin with "20" – however, not 2000. Note that if 2000 is included in the current selection, it will still be included after the modification.
<code>sum({\$<Year = (Year + {"20*",1997}) - {2000} } >} Sales)</code>	Returns almost the same as above, but here 2000 will be excluded, also if it initially is included in the current selection. The example shows the importance of sometimes using brackets to define an order of precedence.
<code>sum({\$<Year = {"*"} - {2000}, Product = {"*bearing*"} } >} Sales)</code>	Returns the sales for the current selection but with a new selection in "Year": all years except 2000; and only for products containing the string 'bearing'.

Set modifiers with implicit set operators

The standard way to write selections in a set modifier is to use an equals sign. For example:

```
Year = {">2015"}
```

The expression to the right of the equals sign in the set modifier is called an element set. It defines a set of distinct field values, in other words a selection.

This notation defines a new selection, disregarding the current selection in the field. So, if the set identifier contains a selection in this field, the old selection will be replaced by the one in the element set.

When you want to base your selection on the current selection in the field, you need to use a different expression

For example, if you want to respect the old selection, and add the requirement that the year is after 2015, you can write the following:

```
Year = Year * {">2015"}
```

The asterisk is a set operator defining an intersection, so you will get the intersection between the current selection in Year, and the additional requirement that the year be after 2015. An alternative way to write this is the following:

```
Year *= {">2015"}
```

That is, the assignment operator (*=) implicitly defines an intersection.

Similarly, implicit unions, exclusions and symmetric differences can be defined using the following: +=, -=, /=

Examples: Chart expressions for set modifiers with implicit set operators

Examples - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
MyTable:  
Load  
Year(Date) as Year,  
Date#(Date,'YYYY-MM-DD') as ISO_Date,  
Date(Date#(Date,'YYYY-MM-DD'), 'M/D/YYYY') as US_Date,  
Country, Product, Amount  
Inline  
[Date, Country, Product, Amount  
2018-02-20, Canada, Washer, 6  
2018-07-08, Germany, Anchor bolt, 10  
2018-07-14, Germany, Anchor bolt, 3  
2018-08-31, France, Nut, 2  
2018-09-02, Czech Republic, Bolt, 1  
2019-02-11, Czech Republic, Bolt, 3  
2019-07-31, Czech Republic, Washer, 6  
2020-03-13, France, Anchor bolt, 1  
2020-07-12, Canada, Anchor bolt, 8  
2020-09-16, France, Washer, 1];
```

Chart expressions with implicit set operators

Create a table in a Qlik Sense sheet with the following chart expressions.

Select Canada and Czech Republic from a list of countries.

Table - Chart expressions with implicit set operators

Country	Sum (Amount)	Sum({<Country*= {Canada}>} Amount)	Sum({<Country-= {Canada}>} Amount)	Sum({<Country+= {France}>} Amount)
Totals	24	14	10	28
Canada	14	14	0	14
Czech Republic	10	0	10	10
France	0	0	0	4

Explanation

- Dimensions:
 - Country

- Measures:

- `Sum(Amount)`

`Sum Amount` for the current selection. Note that only Canada and Czech Republic have non-zero values.

- `Sum({<Country*={Canada}>}Amount)`

`Sum Amount` for the current selection, intersected with the requirement that the country be Canada. If Canada is not part of the user selection, the set expression returns an empty set, and the column will have 0 on all rows.

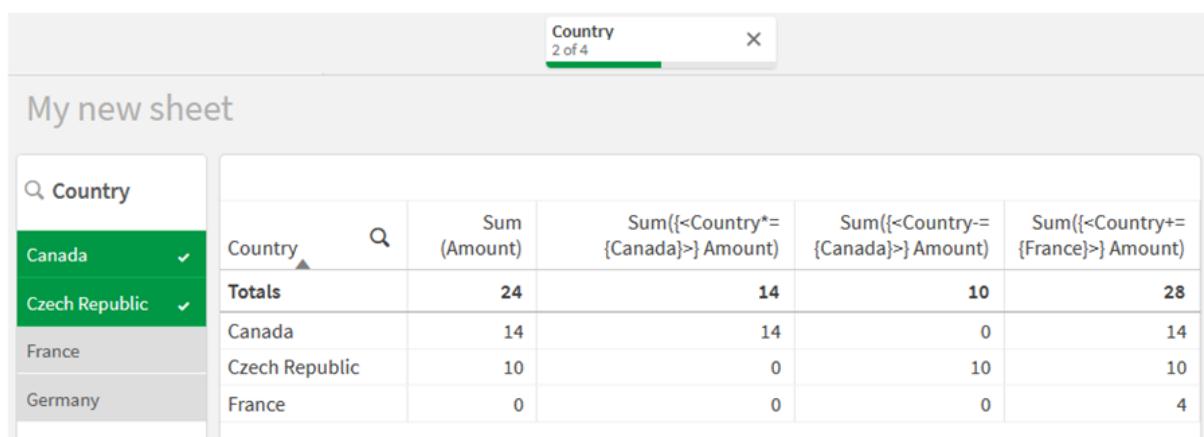
- `Sum({<Country-= {Canada}>}Amount)`

`Sum Amount` for the current selection, but first exclude Canada from the country selection. If Canada is not part of the user selection, the set expression will not change any numbers.

- `Sum({<Country+= {France}>}Amount)`

`Sum Amount` for the current selection, but first add France to the country selection. If France is already part of the user selection, the set expression will not change any numbers.

Set modifiers with implicit set operators



Examples	Results
<code>sum(\${<Product += {OurProduct1, OurProduct2}>} Sales)</code>	Returns the sales for the current selection, but using an implicit union to add the products 'OurProduct1' and 'OurProduct2' to the list of selected products.
<code>sum(\${<Year += {"20*",1997} - {2000}>} Sales)</code>	Returns the sales for the current selection but using an implicit union to add a number of years in the selection: 1997 and all that begin with "20" – however, not 2000. Note that if 2000 is included in the current selection, it will still be included after the modification. Same as <code><Year=Year + {"20*",1997}-{2000}></code> .
<code>sum(\${<Product *= {OurProduct1}>} Sales)</code>	Returns the sales for the current selection, but only for the intersection of currently selected products and the product OurProduct1.

Set modifiers using set functions

Sometimes you need to define a set of field values using a nested set definition. For example, you may want to select all customers that have bought a specific product, without selecting the product.

In such cases, use the element set functions `P()` and `E()`. These return the element sets of possible values and excluded values of a field, respectively. Inside the brackets, you can specify the field in question, and a set expression that defines the scope. For example:

```
P({1<Year = {2021}>} Customer)
```

This will return the set of customers that had transactions in 2021. You can then use this in a set modifier. For example:

```
Sum({<Customer = P({1<Year = {2021}>} Customer)>} Amount)
```

This set expression will select these customers, but it will not restrict the selection to 2021.

These functions cannot be used in other expressions.

Additionally, only natural sets can be used inside the element set functions. That is, a set of records that can be defined by a simple selection.

For example, the set given by `{1-$}` cannot always be defined through a selection, and is therefore not a natural set. Using these functions on non-natural sets will return unexpected results.

Examples: Chart expressions for set modifiers using set functions

Examples - chart expressions

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```
MyTable:  
Load  
Year(Date) as Year,  
Date#(Date,'YYYY-MM-DD') as ISO_Date,  
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,  
Country, Product, Amount  
Inline  
[Date, Country, Product, Amount  
2018-02-20, Canada, Washer, 6  
2018-07-08, Germany, Anchor bolt, 10  
2018-07-14, Germany, Anchor bolt, 3  
2018-08-31, France, Nut, 2  
2018-09-02, Czech Republic, Bolt, 1  
2019-02-11, Czech Republic, Bolt, 3  
2019-07-31, Czech Republic, Washer, 6
```

2020-03-13, France, Anchor bolt, 1
 2020-07-12, Canada, Anchor bolt, 8
 2020-09-16, France, Washer, 1];

Chart expressions

Create a table in a Qlik Sense sheet with the following chart expressions.

Table - Set modifiers using set functions				
Country	Sum (Amount)	Sum({<Country=P ({<Year= {2019}>}Country)>} Amount)	Sum({<Product=P ({<Year= {2019}>}Product)>} Amount)	Sum({<Country=E ({<Product= {Washer}>}Country)>} Amount)
Totals	41	10	17	13
Canada	14	0	6	0
Czech Republic	10	10	10	0
France	4	0	1	0
Germany	13	0	0	13

Explanation

- Dimensions:
 - Country
- Measures:
 - Sum(Amount)
 Sum Amount with no set expression.
 - Sum({<Country=P({<Year={2019}>} Country)>} Amount)
 Sum Amount for the countries that are associated with year 2019. It will however not limit the calculation to 2019.
 - Sum({<Product=P({<Year={2019}>} Product)>} Amount)
 Sum Amount for the products that are associated with year 2019. It will however not limit the calculation to 2019.
 - Sum({<Country=E({<Product={Washer}>} Country)>} Amount)
 Sum Amount for the countries that are not associated with the product Washer.

Set modifiers using set functions

Country	Sum (Amount)	Sum({<Country=P({<Year=[2019]>} Country)>} Amount)	Sum({<Product=P({<Year=[2019]>} Product)>} Amount)	Sum({<Country=E({<Product='Washer'>} Country)>} Amount)
Totals	41	10	17	13
Canada	14	0	6	0
Czech Republic	10	10	10	0
France	4	0	1	0
Germany	13	0	0	13

Examples	Results
<code>sum(\${<Customer = P({1<Product= {'Shoe'}>} Customer)>} Sales)</code>	Returns the sales for current selection, but only those customers that ever have bought the product 'Shoe'. The element function P() here returns a list of possible customers; those that are implied by the selection 'Shoe' in the field Product.
<code>sum(\${<Customer = P({1<Product= {'Shoe'}>})>} Sales)</code>	Same as above. If the field in the element function is omitted, the function will return the possible values of the field specified in the outer assignment.
<code>sum(\${<Customer = P({1<Product= {'Shoe'}>} Supplier)>} Sales)</code>	Returns the sales for current selection, but only those customers that ever have supplied the product 'Shoe', that is, the customer is also a supplier. The element function P() here returns a list of possible suppliers; those that are implied by the selection 'Shoe' in the field Product. The list of suppliers is then used as a selection in the field Customer.
<code>sum(\${<Customer = E({1<Product= {'Shoe'}>})>} Sales)</code>	Returns the sales for current selection, but only those customers that never bought the product 'Shoe'. The element function E() here returns the list of excluded customers; those that are excluded by the selection 'Shoe' in the field Product.

Inner and outer set expressions

Set expressions can be used inside and outside aggregation functions, and are enclosed in curly brackets.

When you use a set expression inside an aggregation function, it can look like this:

Example: Inner set expression

```
sum( {$<Year={2021}>} Sales )
```

Use a set expression outside the aggregation function if you have expressions with multiple aggregations and want to avoid writing the same set expression in every aggregation function.

If you use an outer set expression, it must be placed at the beginning of the scope.

Example: Outer set expression

```
{<Year={2021}>} sum(Sales) / count(distinct Customer)
```

If you use a set expression outside the aggregation function, you can also apply it on existing master measures.

Example: Outer set expression applied to master measure

```
{<Year={2021}>} [Master Measure]
```

A set expression used outside aggregation functions affects the entire expression, unless it is enclosed in brackets then the brackets define the scope. In the lexical scoping example below, the set expression is only applied to the aggregation inside the brackets.

Example: Lexical scoping

```
( {<Year={2021}>} sum(Amount) / count(distinct Customer) ) - avg(customersales)
```

Rules

Lexical scope

The set expression affects the entire expression, unless it is enclosed in brackets. If so, the brackets define the lexical scope.

Position

The set expression must be placed in the beginning of the lexical scope.

Context

The context is the selection that is relevant for the expression. Traditionally, the context has always been the default state of current selection. But if an object is set to an alternate state, the context is the alternate state of the current selection.

You can also define a context in the form of an outer set expression.

Inheritance

Inner set expressions have precedence over outer set expressions. If the inner set expression contains a set identifier, it replaces the context. Otherwise, the context and the set expression will be merged.

- {\$<SetExpression>} - overrides the outer set expression
- {<SetExpression>} - is merged with the outer set expression

Element set assignment

The element set assignment determines how the two selections are merged. If a normal equals sign is used, the selection in the inner set expression has precedence. Otherwise, the implicit set operator will be used.

- {<Field={value}>} - this inner selection replaces any outer selection in “Field”.
- {<Field+={value}>} - this inner selection is merged with the outer selection in “Field”, using the union operator.
- {<Field*={value}>} - this inner selection is merged with the outer selection in “Field”, using the intersection operator.

Inheritance in multiple steps

The inheritance can occur in multiple steps. Examples:

- Current Selection → sum(Amount)
The aggregation function will use the context, which here is the current selection.
- Current Selection → {<Set1>} sum(Amount)
Set1 will inherit from current selection, and the result will be the context for the aggregation function.
- Current Selection → {<Set1>} ({<set2>} sum(Amount))
Set2 will inherit from Set1, which in turn inherits from current selection, and the result will be the context for the aggregation function.

The Aggr() function

The Aggr() function creates a nested aggregation that has two independent aggregations. In the example below, a Count() is calculated for each value of Dim, and the resulting array is aggregated using the sum() function.

Example:

```
Sum(Aggr(Count(X),Dim))
```

Count() is the inner aggregation and sum() is the outer aggregation.

- The inner aggregation does not inherit any context from the outer aggregation.
- The inner aggregation inherits the context from the Aggr() function, which may contain a set expression.
- Both the Aggr() function and the outer aggregation function inherit the context from an outer set expression.

Tutorial - Creating a set expression

You can build set expressions in Qlik Sense to support data analysis. In this context, the analysis is often referred to as set analysis. Set analysis offers a way of defining a scope that is different from the set of records defined by the current selection in an app.

What you will learn

This tutorial provides the data and chart expressions to build set expressions using set modifiers, identifiers and operators.

Who should complete this tutorial

This tutorial is for app developers who are comfortable working with the script editor and chart expressions.

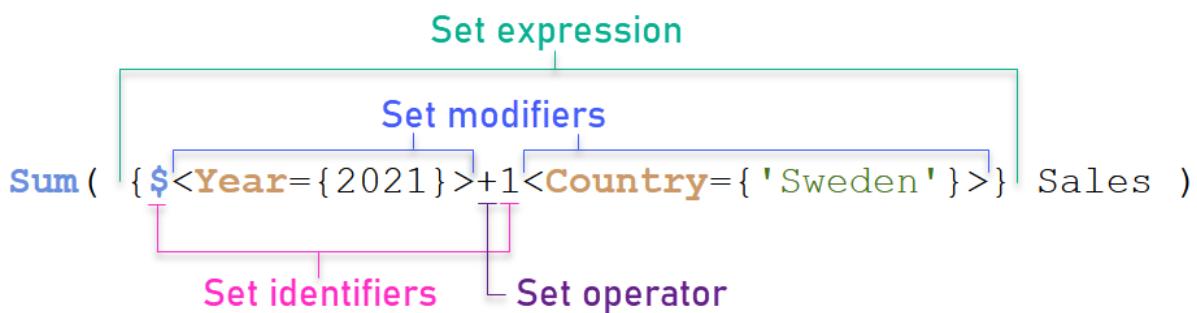
What you need to do before you start

A Qlik Sense Enterprise professional access allocation, which allows you to load data and create apps.

Elements in a set expression

Set expressions are enclosed in an aggregation function, such as `Sum()`, `Max()`, `Min()`, `Avg()`, or `Count()`. Set expressions are constructed from building blocks known as elements. These elements are set modifiers, identifiers, and operators.

Elements in a set expression



The set expression above, for example, is built from the aggregation `Sum(Sales)`. The set expression is enclosed in the outer curly brackets: `{ }`

The first operand in the expression is: `$<Year={2021}>`

This operand returns sales for the year 2021 for the current selection. The modifier, `<year={2021}>`, contains the selection of the year 2021. The `$` set identifier indicates that the set expression is based on current selection.

The second operand in the expression is: `1<Country={'Sweden' }>`

This operand returns Sales for Sweden. The modifier, `<country={'Sweden' }>`, contains the selection of the country Sweden. The `1` set identifier indicates that selections made in the app will be ignored.

Finally, the `+` set operator indicates that the expression returns a set consisting of the records that belongs to any of the two set operands.

Creating a set expression tutorial

Complete the following procedures to create the set expressions shown in this tutorial.

Create a new app and load data

Do the following:

1. Create a new app.
2. Click **Script editor**. Alternatively, click **Prepare > Data load editor** in the navigation bar.
3. Create a new section in the **Data load editor**.
4. Copy the following data and paste it into the new section: *Set expression tutorial data (page 306)*
5. Click **Load data**. The data is loaded as an inline load.

Create set expressions with modifiers

The set modifier consists of one or more field names, each followed by a selection that should be made on the field. The modifier is enclosed by angled brackets. For example, in this set expression:

```
Sum ( {<Year = {2015}>} Sales )
```

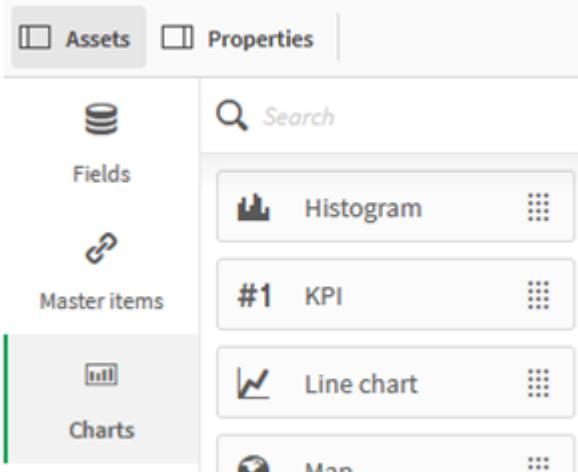
The modifier is:

```
<Year = {2015}>
```

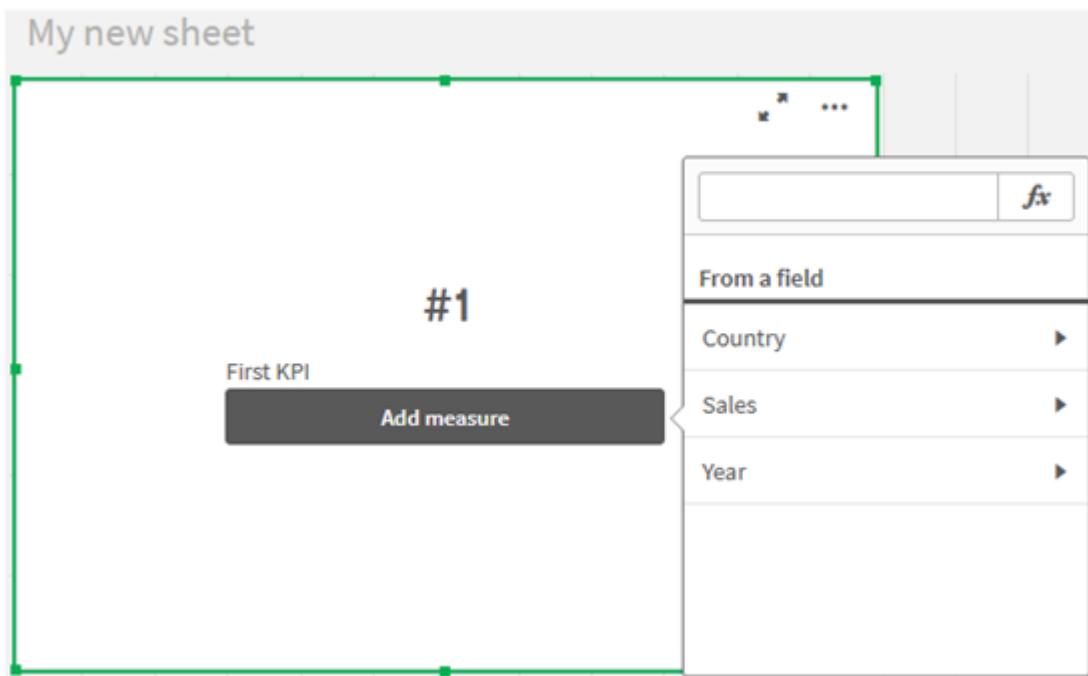
This modifier specifies that data from the year 2015 will be selected. The curly brackets in which the modifier is enclosed indicate a set expression.

Do the following:

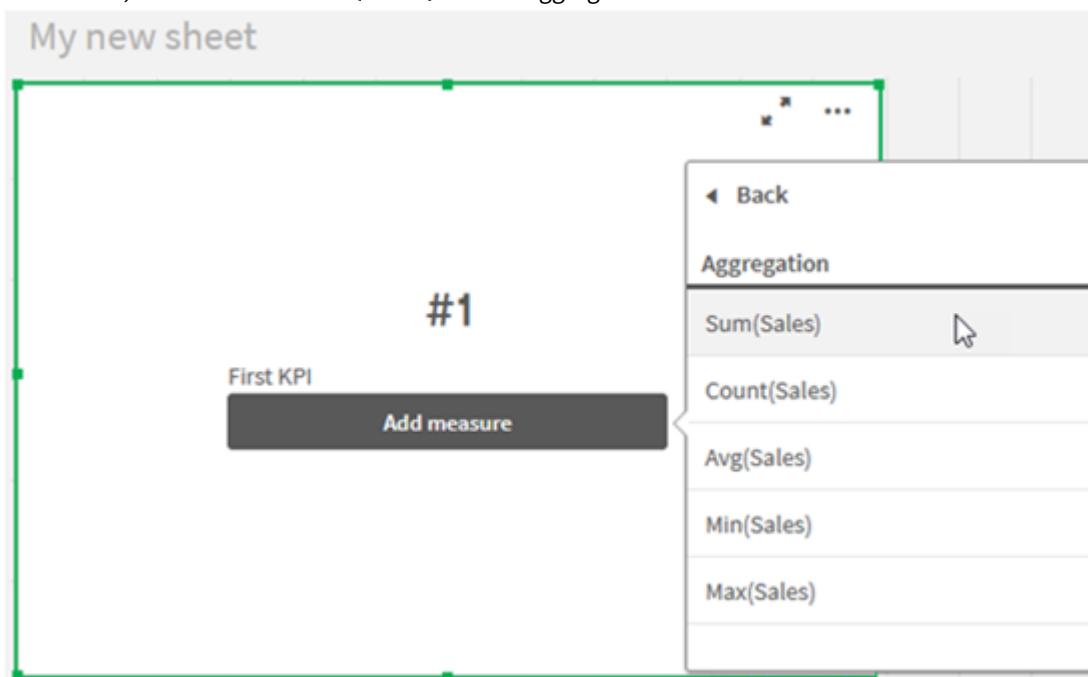
1. In a sheet, open the **Assets** panel from the navigation bar, and then click **Charts**.



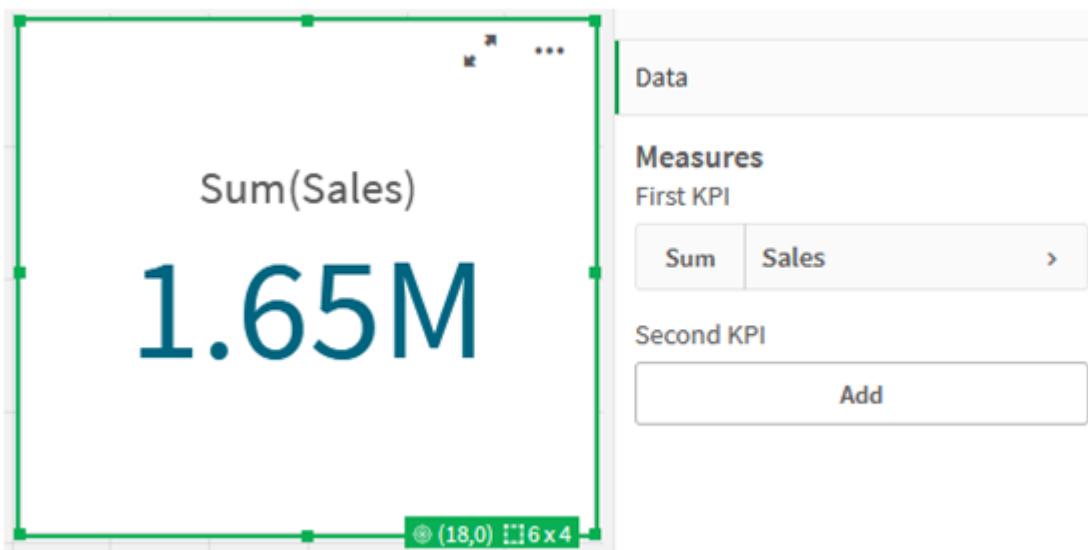
2. Drag a **KPI** onto the sheet, and then click **Add measure**.



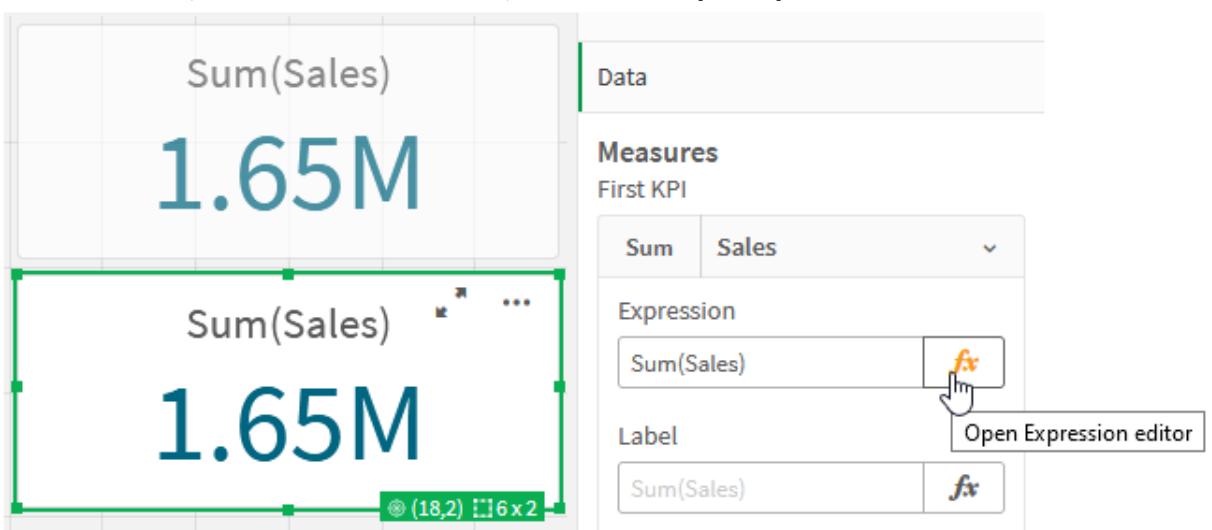
3. Click sales, and then select sum(sales) for the aggregation.



The KPI shows the sum of sales for all years.



4. Copy and paste the KPI to create a new KPI.
5. Click the new KPI, click **Sales** under **Measures**, and then click **Open Expression editor**.



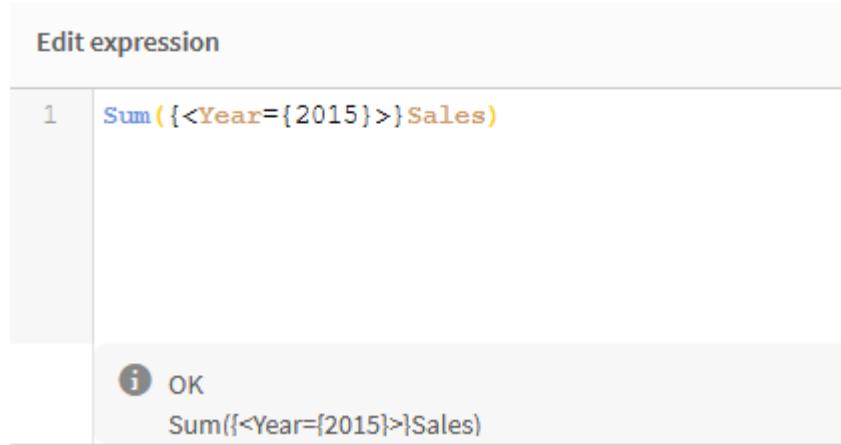
The expression editor open with the aggregation sum(sales).



6. In the expression editor, create an expression to sum Sales for 2015 only:
 - i. Add curly brackets to indicate a set expression: `sum({}sales)`
 - ii. Add angle brackets to indicate a set modifier: `sum({<>}sales)`
 - iii. In the angle brackets, add the field to be selected, in this case the field is `Year`, followed by an equal sign. Next, enclose 2015 in another set of curly brackets. The resulting set modifier is: `{<Year={2015}>}.`

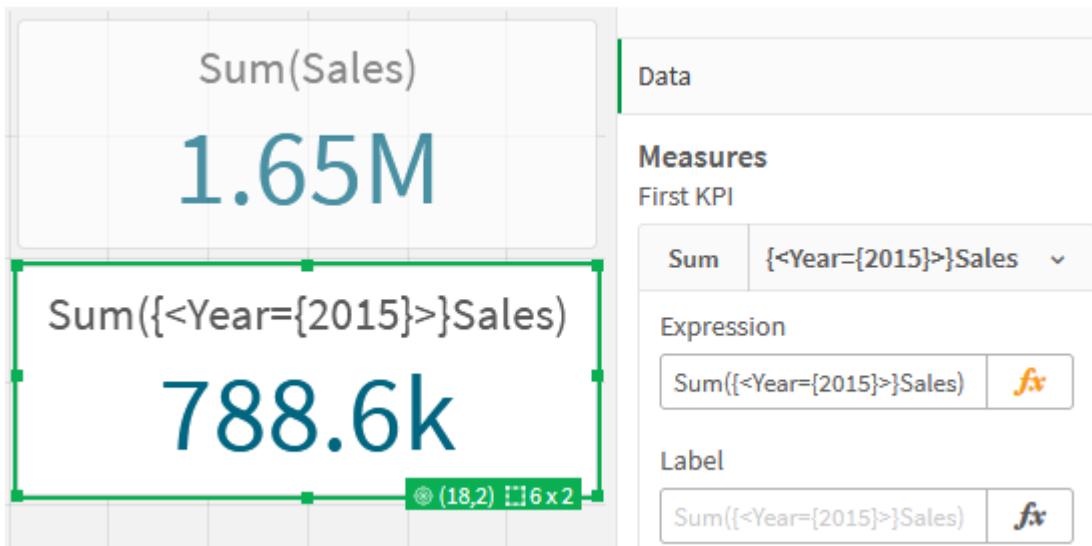
The entire expression is:

`Sum({<Year={2015}>}Sales)`



- iii. Click **Apply** to save the expression and to close the expression editor. The sum of Sales for 2015

is shown in the KPI.



7. Create two more KPIs with the following expressions:

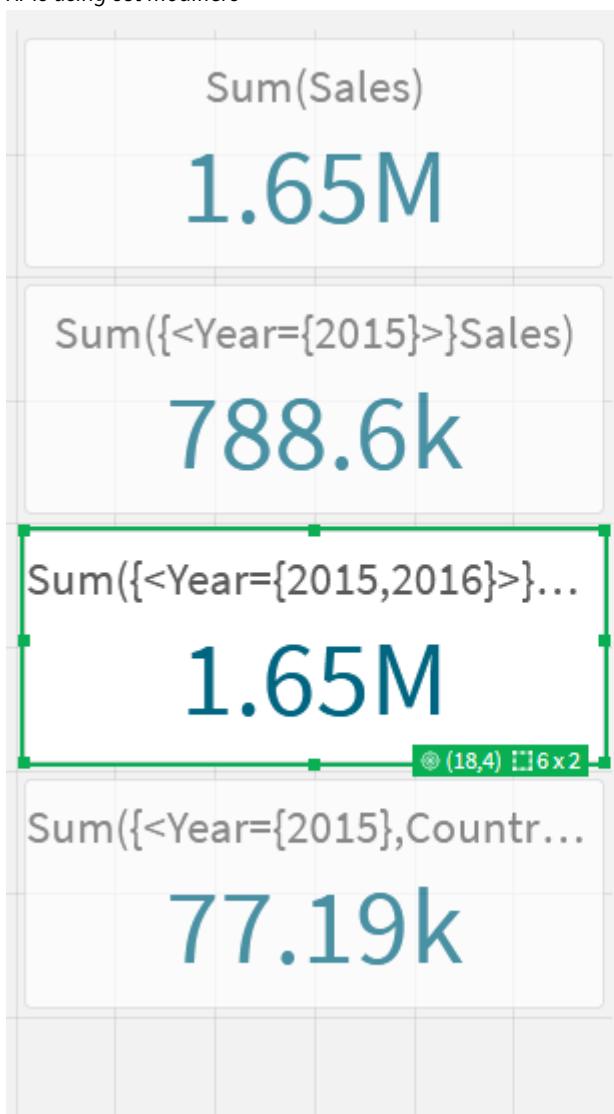
`Sum({<Year=[2015,2016]>}Sales)`

The modifier in the above is `<Year=[2015,2016]>`. The expression will return the sum of Sales for 2015 and 2016.

`Sum({<Year=[2015],Country={'Germany'}>} Sales)`

The modifier in the above is `<Year=[2015], Country={'Germany'}>`. The expression will return the sum of Sales for 2015, where 2015 intersects with Germany.

KPIs using set modifiers



Data

Measures

First KPI

Sum	{<Year=[2015,2016]}S	fx
-----	----------------------	----

Expression

Sum({<Year=[2015,2016]}S)	fx
---------------------------	----

Label

Sum({<Year=[2015,2016]}...)	fx
-----------------------------	----

Number formatting

Auto

Master item

Add new

Delete

Second KPI

Add

Add set identifiers

The set expressions above will use current selections as base, because an identifier was not used. Next, add identifiers to specify the behavior when selections are made.

Do the following:

On your sheet, build or copy the following set expressions:

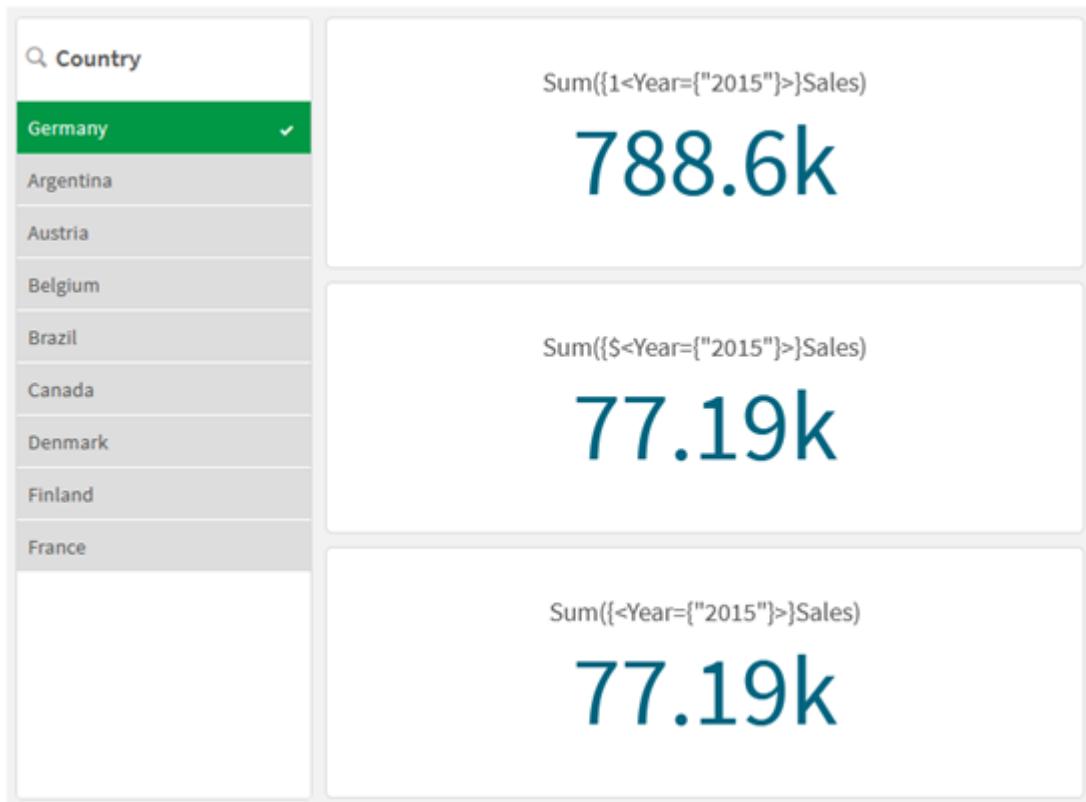
`Sum({$<Year={"2015"}>}Sales)`

The \$ identifier will base the set expression on the current selections made in the data. This is also the default behavior when an identifier is not used.

`Sum({1<Year={"2015"}>}Sales)`

The 1 identifier will cause the aggregation of sum(sales) on 2015 to ignore the current selection. The value of the aggregation will not change when the user makes other selections. For example, when Germany is selected below, the value for the aggregate sum of 2015 does not change.

KPIs using set modifiers and identifiers



Add operators

Set operators are used to include, exclude, or intersect data sets. All operators use sets as operands and return a set as result.

You can use set operators in two different situations:

- To perform a set operation on set identifiers, representing sets of records in data.
- To perform a set operation on the element sets, on the field values, or inside a set modifier.

Do the following:

On your sheet, build or copy the following set expression:

```
Sum({$<Year={2015}>+1<Country={'Germany'}>}Sales)
```

The plus sign (+) operator produces a union of the data sets for 2015 and Germany. As explained with set identifiers above, the dollar sign (\$) identifier means current selections will be used for the first operand, `<Year={2015}>`, will be respected. The 1 identifier means selection will be ignored for the second operand, `<Country={'Germany'}>`.

KPI using plus sign (+) operator

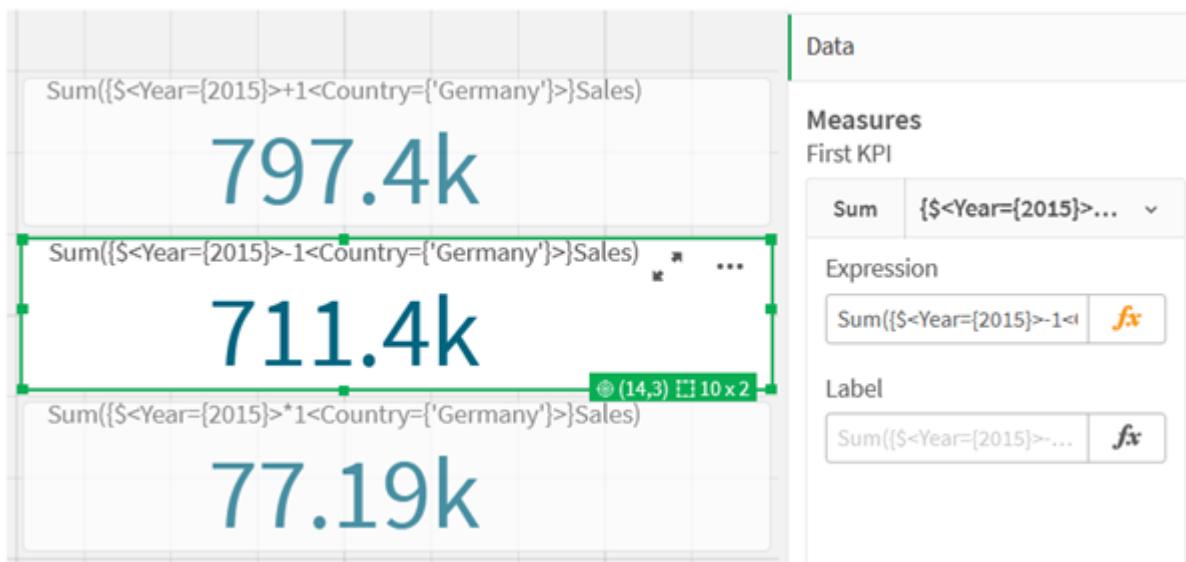


Alternatively, use a minus sign (-) to return a data set that consists of the records that belong to 2015 but not Germany. Or, use an asterisk (*) to return a set consisting of the records that belong to both sets.

`Sum({$<Year=[2015]>-1<Country=['Germany']>}Sales)`

`Sum({$<Year=[2015]>*1<Country=['Germany']>}Sales)`

KPIs using operators



Set expression tutorial data

Load script

Load the following data as an inline load and then create the chart expressions in the tutorial.

```
//Create table SalesByCountry
SalesByCountry:
Load * Inline [
Country, Year, Sales
Argentina, 2016, 66295.03
Argentina, 2015, 140037.89
Austria, 2016, 54166.09
Austria, 2015, 182739.87
```

```

Belgium, 2016, 182766.87
Belgium, 2015, 178042.33
Brazil, 2016, 174492.67
Brazil, 2015, 2104.22
Canada, 2016, 101801.33
Canada, 2015, 40288.25
Denmark, 2016, 45273.25
Denmark, 2015, 106938.41
Finland, 2016, 107565.55
Finland, 2015, 30583.44
France, 2016, 115644.26
France, 2015, 30696.98
Germany, 2016, 8775.18
Germany, 2015, 77185.68
];

```

Syntax for set expressions

The full syntax (not including the optional use of standard brackets to define precedence) is described using Backus-Naur Formalism:

```

set_expression ::= { set_entity { set_operator set_entity } }
set_entity ::= set_identifier [ set_modifier ] | set_modifier
set_identifier ::= 1 | $ | $N | $_N | bookmark_id | bookmark_name
set_operator ::= + | - | * | /
set_modifier ::= < field_selection {, field_selection } >
field_selection ::= field_name [ = | += | -= | *= | /= ] element_set_
expression
element_set_expression ::= [ - ] element_set { set_operator element_set }
element_set ::= [ field_name ] | { element_list } | element_function
element_list ::= element {, element }
element_function ::= ( P | E ) ( [set_expression] [field_name] )
element ::= field_value | " search_mask "

```

3.3 General syntax for chart expressions

The following general syntax structure can be used for chart expressions, with many optional parameters:

```

expression ::= ( constant | expressionname | operator1 expression | expression operator2
expression | function | aggregation function | (expression) )
where:

```

constant is a string (a text, a date or a time) enclosed by single straight quotation marks, or a number.
Constants are written without thousands separator and with a decimal point as decimal separator.

expressionname is the name (label) of another expression in the same chart.

operator1 is a unary operator (working on one expression, the one to the right).

operator2 is a binary operator (working on two expressions, one on each side).

```

function ::= functionname ( parameters )
parameters ::= expression {, expression}

```

The number and types of parameters are not arbitrary. They depend on the function used.

```
aggregationfunction ::= aggregationfunctionname ( parameters2 )
parameters2 ::= aggreexpression { , aggreexpression }
```

The number and types of parameters are not arbitrary. They depend on the function used.

3.4 General syntax for aggregations

The following general syntax structure can be used for aggregations, with many optional parameters:

```
aggreexpression ::= ( fieldref | operator1 aggreexpression | aggreexpression operator2
aggreexpression | functioninaggr | ( aggreexpression ) )
```

fieldref is a field name.

```
functionaggr ::= functionname ( parameters2 )
```

Expressions and functions can thus be nested freely, as long as **fieldref** is always enclosed by exactly one aggregation function and provided the expression returns an interpretable value, Qlik Sense does not give any error messages.

4 Operators

This section describes the operators that can be used in Qlik Sense. There are two types of operators:

- Unary operators (take only one operand)
- Binary operators (take two operands)

Most operators are binary.

The following operators can be defined:

- Bit operators
- Logical operators
- Numeric operators
- Relational operators
- String operators

4.1 Bit operators

All bit operators convert (truncate) the operands to signed integers (32 bit) and return the result in the same way. All operations are performed bit by bit. If an operand cannot be interpreted as a number, the operation will return NULL.

Bit operators

Operator	Full name	Description
bitnot	Bit inverse.	<p>Unary operator. The operation returns the logical inverse of the operand performed bit by bit.</p> <p>Example:</p> <p><code>bitnot 17</code> returns -18</p>
bitand	Bit and.	<p>The operation returns the logical AND of the operands performed bit by bit.</p> <p>Example:</p> <p><code>17 bitand 7</code> returns 1</p>
bitor	Bit or.	<p>The operation returns the logical OR of the operands performed bit by bit.</p> <p>Example:</p> <p><code>17 bitor 7</code> returns 23</p>

Operator	Full name	Description
bitxor	Bit exclusive or.	<p>The operation returns the logical exclusive or of the operands performed bit by bit.</p> <p>Example:</p> <p><code>17 bitxor 7</code> returns 22</p>
>>	Bit right shift.	<p>The operation returns the first operand shifted to the right. The number of steps is defined in the second operand.</p> <p>Example:</p> <p><code>8 >> 2</code> returns 2</p>
<<	Bit left shift.	<p>The operation returns the first operand shifted to the left. The number of steps is defined in the second operand.</p> <p>Example:</p> <p><code>8 << 2</code> returns 32</p>

4.2 Logical operators

All logical operators interpret the operands logically and return True (-1) or False (0) as result.

Logical operators

Operator	Description
not	Logical inverse. One of the few unary operators. The operation returns the logical inverse of the operand.
and	Logical and. The operation returns the logical and of the operands.
or	Logical or. The operation returns the logical or of the operands.
Xor	Logical exclusive or. The operation returns the logical exclusive or of the operands. I.e. like logical or, but with the difference that the result is False if both operands are True.

4.3 Numeric operators

All numeric operators use the numeric values of the operands and return a numeric value as result.

Numeric operators

Operator	Description
+	Sign for positive number (unary operator) or arithmetic addition. The binary operation returns the sum of the two operands.
-	Sign for negative number (unary operator) or arithmetic subtraction. The unary operation returns the operand multiplied by -1, and the binary the difference between the two operands.
*	Arithmetic multiplication. The operation returns the product of the two operands.
/	Arithmetic division. The operation returns the ratio between the two operands.

4.4 Relational operators

All relational operators compare the values of the operands and return True (-1) or False (0) as the result. All relational operators are binary.

Relational operators

Operator	Description
<	Less than. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
<=	Less than or equal. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
>	Greater than. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
>=	Greater than or equal. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
=	Equals. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.
<>	Not equivalent to. A numeric comparison is made if both operands can be interpreted numerically. The operation returns the logical value of the evaluation of the comparison.

Operator	Description
precedes	<p>Unlike the <code><</code> operator no attempt is made to make a numeric interpretation of the argument values before the comparison. The operation returns true if the value to the left of the operator has a text representation which, in string comparison, comes before the text representation of the value on the right.</p> <p>Example:</p> <pre>'1' precedes '2' returns FALSE '1' precedes '2' returns TRUE</pre> <p>as the ASCII value of a space (' ') is of less value than the ASCII value of a number.</p> <p>Compare this to:</p> <pre>'1' < '2' returns TRUE '1' < '2' returns TRUE</pre>
follows	<p>Unlike the <code>></code> operator no attempt is made to make a numeric interpretation of the argument values before the comparison. The operation returns true if the value to the left of the operator has a text representation which, in string comparison, comes after the text representation of the value on the right.</p> <p>Example:</p> <pre>'2' follows '1' returns FALSE '2' follows '1' returns TRUE</pre> <p>as the ASCII value of a space (' ') is of less value than the ASCII value of a number.</p> <p>Compare this to:</p> <pre>'2' > '1' returns TRUE '2' > '1' returns TRUE</pre>

4.5 String operators

There are two string operators. One uses the string values of the operands and return a string as result. The other one compares the operands and returns a boolean value to indicate match.

&

String concatenation. The operation returns a text string, that consists of the two operand strings, one after another.

Example:

```
'abc' & 'xyz' returns 'abcxyz'
```

like

String comparison with wildcard characters. The operation returns a boolean True (-1) if the string before the operator is matched by the string after the operator. The second string may contain the wildcard characters * (any number of arbitrary characters) or ? (one arbitrary character).

Example:

```
'abc' like 'a*' returns True (-1)
```

```
'abcd' like 'a?c*' returns True (-1)
```

```
'abc' like 'a??bc' returns False (0)
```

5 Script and chart functions

Transform and aggregate data using functions in data load scripts and chart expressions.

Many functions can be used in the same way in both data load scripts and chart expressions, but there are a number of exceptions:

- Some functions can only be used in data load scripts, denoted by - script function.
- Some functions can only be used in chart expressions, denoted by - chart function.
- Some functions can be used in both data load scripts and chart expressions, but with differences in parameters and application. These are described in separate topics denoted by - script function or - chart function.

5.1 Analytic connections for server-side extensions (SSE)

Functions enabled by analytic connections will only be visible if you have configured the analytic connections and Qlik Sense has started.

You configure the analytic connections in the QMC, see the topic "Creating an analytic connection" in the guide *Manage Qlik Sense sites*.

In Qlik Sense Desktop, you configure the analytic connections by editing the *Settings.ini* file, see the topic "Configuring analytic connections in Qlik Sense Desktop" in the guide *Qlik Sense Desktop*.

5.2 Aggregation functions

The family of functions known as aggregation functions consists of functions that take multiple field values as their input and return a single result per group, where the grouping is defined by a chart dimension or a **group by** clause in the script statement.

Aggregation functions include **Sum()**, **Count()**, **Min()**, **Max()**, and many more.

Most aggregation functions can be used in both the data load script and chart expressions, but the syntax differs.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

When naming an entity, avoid assigning the same name to more than one field, variable, or measure. There is a strict order of precedence for resolving conflicts between entities with identical names. This order is reflected in any objects or contexts in which these entities are used. This order of precedence is as follows:

- Inside an aggregation, a field has precedence over a variable. Measure labels are not relevant in aggregations and are not prioritized.

- Outside an aggregation, a measure label has precedence over a variable, which in turn has precedence over a field name.
- Additionally, outside an aggregation, a measure can be re-used by referencing its label, unless the label is in fact a calculated one. In that situation, the measure drops in significance in order to reduce risk of self-reference, and in this case the name will always be interpreted first as a measure label, second as a field name, and third as a variable name.

Using aggregation functions in a data load script

Aggregation functions can only be used inside **LOAD** and **SELECT** statements.

Using aggregation functions in chart expressions

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

An aggregation function aggregates over the set of possible records defined by the selection. However, an alternative set of records can be defined by using a set expression in set analysis.

How aggregations are calculated

An aggregation loops over the records of a specific table, aggregating the records in it. For example, **Count(<Field>)** will count the number of records in the table where **<Field>** resides. Should you want to aggregate just the distinct field values, you need to use the **distinct** clause, such as **Count(distinct <Field>)**.

If the aggregation function contains fields from different tables, the aggregation function will loop over the records of the cross product of the tables of the constituent fields. This has a performance penalty, and for this reason such aggregations should be avoided, particularly when you have large amounts of data.

Aggregation of key fields

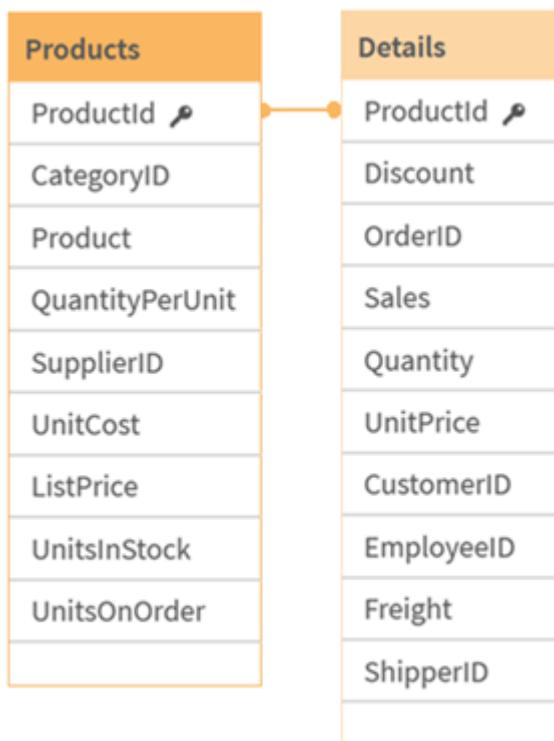
The way aggregations are calculated means that you cannot aggregate key fields because it is not clear which table should be used for the aggregation. For example, if the field **<Key>** links two tables, it is not clear whether **Count(<Key>)** should return the number of records from the first or the second table.

However, if you use the **distinct** clause, the aggregation is well-defined and can be calculated.

So, if you use a key field inside an aggregation function without the **distinct** clause, Qlik Sense will return a number which may be meaningless. The solution is to either use the **distinct** clause, or use a copy of the key – a copy that resides in one table only.

For example, in the following tables, ProductID is the key between the tables.

ProductID key between Products and Details tables



Count(ProductID) can be counted either in the Products table (which has only one record per product – ProductID is the primary key) or it can be counted in the Details table (which most likely has several records per product). If you want to count the number of distinct products, you should use Count(distinct ProductID). If you want to count the number of rows in a specific table, you should not use the key.

Basic aggregation functions

Basic aggregation functions overview

Basic aggregation functions are a group of the most common aggregation functions.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Basic aggregation functions in the data load script

FirstSortedValue

FirstSortedValue() returns the value from the expression specified in **value** that corresponds to the result of sorting the **sort_weight** argument, for example, the name of the product with the lowest unit price. The nth value in the sort order, can be specified in **rank**. If more than one resulting value shares the same **sort_weight** for the specified **rank**, the function returns NULL. The sorted values are iterated over a number of records, as defined by a **group by** clause, or aggregated across the full data set if no **group by** clause is defined.

```
FirstSortedValue ([ distinct ] expression, sort_weight [, rank ])
```

Max

Max() finds the highest numeric value of the aggregated data in the expression, as defined by a **group by** clause. By specifying a **rank** n, the nth highest value can be found.

```
Max ( expression[, rank] )
```

Min

Min() returns the lowest numeric value of the aggregated data in the expression, as defined by a **group by** clause. By specifying a **rank** n, the nth lowest value can be found.

```
Min ( expression[, rank] )
```

Mode

Mode() returns the most commonly-occurring value, the mode value, of the aggregated data in the expression, as defined by a **group by** clause. The **Mode()** function can return numeric values as well as text values.

```
Mode (expression )
```

Only

Only() returns a value if there is one and only one possible result from the aggregated data. If records contain only one value then that value is returned, otherwise NULL is returned. Use the **group by** clause to evaluate over multiple records. The **Only()** function can return numeric and text values.

```
Only (expression )
```

Sum

Sum() calculates the total of the values aggregated in the expression, as defined by a **group by** clause.

```
Sum ([distinct]expression)
```

Basic aggregation functions in chart expressions

Chart aggregation functions can only be used on fields in chart expressions. The argument expression of one aggregation function must not contain another aggregation function.

FirstSortedValue

FirstSortedValue() returns the value from the expression specified in **value** that corresponds to the result of sorting the **sort_weight** argument, for example, the name of the product with the lowest unit price. The nth value in the sort order, can be specified in **rank**. If more than one resulting value shares the same **sort_weight** for the specified **rank**, the function returns NULL.

```
FirstSortedValue - chart function([SetExpression]) [DISTINCT] [TOTAL [<fld  
{, fld}>]] value, sort_weight [,rank])
```

Max

Max() finds the highest value of the aggregated data. By specifying a **rank** n, the nth highest value can be found.

Max - chart function
Max() finds the highest value of the aggregated data. By specifying a rank n, the nth highest value can be found. You might also want to look at **FirstSortedValue** and **rangemax**, which have similar functionality to the **Max** function.
Max([SetExpression]) [TOTAL [<fld {, fld}>]] expr [,rank])

```

numeric ArgumentsArgumentDescriptionexprThe expression or field containing
the data to be measured.rankThe default value of rank is 1, which corresponds
to the highest value. By specifying rank as 2, the second highest value is
returned. If rank is 3, the third highest value is returned, and so
on.SetExpressionBy default, the aggregation function will aggregate over the
set of possible records defined by the selection. An alternative set of
records can be defined by a set analysis expression. TOTALIf the word TOTAL
occurs before the function arguments, the calculation is made over all
possible values given the current selections, and not just those that pertain
to the current dimensional value, that is, it disregards the chart
dimensions. By using TOTAL [<fld {.fld}>], where the TOTAL qualifier is
followed by a list of one or more field names as a subset of the chart
dimension variables, you create a subset of the total possible
values. DataCustomerProductUnitSalesUnitPrice
AstridaAA416AstridaAA1015AstridaBB99BetacabBB510BetacabCC220BetacabDD-
25CanutilityAA815CanutilityCC-19Examples and resultsExamplesResultsMax
(UnitSales)10, because this is the highest value in UnitSales. The value of an
order is calculated from the number of units sold in (UnitSales) multiplied
by the unit price.Max(UnitSales*UnitPrice)150, because this is the highest
value of the result of calculating all possible values of (UnitSales)*
(UnitPrice).Max(UnitSales, 2)9, which is the second highest value.Max
(TOTAL UnitSales)10, because the TOTAL qualifier means the highest possible
value is found, disregarding the chart dimensions. For a chart with Customer
as dimension, the TOTAL qualifier will ensure the maximum value across the
full dataset is returned, instead of the maximum UnitSales for each
customer.Make the selection Customer B.Max({1} TOTAL UnitSales)10,
independent of the selection made, because the Set Analysis expression {1}
defines the set of records to be evaluated as ALL, no matter what selection
is made.Data used in examples:ProductData:LOAD * inline
[Customer|Product|UnitSales|UnitPriceAstrida|AA|4|16Astrida|AA|10|15Astrida|B
B|9|9Betacab|BB|5|10Betacab|CC|2|20Betacab|DD||25Canutility|AA|8|15Canutility
|CC||19] (delimiter is '|'); FirstSortedValue RangeMax ([{SetExpression}]
[DISTINCT] [TOTAL [<fld ,fld>]] expr [,rank])

```

Min

Min() finds the lowest value of the aggregated data. By specifying a **rank n**, the nth lowest value can be found.

```

Min - chart function([{SetExpression}] [DISTINCT] [TOTAL [<fld ,fld>]] expr
[,rank])

```

Mode

Mode() finds the most commonly-occurring value, the mode value, in the aggregated data. The **Mode()** function can process text values as well as numeric values.

```

Mode - chart function ({[SetExpression]} [TOTAL [<fld ,fld>]]} expr)

```

Only

Only() returns a value if there is one and only one possible result from the aggregated data. For example, searching for the only product where the unit price =9 will return NULL if more than one product has a unit price of 9.

```
Only - chart function([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

Sum

Sum() calculates the total of the values given by the expression or field across the aggregated data.

```
Sum - chart function([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

FirstSortedValue

FirstSortedValue() returns the value from the expression specified in **value** that corresponds to the result of sorting the **sort_weight** argument, for example, the name of the product with the lowest unit price. The nth value in the sort order, can be specified in **rank**. If more than one resulting value shares the same **sort_weight** for the specified **rank**, the function returns NULL. The sorted values are iterated over a number of records, as defined by a **group by** clause, or aggregated across the full data set if no **group by** clause is defined.

Syntax:

```
FirstSortedValue ([ distinct ] value, sort-weight [, rank ])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
value Expression	The function finds the value of the expression value that corresponds to the result of sorting sort_weight .
sort-weight Expression	The expression containing the data to be sorted. The first (lowest) value of sort_weight is found, from which the corresponding value of the value expression is determined. If you place a minus sign in front of sort_weight , the function returns the last (highest) sorted value instead.
rank Expression	By stating a rank "n" larger than 1, you get the nth sorted value.
distinct	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

5 Script and chart functions

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting examples

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber UnitSales CustomerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD 12 25 2 Canutility AA 3 8 3 Canutility CC 13 19 3 Divadip AA 9 16 4 Divadip AA 10 16 4 Divadip DD 11 10 4] (delimiter is ' '); FirstSortedValue: LOAD Customer,FirstSortedValue(Product, UnitSales) as MyProductwithSmallestorderByCustomer Resident Temp Group By Customer;</pre>	<p>Customer MyProductwithSmallestorderByCustomer Astrida AA Betacab AA Canutility AA Divadip DD</p> <p>The function sorts UnitSales from smallest to largest, looking for the value of Customer with the smallest value of UnitSales, the smallest order.</p> <p>Because CC corresponds to the smallest order (value of UnitSales=2) for customer Astrida. AA corresponds to the smallest order (4) for customer Betacab, AA corresponds to the smallest order (8) for customer Canutility, and DD corresponds to the smallest order (10) for customer Divadip..</p>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,FirstSortedValue(Product, -UnitSales) as MyProductwithLargestorderByCustomer Resident Temp Group By Customer;</pre>	<p>Customer MyProductwithLargestorderByCustomer Astrida AA Betacab DD Canutility CC Divadip -</p> <p>A minus sign precedes the sort_weight argument, so the function sorts the largest first.</p> <p>Because AA corresponds to the largest order (value of UnitSales:18) for customer Astrida, DD corresponds to the largest order (12) for customer Betacab, and CC corresponds to the largest order (13) for customer Canutility. There are two identical values for the largest order (16) for customer Divadip, therefore this produces a null result.</p>

Example	Result
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer,FirstSortedValue(distinct Product, - Unitsales) as MyProductwithSmallestOrderByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer MyProductwithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip AA</pre> <p>This is the same as the previous example, except the distinct qualifier is used. This causes the duplicate result for Divadip to be disregarded, allowing a non-null value to be returned.</p>

FirstSortedValue - chart function

FirstSortedValue() returns the value from the expression specified in **value** that corresponds to the result of sorting the **sort_weight** argument, for example, the name of the product with the lowest unit price. The nth value in the sort order, can be specified in **rank**. If more than one resulting value shares the same **sort_weight** for the specified **rank**, the function returns NULL.

Syntax:

```
FirstSortedValue([ {SetExpression} ] [DISTINCT] [TOTAL [<fld {,fld}>]]] value,
sort_weight [,rank])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
value	Output field. The function finds the value of the expression value that corresponds to the result of sorting sort_weight .
sort_weight	Input field. The expression containing the data to be sorted. The first (lowest) value of sort_weight is found, from which the corresponding value of the value expression is determined. If you place a minus sign in front of sort_weight , the function returns the last (highest) sorted value instead.
rank	By stating a rank "n" larger than 1, you get the nth sorted value.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples and results

Example	Result
<code>firstsortedvalue (Product, UnitPrice)</code>	BB, which is the Product with the lowest unitPrice(9).
<code>firstsortedvalue (Product, UnitPrice, 2)</code>	BB, which is the Product with the second-lowest unitPrice(10).
<code>firstsortedvalue (Customer, -UnitPrice, 2)</code>	Betacab, which is the Customer with the Product that has second-highest unitPrice(20).
<code>firstsortedvalue (Customer, UnitPrice, 3)</code>	NULL, because there are two values of Customer (Astrida and Canutility) with the same rank (third-lowest) unitPrice(15). Use the distinct qualifier to make sure unexpected null results do not occur.
<code>firstsortedvalue (Customer, -UnitPrice*UnitSales, 2)</code>	Canutility, which is the customer with the second-highest sales order value unitPrice multiplied by unitsales (120).

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|UnitsSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|1|25
Canutility|AA|8|15
Canutility|CC|19
] (delimiter is '|');
```

Max

Max() finds the highest numeric value of the aggregated data in the expression, as defined by a **group by** clause. By specifying a **rank** n, the nth highest value can be found.

Syntax:

```
Max ( expr [, rank])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr Expression	The expression or field containing the data to be measured.
rank Expression	The default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
```

```
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');

Max:
LOAD Customer, Max(UnitSales) as MyMax Resident Temp Group By Customer;
```

Resulting table

Customer	MyMax
Astrida	18
Betacab	5
Canutility	8

Example:

Given that the **Temp** table is loaded as in the previous example:

```
LOAD Customer, Max(UnitSales,2) as MyMaxRank2 Resident Temp Group By Customer;
Resulting table
```

Customer	MyMaxRank2
Astrida	10
Betacab	4
Canutility	-

Max - chart function

Max() finds the highest value of the aggregated data. By specifying a **rank n**, the nth highest value can be found.



You might also want to look at **FirstSortedValue** and **rangemax**, which have similar functionality to the **Max** function.

Syntax:

```
Max([{SetExpression}] [TOTAL [<fld ,fld>]]) expr [,rank])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Argument	Description
rank	The default value of rank is 1, which corresponds to the highest value. By specifying rank as 2, the second highest value is returned. If rank is 3, the third highest value is returned, and so on.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples and results

Examples	Results
Max(UnitSales)	10, because this is the highest value in unitsales.

Examples	Results
The value of an order is calculated from the number of units sold in <code>(unitsales)</code> multiplied by the unit price. <code>Max (UnitSales*UnitPrice)</code>	150, because this is the highest value of the result of calculating all possible values of <code>(unitsales)*(unitPrice)</code> .
<code>Max(Unitsales, 2)</code>	9, which is the second highest value.
<code>Max(TOTAL Unitsales)</code>	10, because the TOTAL qualifier means the highest possible value is found, disregarding the chart dimensions. For a chart with Customer as dimension, the TOTAL qualifier will ensure the maximum value across the full dataset is returned, instead of the maximum UnitSales for each customer.
Make the selection Customer B. <code>Max({1} TOTAL Unitsales)</code>	10, independent of the selection made, because the Set Analysis expression <code>{1}</code> defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:  
LOAD * inline [  
Customer|Product|unitsales|unitPrice  
Astrida|AA|4|16  
Astrida|AA|10|15  
Astrida|BB|9|9  
Betacab|BB|5|10  
Betacab|CC|2|20  
Betacab|DD|1|25  
Canutility|AA|8|15  
Canutility|CC|1|19  
] (delimiter is '|');
```

See also:

- [FirstSortedValue - chart function \(page 321\)](#)
- [RangeMax \(page 1304\)](#)

Min

Min() returns the lowest numeric value of the aggregated data in the expression, as defined by a **group by** clause. By specifying a **rank n**, the nth lowest value can be found.

Syntax:

```
Min ( expr [, rank])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr Expression	The expression or field containing the data to be measured.
rank Expression	The default value of rank is 1, which corresponds to the lowest value. By specifying rank as 2, the second lowest value is returned. If rank is 3, the third lowest value is returned, and so on.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitsSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Min:
LOAD Customer, Min(UnitsSales) as MyMin Resident Temp Group By Customer;
```

Resulting table

Customer	MyMin
Astrida	2
Betacab	4
Canutility	8

Example:

Given that the **Temp** table is loaded as in the previous example:

```
LOAD Customer, Min(UnitSales,2) as MyMinRank2 Resident Temp Group By Customer;
```

Resulting table

Customer	MyMinRank2
Astrida	9
Betacab	5
Canutility	-

Min - chart function

Min() finds the lowest value of the aggregated data. By specifying a **rank** n, the nth lowest value can be found.



You might also want to look at **FirstSortedValue** and **rangemin**, which have similar functionality to the **Min** function.

Syntax:

```
Min({ [SetExpression] [TOTAL [<fld {,fld}>]] } expr [,rank])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
rank	The default value of rank is 1, which corresponds to the lowest value. By specifying rank as 2, the second lowest value is returned. If rank is 3, the third lowest value is returned, and so on.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19



The `Min()` function must return a non-NULL value from the array of values given by the expression, if there is one. So in the examples, because there are NULL values in the data, the function returns the first non-NULL value evaluated from the expression.

Examples and results

Examples	Results
<code>Min(UnitSales)</code>	2, because this is the lowest non-NULL value in <code>unitsales</code> .
The value of an order is calculated from the number of units sold in <code>(unitsales)</code> multiplied by the unit price.	40, because this is the lowest non-NULL value result of calculating all possible values of <code>(unitsales)*(unitPrice)</code> .
<code>Min(UnitSales, 2)</code>	4, which is the second lowest value (after the NULL values).
<code>Min(TOTAL UnitSales)</code>	2, because the <code>TOTAL</code> qualifier means the lowest possible value is found, disregarding the chart dimensions. For a chart with <code>Customer</code> as dimension, the <code>TOTAL</code> qualifier will ensure the minimum value across the full dataset is returned, instead of the minimum <code>UnitSales</code> for each customer.
Make the selection Customer B. <code>Min({1} TOTAL UnitSales)</code>	2, which is independent of the selection of Customer B. The Set Analysis expression <code>{1}</code> defines the set of records to be evaluated as <code>ALL</code> , no matter what selection is made.

Data used in examples:

```
ProductData:  
LOAD * inline [  
Customer|Product|unitsSales|unitPrice  
Astrida|AA|4|16  
Astrida|AA|10|15  
Astrida|BB|9|9  
Betacab|BB|5|10  
Betacab|CC|2|20  
Betacab|DD||25  
Canutility|AA|8|15  
Canutility|CC||19  
] (delimiter is '|');
```

See also:

- [FirstSortedValue - chart function \(page 321\)](#)
- [RangeMin \(page 1307\)](#)

Mode

Mode() returns the most commonly-occurring value, the mode value, of the aggregated data in the expression, as defined by a **group by** clause. The **Mode()** function can return numeric values as well as text values.

Syntax:

```
Mode ( expr )
```

Return data type: dual

Arguments

Argument	Description
expr Expression	The expression or field containing the data to be measured.

Limitations:

If more than one value is equally commonly occurring, NULL is returned.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting examples

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber unitsales customerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility DD 3 8 Canutility CC] (delimiter is ' '); Mode: LOAD Customer, Mode(Product) as MyMostOftenSoldProduct Resident Temp Group By Customer;</pre>	<p>MyMostOftenSoldProduct AA because AA is the only product sold more than once.</p>

Mode - chart function

Mode() finds the most commonly-occurring value, the mode value, in the aggregated data. The **Mode()** function can process text values as well as numeric values.

Syntax:

```
Mode({ [SetExpression] [TOTAL [<fld {,fld}>]] } expr)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples and results

Examples	Results
Mode(UnitPrice) Make the selection Customer A.	15, because this is the most commonly-occurring value in UnitSales. Returns NULL (-). No single value occurs more often than another.
Mode(Product) Make the selection Customer A	AA, because this is the most commonly occurring value in Product. Returns NULL (-). No single value occurs more often than another.
Mode (TOTAL UnitPrice)	15, because the TOTAL qualifier means the most commonly occurring value is still 15, even disregarding the chart dimensions.
Make the selection Customer B. Mode({1} TOTAL UnitPrice)	15, independent of the selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:  
LOAD * inline [  
Customer|Product|unitsales|unitprice  
Astrida|AA|4|16  
Astrida|AA|10|15  
Astrida|BB|9|9  
Betacab|BB|5|10  
Betacab|CC|2|20  
Betacab|DD||25  
Canutility|AA|8|15  
Canutility|CC||19
```

```
] (delimiter is '|');
```

See also:

- [Avg - chart function \(page 390\)](#)
- [Median - chart function \(page 427\)](#)

Only

Only() returns a value if there is one and only one possible result from the aggregated data. If records contain only one value then that value is returned, otherwise NULL is returned. Use the **group by** clause to evaluate over multiple records. The **Only()** function can return numeric and text values.

Syntax:

```
Only ( expr )
```

Return data type: dual

Arguments

Argument	Description
expr Expression	The expression or field containing the data to be measured.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

```
Temp:  
LOAD * inline [  
Customer|Product|orderNumber|unitsales|customerID  
Astrida|AA|1|10|1  
Astrida|AA|7|18|1  
Astrida|BB|4|9|1  
Astrida|CC|6|2|1  
Betocab|AA|5|4|2  
Betocab|BB|2|5|2  
Betocab|DD  
Canutility|DD|3|8  
Canutility|CC  
] (delimiter is '|');  
only:  
LOAD Customer, Only(CustomerID) as MyUniqIDCheck Resident Temp Group By Customer;
```

Resulting table

Customer	MyUniqIDCheck
Astrida	<p>1</p> <p>because only customer Astrida has complete records that include CustomerID.</p>

Only - chart function

Only() returns a value if there is one and only one possible result from the aggregated data. For example, searching for the only product where the unit price =9 will return NULL if more than one product has a unit price of 9.

Syntax:

```
Only([{SetExpression}] [TOTAL [<fld {,fld}>]] expr)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>By using TOTAL [<fld {,fld}>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.</p>



Use Only() when you want a NULL result if there are multiple possible values in the sample data.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15

Customer	Product	UnitSales	UnitPrice
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples and results

Examples	Results
only({<UnitPrice={9}>} Product)	BB, because this is the only Product that has a UnitPrice of '9'.
only({<Product={DD}>} Customer)	Betacab, because it is the only customer selling a Product called 'DD'.
only({<UnitPrice={20}>} UnitSales)	The number of UnitSales where UnitPrice is 20 is 2, because there is only one value of UnitSales where the UnitPrice =20.
only({<UnitPrice={15}>} UnitSales)	NULL, because there are two values of UnitSales where the UnitPrice =15.

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|unitsales|unitprice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

Sum

Sum() calculates the total of the values aggregated in the expression, as defined by a **group by** clause.

Syntax:

```
sum ( [ distinct] expr)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.
expr Expression	The expression or field containing the data to be measured.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Temp:

```
LOAD * inline [
Customer|Product|orderNumber|unitsales|customerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Sum:
LOAD Customer, Sum(unitsales) as MySum Resident Temp Group By Customer;
```

Resulting table

Customer	MySum
Astrida	39
Betacab	9
Canutility	8

Sum - chart function

Sum() calculates the total of the values given by the expression or field across the aggregated data.

Syntax:

```
Sum([{SetExpression}]) [DISTINCT] [TOTAL [<fld {,fld}>]] expr])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.  <i>Although the DISTINCT qualifier is supported, use it only with extreme caution because it may mislead the reader into thinking a total value is shown when some data has been omitted.</i>
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data			
Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

Examples and results

Examples	Results
Sum(unitsales)	38. The total of the values in unitsales.
Sum(Unitsales*UnitPrice)	505. The total of unitPrice multiplied by unitsales aggregated.
Sum (TOTAL Unitsales*unitPrice)	505 for all rows in the table as well as the total, because the TOTAL qualifier means the sum is still 505, disregarding the chart dimensions.
Make the selection customer B. Sum({1} TOTAL Unitsales*unitPrice)	505, independent of the selection made, because the Set Analysis expression {1} defines the set of records to be evaluated as ALL, no matter what selection is made.

Data used in examples:

```
ProductData:  
LOAD * inline [  
Customer|Product|unitsales|unitPrice  
Astrida|AA|4|16  
Astrida|AA|10|15  
Astrida|BB|9|9  
Betocab|BB|5|10  
Betocab|CC|2|20  
Betocab|DD||25  
Canutility|AA|8|15  
Canutility|CC||19  
] (delimiter is '|');
```

Counter aggregation functions

Counter aggregation functions return various types of counts of an expression over a number of records in a data load script, or a number of values in a chart dimension.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Counter aggregation functions in the data load script

Count

Count() returns the number of values aggregated in expression, as defined by a **group by** clause.

```
Count ([distinct] expression | * )
```

MissingCount

MissingCount() returns the number of missing values aggregated in the expression, as defined by a **group by** clause.

```
MissingCount ([ distinct ] expression)
```

NullCount

NullCount() returns the number of NULL values aggregated in the expression, as defined by a **group by** clause.

```
NullCount ([ distinct ] expression)
```

NumericCount

NumericCount() returns the number of numeric values found in the expression, as defined by a **group by** clause.

```
NumericCount ([ distinct ] expression)
```

TextCount

TextCount() returns the number of field values that are non-numeric aggregated in the expression, as defined by a **group by** clause.

```
TextCount ([ distinct ] expression)
```

Counter aggregation functions in chart expressions

The following counter aggregation functions can be used in charts:

Count

Count() is used to aggregate the number of values, text and numeric, in each chart dimension.

```
Count - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

MissingCount

MissingCount() is used to aggregate the number of missing values in each chart dimension. Missing values are all non-numeric values.

```
MissingCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

NullCount

NullCount() is used to aggregate the number of NULL values in each chart dimension.

```
NullCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

NumericCount

NumericCount() aggregates the number of numeric values in each chart dimension.

```
NumericCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

TextCount

TextCount() is used to aggregate the number of field values that are non-numeric in each chart dimension.

```
TextCount - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

Count

Count() returns the number of values aggregated in expression, as defined by a **group by** clause.

Syntax:

```
Count( [distinct] expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting examples

Example	Result
<pre>Temp: LOAD * inline [Customer Product OrderNumber Unitsales UnitPrice Astrida AA 1 4 16 Astrida AA 7 10 15 Astrida BB 4 9 9 Betocab CC 6 5 10 Betocab AA 5 2 20 Betocab BB 1 25 25 Canutility AA 3 8 15 Canutility CC 19 Divadip CC 2 4 16 Divadip DD 3 1 25] (delimiter is ' '); Count1: LOAD Customer,Count(OrderNumber) as OrdersByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer OrdersByCustomer Astrida 3 Betocab 3 Canutility 2 Divadip 2 As long as the dimension Customer is included in the table on the sheet, otherwise the result for OrdersByCustomer is 3, 2.</pre>

Example	Result
Given that the Temp table is loaded as in the previous example: LOAD Count(OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 10
Given that the Temp table is loaded as in the first example: LOAD Count(DISTINCT OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 8 Because there are two values of OrderNumber with the same value, 1, and one null value.

Count - chart function

Count() is used to aggregate the number of values, text and numeric, in each chart dimension.

Syntax:

```
Count({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data				
Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD	1	25	25
Canutility	AA	3	8	15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

The following examples assume that all customers are selected, except where stated.

Examples and results

Example	Result
Count(OrderNumber)	10, because there are 10 fields that could have a value for OrderNumber, and all records, even empty ones, are counted.
	 "0" counts as a value and not an empty cell. However, if a measure aggregates to 0 for a dimension, that dimension will not be included in charts.
Count(Customer)	10, because Count evaluates the number of occurrences in all fields.
Count(DISTINCT [Customer])	4, because using the Distinct qualifier, Count only evaluates unique occurrences.
Given that customer Canutility is selected Count (OrderNumber)/Count({1} TOTAL OrderNumber)	0.2, because the expression returns the number of orders from the selected customer as a percentage of orders from all customers. In this case 2 / 10.

Example	Result
Given that customers Astrida and Canutility are selected Count(TOTAL <Product> OrderNumber)	5, because that is the number of orders placed on products for the selected customers only and empty cells are counted.

Data used in examples:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|Unitsales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|1|25| 25
Canutility|AA|3|8|15
Canutility|CC|||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

MissingCount

MissingCount() returns the number of missing values aggregated in the expression, as defined by a **group by** clause.

Syntax:

```
MissingCount ( [ distinct ] expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr Expression	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting examples

Example	Result
<pre> Temp: LOAD * inline [Customer Product OrderNumber Unitsales UnitPrice Astrida AA 1 4 16 Astrida AA 7 10 15 Astrida BB 4 9 9 Betacab CC 6 5 10 Betacab AA 5 2 20 Betacab BB 25 Canutility AA 15 Canutility CC 19 Divadip CC 2 4 16 Divadip DD 3 1 25] (delimiter is ' '); MissCount1: LOAD Customer,MissingCount(OrderNumber) as MissingOrdersByCustomer Resident Temp Group By Customer; Load MissingCount(OrderNumber) as TotalMissingCount Resident Temp; </pre>	<pre> Customer MissingOrdersByCustomer Astrida 0 Betacab 1 Canutility 2 Divadip 0 The second statement gives: TotalMissingCount 3 in a table with that dimension. </pre>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre> LOAD MissingCount(distinct OrderNumber) as TotalMissingCountDistinct Resident Temp; </pre>	<pre> TotalMissingCountDistinct 1 Because there is only oneOrderNumber one missing value. </pre>

MissingCount - chart function

MissingCount() is used to aggregate the number of missing values in each chart dimension. Missing values are all non-numeric values.

Syntax:

```
MissingCount({ [SetExpression] [DISTINCT] [TOTAL [<fld ,fld>]] } expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Argument	Description
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data				
Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

Examples and results

Example	Result
<code>MissingCount([OrderNumber])</code>	3 because 3 of the 10 OrderNumber fields are empty
<code>MissingCount([OrderNumber])/MissingCount({1} Total [OrderNumber])</code>	The expression returns the number of incomplete orders from the selected customer as a fraction of incomplete orders from all customers. There is a total of 3 missing values for OrderNumber for all customers. So, for each Customer that has a missing value for Product the result is 1/3.

Data used in example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|unitsSales|unitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betocab|CC|6|5|10
Betocab|AA|5|2|20
Betocab|BB||| 25
Canutility|AA|||15
Canutility|CC| |||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

NullCount

NullCount() returns the number of NULL values aggregated in the expression, as defined by a **group by** clause.

Syntax:

```
NullCount ( [ distinct ] expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr Expression	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting examples

Example	Result
<pre>Set NULLINTERPRET = NULL; Temp: LOAD * inline [Customer Product orderNumber unitsales customerID Astrida AA 1 10 1 Astrida AA 7 18 1 Astrida BB 4 9 1 Astrida CC 6 2 1 Betacab AA 5 4 2 Betacab BB 2 5 2 Betacab DD Canutility AA 3 8 Canutility CC NULL] (delimiter is ' '); Set NULLINTERPRET=; NullCount1: LOAD Customer,NullCount(orderNumber) as NullordersByCustomer Resident Temp Group By Customer; LOAD NullCount(orderNumber) as TotalNullCount Resident Temp;</pre>	<pre>Customer NullordersByCustomer Astrida 0 Betacab 0 Canutility 1</pre> <p>The second statement gives:</p> <pre>TotalNullCount 1 in a table with that dimension, because only one record contains a null value.</pre>

NullCount - chart function

NullCount() is used to aggregate the number of NULL values in each chart dimension.

Syntax:

```
NullCount({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Argument	Description
set_expression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Examples and results

Example	Result
NullCount ([OrderNumber])	1 because we have introduced a null value using NullInterpret in the inline LOAD statement.

Data used in example:

```
Set NULLINTERPRET = NULL;
Temp:
LOAD * inline [
Customer|Product|OrderNumber|unitsales|customerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD|||
Canutility|AA|3|8|
Canutility|CC|NULL||
] (delimiter is '|');
Set NULLINTERPRET=;
```

NumericCount

NumericCount() returns the number of numeric values found in the expression, as defined by a **group by** clause.

Syntax:

```
NumericCount ( [ distinct ] expr)
```

Return data type: integer

Arguments:

Arguments	
Argument	Description
expr Expression	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Scripting example	
Example	Result
<pre>LOAD NumericCount(OrderNumber) as TotalNumericCount Resident Temp;</pre>	<p>The second statement gives: TotalNumericCount 7 in a table with that dimension.</p>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD NumericCount(distinct OrderNumber) as TotalNumericCountDistinct Resident Temp;</pre>	<p>TotalNumericCountDistinct 6 Because there is one OrderNumber that duplicates another, so the result is 6 that are not duplicates..</p>

Example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|Unitsales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betocab|CC|6|5|10
Betocab|AA|5|2|20
Betocab|BB||| 25
Canutility|AA||||15
Canutility|CC| |||19
Divadip|CC|2|4|16
Divadip|DD|7|1|25
] (delimiter is '|');
NumCount1:
LOAD Customer,NumericCount(OrderNumber) as NumericCountByCustomer Resident Temp Group By
Customer;
```

Resulting table

Customer	NumericCountByCustomer
Astrida	3
Betacab	2
Canutility	0
Divadip	2

NumericCount - chart function

NumericCount() aggregates the number of numeric values in each chart dimension.

Syntax:

```
NumericCount({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
set_expression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1

Customer	Product	OrderNumber	UnitSales	Unit Price
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

The following examples assume that all customers are selected, except where stated.

Examples and results

Example	Result
NumericCount ([OrderNumber])	7 because three of the 10 fields in OrderNumber are empty.  <i>"0" counts as a value and not an empty cell. However, if a measure aggregates to 0 for a dimension, that dimension will not be included in charts.</i>
NumericCount ([Product])	0 because all product names are in text. Typically you could use this to check that no text fields have been given numeric content.
NumericCount (DISTINCT [OrderNumber])/Count (DISTINCT [OrderNumber])	Counts all the number of distinct numeric order numbers and divides it by the number of order numbers numeric and non-numeric. This will be 1 if all field values are numeric. Typically you could use this to check that all field values are numeric. In the example, there are 7 distinct numeric values for OrderNumber of 8 distinct numeric and non-numeric, so the expression returns 0.875.

Data used in example:

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|unitsales|unitprice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB||| 25
Canutility|AA|||15
Canutility|CC| |||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
```

```
] (delimiter is '|');
```

TextCount

TextCount() returns the number of field values that are non-numeric aggregated in the expression, as defined by a **group by** clause.

Syntax:

```
TextCount ( [ distinct ] expr)
```

Return data type: integer

Arguments:

Arguments	
Argument	Description
expr Expression	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example:

```
Temp:  
LOAD * inline [  
Customer|Product|OrderNumber|UnitsSales|UnitPrice  
Astrida|AA|1|4|16  
Astrida|AA|7|10|15  
Astrida|BB|4|9|9  
Betacab|CC|6|5|10  
Betacab|AA|5|2|20  
Betacab|BB||| 25  
Canutility|AA|||15  
Canutility|CC| |||19  
Divadip|CC|2|4|16  
Divadip|DD|3|1|25  
] (delimiter is '|');  
TextCount1:  
LOAD Customer,TextCount(Product) as ProductTextCount Resident Temp Group By Customer;
```

Resulting table

Customer	ProductTextCount
Astrida	3
Betacab	3
Canutility	2
Divadip	2

Example:

```
LOAD Customer ,TextCount(OrderNumber) as OrderNumberTextCount Resident Temp Group By Customer;
```

Resulting table

Customer	OrderNumberTextCount
Astrida	0
Betacab	1
Canutility	2
Divadip	0

TextCount - chart function

TextCount() is used to aggregate the number of field values that are non-numeric in each chart dimension.

Syntax:

```
TextCount({ [SetExpression] [DISTINCT] [TOTAL [<fld ,fld>]] } expr)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Data				
Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

Examples and results

Example	Result
TextCount ([Product])	10 because all of the 10 fields in Product are text.  <i>"0" counts as a value and not an empty cell. However, if a measure aggregates to 0 for a dimension, that dimension will not be included in charts. Empty cells are evaluated as being non text and are not counted by TextCount.</i>
TextCount ([OrderNumber])	3, because empty cells are counted. Typically, you would use this to check that no numeric fields have been given text values or are non-zero.

Example	Result
TextCount (DISTINCT [Product])/Count ([Product])	Counts all the number of distinct text values of Product (4), and divides it by the total number of values in Product (10). The result is 0.4.

Data used in example:

```
Temp:  
LOAD * inline [  
Customer|Product|OrderNumber|UnitsSales|UnitPrice  
Astrida|AA|1|4|16  
Astrida|AA|7|1|15  
Astrida|BB|4|9|9  
Betacab|CC|6|5|10  
Betacab|AA|5|2|20  
Betacab|BB|||| 25  
Canutility|AA||||15  
Canutility|CC||||19  
Divadip|CC|2|4|16  
Divadip|DD|3|1|25  
] (delimiter is '|');
```

Financial aggregation functions

This section describes aggregation functions for financial operations regarding payments and cash flow.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Financial aggregation functions in the data load script

IRR

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression iterated over a number of records as defined by a group by clause.

```
IRR (expression)
```

XIRR

XIRR() returns the aggregated internal rate of return (yearly) for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in **pmt** and **date** iterated over a number of records as defined by a group by clause. All payments are discounted based on a 365-day year.

```
XIRR (valueexpression, dateexpression )
```

NPV

The **NPV()** script function takes a discount rate and multiple values ordered by period. Inflows (incomes) are positive, and outflows (future payments) are assumed to be negative values for these calculations. These occur at the end of each period.

```
NPV (rate, expression)
```

XNPV

XNPV() returns the aggregated net present value for a schedule of cashflows (not necessarily periodic) represented by paired numbers in **pmt** and **date**. All payments are discounted based on a 365-day year.

```
XNPV (rate, valueexpression, dateexpression)
```

Financial aggregation functions in chart expressions

These financial aggregation functions can be used in charts.

IRR

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression given by **value** iterated over the chart dimensions.

```
IRR - chart function[TOTAL [<fld {,fld}>]] value)
```

NPV

NPV() returns the aggregated net present value of an investment based on a **discount_rate** per period and a series of future payments (negative values) and incomes (positive values,) represented by the numbers in **value**, iterated over the chart dimensions. The payments and incomes are assumed to occur at the end of each period.

```
NPV - chart function([TOTAL [<fld {,fld}>]] discount_rate, value)
```

XIRR

XIRR() returns the aggregated internal rate of return (yearly) for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart dimensions. All payments are discounted based on a 365-day year.

```
XIRR - chart function([TOTAL [<fld {,fld}>]] pmt, date)
```

XNPV

XNPV() returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date**, iterated over the chart dimensions. All payments are discounted based on a 365-day year.

```
XNPV - chart function([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

IRR

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression iterated over a number of records as defined by a group by clause.

These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods. The function needs at least one positive and one negative value to calculate.

This function uses a simplified version of the Newton method for calculating the internal rate of return (IRR).

Syntax:

```
IRR(value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The expression or field containing the data to be measured.

Limitations:

Text values, NULL values and missing values are disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Examples and results:

Examples and results		
Example	Year	IRR2013
<pre>Cashflow: LOAD 2013 as Year, * inline [Date Discount Payments 2013-01-01 0.1 -10000 2013-03-01 0.1 3000 2013-10-30 0.1 4200 2014-02-01 0.2 6800] (delimiter is ' '); Cashflow1: LOAD Year,IRR(Payments) as IRR2013 Resident Cashflow Group By Year;</pre>	2013	0.1634

IRR - chart function

IRR() returns the aggregated internal rate of return for a series of cash flows represented by the numbers in the expression given by **value** iterated over the chart dimensions.

These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods. The function needs at least one positive and one negative value to calculate.

This function uses a simplified version of the Newton method for calculating the internal rate of return (IRR).

Syntax:

```
IRR( [TOTAL [<fld {,fld}>]] value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The expression or field containing the data to be measured.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values are disregarded.

Examples and results:

Examples and results	
Example	Result
IRR (Payments)	0.1634 The payments are assumed to be periodic in nature, for example monthly.  <i>The Date field is used in the XIRR example where payments can be non-periodical as long as you provide the dates on which payments were made.</i>

Data used in examples:

Cashflow:

```
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

See also:

- [XIRR - chart function \(page 370\)](#)
- [Aggr - chart function \(page 525\)](#)

NPV

The **NPV()** script function takes a discount rate and multiple values ordered by period. Inflows (incomes) are positive, and outflows (future payments) are assumed to be negative values for these calculations. These occur at the end of each period.

Net Present Value, or NPV, is used to calculate the current total value of a future stream of cash flows. To calculate NPV, we need to estimate future cash flows for each period and determine the correct discount rate. The **NPV()** script function takes a discount rate and multiple values ordered by period. Inflows (incomes) are positive, and outflows (future payments) are assumed to be negative values for these calculations. These occur at the end of each period.

Syntax:

```
NPV(discount_rate, value)
```

Return data type: numeric. By default, the result will be formatted as currency.

The formula to calculate net present value is:

$$NPV = \sum_{t=1}^n \frac{R_t}{(1+i)^t}$$

where:

- R_t = Net cash inflow-outflows during a single period t
- i = Discount rate or return that could be earned in alternative investments
- t = Number of timer periods

Arguments

Argument	Description
discount_rate	discount_rate is the percentage rate of discount applied. A value of 0.1 would indicate a 10% discount rate.
value	This field holds values for multiple periods ordered by period. The first value is assumed to be the cashflow at the end of period 1, and so on.

Limitations:

The **NPV()** function has the following limitations:

- Text values, NULL values and missing values are disregarded.
- Cashflow values must be in order of ascending period.

When to use it

`NPV()` is a financial function used to check project profitability and to derive other measures. This function is useful when cashflows are available as raw data.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Single payment (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of one project and its cashflow for one period, which is loaded into a table named `cashFlow`.
- A resident load from the `cashFlow` table, which is used to calculate the `NPV` field for the project in a table named `NPV`.
- A hard-coded discount rate of 10% , which is used in the `NPV` calculation.
- A `Group By` statement, which is used to group all the payments for the project.

Load script

```
CashFlow:  
Load  
*  
Inline  
[  
PrjId,PeriodId,values  
1,1,1000  
];  
  
NPV:  
Load
```

```
PrjId,  
NPV(0.1,values) as NPV //Discount Rate of 10%  
Resident CashFlow  
Group By PrjId;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- NPV

Results table

PrjId	NPV
1	\$909.09

For a single payment of \$1000 to be received at the end of one period, at a discount rate of 10% per period, the NPV is equal to \$1000 divided by $(1 + \text{discount rate})$. The effective NPV is equal to \$909.09

Example 2 – Multiple payments (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of one project and its cashflow for multiple periods, which is loaded into a table named CashFlow.
- A resident load from the cashFlow table, which is used to calculate the NPV field for the project in a table named NPV.
- A hard-coded discount rate of 10% (0.1) is used in the NPV calculation.
- A Group By statement, which is used to group all the payments for the project.

Load script

```
CashFlow:  
Load  
*  
Inline  
[  
PrjId,PeriodId,values  
1,1,1000  
1,2,1000  
];
```

```
NPV:  
Load
```

```
PrjId,  
NPV(0.1,values) as NPV //Discount Rate of 10%  
Resident CashFlow  
Group By PrjId;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- NPV

Results table

PrjId	NPV
1	\$1735.54

For payments of \$1000 to be received at the end of two periods, at a discount rate of 10% per period, the effective NPV is equal to \$1735.54.

Example 3 – Multiple payments (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Discount rates for two projects, which is loaded into a table named Project.
- Cashflows for multiple periods for each project by project ID and period ID. This period ID could be used to order the records in case the data is not ordered.
- The combination of NoConcatenate, Resident loads, and the Left Join function to create a temporary table, tmpNPV. The table combines the records of Project and CashFlow tables into one flat table. This table will have discount rates repeated for each period.
- A resident load from the tmpNPV table, which is used to calculate the NPV field for each project in a table named NPV.
- The single value discount rate associated to each project. This is retrieved using the only() function and is used in the NPV calculation for each project.
- A Group By statement, which is used to group all the payments for each project by project ID.

To avoid any synthetic or redundant data being loaded into the data model, the tmpNPV table is dropped at the end of the script.

Load script

```
Project:  
Load * inline [
```

```

PrjId,Discount_Rate
1,0.1
2,0.15
];

CashFlow:
Load
*
Inline
[
PrjId,PeriodId,values
1,1,1000
1,2,1000
1,3,1000
2,1,500
2,2,500
2,3,1000
2,4,1000
];
tmpNPV:
NoConcatenate Load *
Resident Project;
Left Join
Load *
Resident CashFlow;

NPV:
Load
    PrjId,
    NPV(Only(Discount_Rate),values) as NPV //Discount Rate will be 10% for Project 1 and 15% for
Project 2
Resident tmpNPV
Group By PrjId;

Drop table tmpNPV;

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- NPV

Results table

PrjId	NPV
1	\$2486.85
2	\$2042.12

Project ID 1 expects for payments of \$1000 to be received at the end of three periods, at a discount rate of 10% per period. Therefore, the effective NPV is equal to \$2486.85.

Project ID 2 expects two payments of \$500 and two further payments of \$1000 across four periods at a discount rate of 15%. Therefore, the effective NPV is equal to \$2042.12.

Example 4 – Project profitability example (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Discount rates and initial investments (period 0) for two projects, loaded into a table named `Project`.
- Cashflows for multiple periods for each project by project ID and period ID. This period ID could be used to order the records in case the data is not ordered.
- The combination of `NoConcatenate`, Resident loads, and the `Left Join` function to create a temporary table, `tmpNPV`. The table combines the records of `Project` and `CashFlow` tables into one flat table. This table will have discount rates repeated for each period.
- The single value discount rate associated to each project, which is retrieved using the `only()` function and is used in the NPV calculation for each project.
- A resident load from the `tmpNPV` table is used to calculate the NPV field for each project in a table named `NPV`.
- An additional field that divides the NPV by the initial investment of each project is created to calculate the project profitability index.
- A group by statement, grouping by project ID, is used to group all the payments for each project.

To avoid any synthetic or redundant data being loaded into the data model, the `tmpNPV` table is dropped at the end of the script.

Load script

```
Project:  
Load * inline [  
PrjId,Discount_Rate, Initial_Investment  
1,0.1,100000  
2,0.15,100000  
];  
  
CashFlow:  
Load  
*  
Inline  
[  
PrjId,PeriodId,values,  
1,1,35000  
1,2,35000  
1,3,35000  
2,1,30000  
2,2,40000
```

```

2,3,50000
2,4,60000
];

tmpNPV:
NoConcatenate Load *
Resident Project;
Left Join
Load *
Resident CashFlow;

NPV:
Load
    PrjId,
    NPV(Only(Discount_Rate),values) as NPV, //Discount Rate will be 10% for Project 1 and
15% for Project 2
    NPV(Only(Discount_Rate),values)/ Only(Initial_Investment) as Profitability_Index
Resident tmpNPV
Group By PrjId;

Drop table tmpNPV;

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- NPV

Create the following measure:

=only(Profitability_Index)

Results table

PrjId	NPV	=only(Profitability_Index)
1	\$87039.82	0.87
2	\$123513.71	1.24

Project ID 1 has an effective NPV of \$87039.82 and an initial investment of \$100000. Therefore, the profitability index is equal to 0.87. Because it is less than 1, the project is not profitable.

Project ID 2 has an effective NPV of \$123513.71 and an initial investment of \$100000. Therefore, the profitability index is equal to 1.24. Because it is greater than 1, the project is profitable.

NPV - chart function

NPV() returns the aggregated net present value of an investment based on a **discount_rate** per period and a series of future payments (negative values) and incomes (positive values,) represented by the numbers in **value**, iterated over the chart dimensions. The payments and incomes are assumed to occur at the end of each period.

Syntax:

```
NPV([TOTAL [<fld {.fld}>]] discount_rate, value)
```

Return data type: numeric By default, the result will be formatted as currency.

Arguments:

Arguments	
Argument	Description
discount_rate	discount_rate is the rate of discount over the length of the period. discount_rate is the percentage rate of discount applied.
value	The expression or field containing the data to be measured.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>By using TOTAL [<fld {.fld}>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.</p> <p>The TOTAL qualifier may be followed by a list of one or more field names within angle brackets. These field names should be a subset of the chart dimension variables. In this case, the calculation is made disregarding all chart dimension variables except those listed, that is, one value is returned for each combination of field values in the listed dimension fields. Also, fields that are not currently a dimension in a chart may be included in the list. This may be useful in the case of group dimensions, where the dimension fields are not fixed. Listing all of the variables in the group causes the function to work when the drill-down level changes.</p>

Limitations:

discount_rate and **value** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values are disregarded.

Examples and results:

Examples and results	
Example	Result
NPV(Discount, Payments)	-\$540.12

Data used in examples:

Cashflow:

```
LOAD 2013 as Year, * inline [  
Date|Discount|Payments  
2013-01-01|0.1|-10000  
2013-03-01|0.1|3000  
2013-10-30|0.1|4200  
2014-02-01|0.2|6800  
] (delimiter is '|');
```

See also:

- [XNPV - chart function \(page 380\)](#)
- [Aggr - chart function \(page 525\)](#)

XIRR

XIRR() returns the aggregated internal rate of return (yearly) for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in **pmt** and **date** iterated over a number of records as defined by a group by clause. All payments are discounted based on a 365-day year.

Qlik's XIRR functionality (**XIRR()** and **RangeXIRR()** functions) uses the following equation, solving for the **Rate** value, to determine the correct XIRR value:

`XNPV(Rate, pmt, date) = 0`

The equation is solved using a simplified version of the Newton method.

Syntax:

```
XIRR(pmt, date )
```

Return data type: numeric

Arguments

Argument	Description
pmt	Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date .
date	The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt .

When working with this function, the following limitations apply:

- Text values, NULL values and missing values in any or both pieces of a data-pair will result in the entire data-pair to be disregarded.
- This function requires at least one valid negative and at least one valid positive payment (with corresponding valid dates). If these payments are not provided, a NULL value is returned.

These topics might help you work with this function:

- *XNPV* (page 373): Use this function to calculate aggregated net present value for a schedule of cash flows.
- *RangeXIRR* (page 1325): **RangeXIRR()** is the equivalent range function for the **XIRR()** function.



Across different versions of Qlik Sense Client-Managed, there are variations in the underlying algorithm used by this function. For more information about recent updates to the algorithm, see support article [XIRR function Fix and Update](#).

Example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Transaction data for a series of cashflows.
- The use of the **XIRR()** function to compute internal yearly rate of return for these cashflows.

Load script

Cashflow:

```
LOAD 2013 as Year, * inline [  
Date|Payments  
2013-01-01|-10000  
2013-03-01|3000  
2013-10-30|4200  
2014-02-01|6800  
] (delimiter is '|');
```

Cashflow1:

```
LOAD Year,XIRR(Payments, Date) as XIRR2013 Resident Cashflow Group By Year;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- Year
- XIRR2013

Results table

Year	XIRR2013
2013	0.5385

Interpreting the XIRR return value

The XIRR functionality is usually used to analyze an investment, where there is an outgoing (negative) payment in the beginning, and then a series of smaller income (positive) payments later on. Here is a simplified example with only one negative and one positive payment:

```
Cashflow:  
LOAD * inline [  
Date|Payments  
2023-01-01|-100  
2024-01-01|110  
] (delimiter is '|');
```

We make an initial payment of 100 and get 110 back after exactly one year. This represents a rate of return of 10% per year. `XIRR(Payments, Date)` returns a value of 0.1.

The return value of the XIRR functionality can be positive or negative. In the case of an investment, a negative result indicates that the investment is a loss. The amount of gain or loss can be calculated simply by making a sum aggregation over the payments field.

In the example above, we are lending out our money for one year. The rate of return can be thought of as interest. It is also possible to use XIRR's functionality when you are on the other side of the transaction (for example, if you are the borrower instead of the lender).

Consider this example:

```
Cashflow:  
LOAD * inline [  
Date|Payments  
2023-01-01|100  
2024-01-01|-110  
] (delimiter is '|');
```

This is the same as the first example but inverted. Here, we are borrowing 100 for one year and we repay it with a 10% interest. In this example, the XIRR calculation returns 0.1 (10%), the same value as the first example.

Note that in the first example, we received a profit of 10, and in the second example, we experienced a loss of 10, but the return value of the XIRR functionality is positive for both these examples. This is because the XIRR functionality calculates the hidden interest in the transaction, regardless of which side you are on in the transaction.

Limitations with multiple solutions

Qlik's XIRR functionality is defined by the following equation, in which the `Rate` value is solved:

```
XNPV(Rate, pmt, date) = 0
```

It is sometimes possible for this equation to have more than one solution. This is known as the “multiple-IRR problem”, and is caused by a non-normal cash flow stream (also called an unconventional cash flow). The following load script shows an example of this:

```
Cashflow:  
LOAD * inline [  
Date|Payments
```

```
2021-01-01|-200  
2022-01-01|500  
2023-01-01|-250  
] (delimiter is '|');
```

In this example, there is one negative solution and one positive solution (`rate = -0.3` and `rate = 0.8`). **XIRR()** will return 0.8.

When Qlik's XIRR functionality searches for a solution, it starts at `Rate = 0` and increases the rate in steps until it finds a solution. If there is more than one positive solution, it will return the first one that it encounters. If it cannot find a positive solution, it will reset the `Rate` back to zero and start searching for a solution in the negative direction.

Note that a “normal” cash flow stream is guaranteed to have only one solution. “Normal” cash flow stream means that all payments with the same sign (positive or negative) are in a continuous group.

See also:

- ❑ [XNPV \(page 373\)](#)
- ❑ [RangeXIRR \(page 1325\)](#)
- [XIRR function Fix and Update](#)

XIRR - chart function

XIRR() returns the aggregated internal rate of return (yearly) for a schedule of cash flows (that is not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date** iterated over the chart dimensions. All payments are discounted based on a 365-day year.

Qlik's XIRR functionality (**XIRR()** and **RangeXIRR()** functions) uses the following equation, solving for the `Rate` value, to determine the correct XIRR value:

```
XNPV(Rate, pmt, date) = 0
```

The equation is solved using a simplified version of the Newton method.

Syntax:

```
XIRR([TOTAL [<fld {,fld}>]] pmt, date)
```

Return data type: numeric

Arguments

Argument	Description
pmt	Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date .
date	The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt .

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

When working with this function, the following limitations apply:

- **pmt** and **date** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.
- Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.
- This function requires at least one valid negative and at least one valid positive payment (with corresponding valid dates). If these payments are not provided, a NULL value is returned.

These topics might help you work with this function:

- *XNPV - chart function (page 380)*: Use this function to calculate aggregated net present value for a schedule of cash flows.
- *RangeXIRR (page 1325)*: **RangeXIRR()** is the equivalent range function for the **XIRR()** function.



Across different versions of Qlik Sense Client-Managed, there are variations in the underlying algorithm used by this function. For more information about recent updates to the algorithm, see support article [XIRR function Fix and Update](#).

Example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing cashflow transactions.
- Information stored in a table called **cashflow**.

Load script

Cashflow:

```
LOAD 2013 as Year, * inline [
Date|Payments
```

```
2013-01-01|-10000  
2013-03-01|3000  
2013-10-30|4200  
2014-02-01|6800  
] (delimiter is '|');
```

Results

Do the following:

Load the data and open a sheet. Create a new table and add the following calculation as a measure:

```
=XIRR(Payments, Date)
```

Results table	
=XIRR(Payments, Date)	
0.5385	

Interpreting the XIRR return value

The XIRR functionality is usually used to analyze an investment, where there is an outgoing (negative) payment in the beginning, and then a series of smaller income (positive) payments later on. Here is a simplified example with only one negative and one positive payment:

Cashflow:

```
LOAD * inline [  
Date|Payments  
2023-01-01|-100  
2024-01-01|110  
] (delimiter is '|');
```

We make an initial payment of 100 and get 110 back after exactly one year. This represents a rate of return of 10% per year. `XIRR(Payments, Date)` returns a value of 0.1.

The return value of the XIRR functionality can be positive or negative. In the case of an investment, a negative result indicates that the investment is a loss. The amount of gain or loss can be calculated simply by making a sum aggregation over the payments field.

In the example above, we are lending out our money for one year. The rate of return can be thought of as interest. It is also possible to use XIRR's functionality when you are on the other side of the transaction (for example, if you are the borrower instead of the lender).

Consider this example:

```
Cashflow:  
LOAD * inline [  
Date|Payments  
2023-01-01|100  
2024-01-01|-110  
] (delimiter is '|');
```

This is the same as the first example but inverted. Here, we are borrowing 100 for one year and we repay it with a 10% interest. In this example, the XIRR calculation returns 0.1 (10%), the same value as the first example.

Note that in the first example, we received a profit of 10, and in the second example, we experienced a loss of 10, but the return value of the XIRR functionality is positive for both these examples. This is because the XIRR functionality calculates the hidden interest in the transaction, regardless of which side you are on in the transaction.

Limitations with multiple solutions

Qlik's XIRR functionality is defined by the following equation, in which the `Rate` value is solved:

```
XNPV(Rate, pmt, date) = 0
```

It is sometimes possible for this equation to have more than one solution. This is known as the “multiple-IRR problem”, and is caused by a non-normal cash flow stream (also called an unconventional cash flow). The following load script shows an example of this:

```
Cashflow:  
LOAD * inline [  
Date|Payments  
2021-01-01|-200  
2022-01-01|500  
2023-01-01|-250  
] (delimiter is '|');
```

In this example, there is one negative solution and one positive solution (`Rate` = -0.3 and `Rate` = 0.8). **XIRR()** will return 0.8.

When Qlik's XIRR functionality searches for a solution, it starts at `Rate` = 0 and increases the rate in steps until it finds a solution. If there is more than one positive solution, it will return the first one that it encounters. If it cannot find a positive solution, it will reset the `Rate` back to zero and start searching for a solution in the negative direction.

Note that a “normal” cash flow stream is guaranteed to have only one solution. “Normal” cash flow stream means that all payments with the same sign (positive or negative) are in a continuous group.

See also:

-  [IRR - chart function \(page 357\)](#)
-  [Aggr - chart function \(page 525\)](#)
-  [XIRR function Fix and Update](#)

XNPV

XNPV() returns the aggregated net present value for a schedule of cashflows (not necessarily periodic) represented by paired numbers in **pmt** and **date**. All payments are discounted based on a 365-day year.

Syntax:

```
XNPV(discount_rate, pmt, date)
```

Return data type: numeric



By default, the result will be formatted as currency.

The formula to calculate XNPV is shown below:

XNPV aggregation formula

$$XNPV = \sum_{i=1}^n \frac{P_i}{(1+rate)^{(d_i-d_1)/365}}$$

where:

- P_i = Net cash inflow-outflows during a single period i
- d_1 = the first payment date
- d_i = the i^{th} payment date
- rate = discount rate

Net present value, or NPV, is used to calculate the current total value of a future stream of cash flows given a discount rate. To calculate XNPV, we need to estimate future cash flows with corresponding dates. After this, for each payment, we apply the compounded discount rate based on the date of the payment.

Performing the XNPV aggregation over a series of payments is similar to performing a Sum aggregation over those payments. The difference is that each amount is modified (or “discounted”) according to the chosen discount rate (similar to interest rate) and how far into the future the payment is. Performing XNPV with the **discount_rate** parameter set to zero will make XNPV equivalent to a Sum operation (the payments will not be modified before being summed). In general, the closer the **discount_rate** is set to zero, the more similar the XNPV result will be to that of a Sum aggregation.

Arguments

Argument	Description
discount_rate	discount_rate is the yearly rate that the payments should be discounted by. A value of 0.1 would indicate a 10% discount rate.

Argument	Description
pmt	Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date . Positive values are assumed to be inflows, and negative values are assumed to be outflows.
date	The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt . The first value is used as the start date for calculating the time offsets for future cashflows.

When working with this function, the following limitations apply:

- Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

When to use it

- XNPV() is used in financial modeling for calculating the net present value (NPV) of an investment opportunity.
- Due to its higher precision, XNPV is preferred over NPV, for all types of financial models.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Single payment (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of one project and its cashflow for one year, in a table named cashFlow. The initial date for calculation is set to July 1, 2022, with a net cashflow of 0. After one year, a cashflow of \$1000 occurs.
- A resident load from the cashFlow table, which is used to calculate the XNPV field for the project in a table named XNPV.
- A hard-coded discount rate of 10% (0.1) is used in the XNPV calculation.
- A Group By statement is used to group all the payments for the project.

Load script

```
CashFlow:  
Load  
*  
Inline  
[  
PrjId,Dates,values  
1,'07/01/2022',0  
1,'07/01/2023',1000  
];  
  
XNPV:  
Load  
    PrjId,  
    XNPV(0.1,values,Dates) as XNPV //Discount Rate of 10%  
Resident CashFlow  
Group By PrjId;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- XNPV

Results table

PrjId	XNPV
1	\$909.09

As per the formula, the XNPV value for the first record is 0, and for the second record, the XNPV value is \$909.09 Thus, the total XNPV is \$909.09.

Example 2 – Multiple payments (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of one project and its cashflow for one year, in a table named `CashFlow`.
- A resident load from the `CashFlow` table, which is used to calculate the `XNPV` field for the project in a table named `XNPV`.
- A hard-coded discount rate of 10% (0.1) is used in the `XNPV` calculation.
- A `Group By` statement is used to group all the payments for the project.

Load script

```
CashFlow:  
Load  
*  
Inline  
[  
PrjId,Dates,Values  
1,'07/01/2022',0  
1,'07/01/2024',500  
1,'07/01/2023',1000  
];  
  
XNPV:  
Load  
    PrjId,  
    XNPV(0.1,Values,Dates) as XNPV //Discount Rate of 10%  
Resident CashFlow  
Group By PrjId;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `PrjId`
- `XNPV`

Results table

PrjId	XNPV
1	\$1322.21

In this example, a payment of \$1000 is received at the end of first year, and a payment of \$500 is received at the end of second year. With a discount rate of 10% per period, the effective `XNPV` is equal to \$1322.21.

Note that only the first row of data should refer to the base date for calculations. For rest of the rows, order is not important, since the date parameter is used to calculate the elapsed period.

Example 3 – Multiple payments and irregular cashflows (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Discount rates for two projects in a table named `Project`.
- Cashflows for multiple periods for each project by project ID and Dates. The `Dates` field is used to calculate the duration for which discount rate is applied to the cash flow. Apart from the first record (initial cashflow and date), order of records is not important, and changing it should not impact the calculations.
- Using a combination of `Noconcatenate`, Resident loads, and the `Left Join` function, a temporary table, `tmpNPV`, is created that combines the records of the `Project` and `CashFlow` tables in one flat table. This table will have discount rates repeated for each cashflow.
- A resident load from the `tmpNPV` table, which is used to calculate the `XNPV` field for each project in a table named `XNPV`.
- The single value discount rate associated to each project is fetched using the `only()` function and is used in the `XNPV` calculation for each project.
- A `Group By` statement, grouping by project ID, is used to group all the payments and corresponding dates for each project.
- To avoid any synthetic or redundant data being loaded into the data model, the `tmpXNPV` table is dropped at the end of the script.

Load script

```
Project:  
Load * inline [  
PrjId,Discount_Rate  
1,0.1  
2,0.15  
];
```

```
CashFlow:  
Load  
*  
Inline  
[  
PrjId,Dates,Values  
1,'07/01/2021',0  
1,'07/01/2022',1000  
1,'07/01/2023',1000  
2,'07/01/2020',0  
2,'07/01/2023',500  
2,'07/01/2024',1000  
2,'07/01/2022',500  
];
```

```
tmpXNPV:  
NoConcatenate Load *  
Resident Project;  
Left Join  
Load *  
Resident CashFlow;  
  
XNPV:  
Load  
    PrjId,  
    XNPV(Only(DiscOUNT_Rate),values,Dates) as XNPV //Discount Rate will be 10% for Project 1 and  
15% for Project 2  
Resident tmpXNPV  
Group By PrjId;  
  
Drop table tmpXNPV;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- PrjId
- XNPV

Results table

PrjId	XNPV
1	\$1735.54
2	\$278.36

Project ID 1 has an initial cashflow of \$0 on July 1, 2021. There are two payments of \$1000 to be received at the end of two subsequent years, at a discount rate of 10% per period. Therefore, the effective XNPV is equal to \$1735.54.

Project ID 2 has an initial outflow of \$1000 (thus the negative sign) on July 1, 2020. After two years, a payment of \$500 is expected. After 3 years, another \$500 payment is expected. Finally, on July 1, 2024, a payment of \$1000 is expected. With the discount rate of 15%, the effective XNPV is equal to \$278.36.

See also:

- [Drop table \(page 147\)](#)
- [group by \(page 156\)](#)
- [Join \(page 69\)](#)
- [Max \(page 323\)](#)
- [NoConcatenate \(page 87\)](#)
- [NPV - chart function \(page 365\)](#)
- [Only \(page 333\)](#)

XNPV - chart function

XNPV() returns the aggregated net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date**, iterated over the chart dimensions. All payments are discounted based on a 365-day year.

Syntax:

```
XNPV( [TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

Return data type: numeric



By default, the result will be formatted as currency.

The formula to calculate XNPV is shown below:

XNPV aggregation formula

$$XNPV = \sum_{i=1}^n \frac{P_i}{(1+rate)^{(d_i-d_1)/365}}$$

where:

- P_i = Net cash inflow-outflows during a single period i
- d_1 = the first payment date
- d_i = the i^{th} payment date
- rate = discount rate

Net present value, or NPV, is used to calculate the current total value of a future stream of cash flows given a discount rate. To calculate XNPV, we need to estimate future cash flows with corresponding dates. After this, for each payment, we apply the compounded discount rate based on the date of the payment.

Performing the XNPV aggregation over a series of payments is similar to performing a Sum aggregation over those payments. The difference is that each amount is modified (or “discounted”) according to the chosen discount rate (similar to interest rate) and how far into the future the payment is. Performing XNPV with the **discount_rate** parameter set to zero will make XNPV equivalent to a Sum operation (the payments will not be modified before being summed). In general, the closer the **discount_rate** is set to zero, the more similar the XNPV result will be to that of a Sum aggregation.

Arguments

Argument	Description
discount_rate	discount_rate is the yearly rate that the payments should be discounted by. A value of 0.1 would indicate a 10% discount rate.

Argument	Description
pmt	<p>Payments. The expression or field containing the cash flows corresponding to the payment schedule given in date. Positive values are assumed to be inflows, and negative values are assumed to be outflows.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  XNPV() does not discount the initial cash flow since it will always happen on the start date. Subsequent payments are discounted based on a 365-day year. This is different from NPV(), where also the first payment is discounted. </div>
date	<p>The expression or field containing the schedule of dates corresponding to the cash flow payments given in pmt. The first value is used as the start date for calculating the time offsets for future cash flows.</p>
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>By using TOTAL [<fld {.fld}>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.</p>

When working with this function, the following limitations apply:

- **discount_rate**, **pmt** and **date** must not contain aggregation functions, unless these inner aggregations contain the **TOTAL** or **ALL** qualifiers. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.
- Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

When to use it

- **XNPV()** is used in financial modeling for calculating the net present value (NPV) of an investment opportunity.
- Due to its higher precision, XNPV is preferred over NPV, for all types of financial models.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the **SET DateFormat** statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing cashflow transactions.
- Information stored in a table called Cashflow.

Load script

Cashflow:

```
LOAD 2013 as Year, * inline [
Date|Payments
2013-01-01|-10000
2013-03-01|3000
2013-10-30|4200
2014-02-01|6800
] (delimiter is '|');
```

Results

Do the following:

Load the data and open a sheet. Create a new table and add the following calculation as a measure:

```
=XNPV(0.09, Payments, Date)
```

Results table

=XNPV(0.09, Payments, Date)
\$3062.49

See also:

- ❑ [NPV - chart function \(page 365\)](#)
- ❑ [Aggr - chart function \(page 525\)](#)

Statistical aggregation functions

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Statistical aggregation functions in the data load script

The following statistical aggregation functions can be used in scripts.

Avg

Avg() finds the average value of the aggregated data in the expression over a number of records as defined by a **group by** clause.

```
Avg ([distinct] expression)
```

Correl

Correl() returns the aggregated correlation coefficient for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
Correl (x-expression, y-expression)
```

Fractile

Fractile() finds the value that corresponds to the inclusive fractile (quantile) of the aggregated data in the expression over a number of records as defined by a **group by** clause.

```
Fractile (expression, fractile)
```

FractileExc

FractileExc() finds the value that corresponds to the exclusive fractile (quantile) of the aggregated data in the expression over a number of records as defined by a **group by** clause.

```
FractileExc (expression, fractile)
```

Kurtosis

Kurtosis() returns the kurtosis of the data in the expression over a number of records as defined by a **group by** clause.

```
Kurtosis ([distinct] expression )
```

LINEST_B

LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_B (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_df

LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_DF (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_f

This script function returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_F (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_m

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_M  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_r2

LINEST_R2() returns the aggregated r^2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_R2  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_seb

LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_SEB  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_sem

LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_SEM  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_sey

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_SEY  (y-expression, x-expression [, y0 [, x0 ]])
```

LINEST_ssreg

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_SSREG  (y-expression, x-expression [, y0 [, x0 ]])
```

LineST_ssresid

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
LINEST_SSRESID  (y-expression, x-expression [, y0 [, x0 ]])
```

Median

Median() returns the aggregated median of the values in the expression over a number of records as defined by a **group by** clause.

```
Median (expression)
```

Skew

Skew() returns the skewness of expression over a number of records as defined by a **group by** clause.

```
Skew ([ distinct] expression)
```

Stdev

Stdev() returns the standard deviation of the values given by the expression over a number of records as defined by a **group by** clause.

```
Stdev ([distinct] expression)
```

Sterr

Sterr() returns the aggregated standard error (stdev/sqrt(n)) for a series of values represented by the expression iterated over a number of records as defined by a **group by** clause.

```
Sterr ([distinct] expression)
```

STEYX

STEYX() returns the aggregated standard error of the predicted y-value for each x-value in the regression for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

```
STEYX (y-expression, x-expression)
```

Statistical aggregation functions in chart expressions

The following statistical aggregation functions can be used in charts.

Avg

Avg() returns the aggregated average of the expression or field iterated over the chart dimensions.

```
Avg - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

Correl

Correl() returns the aggregated correlation coefficient for two data sets. The correlation function is a measure of the relationship between the data sets and is aggregated for (x,y) value pairs iterated over the chart dimensions.

```
Correl - chart function({[SetExpression] [TOTAL [<fld {, fld}>]]} value1, value2 )
```

Fractile

Fractile() finds the value that corresponds to the inclusive fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.

```
Fractile - chart function({{[SetExpression] [TOTAL [<fld {, fld}>]]}} expr,  
fraction)
```

FractileExc

FractileExc() finds the value that corresponds to the exclusive fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.

```
FractileExc - chart function({{[SetExpression] [TOTAL [<fld {, fld}>]]}} expr,  
fraction)
```

Kurtosis

Kurtosis() finds the kurtosis of the range of data aggregated in the expression or field iterated over the chart dimensions.

```
Kurtosis - chart function({{[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]}}  
expr)
```

LINEST_b

LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_R2 - chart function({{[SetExpression] [TOTAL [<fld{, fld}>]]}} y_value,  
x_value[, y0_const[, x0_const]])
```

LINEST_df

LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_DF - chart function({{[SetExpression] [TOTAL [<fld{, fld}>]]}} y_value,  
x_value [, y0_const [, x0_const]])
```

LINEST_f

LINEST_F() returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and the **y_value**, iterated over the chart dimensions.

```
LINEST_F - chart function({{[SetExpression] [TOTAL [<fld{, fld}>]]}} y_value, x_  
value [, y0_const [, x0_const]])
```

LINEST_m

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_M - chart function({{[SetExpression] [TOTAL [<fld{, fld}>]]}} y_value, x_  
value [, y0_const [, x0_const]])
```

LINEST_r2

LINEST_R2() returns the aggregated r2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_R2 - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value,  
x_value[, y0_const[, x0_const]])
```

LINEST_seb

LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEB - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value,  
x_value[, y0_const[, x0_const]])
```

LINEST_sem

LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEM - chart function([{set_expression}][ distinct ] [total [<fld  
{, fld}>] ] y-expression, x-expression [, y0 [, x0 ]] )
```

LINEST_sey

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SEY - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value,  
x_value[, y0_const[, x0_const]])
```

LINEST_ssreg

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SSREG - chart function({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_  
value, x_value[, y0_const[, x0_const]])
```

LINEST_ssresid

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

```
LINEST_SSRESID - chart functionLINEST_SSRESID() returns the aggregated  
residual sum of squares of a linear regression defined by the equation  $y=mx+b$   
for a series of coordinates represented by paired numbers in the expressions  
given by x_value and y_value, iterated over the chart dimensions. LINEST_  
SSRESID([{SetExpression}] [DISTINCT] [TOTAL [<fld{ , fld}>]] y_value, x_value
```

```
[, y0_const[, x0_const]]) numeric ArgumentsArgumentDescriptiony_valueThe
expression or field containing the range of y-values to be measured.x_
valueThe expression or field containing the range of x-values to be
measured.y0, x0An optional value y0 may be stated forcing the regression line
to pass through the y-axis at a given point. By stating both y0 and x0 it is
possible to force the regression line to pass through a single fixed
coordinate. Unless both y0 and x0 are stated, the function requires at least
two valid data-pairs to calculate. If y0 and x0 are stated, a single data
pair will do. SetExpressionBy default, the aggregation function will
aggregate over the set of possible records defined by the selection. An
alternative set of records can be defined by a set analysis expression.
DISTINCTIf the word DISTINCT occurs before the function arguments, duplicates
resulting from the evaluation of the function arguments are disregarded.
TOTALIf the word TOTAL occurs before the function arguments, the calculation
is made over all possible values given the current selections, and not just
those that pertain to the current dimensional value, that is, it disregards
the chart dimensions. By using TOTAL [<fld{.fld}>], where the TOTAL
qualifier is followed by a list of one or more field names as a subset of the
chart dimension variables, you create a subset of the total possible
values.An optional value y0 may be stated forcing the regression line to pass
through the y-axis at a given point. By stating both y0 and x0 it is possible
to force the regression line to pass through a single fixed coordinate. The
parameter of the aggregation function must not contain other aggregation
functions, unless these inner aggregations contain the TOTAL qualifier. For
more advanced nested aggregations, use the advanced function Aggr, in
combination with a specified dimension. Text values, NULL values and missing
values in any or both pieces of a data-pair result in the entire data-pair
being disregarded. An example of how to use linest functionsavg
({[SetExpression] [TOTAL [<fld{, fld}>]] }y_value, x_value[, y0_const[, x0_
const]])
```

Median

Median() returns the median value of the range of values aggregated in the expression iterated over the chart dimensions.

```
Median - chart function{[SetExpression] [TOTAL [<fld{, fld}>]]} expr
```

MutualInfo

MutualInfo calculates the mutual information (MI) between two fields or between aggregated values in **Aggr()**.

```
MutualInfo - chart function{[SetExpression] [DISTINCT] [TOTAL target, driver
[, datatype [, breakdownbyvalue [, samplesize ]]]]}
```

Skew

Skew() returns the aggregated skewness of the expression or field iterated over the chart dimensions.

```
Skew - chart function{[SetExpression] [DISTINCT] [TOTAL [<fld{ ,fld}>]]} expr)
```

Stdev

Stdev() finds the standard deviation of the range of data aggregated in the expression or field iterated over the chart dimensions.

```
Stdev - chart function{[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

Sterr

Sterr() finds the value of the standard error of the mean, (stdev/sqrt(n)), for the series of values aggregated in the expression iterated over the chart dimensions.

```
Sterr - chart function{[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

STEYX

STEYX() returns the aggregated standard error when predicting y-values for each x-value in a linear regression given by a series of coordinates represented by paired numbers in the expressions given by **y_value** and **x_value**.

```
STEYX - chart function{[SetExpression] [TOTAL [<fld{, fld}>]]} y_value, x_value)
```

Avg

Avg() finds the average value of the aggregated data in the expression over a number of records as defined by a **group by** clause.

Syntax:

```
Avg ([DISTINCT] expr)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
DISTINCT	If the word distinct occurs before the expression, all duplicates will be disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre>Temp: crosstable (Month, Sales) load * inline [Customer Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Astrida 46 60 70 13 78 20 45 65 78 12 78 22 Betacab 65 56 22 79 12 56 45 24 32 78 55 15 Canutility 77 68 34 91 24 68 57 36 44 90 67 27 Divadip 36 44 90 67 27 57 68 47 90 80 94] (delimiter is ' ');</pre> <pre>Avg1: LOAD Customer, Avg(sales) as MyAverageSalesByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer MyAverageSalesByCustomer Astrida 48.916667 Betacab 44.916667 Canutility 56.916667 Divadip 63.083333 This can be checked in the sheet by creating a table including the measure: Sum(Sales)/12</pre>
<p>Given that the Temp table is loaded as in the previous example:</p> <pre>LOAD Customer, Avg(DISTINCT Sales) as MyAvgSalesDistinct Resident Temp Group By Customer;</pre>	<pre>Customer MyAverageSalesByCustomer Astrida 43.1 Betacab 43.909091 Canutility 55.909091 Divadip 61 Only the distinct values are counted. Divide the total by the number of non-duplicate values.</pre>

Avg - chart function

Avg() returns the aggregated average of the expression or field iterated over the chart dimensions.

Syntax:

```
Avg([{SetExpression}] [DISTINCT] [TOTAL [<fld[, fld]>]] expr)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld>]>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

Function examples

Example	Result
Avg(Sales)	For a table including the dimension Customer and the measure Avg([Sales]), if Totals are shown, the result is 2566.
Avg([TOTAL (Sales)])	53.458333 for all values of Customer, because the TOTAL qualifier means that dimensions are disregarded.
Avg(DISTINCT (Sales))	51.862069 for the total, because using the Distinct qualifier means only unique values in Sales for each customer are evaluated.

Data used in examples:

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
```

```
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

 [Aggr - chart function \(page 525\)](#)

Correl

Correl() returns the aggregated correlation coefficient for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
Correl(value1, value2)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value1, value2	The expressions or fields containing the two sample sets for which the correlation coefficient is to be measured.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre> Salary: Load *, 1 as Grp; LOAD * inline ["Employee name" Gender Age Salary Aiden Charles Male 20 25000 Brenda Davies Male 25 32000 Charlotte Edberg Female 45 56000 Daroush Ferrara Male 31 29000 Eunice Goldblum Female 31 32000 Freddy Halvorsen Male 25 26000 Gauri Indu Female 36 46000 Harry Jones Male 38 40000 Ian Underwood Male 40 45000 Jackie Kingsley Female 23 28000] (delimiter is ' '); Correl1: LOAD Grp, Correl(Age,Salary) as Correl_ Salary Resident Salary Group By Grp;</pre>	<p>In a table with the dimension <code>correl_Salary</code>, the result of the <code>Correl()</code> calculation in the data load script will be shown:</p> <p>0.9270611</p>

Correl - chart function

Correl() returns the aggregated correlation coefficient for two data sets. The correlation function is a measure of the relationship between the data sets and is aggregated for (x,y) value pairs iterated over the chart dimensions.

Syntax:

```
Correl([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] value1, value2 )
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value1, value2	The expressions or fields containing the two sample sets for which the correlation coefficient is to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {,fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Function examples

Example	Result
<code>Correl(Age, Salary)</code>	For a table including the dimension Employee name and the measure <code>Correl(Age, Salary)</code> , the result is 0.9270611. The result is only displayed for the totals cell.
<code>Correl(TOTAL Age, Salary)</code>	0.927. This and the following results are shown to three decimal places for readability. If you create a filter pane with the dimension Gender, and make selections from it, you see the result 0.951 when Female is selected and 0.939 if Male is selected. This is because the selection excludes all results that do not belong to the other value of Gender.
<code>Correl({1} TOTAL Age, Salary)</code>	0.927. Independent of selections. This is because the set expression {1} disregards all selections and dimensions.
<code>Correl(TOTAL <Gender> Age, Salary)</code>	0.927 in the total cell, 0.939 for all values of Male, and 0.951 for all values of Female. This corresponds to the results from making the selections in a filter pane based on Gender.

Data used in examples:

```
Salary:  
LOAD * inline [  
"Employee name" | Gender | Age | Salary  
Aiden Charles | Male | 20 | 25000  
Brenda Davies | Male | 25 | 32000  
Charlotte Edberg | Female | 45 | 56000  
Daroush Ferrara | Male | 31 | 29000  
Eunice Goldblum | Female | 31 | 32000  
Freddy Halvorsen | Male | 25 | 26000
```

```
Gauri Indu|Female|36|46000  
Harry Jones|Male|38|40000  
Ian Underwood|Male|40|45000  
Jackie Kingsley|Female|23|28000  
] (delimiter is '|');
```

See also:

- [Aggr - chart function \(page 525\)](#)
- [Avg - chart function \(page 390\)](#)
- [RangeCorrel \(page 1295\)](#)

Fractile

Fractile() finds the value that corresponds to the inclusive fractile (quantile) of the aggregated data in the expression over a number of records as defined by a **group by** clause.



You can use **FractileExc** (page 398) to calculate the exclusive fractile.

Syntax:

```
Fractile(expr, fraction)
```

Return data type: numeric

The function returns the value corresponding to the rank as defined by $\text{rank} = \text{fraction} * (\text{N}-1) + 1$ where N is the number of values in expr. If rank is a non-integer number, an interpolation is made between the two closest values.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to use when calculating the fractile.
fraction	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Fractile1: LOAD Type, Fractile(value,0.75) as MyFractile Resident Table1 Group By Type;</pre>	<p>In a table with the dimensions Type and MyFractile, the results of the Fractile() calculations in the data load script are:</p> <p>Type MyFractile Comparison 27.5 Observation 36</p>

Fractile - chart function

Fractile() finds the value that corresponds to the inclusive fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.



You can use *FractileExc* - chart function (page 400) to calculate the exclusive fractile.

Syntax:

```
Fractile([ {SetExpression} ] [DISTINCT] [TOTAL [<fld{, fld}>]]] expr, fraction)
```

Return data type: numeric

The function returns the value corresponding to the rank as defined by $\text{rank} = \text{fraction} * (\text{N}-1) + 1$ where N is the number of values in expr. If rank is a non-integer number, an interpolation is made between the two closest values.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to use when calculating the fractile.
fraction	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betocab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

Function examples

Example	Result
Fractile (Sales, 0.75)	For a table including the dimension <code>customer</code> and the measure <code>Fractile([Sales])</code> , if Totals are shown, the result is 71.75. This is the point in the distribution of values of sales that 75% of the values fall beneath.

Example	Result
Fractile (TOTAL Sales, 0.75))	71.75 for all values of customer, because the TOTAL qualifier means that dimensions are disregarded.
Fractile (DISTINCT Sales, 0.75)	70 for the total, because using the DISTINCT qualifier means only unique values in Sales for each Customer are evaluated.

Data used in examples:

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

- [Aggr - chart function \(page 525\)](#)

FractileExc

FractileExc() finds the value that corresponds to the exclusive fractile (quantile) of the aggregated data in the expression over a number of records as defined by a **group by** clause.



You can use Fractile (page 395) to calculate the inclusive fractile.

Syntax:

```
FractileExc(expr, fraction)
```

Return data type: numeric

The function returns the value corresponding to the rank as defined by $\text{rank} = \text{fraction} * (\text{N}+1)$ where N is the number of values in expr. If rank is a non-integer number, an interpolation is made between the two closest values.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to use when calculating the fractile.
fraction	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Fractile1: LOAD Type, FractileExc(Value,0.75) as MyFractile Resident Table1 Group By Type;</pre>	<p>In a table with the dimensions Type and MyFractile, the results of the FractileExc() calculations in the data load script are:</p> <p>Type MyFractile Comparison 28.5 Observation 38</p>

FractileExc - chart function

FractileExc() finds the value that corresponds to the exclusive fractile (quantile) of the aggregated data in the range given by the expression iterated over the chart dimensions.



You can use Fractile - chart function (page 396) to calculate the inclusive fractile.

Syntax:

```
FractileExc([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr,
fraction)
```

Return data type: numeric

The function returns the value corresponding to the rank as defined by $\text{rank} = \text{fraction} * (\text{N}+1)$ where N is the number of values in expr. If rank is a non-integer number, an interpolation is made between the two closest values.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to use when calculating the fractile.
fraction	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betocab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

Function examples

Example	Result
FractileExc (Sales, 0.75)	For a table including the dimension Customer and the measure FractileExc([sales]) , if Totals are shown, the result is 75.25. This is the point in the distribution of values of sales that 75% of the values fall beneath.

Example	Result
FractileExc (TOTAL Sales, 0.75))	75.25 for all values of customer, because the TOTAL qualifier means that dimensions are disregarded.
FractileExc (DISTINCT Sales, 0.75)	73.50 for the total, because using the DISTINCT qualifier means only unique values in sales for each customer are evaluated.

Data used in examples:

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

- [Aggr - chart function \(page 525\)](#)

Kurtosis

Kurtosis() returns the kurtosis of the data in the expression over a number of records as defined by a **group by** clause.

Syntax:

```
Kurtosis([distinct] expr )
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data

Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Kurtosis1: LOAD Type, Kurtosis(value) as MyKurtosis1, Kurtosis(DISTINCT value) as MyKurtosis2 Resident Table1 Group By Type;</pre>	<p>In a table with the dimensions Type, MyKurtosis1, and MyKurtosis2, the results of the Kurtosis() calculations in the data load script are:</p> <pre>Type MyKurtosis1 MyKurtosis2 Comparison -1.1612957 -1.4982366 observation -1.1148768 -0.93540144</pre>

Kurtosis - chart function

Kurtosis() finds the kurtosis of the range of data aggregated in the expression or field iterated over the chart dimensions.

Syntax:

```
Kurtosis([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Example table

Type	Value																								
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	9	7				
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2	9	9	5		

Function examples

Example	Result
Kurtosis (Value)	For a table including the dimension Type and the measure Kurtosis(value), if Totals are shown for the table, and number formatting is set to 3 significant figures, the result is 1.252. For comparison it is 1.161 and for observation it is 1.115.
Kurtosis (TOTAL value))	1.252 for all values of Type, because the TOTAL qualifier means that dimensions are disregarded.

Data used in examples:

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

See also:

- [Avg - chart function \(page 390\)](#)

LINEST_B

LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_B (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_B - chart function

LINEST_B() returns the aggregated b value (y-intercept) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_B([ {SetExpression} ] [DISTINCT] [TOTAL [<fld{, fld}>]]] y_value, x_value  
[, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0_const, x0_const	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.
	<p> Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</p>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>By using TOTAL [<fld {.fld}>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.</p>

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

-  Examples of how to use linest functions (page 444)
-  Avg - chart function (page 390)

LINEST_DF

LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_DF (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_DF - chart function

LINEST_DF() returns the aggregated degrees of freedom of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_DF([{SetExpression}] [DISTINCT] [TOTAL [<fld[, fld]>]] y_value, x_value [, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

-  [Examples of how to use linest functions \(page 444\)](#)
-  [Avg - chart function \(page 390\)](#)

LINEST_F

This script function returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_F (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_F - chart function

LINEST_F() returns the aggregated F statistic ($r^2/(1-r^2)$) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and the **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_F( [{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]]] y_value, x_value  
[, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)
- Avg - chart function (page 390)

LINEST_M

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_M (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_M - chart function

LINEST_M() returns the aggregated m value (slope) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_M( [{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]]] y_value, x_value  
[, y0_const [, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

-  [Examples of how to use linest functions \(page 444\)](#)
-  [Avg - chart function \(page 390\)](#)

LINEST_R2

LINEST_R2() returns the aggregated r^2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_R2 (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_R2 - chart function

LINEST_R2() returns the aggregated r2 value (coefficient of determination) of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_R2([{SetExpression}] [DISTINCT] [TOTAL [<fld[, fld]>]] y_value, x_
value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)
- Avg - chart function (page 390)

LINEST_SEB

LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_SEB (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_SEB - chart function

LINEST_SEB() returns the aggregated standard error of the b value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SEB([{SetExpression}] [DISTINCT] [TOTAL [<fld{}, fld>]]) y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)
- Avg - chart function (page 390)

LINEST_SEM

LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_SEM (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_SEM - chart function

LINEST_SEM() returns the aggregated standard error of the m value of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SEM([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_
value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)
- Avg - chart function (page 390)

LINEST_SEY

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_SEY (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_SEY - chart function

LINEST_SEY() returns the aggregated standard error of the y estimate of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SEY([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_
value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

-  [Examples of how to use linest functions \(page 444\)](#)
-  [Avg - chart function \(page 390\)](#)

LINEST_SSREG

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_SSREG (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_SSREG - chart function

LINEST_SSREG() returns the aggregated regression sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers given by the expressions **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SSREG([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)
- Avg - chart function (page 390)

LINEST_SSRESID

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
LINEST_SSRESID (y_value, x_value[, y0 [, x0 ]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.
y(0), x(0)	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate. Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

- Examples of how to use linest functions (page 444)

LINEST_SSRESID - chart function

LINEST_SSRESID() returns the aggregated residual sum of squares of a linear regression defined by the equation $y=mx+b$ for a series of coordinates represented by paired numbers in the expressions given by **x_value** and **y_value**, iterated over the chart dimensions.

Syntax:

```
LINEST_SSRESID([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value,  
x_value[, y0_const[, x0_const]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Argument	Description
y0, x0	An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.  <i>Unless both y0 and x0 are stated, the function requires at least two valid data-pairs to calculate. If y0 and x0 are stated, a single data pair will do.</i>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

An optional value y0 may be stated forcing the regression line to pass through the y-axis at a given point. By stating both y0 and x0 it is possible to force the regression line to pass through a single fixed coordinate.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

See also:

-  Examples of how to use linest functions (page 444)
-  Avg - chart function (page 390)

Median

Median() returns the aggregated median of the values in the expression over a number of records as defined by a **group by** clause.

Syntax:

Median (expr)

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.

Example: Script expression using Median

Example - script expression

Load script

Load the following inline data and script expression in the data load editor for this example.

Table 1:

```
Load RecNo() as RowNo, Letter, Number Inline  
[Letter, Number  
A,1  
A,3  
A,4  
A,9  
B,2  
B,8  
B,9];
```

Median:

```
LOAD Letter,  
Median(Number) as MyMedian  
Resident Table1 Group By Letter;
```

Create a visualization

Create a table visualization in a Qlik Sense sheet with **Letter** and **MyMedian** as dimensions.

Result

Letter	MyMedian
A	3.5
B	8

Explanation

The median is considered the "middle" number when the numbers have been sorted in order from smallest to greatest. If the data set has an even number of values, the function returns the average of the two middle values. In this example, the median is calculated for each set of values of **A** and **B**, which is 3.5 and 8, respectively.

Median - chart function

Median() returns the median value of the range of values aggregated in the expression iterated over the chart dimensions.

Syntax:

```
Median([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Example: Chart expression using Median

Example - chart expression

Load script

Load the following data as an inline load in the data load editor to create the chart expression example below.

```
Load RecNo() as RowNo, Letter, Number Inline
[Letter, Number
A,1
A,3
A,4
A,9
B,2
```

```
B,8  
B,9];
```

Create a visualization

Create a table visualization in a Qlik Sense sheet with **Letter** as a dimension.

Chart expression

Add the following expression to the table, as a measure:

```
Median(Number)
```

Result

The screenshot shows a table with two columns: 'Letter' and 'Median(Number)'. The 'Letter' column has three rows: 'Totals', 'A', and 'B'. The 'Median(Number)' column has three corresponding values: 4, 3.5, and 8. The table has a header row with a search icon and a sorting arrow. The 'Totals' row is bolded.

Letter	Median(Number)
Totals	4
A	3.5
B	8

Explanation

The median is considered the "middle" number when the numbers have been sorted in order from smallest to greatest. If the data set has an even number of values, the function returns the average of the two middle values. In this example, the median is calculated for each set of values of **A** and **B**, which is 3.5 and 8, respectively.

The median for **Totals** is calculated from all values, which equals 4.

See also:

[Avg - chart function \(page 390\)](#)

MutualInfo - chart function

MutualInfo calculates the mutual information (MI) between two fields or between aggregated values in **Aggr()**.

MutualInfo returns the aggregated mutual information for two datasets. This allows key driver analysis between a field and a potential driver. Mutual information measures the relationship between the datasets and is aggregated for (x,y) pair values iterated over the chart dimensions. Mutual information is measured between 0 and 1 and can be formatted as a percentile value. **MutualInfo** is defined by either selections or by a set expression.

MutualInfo allows different kinds of MI analysis:

- Pair-wise MI: Calculate the MI between a driver field and a target field.
- Driver breakdown by value: The MI is calculated between individual field values in the driver and target fields.
- Feature selection: Use **MutualInfo** in a grid chart to create a matrix where all fields are compared to each other based on MI.

MutualInfo does not necessarily indicate causality between fields sharing mutual information. Two fields may share mutual information, but may not be equal drivers for each other. For example, when comparing ice cream sales and outdoor temperature, **MutualInfo** will show mutual information between the two. It will not indicate if it is outdoor temperature driving ice cream sales, which is likely, or if it is ice cream sales that drives outdoor temperature, which is unlikely.

When calculating mutual information, associations affect the correspondence between and the frequency of values from fields that are from different tables.

Returned values for the same fields or selections may vary slightly. This is due to each **MutualInfo** call operating on a randomly selected sample and the inherent randomness of the **MutualInfo** algorithm.

MutualInfo can be applied to the **Aggr()** function.

Syntax:

```
MutualInfo([{SetExpression}] [DISTINCT] [TOTAL] field1, field2 , datatype [, breakdownbyvalue [, samplesize ]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
field1, field2	The expressions or fields containing the two sample sets for which the mutual information to be measured.
datatype	The data types contained in the target and driver, 1 or 'dd' for discrete:discrete 2 or 'cc' for continuous:continuous 3 or 'cd' for continuous:discrete 4 or 'dc' for discrete:continuous Data types are not case sensitive.

Argument	Description
breakdownbyvalue	A static value corresponding to a value in the driver. If supplied, the calculation will calculate the MI contribution for that value. You can use ValueList() or ValueLoop() . If Null() is added, the calculation will calculate the overall MI for all values in the driver. Breaking down by value requires the driver contain discrete data.
samplesize	The number of values to sample from the target and driver. Sampling is random. MutualInfo requires a minimum sample size of 80. By default, MutualInfo only samples up to 10,000 data-pairs as MutualInfo can be resource intensive. You can specify greater numbers of data-pairs in the sample size. If MutualInfo times out, reduce the sample size.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Function examples

Example	Result
<code>mutualinfo(Age, Salary, 1)</code>	For a table including the dimension Employee name and the measure <code>mutualinfo(Age, salary, 1)</code> , the result is 0.99820986. The result is only displayed for the totals cell.
<code>mutualinfo(TOTAL Age, salary, 1, null (), 81)</code>	If you create a filter pane with the dimension Gender, and make selections from it, you see the result 0.99805677 when Female is selected and 0.99847373 if Male is selected. This is because the selection excludes all results that do not belong to the other value of Gender.

Example	Result
<code>mutualinfo (TOTAL Age, Gender, 1, valueLoop (25,35))</code>	0.68196996. Selecting any value from Gender will change this to 0.
<code>mutualinfo({1} TOTAL Age, Salary, 1, null)</code>	0.99820986. This is independent of selections. The set expression {1} disregards all selections and dimensions.

Data used in examples:

```
Salary:  
LOAD * inline [  
"Employee name"|Age|Gender|Salary  
Aiden Charles|20|Male|25000  
Ann Lindquist|69|Female|58000  
Anna Johansen|37|Female|36000  
Anna Karlsson|42|Female|23000  
Antonio Garcia|20|Male|61000  
Benjamin Smith|42|Male|27000  
Bill Yang|49|Male|50000  
Binh Protzmann|69|Male|21000  
Bob Park|51|Male|54000  
Brenda Davies|25|Male|32000  
Celine Gagnon|48|Female|38000  
Cezar Sandu|50|Male|46000  
Charles Ingvar Jönsson|27|Male|58000  
Charlotte Edberg|45|Female|56000  
Cindy Lynn|69|Female|28000  
Clark Wayne|63|Male|31000  
Daroush Ferrara|31|Male|29000  
David Cooper|37|Male|64000  
David Leg|58|Male|57000  
Eunice Goldblum|31|Female|32000  
Freddy Halvorsen|25|Male|26000  
Gauri Indu|36|Female|46000  
George van Zaant|59|Male|47000  
Glenn Brown|58|Male|40000  
Harry Jones|38|Male|40000  
Helen Brolin|52|Female|66000  
Hiroshi Ito|24|Male|42000  
Ian Underwood|40|Male|45000  
Ingrid Hendrix|63|Female|27000  
Ira Baumel|39|Female|39000  
Jackie Kingsley|23|Female|28000  
Jennica Williams|36|Female|48000  
Jerry Tesse|31|Male|57000  
Jim Bond|50|Male|58000  
Joan Collins|60|Female|65000  
Joan Cleaves|25|Female|61000  
Joe Cheng|61|Male|41000  
John Doe|36|Male|59000  
John Lemon|43|Male|21000  
Karen Helmkey|54|Female|25000
```

```
Karl Berger|38|Male|68000
Karl Straubau|m 30|Male|40000
Kaya Alpan|32|Female|60000
Kenneth Finley|21|Male|25000
Leif Shine|63|Male|70000
Lennart Skoglund|63|Male|24000
Leona Korhonen|46|Female|50000
Lina André|50|Female|65000
Louis Presley|29|Male|36000
Luke Langston|50|Male|63000
Marcus Salvatori|31|Male|46000
Marie Simon|57|Female|23000
Mario Rossi|39|Male|62000
Markus Danzig|26|Male|48000
Michael Carlen|21|Male|45000
Michelle Tyson|44|Female|69000
Mike Ashkenaz|45|Male|68000
Miro Ito|40|Male|39000
Nina Mihm|62|Female|57000
olivia Nguyen|35|Female|51000
olivier Simenon|44|Male|31000
Östen Ärlig|68|Male|57000
Pamala Garcia|69|Female|29000
Paolo Romano|34|Male|45000
Pat Taylor|67|Female|69000
Paul Dupont|34|Male|38000
Peter Smith|56|Male|53000
Pierre Clouseau|21|Male|37000
Preben Jørgensen|35|Male|38000
Rey Jones|65|Female|20000
Ricardo Gucci|55|Male|65000
Richard Ranieri|30|Male|64000
Rob Carsson|46|Male|54000
Rolf Wesenlund|25|Male|51000
Ronaldo Costa|64|Male|39000
Sabrina Richards|57|Female|40000
Sato Hiromu|35|Male|21000
Sehoon Daw|57|Male|24000
Stefan Lind|67|Male|35000
Steve Cioazzi|58|Male|23000
Sunil Gupta|45|Male|40000
Sven Svensson|45|Male|55000
Tom Lindwall|46|Male|24000
Tomas Nilsson|27|Male|22000
Trinity Rizzo|52|Female|48000
Vanessa Lambert|54|Female|27000
] (delimiter is '|');
```

Skew

Skew() returns the skewness of expression over a number of records as defined by a **group by** clause.

Syntax:

```
Skew([ distinct] expr)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
DISTINCT	If the word distinct occurs before the expression, all duplicates will be disregarded.

Examples and results:

Add the example script to your app and run it. Then build a straight table with `Type` and `MySkew` as dimensions.

Resulting data	
Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Skew1: LOAD Type, Skew(Value) as Myskew Resident Table1 Group By Type;</pre>	<p>The results of the <code>Skew()</code> calculation are:</p> <ul style="list-style-type: none"> • <code>Type</code> is <code>Myskew</code> • <code>Comparison</code> is <code>0.86414768</code> • <code>Observation</code> is <code>0.32625351</code>

Skew - chart function

`Skew()` returns the aggregated skewness of the expression or field iterated over the chart dimensions.

Syntax:

```
Skew([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Add the example script to your app and run it. Then build a straight table with Type as dimension and skew (value) as measure.

Totals should be enabled in the properties of the table.

Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' ');</pre>	<p>The results of the Skew(Value) calculation are:</p> <ul style="list-style-type: none"> • Total is 0.23522195 • Comparison is 0.86414768 • Observation is 0.32625351

See also:

□ [Avg - chart function \(page 390\)](#)

Stdev

Stdev() returns the standard deviation of the values given by the expression over a number of records as defined by a **group by** clause.

Syntax:

```
Stdev([distinct] expr)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Examples and results:

Add the example script to your app and run it. Then build a straight table with Type and MyStdev as dimensions.

Resulting data	
Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Stdev1: LOAD Type, Stdev(value) as MyStdev Resident Table1 Group By Type;</pre>	<p>The results of the Stdev() calculation are:</p> <ul style="list-style-type: none"> • Type is MyStdev • Comparison is 14.61245 • observation is 12.507997

Stdev - chart function

Stdev() finds the standard deviation of the range of data aggregated in the expression or field iterated over the chart dimensions.

Syntax:

```
Stdev( [ {SetExpression} ] [ DISTINCT ] [ TOTAL [ <fld{, fld}> ] ] expr )
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Examples and results:

Add the example script to your app and run it. Then build a straight table with Type as dimension and stdev (Value) as measure.

Totals should be enabled in the properties of the table.

Example	Result
<pre>Stdev(value) Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' ');</pre>	<p>The results of the Stdev(Value) calculation are:</p> <ul style="list-style-type: none"> • Total is 15.47529 • Comparison is 14.61245 • Observation is 12.507997

See also:

- [Avg - chart function \(page 390\)](#)
- [STEYX - chart function \(page 442\)](#)

Sterr

Sterr() returns the aggregated standard error (stdev/\sqrt{n}) for a series of values represented by the expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
Sterr ([distinct] expr)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
distinct	If the word distinct occurs before the expression, all duplicates will be disregarded.

Limitations:

Text values, NULL values and missing values are disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' '); Sterr1: LOAD Type, Sterr(Value) as MySterr Resident Table1 Group By Type;</pre>	<p>In a table with the dimensions Type and MySterr, the results of the Sterr() calculation in the data load script are:</p> <p>Type MySterr Comparison 3.2674431 Observation 2.7968733</p>

Sterr - chart function

Sterr() finds the value of the standard error of the mean, ($stdev/\sqrt{n}$), for the series of values aggregated in the expression iterated over the chart dimensions.

Syntax:

```
Sterr([ {SetExpression} ] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values are disregarded.

Examples and results:

Add the example script to your app and run it. Then build a straight table with **Type** as dimension and **sterr (value)** as measure.

Totals should be enabled in the properties of the table.

Example	Result
<pre>Table1: crosstable LOAD recno() as ID, * inline [Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2] (delimiter is ' ');</pre>	<p>The results of the <code>Sterr(Value)</code> calculation are:</p> <ul style="list-style-type: none"> • Total is 2.4468583 • Comparison is 3.2674431 • Observation is 2.7968733

See also:

- [Avg - chart function \(page 390\)](#)
- [STEYX - chart function \(page 442\)](#)

STEYX

STEYX() returns the aggregated standard error of the predicted y-value for each x-value in the regression for a series of coordinates represented by paired numbers in x-expression and y-expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
STEYX (y_value, x_value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
y_value	The expression or field containing the range of y-values to be measured.
x_value	The expression or field containing the range of x-values to be measured.

Limitations:

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data	
Example	Result
<pre>Trend: Load *, 1 as Grp; LOAD * inline [Month KnownY KnownX Jan 2 6 Feb 3 5 Mar 9 11 Apr 6 7 May 8 5 Jun 7 4 Jul 5 5 Aug 10 8 Sep 9 10 Oct 12 14 Nov 15 17 Dec 14 16] (delimiter is ' '); STEYX1: LOAD Grp, STEYX(KnownY, KnownX) as MySTEYX Resident Trend Group By Grp;</pre>	In a table with the dimension MYSTEYX, the result of the STEYX() calculation in the data load script is 2.0714764.

STEYX - chart function

STEYX() returns the aggregated standard error when predicting y-values for each x-value in a linear regression given by a series of coordinates represented by paired numbers in the expressions given by **y_value** and **x_value**.

Syntax:

```
STEYX([{SetExpression}][, [DISTINCT] [TOTAL [<fld{, fld}>]]] y_value, x_value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
y_value	The expression or field containing the range of known y-values to be measured.
x_value	The expression or field containing the range of known x-values to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

The parameter of the aggregation function must not contain other aggregation functions, unless these inner aggregations contain the **TOTAL** qualifier. For more advanced nested aggregations, use the advanced function **Aggr**, in combination with a specified dimension.

Text values, NULL values and missing values in any or both pieces of a data-pair result in the entire data-pair being disregarded.

Examples and results:

Add the example script to your app and run it. Then build a straight table with `KnownY` and `KnownX` as dimension and `Steyx(KnownY, KnownX)` as measure.

Totals should be enabled in the properties of the table.

Example	Result
<pre>Trend: LOAD * inline [Month KnownY KnownX Jan 2 6 Feb 3 5 Mar 9 11 Apr 6 7 May 8 5 Jun 7 4 Jul 5 5 Aug 10 8 Sep 9 10 Oct 12 14 Nov 15 17 Dec 14 16] (delimiter is ' ');</pre>	The result of the STEYX(KnownY, KnownX) calculation is 2.071 (If number formatting is set to 3 decimal places.)

See also:

- [Avg - chart function \(page 390\)](#)
- [Sterr - chart function \(page 439\)](#)

Examples of how to use linest functions

The linest functions are used to find values associated with linear regression analysis. This section describes how to build visualizations using sample data to find the values of the linest functions available in Qlik Sense. The linest functions can be used in the data load script and in chart expressions.

Refer to the individual linest chart function and script function topics for descriptions of syntax and arguments.

Data and script expressions used in the examples

Load the following inline data and script expressions in the data load editor for the linest() examples below.

```
T1:  
LOAD *, 1 as Grp;  
LOAD * inline [  
X|Y  
1|0  
2|1  
3|3  
4|8  
5|14  
6|20  
7|0  
8|50  
9|25  
10|60  
11|38  
12|19  
13|26  
14|143  
15|98
```

```

16|27
17|59
18|78
19|158
20|279 ] (delimiter is '|');

R1:
LOAD
Grp,
Linest_B(Y,X) as Linest_B,
Linest_DF(Y,X) as Linest_DF,
Linest_F(Y,X) as Linest_F,
Linest_M(Y,X) as Linest_M,
Linest_R2(Y,X) as Linest_R2,
Linest_SEB(Y,X,1,1) as Linest_SEB,
Linest_SEM(Y,X) as Linest_SEM,
Linest_SEY(Y,X) as Linest_SEY,
Linest_SSREG(Y,X) as Linest_SSREG,
Linest_SSRESID(Y,X) as Linest_SSRESID
resident T1 group by Grp;

```

Example 1: Script expressions using linest

Example: Script expressions

Create a visualization from the data load script calculations

Create a table visualization in a Qlik Sense sheet with the following fields as columns:

- Linest_B
- Linest_DF
- Linest_F
- Linest_M
- Linest_R2
- Linest_SEB
- Linest_SEM
- Linest_SEY
- Linest_SSREG
- Linest_SSRESID

Result

The table containing the results of the linest calculations made in the data load script should look like this:

Results table

Linest_B	Linest_DF	Linest_F	Linest_M	Linest_R2	Linest_SEB
-35.047	18	20.788	8.605	0.536	22.607

Results table

Linest_SEM	Linest_SEY	Linest_SSREG	Linest_SSRESID
1.887	48.666	49235.014	42631.186

Example 2: Chart expressions using linest

Example: Chart expressions

Create a table visualization in a Qlik Sense sheet with the following fields as dimensions:

```
valueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID')
```

This expression uses the synthetic dimensions function to create labels for the dimensions with the names of the linest functions. You can change the label to **Linest functions** to save space.

Add the following expression to the table as a measure:

```
Pick(Match(ValueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID'),'Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID'),Linest_b(Y,X),Linest_df(Y,X),Linest_f(Y,X),Linest_m(Y,X),Linest_r2(Y,X),Linest_SEB(Y,X,1,1),Linest_SEM(Y,X),Linest_SEY(Y,X),Linest_SSREG(Y,X),Linest_SSRESID(Y,X) )
```

This expression displays the value of the result of each linest function against the corresponding name in the synthetic dimension. The result of Linest_b(Y,X) is displayed next to **linest_b**, and so on.

Result

Results table

Linest functions	Linest function results
Linest_b	-35.047
Linest_df	18
Linest_f	20.788
Linest_m	8.605
Linest_r2	0.536
Linest_SEB	22.607
Linest_SEM	1.887
Linest_SEY	48.666
Linest_SSREG	49235.014
Linest_SSRESID	42631.186

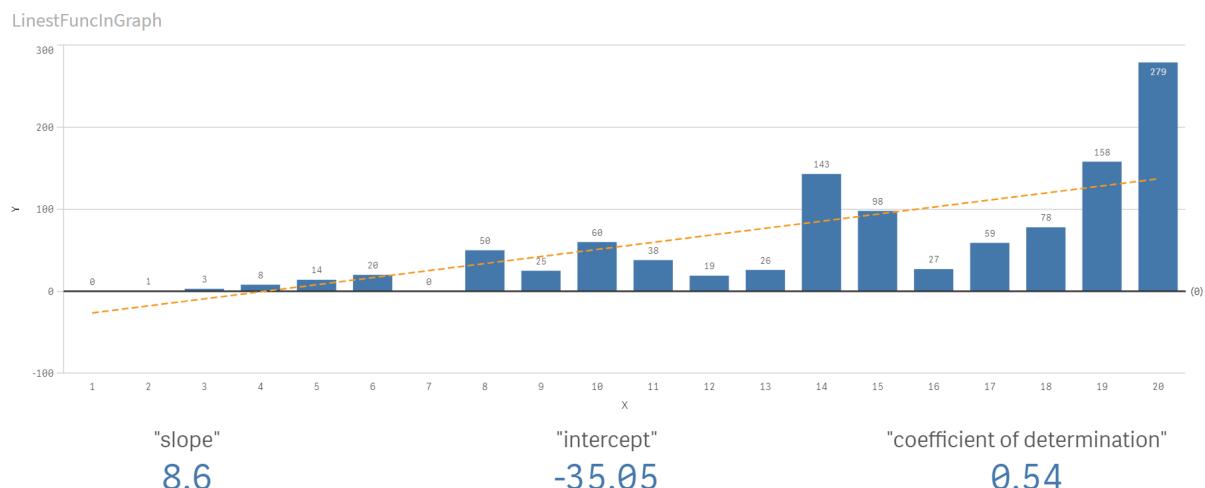
Example 3: Chart expressions using linest

Example: Chart expressions

1. Create a barchart visualization in a Qlik Sense sheet with **X** as a dimension and **Y** as a measure.
2. Add a linear trend line to the Y measure.

3. Add a KPI visualization to the sheet.
 1. Add *slope* as a label for the KPI.
 2. Add `sum(Linest_M)` as an expression for the KPI.
4. Add a second KPI visualization to the sheet.
 1. Add *intercept* as a label for the KPI.
 2. Add `sum(Linest_B)` as an expression for the KPI.
5. Add a third KPI visualization to the sheet.
 1. Add *coefficient of determination* as a label for the KPI.
 2. Add `sum(Linest_R2)` as an expression for the KPI.

Result



Explanation

The barchart shows the plotting of the X and Y data. Relevant `linest()` functions provide values for the linear regression equation that the trend line is based on, namely $y = m * x + b$. The equation uses the "least squares" method to calculate a straight line (trend line) by returning an array that describes a line that best fits the data.

The KPIs display the results of the `linest()` functions `sum(Linest_M)` for slope and `sum(Linest_B)` for the Y intercept, which are variables in the linear regression equation, and the corresponding aggregated R2 value for coefficient of determination.

Statistical test functions

Statistical test functions can be used in both the data load script and chart expressions, but the syntax differs.

Chi-2 test functions

Generally used in the study of qualitative variables. One can compare observed frequencies in a one-way frequency table with expected frequencies, or study the connection between two variables in a contingency table.

T-test functions

T-test functions are used for statistical examination of two population means. A two-sample t-test examines whether two samples are different and is commonly used when two normal distributions have unknown variances and when an experiment uses a small sample size.

Z-test functions

A statistical examination of two population means. A two sample z-test examines whether two samples are different and is commonly used when two normal distributions have known variances and when an experiment uses a large sample size.

Chi2-test functions

Generally used in the study of qualitative variables. One can compare observed frequencies in a one-way frequency table with expected frequencies, or study the connection between two variables in a contingency table. Chi-squared test functions are used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more groups. Often a histogram is used, and the different bins are compared to an expected distribution.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Chi2Test_chi2

Chi2Test_chi2() returns the aggregated chi²-test value for one or two series of values.

```
Chi2Test_chi2() returns the aggregated chi2-test value for one or two series  
of values.(col, row, actual_value[, expected_value])
```

Chi2Test_df

Chi2Test_df() returns the aggregated chi²-test df value (degrees of freedom) for one or two series of values.

```
Chi2Test_df() returns the aggregated chi2-test df value (degrees of freedom)  
for one or two series of values.(col, row, actual_value[, expected_value])
```

Chi2Test_p

Chi2Test_p() returns the aggregated chi²-test p value (significance) for one or two series of values.

```
Chi2Test_p - chart function(col, row, actual_value[, expected_value])
```

See also:

- [T-test functions \(page 451\)](#)
- [Z-test functions \(page 485\)](#)

Chi2Test_chi2

Chi2Test_chi2() returns the aggregated chi²-test value for one or two series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.



All Qlik Sense chi² -test functions have the same arguments.

Syntax:

```
Chi2Test_chi2(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
col, row	The specified column and row in the matrix of values being tested.
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_chi2( Grp, Grade, Count )
Chi2Test_chi2( Gender, Description, Observed, Expected )
```

See also:

- ❑ Examples of how to use chi2-test functions in charts (page 500)
- ❑ Examples of how to use chi2-test functions in the data load script (page 503)

Chi2Test_df

Chi2Test_df() returns the aggregated chi²-test df value (degrees of freedom) for one or two series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.



All Qlik Sense χ^2 -test functions have the same arguments.

Syntax:

```
Chi2Test_df(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
col, row	The specified column and row in the matrix of values being tested.
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_df( Grp, Grade, Count )
Chi2Test_df( Gender, Description, Observed, Expected )
```

See also:

- Examples of how to use chi2-test functions in charts (page 500)
- Examples of how to use chi2-test functions in the data load script (page 503)

Chi2Test_p - chart function

Chi2Test_p() returns the aggregated χ^2 -test p value (significance) for one or two series of values. The test can be done either on the values in **actual_value**, testing for variations within the specified **col** and **row** matrix, or by comparing values in **actual_value** with corresponding values in **expected_value**, if specified.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.



All Qlik Sense χ^2 -test functions have the same arguments.

Syntax:

```
Chi2Test_p(col, row, actual_value[, expected_value])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
col, row	The specified column and row in the matrix of values being tested.
actual_value	The observed value of the data at the specified col and row .
expected_value	The expected value for the distribution at the specified col and row .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
Chi2Test_p( Grp, Grade, Count )
Chi2Test_p( Gender, Description, Observed, Expected )
```

See also:

- Examples of how to use chi2-test functions in charts (page 500)
- Examples of how to use chi2-test functions in the data load script (page 503)

T-test functions

T-test functions are used for statistical examination of two population means. A two-sample t-test examines whether two samples are different and is commonly used when two normal distributions have unknown variances and when an experiment uses a small sample size.

In the following sections, the t-test statistical test functions are grouped according to the sample student test that applies to each type of function.

Creating a typical t-test report (page 504)

Two independent samples t-tests

The following functions apply to two independent samples student's t-tests.

ttest_conf

TTest_conf returns the aggregated t-test confidence interval value for two independent samples.

```
TTest_conf returns the aggregated t-test confidence interval value for two
independent samples. ( grp, value [, sig[, eq_var]])
```

ttest_df

TTest_df() returns the aggregated student's t-test value (degrees of freedom) for two independent series of values.

```
TTest_df() returns the aggregated student's t-test value (degrees of freedom)
for two independent series of values. (grp, value [, eq_var])
```

ttest_dif

TTest_dif() is a numeric function that returns the aggregated student's t-test mean difference for two independent series of values.

```
TTest_dif() is a numeric function that returns the aggregated student's t-
test mean difference for two independent series of values. (grp, value)
```

ttest_lower

TTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

```
TTest_lower() returns the aggregated value for the lower end of the
confidence interval for two independent series of values. (grp, value [, sig
[, eq_var]])
```

ttest_sig

TTest_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values.

```
TTest_sig() returns the aggregated student's t-test 2-tailed level of
significance for two independent series of values. (grp, value [, eq_var])
```

ttest_sterr

TTest_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values.

```
TTest_sterr() returns the aggregated student's t-test standard error of the
mean difference for two independent series of values. (grp, value [, eq_var])
```

ttest_t

TTest_t() returns the aggregated t value for two independent series of values.

```
TTest_t() returns the aggregated t value for two independent series of
values. (grp, value [, eq_var])
```

ttest_upper

TTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

```
TTest_upper() returns the aggregated value for the upper end of the
confidence interval for two independent series of values. (grp, value [, sig
[, eq_var]])
```

Two independent weighted samples t-tests

The following functions apply to two independent samples student's t-tests where the input data series is given in weighted two-column format.

ttestw_conf

TTestw_conf() returns the aggregated t value for two independent series of values.

```
TTestw_conf() returns the aggregated t value for two independent series of values. (weight, grp, value [, sig[, eq_var]])
```

ttestw_df

TTestw_df() returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values.

```
TTestw_df() returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values. (weight, grp, value [, eq_var])
```

ttestw_dif

TTestw_dif() returns the aggregated student's t-test mean difference for two independent series of values.

```
TTestw_dif() returns the aggregated student's t-test mean difference for two independent series of values. ( weight, grp, value)
```

ttestw_lower

TTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

```
TTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values. (weight, grp, value [, sig[, eq_var]])
```

ttestw_sig

TTestw_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values.

```
TTestw_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values. ( weight, grp, value [, eq_var])
```

ttestw_sterr

TTestw_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values.

```
TTestw_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values. (weight, grp, value [, eq_var])
```

ttestw_t

TTestw_t() returns the aggregated t value for two independent series of values.

```
TTestw_t() returns the aggregated t value for two independent series of values. (weight, grp, value [, eq_var])
```

ttestw_upper

TTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

```
TTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values. (weight, grp, value [, sig [, eq_var]])
```

One sample t-tests

The following functions apply to one-sample student's t-tests.

ttest1_conf

TTest1_conf() returns the aggregated confidence interval value for a series of values.

```
TTest1_conf() returns the aggregated confidence interval value for a series of values. (value [, sig])
```

ttest1_df

TTest1_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values.

```
TTest1_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values. (value)
```

ttest1_dif

TTest1_dif() returns the aggregated student's t-test mean difference for a series of values.

```
TTest1_dif() returns the aggregated student's t-test mean difference for a series of values. (value)
```

ttest1_lower

TTest1_lower() returns the aggregated value for the lower end of the confidence interval for a series of values.

```
TTest1_lower() returns the aggregated value for the lower end of the confidence interval for a series of values. (value [, sig])
```

ttest1_sig

TTest1_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values.

```
TTest1_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values. (value)
```

ttest1_sterr

TTest1_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values.

```
TTest1_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values. (value)
```

ttest1_t

TTest1_t() returns the aggregated t value for a series of values.

```
TTest1_t() returns the aggregated t value for a series of values. (value)
```

ttest1_upper

TTest1_upper() returns the aggregated value for the upper end of the confidence interval for a series of values.

```
TTest1_upper() returns the aggregated value for the upper end of the  
confidence interval for a series of values. (value [, sig])
```

One weighted sample t-tests

The following functions apply to one-sample student's t-tests where the input data series is given in weighted two-column format.

ttest1w_conf

TTest1w_conf() is a **numeric** function that returns the aggregated confidence interval value for a series of values.

```
TTest1w_conf() is a numeric function that returns the aggregated confidence  
interval value for a series of values. (weight, value [, sig])
```

ttest1w_df

TTest1w_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values.

```
TTest1w_df() returns the aggregated student's t-test df value (degrees of  
freedom) for a series of values. (weight, value)
```

ttest1w_dif

TTest1w_dif() returns the aggregated student's t-test mean difference for a series of values.

```
TTest1w_dif() returns the aggregated student's t-test mean difference for a  
series of values. (weight, value)
```

ttest1w_lower

TTest1w_lower() returns the aggregated value for the lower end of the confidence interval for a series of values.

```
TTest1w_lower() returns the aggregated value for the lower end of the  
confidence interval for a series of values. (weight, value [, sig])
```

ttest1w_sig

TTest1w_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values.

```
TTest1w_sig() returns the aggregated student's t-test 2-tailed level of  
significance for a series of values. (weight, value)
```

ttest1w_sterr

TTest1w_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values.

```
TTest1w_sterr() returns the aggregated student's t-test standard error of the  
mean difference for a series of values. (weight, value)
```

ttest1w_t

TTest1w_t() returns the aggregated t value for a series of values.

```
TTest1w_t() returns the aggregated t value for a series of values. ( weight,  
value)
```

ttest1w_upper

TTest1w_upper() returns the aggregated value for the upper end of the confidence interval for a series of values.

```
TTest1w_upper() returns the aggregated value for the upper end of the  
confidence interval for a series of values. (weight, value [, sig])
```

TTest_conf

TTest_conf returns the aggregated t-test confidence interval value for two independent samples.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_conf ( grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Argument	Description
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_conf( Group, value )
TTest_conf( Group, value, sig, false )
```

See also:

□ *Creating a typical t-test report (page 504)*

TTest_df

TTest_df() returns the aggregated student's t-test value (degrees of freedom) for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_df (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_df( Group, value )
TTest_df( Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest_dif

TTest_dif() is a numeric function that returns the aggregated student's t-test mean difference for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_dif (grp, value [, eq_var] )
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_dif( Group, value )
TTest_dif( Group, value, false )
```

See also:

- Creating a typical t-test report (page 504)*

TTest_lower

TTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_lower (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTtest_lower( Group, value )
TTtest_lower( Group, value, Sig, false )
```

See also:

 [Creating a typical t-test report \(page 504\)](#)

TTest_sig

TTest_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_sig (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_sig( Group, value )
TTest_sig( Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest_sterr

TTest_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_sterr (grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest_sterr( Group, value )
TTest_sterr( Group, value, false )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)
-

TTest_t

TTest_t() returns the aggregated t value for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest_t(grp, value[, eq_var])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest_t( Group, Value, false )
```

See also:

- Creating a typical t-test report (page 504)*

TTest_upper

TTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTTest_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTTest_upper( Group, value )
TTTest_upper( Group, value, sig, false )
```

See also:

 *Creating a typical t-test report (page 504)*

TTestw_conf

TTestw_conf() returns the aggregated t value for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_conf (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_conf( weight, Group, Value )
TTestw_conf( weight, Group, Value, sig, false )
```

See also:

 [Creating a typical t-test report \(page 504\)](#)

TTestw_df

TTestw_df() returns the aggregated student's t-test df value (degrees of freedom) for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_df (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_df( weight, Group, value )
TTestw_df( weight, Group, value, false )
```

See also:

 [Creating a typical t-test report \(page 504\)](#)

TTestw_dif

TTestw_dif() returns the aggregated student's t-test mean difference for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_dif (weight, grp, value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_dif( weight, Group, Value )
TTestw_dif( weight, Group, value, false )
```

See also:

□ [Creating a typical t-test report \(page 504\)](#)

TTestw_lower

TTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_lower (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_lower( weight, Group, value )
TTestw_lower( weight, Group, Value, sig, false )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTestw_sig

TTestw_sig() returns the aggregated student's t-test 2-tailed level of significance for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_sig ( weight, grp, value [, eq_var] )
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_sig( weight, Group, Value )
TTestw_sig( weight, Group, Value, false )
```

See also:

□ [Creating a typical t-test report \(page 504\)](#)

TTestw_sterr

TTestw_sterr() returns the aggregated student's t-test standard error of the mean difference for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_sterr (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_sterr( weight, Group, value )
TTestw_sterr( weight, Group, value, false )
```

See also:

 *Creating a typical t-test report (page 504)*

TTestw_t

TTestw_t() returns the aggregated t value for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ttestw_t (weight, grp, value [, eq_var])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_t( weight, Group, value )
TTestw_t( weight, Group, value, false )
```

See also:

□ [Creating a typical t-test report \(page 504\)](#)

TTestw_upper

TTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

This function applies to two independent samples student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTestw_upper (weight, grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTestw_upper( weight, Group, value )
TTestw_upper( weight, Group, Value, sig, false )
```

See also:

 *Creating a typical t-test report (page 504)*

TTest1_conf

TTest1_conf() returns the aggregated confidence interval value for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_conf (value [, sig ])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_conf( value )
TTest1_conf( value, 0.005 )
```

See also:

□ *Creating a typical t-test report (page 504)*

TTest1_df

TTest1_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_df (value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_df( value )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest1_dif

TTest1_dif() returns the aggregated student's t-test mean difference for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_dif (value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_dif( value )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest1_lower

TTest1_lower() returns the aggregated value for the lower end of the confidence interval for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_lower (value [, sig])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_lower( value )
TTest1_lower( value, 0.005 )
```

See also:

 *Creating a typical t-test report (page 504)*

TTest1_sig

TTest1_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_sig (value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_sig( value )
```

See also:

 *Creating a typical t-test report (page 504)*

TTest1_sterr

TTest1_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_sterr (value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_sterr( value )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest1_t

TTest1_t() returns the aggregated t value for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_t (value)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1_t( value )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

TTest1_upper

TTest1_upper() returns the aggregated value for the upper end of the confidence interval for a series of values.

This function applies to one-sample student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1_upper (value [, sig])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1_upper( value )
TTest1_upper( value, 0.005 )
```

See also:

- *Creating a typical t-test report (page 504)*

TTest1w_conf

TTest1w_conf() is a **numeric** function that returns the aggregated confidence interval value for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_conf (weight, value [, sig ])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1w_conf( weight, value )
TTest1w_conf( weight, value, 0.005 )
```

See also:

 *Creating a typical t-test report (page 504)*

TTest1w_df

TTest1w_df() returns the aggregated student's t-test df value (degrees of freedom) for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_df (weight, value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_df( weight, value )
```

See also:

 *Creating a typical t-test report (page 504)*

TTest1w_dif

TTest1w_dif() returns the aggregated student's t-test mean difference for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_dif (weight, value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_dif( weight, value )
```

See also:

□ [Creating a typical t-test report \(page 504\)](#)

TTest1w_lower

TTest1w_lower() returns the aggregated value for the lower end of the confidence interval for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_lower (weight, value [, sig ])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTest1w_lower( weight, value )
TTest1w_lower( weight, value, 0.005 )
```

See also:

□ *Creating a typical t-test report (page 504)*

TTest1w_sig

TTest1w_sig() returns the aggregated student's t-test 2-tailed level of significance for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_sig (weight, value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_sig( weight, value )
```

See also:

- *Creating a typical t-test report (page 504)*

TTest1w_sterr

TTest1w_sterr() returns the aggregated student's t-test standard error of the mean difference for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_sterr (weight, value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_sterr( weight, value )
```

See also:

□ [Creating a typical t-test report \(page 504\)](#)

TTest1w_t

TTest1w_t() returns the aggregated t value for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_t ( weight, value)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
TTest1w_t( weight, value )
```

See also:

- [Creating a typical t-test report \(page 504\)](#)

[TTest1w_upper](#)

TTest1w_upper() returns the aggregated value for the upper end of the confidence interval for a series of values.

This function applies to one-sample student's t-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
TTest1w_upper (weight, value [, sig])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The samples to be evaluated. If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
weight	Each value in value can be counted one or more times according to a corresponding weight value in weight .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
TTTest1w_upper( weight, value )
TTTest1w_upper( weight, value, 0.005 )
```

See also:

□ *Creating a typical t-test report (page 504)*

Z-test functions

A statistical examination of two population means. A two sample z-test examines whether two samples are different and is commonly used when two normal distributions have known variances and when an experiment uses a large sample size.

The z-test statistical test functions are grouped according the type of input data series that applies to the function.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Examples of how to use z-test functions (page 507)

One column format functions

The following functions apply to z-tests with simple input data series.

ztest_conf

ZTest_conf() returns the aggregated z value for a series of values.

```
ZTest_conf() returns the aggregated z value for a series of values. (value [, sigma [, sig ]])
```

ztest_dif

ZTest_dif() returns the aggregated z-test mean difference for a series of values.

```
ZTest_dif() returns the aggregated z-test mean difference for a series of values. (value [, sigma])
```

ztest_sig

ZTest_sig() returns the aggregated z-test 2-tailed level of significance for a series of values.

```
ZTest_sig() returns the aggregated z-test 2-tailed level of significance for a series of values. (value [, sigma])
```

ztest_sterr

ZTest_sterr() returns the aggregated z-test standard error of the mean difference for a series of values.

```
ZTest_sterr() returns the aggregated z-test standard error of the mean difference for a series of values. (value [, sigma])
```

ztest_z

ZTest_z() returns the aggregated z value for a series of values.

```
ZTest_z() returns the aggregated z value for a series of values. (value [, sigma])
```

ztest_lower

ZTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

```
ZTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values. (grp, value [, sig [, eq_var]])
```

ztest_upper

ZTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

```
ZTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values. (grp, value [, sig [, eq_var]])
```

Weighted two-column format functions

The following functions apply to z-tests where the input data series is given in weighted two-column format.

ztestw_conf

ZTestw_conf() returns the aggregated z confidence interval value for a series of values.

```
ZTestw_conf() returns the aggregated z confidence interval value for a series of values. (weight, value [, sigma [, sig]])
```

ztestw_dif

ZTestw_dif() returns the aggregated z-test mean difference for a series of values.

```
ZTestw_dif() returns the aggregated z-test mean difference for a series of values. (weight, value [, sigma])
```

ztestw_lower

ZTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

```
ZTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values. (weight, value [, sigma])
```

ztestw_sig

ZTestw_sig() returns the aggregated z-test 2-tailed level of significance for a series of values.

```
ZTestw_sig() returns the aggregated z-test 2-tailed level of significance for a series of values. (weight, value [, sigma])
```

ztestw_sterr

ZTestw_sterr() returns the aggregated z-test standard error of the mean difference for a series of values.

```
ZTestw_sterr() returns the aggregated z-test standard error of the mean difference for a series of values. (weight, value [, sigma])
```

ztestw_upper

ZTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

```
ZTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values. (weight, value [, sigma])
```

ztestw_z

ZTestw_z() returns the aggregated z value for a series of values.

```
ZTestw_z() returns the aggregated z value for a series of values. (weight, value [, sigma])
```

ZTest_z

ZTest_z() returns the aggregated z value for a series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_z(value[, sigma])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_z( value-Testvalue )
```

See also:

□ *Examples of how to use z-test functions (page 507)*

[ZTest_sig](#)

ZTest_sig() returns the aggregated z-test 2-tailed level of significance for a series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_sig(value[, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_sig(value-Testvalue)
```

See also:

□ *Examples of how to use z-test functions (page 507)*

ZTest_dif

ZTest_dif() returns the aggregated z-test mean difference for a series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_dif(value[, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_dif(value-Testvalue)
```

See also:

 [Examples of how to use z-test functions \(page 507\)](#)

ZTest_sterr

ZTest_sterr() returns the aggregated z-test standard error of the mean difference for a series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_sterr(value[, sigma])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_sterr(value-Testvalue)
```

See also:

 [Examples of how to use z-test functions \(page 507\)](#)

ZTest_conf

ZTest_conf() returns the aggregated z value for a series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_conf(value[, sigma[, sig]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTest_conf(value-Testvalue)
```

See also:

- Examples of how to use z-test functions (page 507)

ZTest_lower

ZTest_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_lower (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTest_lower( Group, value )  
ZTest_lower( Group, value, sig, false )
```

See also:

 *Examples of how to use z-test functions (page 507)*

ZTest_upper

ZTest_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTest_upper( Group, value )
ZTest_upper( Group, value, sig, false )
```

See also:

□ [Examples of how to use z-test functions \(page 507\)](#)

ZTestw_z

ZTestw_z() returns the aggregated z value for a series of values.

This function applies to z-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_z (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_z( weight, value-Testvalue)
```

See also:

□ *Examples of how to use z-test functions (page 507)*

ZTestw_sig

ZTestw_sig() returns the aggregated z-test 2-tailed level of significance for a series of values.

This function applies to z-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_sig (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_sig( weight, value-Testvalue)
```

See also:

□ *Examples of how to use z-test functions (page 507)*

ZTestw_dif

ZTestw_dif() returns the aggregated z-test mean difference for a series of values.

This function applies to z-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_dif ( weight, value [, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_dif( weight, value-Testvalue)
```

See also:

□ *Examples of how to use z-test functions (page 507)*

ZTestw_sterr

ZTestw_sterr() returns the aggregated z-test standard error of the mean difference for a series of values.

This function applies to z-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_sterr (weight, value [, sigma])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The values should be returned by value . A sample mean of 0 is assumed. If you want the test to be performed around another mean, subtract that value from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
ZTestw_sterr( weight, value-Testvalue)
```

See also:

□ [Examples of how to use z-test functions \(page 507\)](#)

ZTestw_conf

ZTestw_conf() returns the aggregated z confidence interval value for a series of values.

This function applies to z-tests where the input data series is given in weighted two-column format.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTest_conf(weight, value[, sigma[, sig]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. A population mean of 0 is assumed. If you want the test to be performed around another mean, subtract that mean from the sample values.
weight	Each sample value in value can be counted one or more times according to a corresponding weight value in weight .
sigma	If known, the standard deviation can be stated in sigma . If sigma is omitted the actual sample standard deviation will be used.
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Example:

```
zTestw_conf( weight, value-Testvalue)
```

See also:

 [Examples of how to use z-test functions \(page 507\)](#)

[ZTestw_lower](#)

ZTestw_lower() returns the aggregated value for the lower end of the confidence interval for two independent series of values.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_lower (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTestw_lower( Group, value )
ZTestw_lower( Group, value, sig, false )
```

See also:

 [Examples of how to use z-test functions \(page 507\)](#)

[ZTestw_upper](#)

ZTestw_upper() returns the aggregated value for the upper end of the confidence interval for two independent series of values.

This function applies to independent samples student's t-tests.

If the function is used in the data load script, the values are iterated over a number of records as defined by a group by clause.

If the function is used in a chart expression, the values are iterated over the chart dimensions.

Syntax:

```
ZTestw_upper (grp, value [, sig [, eq_var]])
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
value	The sample values to be evaluated. The sample values must be logically grouped as specified by exactly two values in group . If a field name for the sample values is not provided in the load script, the field will automatically be named Value .
grp	The field containing the names of each of the two sample groups. If a field name for the group is not provided in the load script, the field will automatically be given the name Type .
sig	The two-tailed level of significance can be specified in sig . If omitted, sig is set to 0.025, resulting in a 95% confidence interval.
eq_var	If eq_var is specified as False (0), separate variances of the two samples will be assumed. If eq_var is specified as True (1), equal variances between the samples will be assumed.

Limitations:

Text values, NULL values and missing values in the expression value will result in the function returning NULL.

Examples:

```
ZTestw_upper( Group, value )
ZTestw_upper( Group, value, sig, false )
```

See also:

 *Examples of how to use z-test functions (page 507)*

Statistical test function examples

This section includes examples of statistical test functions as applied to charts and the data load script.

Examples of how to use chi2-test functions in charts

The chi2-test functions are used to find values associated with chi squared statistical analysis.

This section describes how to build visualizations using sample data to find the values of the chi-squared distribution test functions available in Qlik Sense. Please refer to the individual chi2-test chart function topics for descriptions of syntax and arguments.

Loading the data for the samples

There are three sets of sample data describing three different statistical samples to be loaded into the script.

Do the following:

1. Create a new app.

2. In the data load, enter the following:

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the
top of the script.
Sample_1:
LOAD * inline [
Grp,Grade,Count
I,A,15
I,B,7
I,C,9
I,D,20
I,E,26
I,F,19
II,A,10
II,B,11
II,C,7
II,D,15
II,E,21
II,F,16
];
// Sample_2 data is pre-aggregated: If raw data is used, it must be aggregated using
count()...
Sample_2:
LOAD * inline [
Sex,Opinion,OpCount
1,2,58
1,1,11
1,0,10
2,2,35
2,1,25
2,0,23 ] (delimiter is ',');
// Sample_3a data is transformed using the crosstable statement...
Sample_3a:
crosstable(Gender, Actual) LOAD
Description,
[Men (Actual)] as Men,
[Women (Actual)] as Women;
LOAD * inline [
Men (Actual),Women (Actual),Description
58,35,Agree
11,25,Neutral
10,23,Disagree ] (delimiter is ',');
// Sample_3b data is transformed using the crosstable statement...
Sample_3b:
crosstable(Gender, Expected) LOAD
Description,
[Men (Expected)] as Men,
[Women (Expected)] as Women;
LOAD * inline [
Men (Expected),Women (Expected),Description
45.35,47.65,Agree
17.56,18.44,Neutral
16.09,16.91,Disagree ] (delimiter is ',');
// Sample_3a and Sample_3b will result in a (fairly harmless) synthetic key...
```

3. Click  to load data.

Creating the chi2-test chart function visualizations

Example: Sample 1

Do the following:

1. In the data load editor, click  to go to the app view and then click the sheet you created before.
The sheet view is opened.
2. Click  **Edit sheet** to edit the sheet.
3. From **Charts** add a table, and from **Fields** add Grp, Grade, and Count as dimensions.
This table shows the sample data.
4. Add another table with the following expression as a dimension:
`valueList('p', 'df', 'Chi2')`
This uses the synthetic dimensions function to create labels for the dimensions with the names of the three chi2-test functions.
5. Add the following expression to the table as a measure:
`IF(valueList('p', 'df', 'Chi2')='p', Chi2Test_p(Grp, Grade, Count),
IF(valueList('p', 'df', 'Chi2')='df', Chi2Test_df(Grp, Grade, Count),
Chi2Test_chi2(Grp, Grade, Count)))`
This has the effect of putting the resulting value of each chi2-test function in the table next to its associated synthetic dimension.
6. Set the **Number formatting** of the measure to **Number** and **3 Significant figures**.



In the expression for the measure, you could use the following expression instead: `Pick(Match(valueList('p', 'df', 'chi2'), 'p', 'df', 'chi2'), Chi2Test_p(Grp, Grade, Count), Chi2Test_df(Grp, Grade, Count), Chi2Test_Chisq(Grp, Grade, Count))`

Result:

The resulting table for the chi2-test functions for the Sample 1 data will contain the following values:

Results table

p	df	Chi2
0.820	5	2.21

Example: Sample 2

Do the following:

1. In the sheet you were editing in the example Sample 1, from **Charts** add a table, and from **Fields** add Sex, Opinion, and OpCount as dimensions.
2. Make a copy of the results table from Sample 1 using the **Copy** and **Paste** commands. Edit the expression in the measure and replace the arguments in all three chi2-test functions with the names of the fields used in the Sample 2 data, for example: `Chi2Test_p(Sex, Opinion, OpCount)`.

Result:

The resulting table for the chi2-test functions for the Sample 2 data will contain the following values:

Results table		
p	df	Chi2
0.000309	2	16.2

Example: Sample 3

Do the following:

1. Create two more tables in the same way as in the examples for Sample 1 and Sample 2 data. In the dimensions table, use the following fields as dimensions: Gender, Description, Actual, and Expected.
2. In the results table, use the names of the fields used in the Sample 3 data, for example: chi2Test_p(Gender,Description,Actual,Expected).

Result:

The resulting table for the chi2-test functions for the Sample 3 data will contain the following values:

Results table		
p	df	Chi2
0.000308	2	16.2

Examples of how to use chi2-test functions in the data load script

The chi2-test functions are used to find values associated with chi squared statistical analysis. This section describes how to use the chi-squared distribution test functions available in Qlik Sense in the data load script. Please refer to the individual chi2-test script function topics for descriptions of syntax and arguments.

This example uses a table containing the number of students achieving a grade (A-F) for two groups of students (I and II).

Data table

Group	A	B	C	D	E	F
I	15	7	9	20	26	19
II	10	11	7	15	21	16

Loading the sample data

Do the following:

1. Create a new app.
2. In the data load editor, enter the following:

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top of the script.  
Sample_1:
```

```
LOAD * inline [
    Grp,Grade,Count
    I,A,15
    I,B,7
    I,C,9
    I,D,20
    I,E,26
    I,F,19
    II,A,10
    II,B,11
    II,C,7
    II,D,15
    II,E,21
    II,F,16
];
```

3. Click  to load data.

You have now loaded the sample data.

Loading the chi2-test function values

Now we will load the chi2-test values based on the sample data in a new table, grouped by Grp.

Do the following:

1. In the data load editor, add the following at the end of the script:

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the
// top of the script.
Chi2_table:
LOAD Grp,
    Chi2Test_chi2(Grp, Grade, Count) as chi2,
    Chi2Test_df(Grp, Grade, Count) as df,
    Chi2Test_p(Grp, Grade, Count) as p
resident Sample_1 group by Grp;
```

2. Click  to load data.

You have now loaded the chi2-test values in a table named Chi2_table.

Results

You can view the resulting chi2-test values in the data model viewer under **Preview**, they should look like this:

Results

Grp	chi2	df	p
I	16.00	5	0.007
II	9.40	5	0.094

Creating a typical t-test report

A typical student t-test report can include tables with **Group Statistics** and **Independent Samples Test** results.

In the following sections we will build these tables using Qlik Sense-test functions applied to two independent groups of samples, Observation and Comparison. The corresponding tables for these samples would look like this:

Group statistics				
Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

Independent Sample Test

Independent Sample Test								
Type	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)	
Equal Variance not Assumed	3.534	37.11671733582 3	0.001	15.2	4.30101	6.48625	23.9137	
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069	

Loading the sample data

Do the following:

1. Create a new app with a new sheet and open that sheet.
2. Enter the following in the data load editor:

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2]
```

```
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

In this load script, **recno()** is included because **crosstable** requires three arguments. So, **recno()** simply provides an extra argument, in this case an ID for each row. Without it, **Comparison** sample values would not be loaded.

3. Click  to load data.

Creating the Group Statistics table

Do the following:

1. In the data load editor, click  to go to app view, and then click the sheet you created before. This opens the sheet view.
2. Click  **Edit sheet** to edit the sheet.
3. From **Charts**, add a table, and from **Fields**, add the following expressions as measures:

Example expressions

Label	Expression
N	Count(Value)
Mean	Avg(Value)
Standard Deviation	Stdev(Value)
Standard Error Mean	Sterr(Value)

4. Add Type as a dimension to the table.
5. Click **Sorting** and move Type to the top of the sorting list.

Result:

A Group Statistics table for these samples would look like this:

Group statistics

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

Creating the Two Independent Sample Student's T-test table

Do the following:

1. Click  **Edit sheet** to edit the sheet.
2. Add the following expression as a dimension to the table. =valueList (Dual('Equal variance not Assumed', 0), Dual('Equal variance Assumed', 1))

3. From **Charts** add a table with the following expressions as measures:

Example expressions

Label	Expression
conf	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_conf(Type, Value),TTest_conf(Type, Value, 0))
t	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_t(Type, Value),TTest_t(Type, Value, 0))
df	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_df(Type, Value),TTest_df(Type, Value, 0))
Sig. (2-tailed)	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sig(Type, Value),TTest_sig(Type, Value, 0))
Mean Difference	TTest_dif(Type, Value)
Standard Error Difference	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sterr(Type, Value),TTest_sterr(Type, Value, 0))
95% Confidence Interval of the Difference (Lower)	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_lower(Type, Value,(1-(95)/100)/2),TTest_lower(Type, Value,(1-(95)/100)/2, 0))
95% Confidence Interval of the Difference (Upper)	if(ValuePairList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_upper(Type, Value,(1-(95)/100)/2),TTest_upper(Type, Value,(1-(95)/100)/2, 0))

Result:

Independent sample test

Type	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.1167173358 23	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

Examples of how to use z-test functions

The z-test functions are used to find values associated with z-test statistical analysis for large data samples, usually greater than 30, and where the variance is known.

This section describes how to build visualizations using sample data to find the values of the z-test functions available in Qlik Sense. Please refer to the individual z-test chart function topics for descriptions of syntax and arguments.

Loading the sample data

The sample data used here is the same as that used in the t-test function examples. The sample data size would normally be considered too small for z-test analysis, but is sufficient for the purposes of illustrating the use of the different z-test functions in Qlik Sense.

Do the following:

1. Create a new app with a new sheet and open that sheet.



If you created an app for the t-test functions, you could use that and create a new sheet for these functions.

2. In the data load editor, enter the following:

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

In this load script, **recno()** is included because **crosstable** requires three arguments. So, **recno()** simply provides an extra argument, in this case an ID for each row. Without it, **Comparison** sample values would not be loaded.

3. Click to load data.

Creating z-test chart function visualizations

Do the following:

1. In the data load editor, click to go to app view, and then click the sheet you created when loading the data.
The sheet view is opened.

2. Click  **Edit sheet** to edit the sheet.
3. From **Charts** add a table, and from **Fields** add Type as a dimension.
4. Add the following expressions to the table as measures.

Example expressions

Label	Expression
ZTest Conf	ZTest_conf(Value)
ZTest Dif	ZTest_dif(Value)
ZTest Sig	ZTest_sig(Value)
ZTest Sterr	ZTest_sterr(Value)
ZTest Z	ZTest_z(Value)



You might wish to adjust the number formatting of the measures in order to see meaningful values. The table will be easier to read if you set number formatting on most of the measures to **Number>Simple**, instead of **Auto**. But for ZTest Sig, for example, use the number formatting: **Custom**, and then adjust the format pattern to # ##.

Result:

The resulting table for the z-test functions for the sample data will contain the following values:

Results table

Type	ZTest Conf	ZTest Dif	ZTest Sig	ZTest Sterr	ZTest Z
Comparison	6.40	11.95	0.000123	3.27	3.66
Value	5.48	27.15	0.001	2.80	9.71

Creating z-testw chart function visualizations

The z-testw functions are for use when the input data series occurs in weighted two-column format. The expressions require a value for the argument weight. The examples here use the value 2 throughout, but you could use an expression, which would define a value for weight for each observation.

Examples and results:

Using the same sample data and number formatting as for the z-test functions, the resulting table for the z-testw functions will contain the following values:

Results table

Type	ZTestw Conf	ZTestw Dif	ZTestw Sig	ZTestw Sterr	ZTestw Z
Comparison	3.53	2.95	5.27e-005	1.80	3.88
Value	2.97	34.25	0	4.52	20.49

String aggregation functions

This section describes string-related aggregation functions.

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

String aggregation functions in the data load script

Concat

Concat() is used to combine string values. The script function returns the aggregated string concatenation of all values of the expression iterated over a number of records as defined by a **group by** clause.

```
Concat ([ distinct ] expression [, delimiter [, sort-weight]])
```

FirstValue

FirstValue() returns the value that was loaded first from the records defined by the expression, sorted by a **group by** clause.



This function is only available as a script function.

```
FirstValue (expression)
```

LastValue

LastValue() returns the value that was loaded last from the records defined by the expression, sorted by a **group by** clause.



This function is only available as a script function.

```
LastValue (expression)
```

MaxString

MaxString() finds string values in the expression and returns the last text value sorted alphabetically over a number of records, as defined by a **group by** clause.

```
MaxString (expression )
```

MinString

MinString() finds string values in the expression and returns the first text value sorted alphabetically over a number of records, as defined by a **group by** clause.

```
MinString (expression )
```

String aggregation functions in charts

The following chart functions are available for aggregating strings in charts.

Concat

Concat() is used to combine string values. The function returns the aggregated string concatenation of all the values of the expression evaluated over each dimension.

```
Concat - chart function({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} string[, delimiter[, sort_weight]])
```

MaxString

MaxString() finds string values in the expression or field and returns the last text value in alphabetical sort order.

```
MaxString - chart function({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

MinString

MinString() finds string values in the expression or field and returns the first text value in alphabetical sort order.

```
MinString - chart function({[SetExpression] [TOTAL [<fld {, fld}>]]} expr)
```

Concat

Concat() is used to combine string values. The script function returns the aggregated string concatenation of all values of the expression iterated over a number of records as defined by a **group by** clause.

Syntax:

```
Concat ([ distinct ] string [, delimiter [, sort-weight]])
```

Return data type: string

Arguments:

The expression or field containing the string to be processed.

Arguments

Argument	Description
string	The expression or field containing the string to be processed.
delimiter	Each value may be separated by the string found in delimiter.
sort-weight	The order of concatenation may be determined by the value of the dimension sort-weight , if present, with the string corresponding to the lowest value appearing first in the concatenation.
distinct	If the word distinct occurs before the expression, all duplicates are disregarded.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Examples and results

Example	Result	Results once added to a sheet
<pre>TeamData: LOAD * inline [SalesGroup Team Date Amount East Gamma 01/05/2013 20000 East Gamma 02/05/2013 20000 West Zeta 01/06/2013 19000 East Alpha 01/07/2013 25000 East Delta 01/08/2013 14000 West Epsilon 01/09/2013 17000 West Eta 01/10/2013 14000 East Beta 01/11/2013 20000 West Theta 01/12/2013 23000] (delimiter is ' '); Concat1: LOAD SalesGroup,Concat(Team) as TeamConcat1 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	TeamConcat1 AlphaBetaDeltaGammaGamma EpsilonEtaThetaZeta
<p>Given that the TeamData table is loaded as in the previous example:</p> <pre>LOAD SalesGroup,Concat(distinct Team,'-') as TeamConcat2 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	TeamConcat2 Alpha-Beta-Delta-Gamma Epsilon-Eta-Theta-Zeta
<p>Given that the TeamData table is loaded as in the previous example. Because the argument for sort-weight is added, the results are ordered by the value of the dimension Amount:</p> <pre>LOAD SalesGroup,Concat(distinct Team,'-',Amount) as TeamConcat2 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	TeamConcat2 Delta-Beta-Gamma-Alpha Eta-Epsilon-Zeta-Theta

Concat - chart function

Concat() is used to combine string values. The function returns the aggregated string concatenation of all the values of the expression evaluated over each dimension.

Syntax:

```
Concat({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} string[, delimiter [, sort_weight]])
```

Return data type: string

Arguments:

Arguments	
Argument	Description
string	The expression or field containing the string to be processed.
delimiter	Each value may be separated by the string found in delimiter.
sort-weight	The order of concatenation may be determined by the value of the dimension sort-weight , if present, with the string corresponding to the lowest value appearing first in the concatenation.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the word DISTINCT occurs before the function arguments, duplicates resulting from the evaluation of the function arguments are disregarded.
TOTAL	<p>If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions.</p> <p>By using TOTAL [<fld {.fld}>], where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.</p>

Examples and results:

Results table

SalesGroup	Amount	Concat(Team)	Concat(TOTAL <SalesGroup> Team)
East	25000	Alpha	AlphaBetaDeltaGammaGamma
East	20000	BetaGammaGamma	AlphaBetaDeltaGammaGamma
East	14000	Delta	AlphaBetaDeltaGammaGamma
West	17000	Epsilon	EpsilonEtaThetaZeta
West	14000	Eta	EpsilonEtaThetaZeta
West	23000	Theta	EpsilonEtaThetaZeta
West	19000	Zeta	EpsilonEtaThetaZeta

Function examples

Example	Result
Concat(Team)	The table is constructed from the dimensions SalesGroup and Amount, and variations on the measure Concat(Team). Ignoring the Totals result, note that even though there is data for eight values of Team spread across two values of SalesGroup, the only result of the measure Concat(Team) that concatenates more than one Team string value in the table is the row containing the dimension Amount 20000, which gives the result BetaGammaGamma. This is because there are three values for the Amount 20000 in the input data. All other results remain unconcatenated when the measure is spanned across the dimensions because there is only one value of Team for each combination of SalesGroup and Amount.
Concat(DISTINCT Team, ', ')	Beta, Gamma. because the DISTINCT qualifier means the duplicate Gamma result is disregarded. Also, the delimiter argument is defined as a comma followed by a space.
Concat (TOTAL <SalesGroup> Team)	All the string values for all values of Team are concatenated if the TOTAL qualifier is used. With the field selection <SalesGroup> specified, this divides the results into the two values of the dimension SalesGroup. For the SalesGroupEast, the results are AlphaBetaDeltaGammaGamma. For the SalesGroupWest, the results are EpsilonEtaThetaZeta.
Concat (TOTAL <SalesGroup> Team, ';' , Amount)	By adding the argument for sort-weight : Amount, the results are ordered by the value of the dimension Amount. The results becomes DeltaBetaGammaGammaAlpha and EtaEpsilonZEtaTheta.

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

FirstValue

FirstValue() returns the value that was loaded first from the records defined by the expression, sorted by a **group by** clause.



This function is only available as a script function.

Syntax:

```
FirstValue ( expr)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Limitations:

If no text value is found, NULL is returned.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data

Example	Result	Results on a sheet
TeamData: LOAD * inline [SalesGroup Team Date Amount East Gamma 01/05/2013 20000 East Gamma 02/05/2013 20000 West Zeta 01/06/2013 19000 East Alpha 01/07/2013 25000 East Delta 01/08/2013 14000 West Epsilon 01/09/2013 17000 West Eta 01/10/2013 14000 East Beta 01/11/2013 20000 West Theta 01/12/2013 23000] (delimiter is ' ');	SalesGroup East West	FirstTeamLoaded Gamma Zeta
FirstValue1: LOAD SalesGroup,Firstvalue(Team) as FirstTeamLoaded Resident TeamData Group By SalesGroup;		

LastValue

LastValue() returns the value that was loaded last from the records defined by the expression, sorted by a **group by** clause.



This function is only available as a script function.

Syntax:

```
LastValue ( expr )
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.

Limitations:

If no text value is found, NULL is returned.

Examples and results:

Add the example script to your app and run it. Then add, at least, the fields listed in the results column to a sheet in our app to see the result.

To get the same look as in the result column below, in the properties panel, under Sorting, switch from Auto to Custom, then deselect numerical and alphabetical sorting.

Example	Result	Result with custom sorting
TeamData: LOAD * inline [SalesGroup Team Date Amount East Gamma 01/05/2013 20000 East Gamma 02/05/2013 20000 West Zeta 01/06/2013 19000 East Alpha 01/07/2013 25000 East Delta 01/08/2013 14000 West Epsilon 01/09/2013 17000 West Eta 01/10/2013 14000 East Beta 01/11/2013 20000 West Theta 01/12/2013 23000] (delimiter is ' ');	SalesGroup East West	LastTeamLoaded Beta Theta
LastValue1: LOAD SalesGroup,Lastvalue(Team) as LastTeamLoaded Resident TeamData Group By SalesGroup;		

MaxString

MaxString() finds string values in the expression and returns the last text value sorted alphabetically over a number of records, as defined by a **group by** clause.

Syntax:

```
MaxString ( expr )
```

Return data type: dual

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.

Limitations:

If no text value is found, NULL is returned.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Example	Result	
<pre>TeamData: LOAD * inline [SalesGroup Team Date Amount East Gamma 01/05/2013 20000 East Gamma 02/05/2013 20000 West Zeta 01/06/2013 19000 East Alpha 01/07/2013 25000 East Delta 01/08/2013 14000 West Epsilon 01/09/2013 17000 West Eta 01/10/2013 14000 East Beta 01/11/2013 20000 West Theta 01/12/2013 23000] (delimiter is ' ');</pre>	SalesGroup East West	MaxString1 Gamma Zeta
<pre>Concat1: LOAD SalesGroup,MaxString(Team) as MaxString1 Resident TeamData Group By SalesGroup;</pre>		
<pre>Given that the TeamData table is loaded as in the previous example, and your data load script has the SET statement: SET DateFormat='DD/MM/YYYY';' LOAD SalesGroup,MaxString(Date) as MaxString2 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	MaxString2 01/11/2013 01/12/2013

MaxString - chart function

MaxString() finds string values in the expression or field and returns the last text value in alphabetical sort order.

Syntax:

```
MaxString({ [SetExpression] [TOTAL <fld{, fld}>] } expr)
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld}>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Limitations:

If the expression contains no values with a string representation NULL is returned.

Examples and results:

Results table			
SalesGroup	Amount	MaxString(Team)	MaxString(Date)
East	14000	Delta	2013/08/01
East	20000	Gamma	2013/11/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01

Function examples

Example	Result
MaxString(Team)	There are three values of 20000 for the dimension Amount: two of Gamma (on different dates), and one of Beta. The result of the measure MaxString (Team) is therefore Gamma, because this is the highest value in the sorted strings.

Example	Result
MaxString (Date)	2013/11/01 is the greatest Date value of the three associated with the dimension Amount. This assumes your script has the SET statement SET DateFormat='YYYY-MM-DD';'

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

MinString

MinString() finds string values in the expression and returns the first text value sorted alphabetically over a number of records, as defined by a **group by** clause.

Syntax:

```
MinString ( expr )
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Limitations:

If no text value is found, NULL is returned.

Examples and results:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Resulting data			
Example	Result		
<pre>TeamData: LOAD * inline [SalesGroup Team Date Amount East Gamma 01/05/2013 20000 East Gamma 02/05/2013 20000 West Zeta 01/06/2013 19000 East Alpha 01/07/2013 25000 East Delta 01/08/2013 14000 West Epsilon 01/09/2013 17000 West Eta 01/10/2013 14000 East Beta 01/11/2013 20000 West Theta 01/12/2013 23000] (delimiter is ' ');</pre> <pre>Concat1: LOAD SalesGroup,MinString(Team) as MinString1 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	MinString1 Alpha Epsilon	
<p>Given that the TeamData table is loaded as in the previous example, and your data load script has the SET statement:</p> <pre>SET DateFormat='DD/MM/YYYY';:</pre> <pre>LOAD SalesGroup,Minstring(Date) as MinString2 Resident TeamData Group By SalesGroup;</pre>	SalesGroup East West	MinString2 01/05/2013 01/06/2013	

MinString - chart function

MinString() finds string values in the expression or field and returns the first text value in alphabetical sort order.

Syntax:

```
MinString({{SetExpression} [TOTAL [<fld {, fld}>]} } expr)
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.

Argument	Description
TOTAL	If the word TOTAL occurs before the function arguments, the calculation is made over all possible values given the current selections, and not just those that pertain to the current dimensional value, that is, it disregards the chart dimensions. By using TOTAL [<fld {.fld>] , where the TOTAL qualifier is followed by a list of one or more field names as a subset of the chart dimension variables, you create a subset of the total possible values.

Examples and results:

Sample data

SalesGroup	Amount	MinString(Team)	MinString(Date)
East	14000	Delta	2013/08/01
East	20000	Beta	2013/05/01
East	25000	Alpha	2013/07/01
West	14000	Eta	2013/10/01
West	17000	Epsilon	2013/09/01
West	19000	Zeta	2013/06/01
West	23000	Theta	2013/12/01

Function examples

Examples	Results
MinString (Team)	There are three values of 20000 for the dimension Amount: two of Gamma (on different dates), and one of Beta. The result of the measure MinString (Team) is therefore Beta, because this is the first value in the sorted strings.
MinString (Date)	2013/11/01 is the earliest Date value of the three associated with the dimension Amount. This assumes your script has the SET statement <code>SET DateFormat='YYYY-MM-DD';</code>

Data used in example:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

Synthetic dimension functions

A synthetic dimension is created in the app from values generated from the synthetic dimension functions and not directly from fields in the data model. When values generated by a synthetic dimension function are used in a chart as a calculated dimension, this creates a synthetic dimension. Synthetic dimensions allow you to create, for example, charts with dimensions with values arising from your data, that is, dynamic dimensions.



Synthetic dimensions are not affected by selections.

The following synthetic dimension functions can be used in charts.

ValueList

ValueList() returns a set of listed values, which, when used in a calculated dimension, will form a synthetic dimension.

ValueList - chart function (v1 {, Expression})

ValueLoop

ValueLoop() returns a set of iterated values which, when used in a calculated dimension, will form a synthetic dimension.

ValueLoop - chart function(from [, to [, step]])

ValueList - chart function

ValueList() returns a set of listed values, which, when used in a calculated dimension, will form a synthetic dimension.



*In charts with a synthetic dimension created with the **ValueList** function it is possible to reference the dimension value corresponding to a specific expression cell by restating the **ValueList** function with the same parameters in the chart expression. The function may of course be used anywhere in the layout, but apart from when used for synthetic dimensions it will only be meaningful inside an aggregation function.*



Synthetic dimensions are not affected by selections.

Syntax:

ValueList(v1 {, . . . })

Return data type: dual

Arguments:

Arguments	
Argument	Description
v1	Static value (usually a string, but can be a number).
{...}	Optional list of static values.

Examples and results:

Function examples																																	
Example	Result																																
<pre>valueList ('Number of orders', 'Average Order Size', 'Total Amount')</pre> <pre>=IF(ValueList ('Number of orders', 'Average Order Size', 'Total Amount') = 'Number of orders', count (saleID), IF(ValueList ('Number of orders', 'Average Order Size', 'Total Amount') = 'Average Order Size', avg (Amount), sum (Amount)))</pre>	<p>When used to create a dimension in a table, for example, this results in the three string values as row labels in the table. These can then be referenced in an expression.</p> <p>This expression takes the values from the created dimension and references them in a nested IF statement as input to three aggregation functions:</p> <table border="1"> <thead> <tr> <th>ValueList()</th> <th>Created dimension</th> <th>Year</th> <th>Added expression</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td>522.00</td> </tr> <tr> <td>Number of Orders</td> <td></td> <td>2012</td> <td>5.00</td> </tr> <tr> <td>Number of Orders</td> <td></td> <td>2013</td> <td>7.00</td> </tr> <tr> <td>Average Order Size</td> <td></td> <td>2012</td> <td>13.20</td> </tr> <tr> <td>Average Order Size</td> <td></td> <td>2013</td> <td>15.43</td> </tr> <tr> <td>Total Amount</td> <td></td> <td>2012</td> <td>66.00</td> </tr> <tr> <td>Total Amount</td> <td></td> <td>2013</td> <td>108.00</td> </tr> </tbody> </table>	ValueList()	Created dimension	Year	Added expression				522.00	Number of Orders		2012	5.00	Number of Orders		2013	7.00	Average Order Size		2012	13.20	Average Order Size		2013	15.43	Total Amount		2012	66.00	Total Amount		2013	108.00
ValueList()	Created dimension	Year	Added expression																														
			522.00																														
Number of Orders		2012	5.00																														
Number of Orders		2013	7.00																														
Average Order Size		2012	13.20																														
Average Order Size		2013	15.43																														
Total Amount		2012	66.00																														
Total Amount		2013	108.00																														

Data used in examples:

SalesPeople:

```
LOAD * INLINE [
SaleID|SalesPerson|Amount|Year
1|1|12|2013
2|1|23|2013
3|1|17|2013
4|2|9|2013
5|2|14|2013
6|2|29|2013
```

```
7|2|4|2013  
8|1|15|2012  
9|1|16|2012  
10|2|11|2012  
11|2|17|2012  
12|2|7|2012  
] (delimiter is '|');
```

ValueLoop - chart function

ValueLoop() returns a set of iterated values which, when used in a calculated dimension, will form a synthetic dimension.

The values generated will start with the **from** value and end with the **to** value including intermediate values in increments of step.



*In charts with a synthetic dimension created with the **ValueLoop** function it is possible to reference the dimension value corresponding to a specific expression cell by restating the **ValueLoop** function with the same parameters in the chart expression. The function may of course be used anywhere in the layout, but apart from when used for synthetic dimensions it will only be meaningful inside an aggregation function.*



Synthetic dimensions are not affected by selections.

Syntax:

```
ValueLoop(from [, to [, step ]])
```

Return data type: dual

Arguments:

Arguments

Arguments	Description
from	Start value in the set of values to be generated.
to	End value in the set of values to be generated.
step	Size of increment between values.

Examples and results:

Function examples

Example	Result
ValueLoop(1, 10)	This creates a dimension in a table, for example, that can be used for purposes such as numbered labeling. The example here results in values numbered 1 to 10. These values can then be referenced in an expression.
ValueLoop(2, 10, 2)	This example results in values numbered 2, 4, 6, 8, and 10 because the argument step has a value of 2.

Nested aggregations

You may come across situations where you need to apply an aggregation to the result of another aggregation. This is referred to as nesting aggregations.

You cannot nest aggregations in most chart expressions. You can, however, nest aggregations if you use the **TOTAL** qualifier in the inner aggregation function.



No more than 100 levels of nesting is allowed.

Nested aggregations with the TOTAL qualifier

Example:

You want to calculate the sum of the field **Sales**, but only include transactions with an **OrderDate** equal to the last year. The last year can be obtained via the aggregation function **Max(TOTAL Year(OrderDate))**.

The following aggregation would return the desired result:

```
Sum(If(Year(OrderDate)=Max(TOTAL Year(OrderDate)), Sales))
```

Qlik Sense requires the inclusion of the **TOTAL** qualifier this type of nesting. It is necessary for the desired comparison. This type of nesting need is quite common and is a good practice.

See also:

- [Aggr - chart function \(page 525\)](#)

5.3 Aggr - chart function

Aggr() returns an array of values for the expression calculated over the stated dimension or dimensions. For example, the maximum value of sales, per customer, per region.

The **Aggr** function is used for nested aggregations, in which its first parameter (the inner aggregation) is calculated once per dimensional value. The dimensions are specified in the second parameter (and subsequent parameters).

In addition, the **Aggr** function should be enclosed in an outer aggregation function, using the array of results from the **Aggr** function as input to the aggregation in which it is nested.

Syntax:

```
Aggr({SetExpression} [DISTINCT] [NODISTINCT] expr, StructuredParameter{, StructuredParameter})
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	An expression consisting of an aggregation function. By default, the aggregation function will aggregate over the set of possible records defined by the selection.
StructuredParameter	<p>StructuredParameter consists of a dimension and optionally, sorting criteria in the format: (Dimension(sort-type, ordering))</p> <p>The dimension is a single field and cannot be an expression. The dimension is used to determine the array of values the Aggr expression is calculated for.</p> <p>If sorting criteria are included, the array of values created by the Aggr function, calculated for the dimension, is sorted. This is important when the sort order affects the result of the expression the Aggr function is enclosed in.</p> <p>For details of how to use sorting criteria, see Adding sorting criteria to the dimension in the structured parameter.</p>
SetExpression	By default, the aggregation function will aggregate over the set of possible records defined by the selection. An alternative set of records can be defined by a set analysis expression.
DISTINCT	If the expression argument is preceded by the distinct qualifier or if no qualifier is used at all, each distinct combination of dimension values will generate only one return value. This is the normal way aggregations are made – each distinct combination of dimension values will render one line in the chart.
NODISTINCT	If the expression argument is preceded by the nodistinct qualifier, each combination of dimension values may generate more than one return value, depending on underlying data structure. If there is only one dimension, the aggr function will return an array with the same number of elements as there are rows in the source data.

Basic aggregation functions, such as **Sum**, **Min**, and **Avg**, return a single numerical value, whereas the **Aggr()** function can be compared to creating a temporary staged result set (a virtual table), over which another aggregation can be made. For example, by computing an average sales value by summing the sales by customer in an **Aggr()** statement, and then calculating the average of the summed results: **Avg(TOTAL Aggr (Sum(Sales),Customer))**.



Use the Aggr() function in calculated dimensions if you want to create nested chart aggregations on multiple levels.

Limitations:

Each dimension in an Aggr() function must be a single field, and cannot be an expression (calculated dimension).

Adding sorting criteria to the dimension in the structured parameter

In its basic form, the argument StructuredParameter in the Aggr function syntax is a single dimension. The expression: Aggr(Sum(Sales, Month)) finds the total value of sales for each month. However, when enclosed in another aggregation function, there can be unexpected results unless sorting criteria are used. This is because some dimensions can be sorted numerically or alphabetically, and so on.

In the StructuredParameter argument in the Aggr function, you can specify sorting criteria on the dimension in your expression. This way, you impose a sort order on the virtual table that is produced by the Aggr function.

The argument StructuredParameter has the following syntax:

```
(FieldName, (Sort-type, Ordering))
```

Structured parameters can be nested:

```
(FieldName, (FieldName2, (Sort-type, Ordering)))
```

Sort-type can be: NUMERIC, TEXT, FREQUENCY, or LOAD_ORDER.

The Ordering types associated with each Sort-type are as follows:

Allowed ordering types	
Sort-type	Allowed Ordering types
NUMERIC	ASCENDING, DESCENDING, or REVERSE
TEXT	ASCENDING, A2Z, DESCENDING, REVERSE, or Z2A
FREQUENCY	DESCENDING, REVERSE or ASCENDING
LOAD_ORDER	ASCENDING, ORIGINAL, DESCENDING, or REVERSE

The ordering types REVERSE and DESCENDING are equivalent.

For Sort-type TEXT, the ordering types ASCENDING and A2Z are equivalent, and DESCENDING, REVERSE, and Z2A are equivalent.

For Sort-type LOAD_ORDER, the ordering types ASCENDING and ORIGINAL are equivalent.

Examples: Chart expressions using Aggr

Examples - chart expressions

Chart expression example 1

Load script

Load the following data as an inline load in the data load editor to create the chart expression example below.

ProductData:

```
LOAD * inline [
Customer|Product|unitsales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|25|25
Canutility|AA|8|15
Canutility|CC|0|19
] (delimiter is '|');
```

Chart expression

Create a KPI visualization in a Qlik Sense sheet. Add the following expression to the KPI, as a measure:

```
Avg(Aggr(Sum(unitsales*UnitPrice), customer))
```

Result

376.7

Explanation

The expression `Aggr(Sum(unitsales*UnitPrice), customer)` finds the total value of sales by **Customer**, and returns an array of values: 295, 715, and 120 for the three **Customer** values.

Effectively, we have built a temporary list of values without having to create an explicit table or column containing those values.

These values are used as input to the **Avg()** function to find the average value of sales, 376.7.

Chart expression example 2

Load script

Load the following data as an inline load in the data load editor to create the chart expression example below.

ProductData:

```
LOAD * inline [
Customer|Product|unitsales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
```

```
Astrida|BB|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|BB|7|12
Betacab|CC|2|22
Betacab|CC|4|20
Betacab|DD|25|25
Canutility|AA|8|15
Canutility|AA|5|11
Canutility|CC|0|19
] (delimiter is '|');
```

Chart expression

Create a table visualization in a Qlik Sense sheet with **Customer**, **Product**, **UnitPrice**, and **UnitSales** as dimensions. Add the following expression to the table, as a measure:

```
Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
```

Result

Customer	Product	UnitPrice	UnitSales	Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
Astrida	AA	15	10	16
Astrida	AA	16	4	16
Astrida	BB	9	9	15
Astrida	BB	15	10	15
Betacab	BB	10	5	12
Betacab	BB	12	7	12
Betacab	CC	20	4	22
Betacab	CC	22	2	22
Betacab	DD	25	25	25
Canutility	AA	11	5	15
Canutility	AA	15	8	15
Canutility	CC	19	0	19

Explanation

An array of values: 16, 16, 15, 15, 12, 12, 22, 22, 25, 15, 15, and 19. The **nodistinct** qualifier means that the array contains one element for each row in the source data: each is the maximum **UnitPrice** for each **Customer** and **Product**.

Chart expression example 3

Load script

Load the following data as an inline load in the data load editor to create the chart expression example below.

```
Set vNumberoforders = 1000;

OrderLines:
Load
    RowNo() as OrderLineID,
    OrderID,
    OrderDate,
    Round((Year(OrderDate)-2005)*1000*Rand()*Rand()*Rand1) as Sales
    while Rand()<=0.5 or IterNo()=1;
Load * where OrderDate<=Today();
Load
    Rand() as Rand1,
    Date(MakeDate(2013)+Floor((365*4+1)*Rand())) as OrderDate,
    RecNo() as OrderID
    Autogenerate vNumberoforders;
```

Calendar:

```
Load distinct
    Year(OrderDate) as Year,
    Month(OrderDate) as Month,
    OrderDate
    Resident OrderLines;
```

Chart expressions

Create a table visualization in a Qlik Sense sheet with **Year** and **Month** as dimensions. Add the following expressions to the table as measures:

- Sum(Sales)
- Sum(Agg(Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending)))) labeled as Structured Aggr() in the table.

Result

Year	Month	Sum(Sales)	Structured Aggr()
2013	Jan	53495	53495
2013	Feb	48580	102075
2013	Mar	25651	127726
2013	Apr	36585	164311
2013	May	61211	225522
2013	Jun	23689	249211

Year	Month	Sum(Sales)	Structured Aggr()
2013	Jul	42311	291522
2013	Aug	41913	333435
2013	Sep	28886	362361
2013	Oct	25977	388298
2013	Nov	44455	432753
2013	Dec	64144	496897
2014	Jan	67775	67775

Explanation

This example displays the aggregated values over a twelve month period for each year in chronological ascending order, hence the structured parameters (Numeric, Ascending) part of the **Aggr()** expression. Two specific dimensions are required as structured parameters: **Year** and **Month**, sorted (1) **Year** (numeric) and (2) **Month** (numeric). These two dimensions must be used in the table or chart visualization. This is necessary for the dimension list of the **Aggr()** function to correspond with the dimensions of the object used in the visualization.

You can compare the difference between these measures in a table or in separate line charts:

- `Sum(Aggr(Rangesum(Above(Sum(Sales),0,12)), (Year), (Month)))`
- `Sum(Aggr(Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending))))`

It should be clear to see that only the latter expression performs the desired accumulation of aggregated values.

See also:

 [Basic aggregation functions \(page 316\)](#)

5.4 Color functions

These functions can be used in expressions associated with setting and evaluating the color properties of chart objects, as well as in data load scripts.



*Qlik Sense supports the color functions **Color()**, **qliktechblue**, and **qliktechgray** for backwards compatibility reasons, but use of them is not recommended.*

ARGB

ARGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**, with an alpha factor (opacity) of **alpha**.

```
ARGB(alpha, r, g, b)
```

HSL

HSL() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by values of **hue**, **saturation**, and **luminosity** between 0 and 1.

```
HSL (hue, saturation, luminosity)
```

RGB

RGB() returns an integer corresponding to the color code of the color defined by the three parameters: the red component r, the green component g, and the blue component b. These components must have integer values between 0 and 255. The function can be used in expressions to set or evaluate the color properties of a chart object.

```
RGB (r, g, b)
```

Colormix1

Colormix1() is used in expressions to return an ARGB color representation from a two color gradient, based on a value between 0 and 1.

```
Colormix1 (Value , ColorZero , ColorOne)
```

Value is a real number between 0 and 1.

- If Value = 0 ColorZero is returned.
- If Value = 1 ColorOne is returned.
- If $0 < \text{Value} < 1$ the appropriate intermediate shading is returned.

ColorZero is a valid RGB color representation for the color to be associated with the low end of the interval.

ColorOne is a valid RGB color representation for the color to be associated with the high end of the interval.

Example:

```
colormix1(0.5, red(), blue())  
returns:
```

ARGB(255,64,0,64) (purple)

Colormix2

Colormix2() is used in expressions to return an ARGB color representation from a two color gradient, based on a value between -1 and 1, with the possibility to specify an intermediate color for the center (0) position.

```
Colormix2 (Value ,ColorMinusOne , ColorOne[ , ColorZero])
```

Value is a real number between -1 and 1.

- If Value = -1 the first color is returned.
- If Value = 1 the second color is returned.
- If $-1 < \text{Value} < 1$ the appropriate color mix is returned.

ColorMinusOne is a valid RGB color representation for the color to be associated with the low end of the interval.

ColorOne is a valid RGB color representation for the color to be associated with the high end of the interval.

ColorZero is an optional valid RGB color representation for the color to be associated with the center of the interval.

SysColor

SysColor() returns the ARGB color representation for the Windows system color nr, where nr corresponds to the parameter to the Windows API function **GetSysColor(nr)**.

SysColor (nr)

ColorMapHue

ColorMapHue() returns an ARGB value of a color from a colormap that varies the hue component of the HSV color model. The colormap starts with red, passes through yellow, green, cyan, blue, magenta, and returns to red. x must be specified as a value between 0 and 1.

ColorMapHue (x)

ColorMapJet

ColorMapJet() returns an ARGB value of a color from a colormap that starts with blue, passes through cyan, yellow and orange, and returns to red. x must be specified as a value between 0 and 1.

ColorMapJet (x)

Pre-defined color functions

The following functions can be used in expressions for pre-defined colors. Each function returns an RGB color representation.

Optionally a parameter for alpha factor can be given, in which case an ARGB color representation is returned. An alpha factor of 0 corresponds to full transparency, and an alpha factor of 255 corresponds to full opacity. If a value for alpha is not entered, it is assumed to be 255.

Pre-defined color functions

Color function	RGB value
black ([alpha])	(0,0,0)
blue([alpha])	(0,0,128)
brown([alpha])	(128,128,0)
cyan([alpha])	(0,128,128)
darkgray([alpha])	(128,128,128)

green([alpha])	(0,128,0)
lightblue([alpha])	(0,0,255)
lightcyan([alpha])	(0,255,255)
lightgray([alpha])	(192,192,192)
lightgreen([alpha])	(0,255,0)
lightmagenta([alpha])	(255,0,255)
lightred([alpha])	(255,0,0)
magenta([alpha])	(128,0,128)
red([alpha])	(128,0,0)
white([alpha])	(255,255,255)
yellow([alpha])	(255,255,0)

Examples and results:

Examples and results

Examples	Results
Blue()	RGB(0,0,128)
Blue(128)	ARGB(128,0,0,128)

ARGB

ARGB() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by a red component **r**, a green component **g**, and a blue component **b**, with an alpha factor (opacity) of **alpha**.

Syntax:

```
ARGB(alpha, r, g, b)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
alpha	Transparency value in the range 0 - 255. 0 corresponds to full transparency and 255 corresponds to full opacity.
r, g, b	Red, green, and blue component values. A color component of 0 corresponds to no contribution and one of 255 to full contribution.



All arguments must be expressions that resolve to integers in the range 0 to 255.

If interpreting the numeric component and formatting it in hexadecimal notation, the values of the color components are easier to see. For example, light green has the number 4 278 255 360, which in hexadecimal notation is FF00FF00. The first two positions ‘FF’ (255) denote the **alpha** channel. The next two positions ‘00’ denote the amount of **red**, the next two positions ‘FF’ denote the amount of **green**, and the final two positions ‘00’ denote the amount of **blue**.

RGB

RGB() returns an integer corresponding to the color code of the color defined by the three parameters: the red component r, the green component g, and the blue component b. These components must have integer values between 0 and 255. The function can be used in expressions to set or evaluate the color properties of a chart object.

Syntax:

```
RGB (r, g, b)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
r, g, b	Red, green, and blue component values. A color component of 0 corresponds to no contribution and one of 255 to full contribution.



All arguments must be expressions that resolve to integers in the range 0 to 255.

If interpreting the numeric component and formatting it in hexadecimal notation, the values of the color components are easier to see. For example, light green has the number 4 278 255 360, which in hexadecimal notation is FF00FF00. The first two positions ‘FF’ (255) denote the **alpha** channel. In the functions **RGB** and **HSL**, this is always ‘FF’ (opaque). The next two positions ‘00’ denote the amount of **red**, the next two positions ‘FF’ denote the amount of **green**, and the final two positions ‘00’ denote the amount of **blue**.

Example: Chart expression

This example applies a custom color to a chart:

Data used in this example:

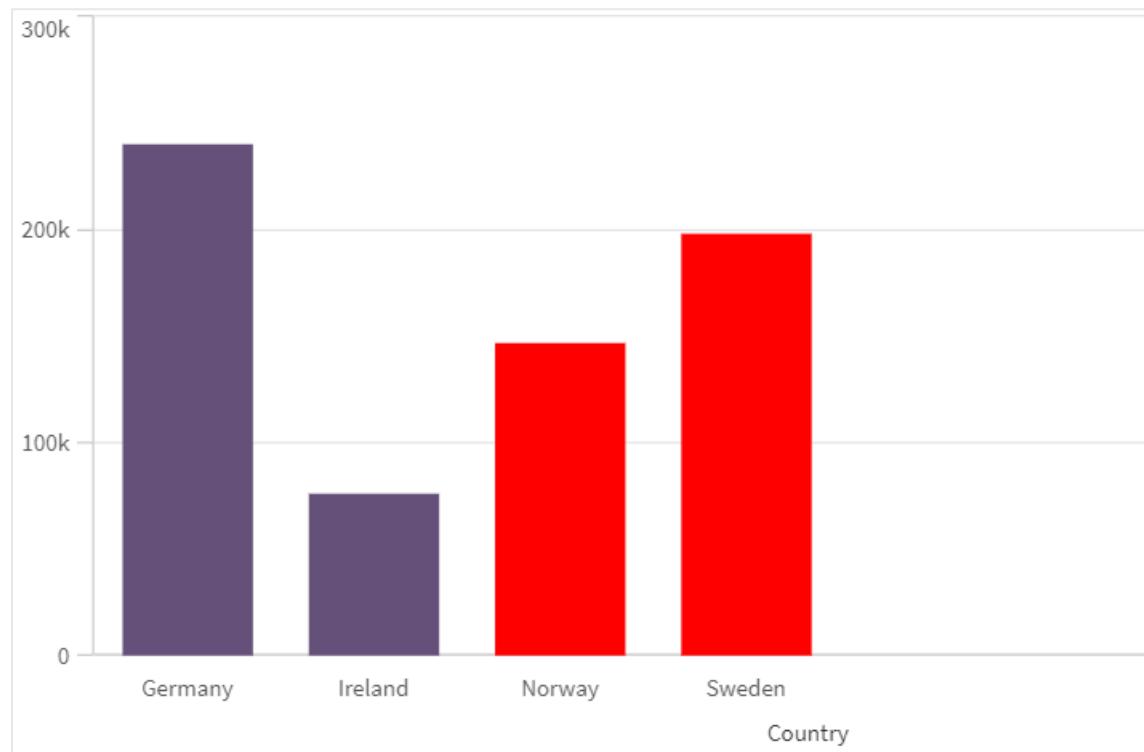
```
ProductSales:  
Load * Inline  
[Country,Sales,Budget  
Sweden,100000,50000  
Germany, 125000, 175000  
Norway, 74850, 68500
```

```
Ireland, 45000, 48000  
Sweden,98000,50000  
Germany, 115000, 175000  
Norway, 71850, 68500  
Ireland, 31000, 48000  
] (delimiter is ',');
```

Enter the following expression in the **Colors and legend** properties panel:

```
If (sum(Sales)>Sum(Budget),RGB(255,0,0),RGB(100,80,120))
```

Result:



Example: Load script

The following example displays the equivalent RGB values for values in hex format:

```
Load  
Text(R & G & B) as Text,  
RGB(R,G,B)      as Color;  
Load  
Num#(R,'(HEX)') as R,  
Num#(G,'(HEX)') as G,  
Num#(B,'(HEX)') as B  
Inline  
[R,G,B  
01,02,03  
AA,BB,CC];  
Result:
```

Text	Color
010203	RGB(1,2,3)
AABBCC	RGB(170,187,204)

HSL

HSL() is used in expressions to set or evaluate the color properties of a chart object, where the color is defined by values of **hue**, **saturation**, and **luminosity** between 0 and 1.

Syntax:

```
HSL (hue, saturation, luminosity)
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
hue, saturation, luminosity	hue, saturation, and luminosity component values ranging between 0 and 1.



All arguments must be expressions that resolve to integers in the range 0 to 1.

If interpreting the numeric component and formatting it in hexadecimal notation, the RGB values of the color components are easier to see. For example, light green has the number 4 278 255 360, which in hexadecimal notation is FF00FF00 and RGB (0,255,0). This is equivalent to HSL (80/240, 240/240, 120/240) - a HSL value of (0.33, 1, 0.5).

5.5 Conditional functions

The conditional functions all evaluate a condition and then return different answers depending on the condition value. The functions can be used in the data load script and in chart expressions.

Conditional functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

alt

The **alt** function returns the first of the parameters that has a valid number representation. If no such match is found, the last parameter will be returned. Any number of parameters can be used.

```
alt (expr1[ , expr2 , expr3 , ...] , else)
```

class

The **class** function assigns the first parameter to a class interval. The result is a dual value with $a \leq x < b$ as the textual value, where a and b are the upper and lower limits of the bin, and the lower bound as numeric value.

```
class (expression, interval [ , label [ , offset ]])
```

coalesce

The **coalesce** function returns the first of the parameters that has a valid non-NULL representation. Any number of parameters can be used.

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

if

The **if** function returns a value depending on whether the condition provided with the function evaluates as True or False.

```
if (condition , then , else)
```

match

The **match** function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case sensitive.

```
match ( str, expr1 [ , expr2,...exprN ])
```

mixmatch

The **mixmatch** function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case insensitive.

```
mixmatch ( str, expr1 [ , expr2,...exprN ])
```

pick

The pick function returns the n :th expression in the list.

```
pick (n, expr1[ , expr2,...exprN])
```

wildmatch

The **wildmatch** function compares the first parameter with all the following ones and returns the number of the expression that matches. It permits the use of wildcard characters (* and ?) in the comparison strings. * matches any sequence of characters. ? matches any single character. The comparison is case insensitive.

```
wildmatch ( str, expr1 [ , expr2,...exprN ])
```

alt

The **alt** function returns the first of the parameters that has a valid number representation. If no such match is found, the last parameter will be returned. Any number of parameters can be used.

Syntax:

```
alt(expr1[ , expr2 , expr3 , ...] , else)
```

Arguments:

Arguments

Argument	Description
expr1	The first expression to check for a valid number representation.
expr2	The second expression to check for a valid number representation.
expr3	The third expression to check for a valid number representation.
else	Value to return if none of the previous parameters has a valid number representation.

The alt function is often used with number or date interpretation functions. This way, Qlik Sense can test different date formats in a prioritized order. It can also be used to handle NULL values in numerical expressions.

Examples:

Examples

Example	Result
<code>alt(date#(dat , 'YYYY/MM/DD'), date#(dat , 'MM/DD/YYYY'), date#(dat , 'MM/DD/YY'), 'No valid date')</code>	This expression will test if the field date contains a date according to any of the three specified date formats. If so, it will return a dual value containing the original string and a valid number representation of a date. If no match is found, the text 'No valid date' will be returned (without any valid number representation).
<code>alt(Sales,0) + alt(Margin,0)</code>	This expression adds the fields Sales and Margin, replacing any missing value (NULL) with a 0.

class

The **class** function assigns the first parameter to a class interval. The result is a dual value with $a \leq x < b$ as the textual value, where a and b are the upper and lower limits of the bin, and the lower bound as numeric value.

Syntax:

```
class(expression, interval [ , label [ , offset ]])
```

Arguments:

Arguments

Argument	Description
interval	A number that specifies the bin width.

Argument	Description
label	An arbitrary string that can replace the 'x' in the result text.
offset	A number that can be used as offset from the default starting point of the classification. The default starting point is normally 0.

Examples:

Examples

Example	Result
class(var,10) with var = 23	returns '20<=x<30'
class(var,5,'value') with var = 23	returns '20<= value <25'
class(var,10,'x',5) with var = 23	returns '15<=x<25'

Example - Load script using class

Example: load script

Load script

In this example, we load a table containing name and age of people. We want to add a field that classifies each person according to an age group with a ten year interval. The original source table looks like the following.

Results

Name	Age
John	25
Karen	42
Yoshi	53

To add the age group classification field, you can add a preceding load statement using the **class** function.

Create a new tab in the data load editor, and then load the following data as an inline load. Create the table below in Qlik Sense to see the results.

```
LOAD *,  
class(Age, 10, 'age') As Agegroup;  
  
LOAD * INLINE  
[ Age, Name  
25, John  
42, Karen  
53, Yoshi];
```

Results

Results

Name	Age	Agegroup
John	25	20 <= age < 30
Karen	42	40 <= age < 50
Yoshi	53	50 <= age < 60

coalesce

The **coalesce** function returns the first of the parameters that has a valid non-NUL representation. Any number of parameters can be used.

Syntax:

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

Arguments:

Arguments

Argument	Description
expr1	The first expression to check for a valid non-NUL representation.
expr2	The second expression to check for a valid non-NUL representation.
expr3	The third expression to check for a valid non-NUL representation.

Examples:

Examples

Example	Result
	This expression changes all the NULL values of a field to 'N/A'.
Coalesce(ProductDescription, ProductName, ProductCode, 'no description available')	This expression will select between three different product description fields, for when some fields may not have values for the product. The first of the fields, in the order given, with a non-null value will be returned. If none of the fields contain a value, the result will be 'no description available'.
Coalesce(TextBetween(FileName, "'", "''), FileName)	This expression will trim potential enclosing quotes from the field <i>FileName</i> . If the <i>FileName</i> given is quoted, these are removed, and the enclosed, unquoted <i>FileName</i> is returned. If the <i>TextBetween</i> function doesn't find the delimiters it returns null, which the Coalesce rejects, returning instead the raw <i>FileName</i> .

if

The **if** function returns a value depending on whether the condition provided with the function evaluates as True or False.

Syntax:

```
if(condition , then [, else])
```

Arguments

Argument	Description
condition	Expression that is interpreted logically.
then	Expression that can be of any type. If the <i>condition</i> is True, then the if function returns the value of the <i>then</i> expression.
else	Expression that can be of any type. If the <i>condition</i> is False, then the if function returns the value of the <i>else</i> expression. This parameter is optional. If the <i>condition</i> is False, NULL is returned if you have not specified else.

Example

Example	Result
<code>if(Amount>= 0, 'OK' , 'Alarm')</code>	This expression tests if the amount is a positive number (0 or larger) and return 'OK' if it is. If the amount is less than 0, 'Alarm' is returned.

Example - Load script using if

Example: Load script

Load script

If can be used in load script with other methods and objects, including variables. For example, if you set a variable *threshold* and want to include a field in the data model based on that threshold, you can do the following.

Create a new tab in the data load editor, and then load the following data as an inline load. Create the table below in Qlik Sense to see the results.

```
Transactions:  
Load * Inline [  
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,  
color_code  
3750, 20180830, 23.56, 2, 2038593, L, Red  
3751, 20180907, 556.31, 6, 203521, m, orange  
3752, 20180916, 5.75, 1, 5646471, s, blue  
3753, 20180922, 125.00, 7, 3036491, l, Black  
3754, 20180922, 484.21, 13, 049681, xs, Red  
3756, 20180922, 59.18, 2, 2038593, M, Blue
```

```
3757, 20180923, 177.42, 21, 203521, XL, Black
];

set threshold = 100;

/* Create new table called Transaction_Buckets
Compare transaction_amount field from Transaction table to threshold of 100.
Output results into a new field called Compared to Threshold
*/
Transaction_Buckets:
Load
    transaction_id,
    If(transaction_amount > $(threshold), 'Greater than $(threshold)', 'Less than $(threshold)')
as [Compared to Threshold]
Resident Transactions;
```

Results

Qlik Sense table showing the output from using the *if* function in the load script.

transaction_id	Compared to Threshold
3750	Less than 100
3751	Greater than 100
3752	Less than 100
3753	Greater than 100
3754	Greater than 100
3756	Less than 100
3757	Greater than 100

Examples - Chart expressions using if

Examples: Chart expressions

Chart expression 1

Load script

Create a new tab in the data load editor, and then load the following data as an inline load. After loading the data, create the chart expression examples below in a Qlik Sense table.

```
MyTable:
LOAD * inline [Date, Location, Incidents
1/3/2016, Beijing, 0
1/3/2016, Boston, 12
1/3/2016, Stockholm, 3
1/3/2016, Toronto, 0
```

5 Script and chart functions

```
1/4/2016, Beijing, 0  
1/4/2016, Boston, 8];
```

Qlik Sense table showing examples of the *if* function in a chart expression.

Date	Location	Incidents	if(Incidents>=10, 'Critical', 'Ok')	if(Incidents>=10, 'Critical', If(Incidents>=1 and Incidents<10, 'Warning', 'Ok'))
1/3/2016	Beijing	0	Ok	Ok
1/3/2016	Boston	12	Critical	Critical
1/3/2016	Stockholm	3	Ok	Warning
1/3/2016	Toronto	0	Ok	Ok
1/4/2016	Beijing	0	Ok	Ok
1/4/2016	Boston	8	Ok	Warning

Chart expression 2

In a new app, add the following script in a new tab in the data load editor, and then load the data. You can then create the table with the chart expressions below.

```
SET FirstWeekDay=0;  
Load  
Date(MakeDate(2022)+RecNo()-1) as Date  
Autogenerate 14;
```

Qlik Sense table showing an example of the *if* function in a chart expression.

Date	WeekDay(Date)	If(WeekDay (Date)>=5,'WeekEnd','Normal Day')
1/1/2022	Sat	WeekEnd
1/2/2022	Sun	WeekEnd
1/3/2022	Mon	Normal Day
1/4/2022	Tue	Normal Day
1/5/2022	Wed	Normal Day
1/6/2022	Thu	Normal Day
1/7/2022	Fri	Normal Day
1/8/2022	Sat	WeekEnd
1/9/2022	Sun	WeekEnd
1/10/2022	Mon	Normal Day
1/11/2022	Tue	Normal Day

Date	WeekDay(Date)	If(WeekDay (Date)>=5,'WeekEnd','Normal Day')
1/12/2022	Wed	Normal Day
1/13/2022	Thu	Normal Day
1/14/2022	Fri	Normal Day

match

The **match** function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case sensitive.

Syntax:

```
match( str, expr1 [ , expr2,...exprN ] )
```



If you want to use case insensitive comparison, use the **mixmatch** function. If you want to use case insensitive comparison and wildcards, use the **wildmatch** function.

Example: Load script using match

Example: Load script

Load script

You can use match to load a subset of data. For example, you can return a numeric value for an expression in the function. You can then limit the data loaded based on the numeric value. Match returns 0 if there is no match. All expressions that are not matched in this example will therefore return 0 and will be excluded from the data load by the WHERE statement.

Create a new tab in the data load editor, and then load the following data as an inline load. Create the table below in Qlik Sense to see the results.

```
Transactions:
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange
3752, 20180916, 5.75, 1, 5646471, S, blue
3753, 20180922, 125.00, 7, 3036491, l, Black
3754, 20180922, 484.21, 13, 049681, xs, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
];
/*
Create new table called Transaction_Buckets
Create new fields called customer, and color code - Blue and Black
```

```
Load Transactions table.  
Match returns 1 for 'Blue', 2 for 'Black'.  
Does not return a value for 'blue' because match is case sensitive.  
Only values that returned numeric value greater than 0  
are loaded by WHERE statement into Transactions_Buckets table.  
*/
```

```
Transaction_Buckets:  
Load  
    customer_id,  
    customer_id as [Customer],  
    color_code as [Color Code Blue and Black]  
Resident Transactions  
where match(color_code,'Blue','Black') > 0;
```

Results

Qlik Sense table showing the output from using the match function in the load script

Color Code Blue and Black	Customer
Black	203521
Black	3036491
Blue	2038593

Examples - Chart expressions using match

Examples: Chart expressions

Chart expression 1

Load script

Create a new tab in the data load editor, and then load the following data as an inline load. After loading the data, create the chart expression examples below in a Qlik Sense table.

```
MyTable:  
Load * inline [Cities, Count  
Toronto, 123  
Toronto, 234  
Toronto, 231  
Boston, 32  
Boston, 23  
Boston, 1341  
Beijing, 234  
Beijing, 45  
Beijing, 235  
Stockholm, 938  
Stockholm, 39  
Stockholm, 189  
zurich, 2342
```

```
zurich, 9033
zurich, 0039];
```

The first expression in the table below returns 0 for Stockholm because 'Stockholm' is not included in the list of expressions in the **match** function. It also returns 0 for 'Zurich' because the **match** comparison is case-sensitive.

Qlik Sense table showing examples of the *match* function in a chart expression

Cities	match(Cities,'Toronto','Boston','Beijing','Zu rich')	match(Cities,'Toronto','Boston','Beijing','Stockholm',' zurich')
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	0	5

Chart expression 2

You can use match to perform a custom sort for an expression.

By default, columns sort numerically or alphabetically, depending on the data.

Qlik Sense table showing an example of the default sort order

Cities
Beijing
Boston
Stockholm
Toronto
zurich

To change the order, do the following:

1. Open the **Sorting** section for your chart in the **Properties** panel.
2. Turn off auto sorting for the column on which you want to do a custom sort.
3. Deselect **Sort numerically** and **Sort alphabetically**.
4. Select **Sort by expression**, and then enter an expression similar to the following:
`=match(Cities, 'Toronto', 'Boston', 'Beijing', 'Stockholm', 'zurich')`
The sort order on the Cities column changes.

Qlik Sense table showing an example of changing the sort order using the *match* function

Cities
Toronto
Boston
Beijing
Stockholm
zurich

You can also view the numeric value that is returned.

Qlik Sense table showing an example of the numeric values that are returned from the *match* function

Cities	Cities & ' - ' & match (Cities, 'Toronto','Boston', 'Beijing','Stockholm','zurich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

mixmatch

The **mixmatch** function compares the first parameter with all the following ones and returns the numeric location of the expressions that match. The comparison is case insensitive.

Syntax:

```
mixmatch( str, expr1 [ , expr2,...exprN ] )
```

If you instead want to use case sensitive comparison, use the **match** function. If you want to use case insensitive comparison and wildcards, use the **wildmatch** function.

Example - Load script using mixmatch

Example: Load script

Load script

You can use mixmatch to load a subset of data. For example, you can return a numeric value for an expression in the function. You can then limit the data loaded based on the numeric value. Mixmatch returns 0 if there is no match. All expressions that are not matched in this example will therefore return 0 and will be excluded from the data load by the WHERE statement.

Create a new tab in the data load editor, and then load the following data as an inline load. Create the table below in Qlik Sense to see the results.

```
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange
3752, 20180916, 5.75, 1, 5646471, s, blue
3753, 20180922, 125.00, 7, 3036491, l, Black
3754, 20180922, 484.21, 13, 049681, xs, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
];

/*
Create new table called Transaction_Buckets
Create new fields called Customer, and Color code - Black, Blue, blue
Load Transactions table.
Mixmatch returns 1 for 'Black', 2 for 'Blue'.
Also returns 3 for 'blue' because mixmatch is not case sensitive.
Only values that returned numeric value greater than 0
are loaded by WHERE statement into Transactions_Buckets table.
*/
Transaction_Buckets:
Load
    customer_id,
    customer_id as [Customer],
    color_code as [Color Code - Black, Blue, blue]
Resident Transactions
where mixmatch(color_code,'Black','Blue') > 0;
```

Results

Qlik Sense table showing the output from using
the mixmatch function in the load script.

Color Code Black, Blue, blue	Customer
Black	203521
Black	3036491
Blue	2038593
blue	5646471

Examples - Chart expressions using mixmatch

Examples: Chart expressions

Create a new tab in the data load editor, and then load the following data as an inline load. After loading the data, create the chart expression examples below in a Qlik Sense table.

Chart expression 1

```
MyTable:
Load * inline [Cities, Count
Toronto, 123
Toronto, 234
Toronto, 231
Boston, 32
Boston, 23
Boston, 1341
Beijing, 234
Beijing, 45
Beijing, 235
Stockholm, 938
Stockholm, 39
Stockholm, 189
zurich, 2342
zurich, 9033
zurich, 0039];
```

The first expression in the table below returns 0 for Stockholm because 'Stockholm' is not included in the list of expressions in the **mixmatch** function. It returns 4 for 'Zurich' because the **mixmatch** comparison is not case-sensitive.

Qlik Sense table showing examples of the *mixmatch* function in a chart expression

Cities	mixmatch(Cities,'Toronto','Boston','Beijing','Zu rich')	mixmatch(Cities,'Toronto','Boston','Beijing','Stockholm',' Zurich')
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

Chart expression 2

You can use mixmatch to perform a custom sort for an expression.

By default, columns sort alphabetically or numerically, depending on the data.

Qlik Sense table showing an example of the default sort order

Cities
Beijing
Boston

Cities
Stockholm
Toronto
zurich

To change the order, do the following:

1. Open the **Sorting** section for your chart in the **Properties** panel.
2. Turn off auto sorting for the column on which you want to do a custom sort.
3. Deselect **Sort numerically** and **Sort alphabetically**.
4. Select **Sort by expression**, and then enter the following expression:
`=mixmatch(Cities, 'Toronto', 'Boston', 'Beijing', 'Stockholm', 'Zurich')`
The sort order on the Cities column changes.

Qlik Sense table showing an example of changing the sort order using the *mixmatch* function.

Cities
Toronto
Boston
Beijing
Stockholm
zurich

You can also view the numeric value that is returned.

Qlik Sense table showing an example of the numeric values that are returned from the *mixmatch* function.

Cities	Cities & ' - ' & mixmatch (Cities, 'Toronto','Boston', 'Beijing','Stockholm','Zurich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

pick

The pick function returns the *n*:th expression in the list.

Syntax:

```
pick(n, expr1[ , expr2,...exprN])
```

Arguments:

Arguments

Argument	Description
n	<i>n</i> is an integer between 1 and N.

Example:

Example

Example	Result
<code>pick(N, 'A', 'B', 4, 6)</code>	returns 'B' if N = 2 returns 4 if N = 3

wildmatch

The **wildmatch** function compares the first parameter with all the following ones and returns the number of the expression that matches. It permits the use of wildcard characters (* and ?) in the comparison strings. * matches any sequence of characters. ? matches any single character. The comparison is case insensitive.

Syntax:

```
wildmatch( str, expr1 [ , expr2,...exprN ])
```

If you want to use comparison without wildcards, use the **match** or **mixmatch** functions.

Example: Load script using wildmatch**Example: Load script****Load script**

You can use wildmatch to load a subset of data. For example, you can return a numeric value for an expression in the function. You can then limit the data loaded based on the numeric value. Wildmatch returns 0 if there is no match. All expressions that are not matched in this example will therefore return 0 and will be excluded from the data load by the WHERE statement.

Create a new tab in the data load editor, and then load the following data as an inline load. Create the table below in Qlik Sense to see the results.

Transactions:

```
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange
3752, 20180916, 5.75, 1, 5646471, s, blue
3753, 20180922, 125.00, 7, 3036491, l, Black
```

```
3754, 20180922, 484.21, 13, 049681, xs, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
];

/*
Create new table called Transaction_Buckets
Create new fields called Customer, and Color code - Black, Blue, blue, red
Load Transactions table.
wildmatch returns 1 for 'Black', 'Blue', and 'blue', and 2 for 'Red'.
Only values that returned numeric value greater than 0
are loaded by WHERE statement into Transactions_Buckets table.
*/

Transaction_Buckets:
Load
    customer_id,
    customer_id as [Customer],
    color_code as [Color Code Black, Blue, blue, Red]
Resident Transactions
where wildmatch(color_code, 'Bl*', 'R??') > 0;
```

Results

Qlik Sense table showing the output from using the *wildmatch* function in the load script

Color Code Black, Blue, blue, Red	Customer
Black	203521
Black	3036491
Blue	2038593
blue	5646471
Red	049681
Red	2038593

Examples: Chart expressions using wildmatch

Example: Chart expression

Chart expression 1

Create a new tab in the data load editor, and then load the following data as an inline load. After loading the data, create the chart expression examples below in a Qlik Sense table.

```
MyTable:
Load * inline [Cities, Count
Toronto, 123
Toronto, 234
Toronto, 231
```

```
Boston, 32
Boston, 23
Boston, 1341
Beijing, 234
Beijing, 45
Beijing, 235
Stockholm, 938
Stockholm, 39
Stockholm, 189
zurich, 2342
zurich, 9033
zurich, 0039];
```

The first expression in the table below returns 0 for Stockholm because 'Stockholm' is not included in the list of expressions in the **wildmatch** function. It also returns 0 for 'Boston' because ? only matches on a single character.

Qlik Sense table showing examples of the *wildmatch* function in a chart expression

Cities	wildmatch(Cities,'Tor*','?ton','Beijing','*uric h')	wildmatch(Cities,'Tor*','???ton','Beijing','Stockholm','*uric h')
Beijing	3	3
Boston	0	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

Chart expression 2

You can use wildmatch to perform a custom sort for an expression.

By default, columns sort numerically or alphabetically, depending on the data.

Qlik Sense table showing an example of the default sort order

Cities
Beijing
Boston
Stockholm
Toronto
zurich

To change the order, do the following:

1. Open the **Sorting** section for your chart in the **Properties** panel.
2. Turn off auto sorting for the column on which you want to do a custom sort.
3. Deselect **Sort numerically** and **Sort alphabetically**.
4. Select **Sort by expression**, and then enter an expression similar to the following:
`=wildmatch(Cities, 'Tor*', '???ton', 'Beijing', 'Stockholm', '*urich')`
The sort order on the Cities column changes.

Qlik Sense table showing an example of changing the sort order using the *wildmatch* function.

Cities
Toronto
Boston
Beijing
Stockholm
zurich

You can also view the numeric value that is returned.

Qlik Sense table showing an example of the numeric values that are returned from the *wildmatch* function

Cities	Cities & ' - ' & wildmatch (Cities, 'Tor*', '???ton', 'Beijing', 'Stockholm', '*urich')
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

5.6 Counter functions

This section describes functions related to record counters during **LOAD** statement evaluation in the data load script. The only function that can be used in chart expressions is **RowNo()**.

Some counter functions do not have any parameters, but the trailing parentheses are however still required.

Counter functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

autonumber

This script function returns a unique integer value for each distinct evaluated value of *expression* encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.

autonumber (*expression*[, *AutoID*])

autonumberhash128

This script function calculates a 128-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used for example for creating a compact memory representation of a complex key.

autonumberhash128 (*expression* {, *expression*})

autonumberhash256

This script function calculates a 256-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.

autonumberhash256 (*expression* {, *expression*})

IterNo

This script function returns an integer indicating for which time one single record is evaluated in a **LOAD** statement with a **while** clause. The first iteration has number 1. The **IterNo** function is only meaningful if used together with a **while** clause.

IterNo ()

RecNo

This script functions returns an integer for the number of the currently read row of the current table. The first record is number 1.

RecNo ()

RowNo - script function

This function returns an integer for the position of the current row in the resulting Qlik Sense internal table. The first row is number 1.

RowNo ()

RowNo - chart function

RowNo() returns the number of the current row within the current column segment in a table. For bitmap charts, **RowNo()** returns the number of the current row within the chart's straight table equivalent.

RowNo - chart function([TOTAL])

autonumber

This script function returns a unique integer value for each distinct evaluated value of *expression* encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.



You can only connect **autonumber** keys that have been generated in the same data load, as the integer is generated according to the order the table is read. If you need to use keys that are persistent between data loads, independent of source data sorting, you should use the **hash128**, **hash160** or **hash256** functions.

Syntax:

```
autonumber(expression[ , AutoID])
```

Arguments:

Argument	Description
AutoID	In order to create multiple counter instances if the autonumber function is used on different keys within the script, the optional parameter <i>AutoID</i> can be used for naming each counter.

Example: Creating a composite key

In this example we create a composite key using the **autonumber** function to conserve memory. The example is brief for demonstration purpose, but would be meaningful with a table containing a large number of rows.

Example data

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

The source data is loaded using inline data. Then we add a preceding load which creates a composite key from the Region, Year and Month fields.

```
Regionsales:
LOAD *,
AutoNumber(Region&Year&Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Sales
North, 2014, May, 245
North, 2014, May, 347
North, 2014, June, 127
South, 2014, June, 645
South, 2013, May, 367
South, 2013, May, 221
];
```

The resulting table looks like this:

Results table

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

In this example you can refer to the RYMkey, for example 1, instead of the string 'North2014May' if you need to link to another table.

Now we load a source table of costs in a similar way. The Region, Year and Month fields are excluded in the preceding load to avoid creating a synthetic key, we are already creating a composite key with the **autonumber** function, linking the tables.

```
RegionCosts:
LOAD Costs,
AutoNumber(Region&Year&Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

Now we can add a table visualization to a sheet, and add the Region, Year and Month fields, as well as Sum measures for the sales and the costs. The table will look like this:

Results table

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

autonumberhash128

This script function calculates a 128-bit hash of the combined input expression values and the returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used for example for creating a compact memory representation of a complex key.



*You can only connect **autonumberhash128** keys that have been generated in the same data load, as the integer is generated according to the order the table is read. If you need to use keys that are persistent between data loads, independent of source data sorting, you should use the **hash128**, **hash160** or **hash256** functions.*

Syntax:

```
autonumberhash128(expression {, expression})
```

Example: Creating a composite key

In this example we create a composite key using the **autonumberhash128** function to conserve memory. The example is brief for demonstration purpose, but would be meaningful with a table containing a large number of rows.

Example data

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

The source data is loaded using inline data. Then we add a preceding load which creates a composite key from the Region, Year and Month fields.

```
Regionsales:
LOAD *, 
AutoNumberHash128(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Sales
North, 2014, May, 245
North, 2014, May, 347
North, 2014, June, 127
South, 2014, June, 645
South, 2013, May, 367
```

```
South, 2013, May, 221
];
```

The resulting table looks like this:

Results table

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

In this example you can refer to the RYMkey, for example 1, instead of the string 'North2014May' if you need to link to another table.

Now we load a source table of costs in a similar way. The Region, Year and Month fields are excluded in the preceding load to avoid creating a synthetic key, we are already creating a composite key with the **autonumberhash128** function, linking the tables.

```
RegionCosts:
LOAD Costs,
AutoNumberHash128(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

Now we can add a table visualization to a sheet, and add the Region, Year and Month fields, as well as Sum measures for the sales and the costs. The table will look like this:

Results table

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784
North	2014	June	127	199
North	2014	May	592	56

Region	Year	Month	Sum([Sales])	Sum([Costs])
South	2014	June	645	64
South	2013	May	588	465

autonumberhash256

This script function calculates a 256-bit hash of the combined input expression values and returns a unique integer value for each distinct hash value encountered during the script execution. This function can be used e.g. for creating a compact memory representation of a complex key.



You can only connect **autonumberhash256** keys that have been generated in the same data load, as the integer is generated according to the order the table is read. If you need to use keys that are persistent between data loads, independent of source data sorting, you should use the **hash128**, **hash160** or **hash256** functions.

Syntax:

```
autonumberhash256(expression {, expression})
```

Example: Creating a composite key

In this example we create a composite key using the **autonumberhash256** function to conserve memory. The example is brief for demonstration purpose, but would be meaningful with a table containing a large number of rows.

Example table

Region	Year	Month	Sales
North	2014	May	245
North	2014	May	347
North	2014	June	127
South	2014	June	645
South	2013	May	367
South	2013	May	221

The source data is loaded using inline data. Then we add a preceding load which creates a composite key from the Region, Year and Month fields.

```
Regionsales:
LOAD * ,
AutoNumberHash256(Region, Year, Month) as RYMkey;

LOAD * INLINE
```

```
[ Region, Year, Month, Sales
North, 2014, May, 245
North, 2014, May, 347
North, 2014, June, 127
South, 2014, June, 645
South, 2013, May, 367
South, 2013, May, 221
];
```

The resulting table looks like this:

Results table

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1
North	2014	May	347	1
North	2014	June	127	2
South	2014	June	645	3
South	2013	May	367	4
South	2013	May	221	4

In this example you can refer to the RYMkey, for example 1, instead of the string 'North2014May' if you need to link to another table.

Now we load a source table of costs in a similar way. The Region, Year and Month fields are excluded in the preceding load to avoid creating a synthetic key, we are already creating a composite key with the **autonumberhash256** function, linking the tables.

```
RegionCosts:
LOAD Costs,
AutoNumberHash256(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

Now we can add a table visualization to a sheet, and add the Region, Year and Month fields, as well as Sum measures for the sales and the costs. The table will look like this:

Results table

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals	-	-	1952	784
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

IterNo

This script function returns an integer indicating for which time one single record is evaluated in a **LOAD** statement with a **while** clause. The first iteration has number 1. The **IterNo** function is only meaningful if used together with a **while** clause.

Syntax:

```
IterNo( )
```

Examples and results:

Example:

```
LOAD
  IterNo() as Day,
  Date( StartDate + IterNo() - 1 ) as Date
  while StartDate + IterNo() - 1 <= EndDate;

LOAD * INLINE
[StartDate, EndDate
2014-01-22, 2014-01-26
];
```

This **LOAD** statement will generate one record per date within the range defined by **StartDate** and **EndDate**.

In this example, the resulting table will look like this:

Results table

Day	Date
1	2014-01-22
2	2014-01-23
3	2014-01-24
4	2014-01-25
5	2014-01-26

RecNo

This script function returns an integer for the number of the currently read row of the current table. The first record is number 1.

Syntax:

```
RecNo( )
```

In contrast to **RowNo()**, which counts rows in the resulting Qlik Sense table, **RecNo()**, counts the records in the raw data table and is reset when a raw data table is concatenated to another.

Example: Data load script

Raw data table load:

```
Tab1:  
LOAD * INLINE  
[A, B  
1, aa  
2,cc  
3,ee];
```

```
Tab2:  
LOAD * INLINE  
[C, D  
5, xx  
4,yy  
6,zz];
```

Loading record and row numbers for selected rows:

```
QTab:  
LOAD *,  
RecNo(),  
RowNo()  
resident Tab1 where A<>2;  
  
LOAD  
C as A,  
D as B,  
RecNo(),  
RowNo()  
resident Tab2 where A<>5;  
  
//We don't need the source tables anymore, so we drop them  
Drop tables Tab1, Tab2;
```

The resulting Qlik Sense internal table:

Results table

A	B	RecNo()	RowNo()
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo

This function returns an integer for the position of the current row in the resulting Qlik Sense internal table. The first row is number 1.

Syntax:

```
RowNo ( [TOTAL] )
```

In contrast to **RecNo()**, which counts the records in the raw data table, the **RowNo()** function does not count records that are excluded by **where** clauses and is not reset when a raw data table is concatenated to another.



If you use preceding load, that is, a number of stacked **LOAD** statements reading from the same table, you can only use **RowNo()** in the top **LOAD** statement. If you use **RowNo()** in subsequent **LOAD** statements, 0 is returned.

Example: Data load script

Raw data table load:

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2,cc
3,ee];
```

```
Tab2:
LOAD * INLINE
[C, D
5, xx
4,yy
6,zz];
```

Loading record and row numbers for selected rows:

```
QTab:
LOAD *,
RecNo( ),
RowNo( )
```

```

resident Tab1 where A<>2;

LOAD
C as A,
D as B,
RecNo( ),
RowNo( )
resident Tab2 where A<>5;

//We don't need the source tables anymore, so we drop them
Drop tables Tab1, Tab2;
The resulting Qlik Sense internal table:

```

Results table

A	B	RecNo()	RowNo()
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo - chart function

RowNo() returns the number of the current row within the current column segment in a table. For bitmap charts, **RowNo()** returns the number of the current row within the chart's straight table equivalent.

If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Column segments

Region	Country	Population	Rank(Population)
Americas	Canada	37,954,554	3
	United States of America	331,002,451	1
	Sweden	10,099,265	4
Europe	United Kingdom	67,886,011	2
	France	65,273,511	3
	Germany	83,783,942	1



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Syntax:

```
RowNo ( [TOTAL] )
```

Return data type: integer

Arguments:

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

Example: Chart expression using RowNo

Example - chart expression

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

Temp:

```
LOAD * inline [
Customer|Product|orderNumber|unitsales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|1|25| 25
Canutility|AA|3|8|15
Canutility|CC|5|4|19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

Chart expression

Create a table visualization in a Qlik Sense sheet with **Customer** and **UnitSales** as dimensions. Add **RowNo()** and **RowNo(TOTAL)** as measures labeled **Row in Segment** and **Row Number**, respectively. Add the following expression to the table as a measure:

```
if( RowNo( )=1, 0, unitsales / Above( unitsales ))
```

Result

Customer	UnitSales	Row in Segment	Row Number	If(RowNo()=1, 0, Unitsales / Above(UnitSales))
Astrida	4	1	1	0
Astrida	9	2	2	2.25
Astrida	10	3	3	1.11111111111111

Customer	UnitSales	Row in Segment	Row Number	If(RowNo()=1, 0, UnitSales / Above(UnitSales))
Betacab	2	1	4	0
Betacab	5	2	5	2.5
Betacab	25	3	6	5
Canutility	4	1	7	0
Canutility	8	2	8	2
Divadip	1	1	9	0
Divadip	4	2	10	4

Explanation

The **Row in Segment** column shows the results 1,2,3 for the column segment containing the values of UnitSales for customer Astrida. The row numbering then begins at 1 again for the next column segment, which is Betacab.

The **Row Number** column disregards the dimensions because of the TOTAL argument for RowNo() and counts the rows in the table.

This expression returns 0 for the first row in each column segment, so the column shows:

0, 2.25, 1.1111111, 0, 2.5, 5, 0, 2, 0, and 4.

See also:

- [Above - chart function \(page 1235\)](#)

5.7 Date and time functions

Qlik Sense date and time functions are used to transform and convert date and time values. All functions can be used in both the data load script and in chart expressions.

Functions are based on a date-time serial number that equals the number of days since December 30, 1899. The integer value represents the day and the fractional value represents the time of the day.

Qlik Sense uses the numerical value of the parameter, so a number is valid as a parameter also when it is not formatted as a date or a time. If the parameter does not correspond to numerical value, for example, because it is a string, then Qlik Sense attempts to interpret the string according to the date and time environment variables.

If the time format used in the parameter does not correspond to the one set in the environment variables, Qlik Sense will not be able to make a correct interpretation. To resolve this, either change the settings or use an interpretation function.

In the examples for each function, the default time and date formats hh:mm:ss and YYYY-MM-DD (ISO 8601) are assumed.



When processing a timestamp with a date or time function, Qlik Sense ignores any daylight savings time parameters unless the date or time function includes a geographical position.

For example, `convertToLocalTime(filetime('Time.qvd'), 'Paris')` would use daylight savings time parameters while `convertToLocalTime(filetime('Time.qvd'), 'GMT-01:00')` would not use daylight savings time parameters.

Date and time functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Integer expressions of time

second

This function returns an integer representing the second when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

second (**expression**)

minute

This function returns an integer representing the minute when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

minute (**expression**)

hour

This function returns an integer representing the hour when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

hour (**expression**)

day

This function returns an integer representing the day when the fraction of the **expression** is interpreted as a date according to the standard number interpretation.

day (**expression**)

week

This function returns an integer representing the week number according to ISO 8601. The week number is calculated from the date interpretation of the expression, according to the standard number interpretation.

week (**expression**)

month

This function returns a dual value: a month name as defined in the environment variable **MonthNames** and an integer between 1-12. The month is calculated from the date interpretation of the expression, according to the standard number interpretation.

```
month (expression)
```

year

This function returns an integer representing the year when the **expression** is interpreted as a date according to the standard number interpretation.

```
year (expression)
```

weekyear

This function returns the year to which the week number belongs according to the environment variables. The week number ranges between 1 and approximately 52.

```
weekyear (expression)
```

weekday

This function returns a dual value with:

- A day name as defined in the environment variable **DayNames**.
- An integer between 0-6 corresponding to the nominal day of the week (0-6).

```
weekday (date)
```

Timestamp functions

now

This function returns a timestamp of the current time. The function returns values in the **TimeStamp** system variable format. The default **timer_mode** value is 1.

```
now ([timer_mode])
```

today

This function returns the current date. The function returns values in the **dateFormat** system variable format.

```
today ([timer_mode])
```

LocalTime

This function returns a timestamp of the current time for a specified time zone.

```
localtime ([timezone [, ignoreDST ]])
```

Make functions

makedate

This function returns a date calculated from the year **YYYY**, the month **MM** and the day **DD**.

```
makedate (YYYY [, MM [, DD ] ])
```

makeweekdate

This function returns a date calculated from the year, the week number, and the day of week .

```
makeweekdate (YYYY [ , WW [ , D ] ])
```

maketime

This function returns a time calculated from the hour **hh**, the minute **mm**, and the second **ss**.

```
maketime (hh [ , mm [ , ss [ .fff ] ] ])
```

Other date functions

AddMonths

This function returns the date occurring **n** months after **startdate** or, if **n** is negative, the date occurring **n** months before **startdate**.

```
addmonths (startdate, n , [ , mode])
```

AddYears

This function returns the date occurring **n** years after **startdate** or, if **n** is negative, the date occurring **n** years before **startdate**.

```
addyears (startdate, n)
```

yeartodate

This function finds if the input timestamp falls within the year of the date the script was last loaded, and returns True if it does, False if it does not.

```
yeartodate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

Timezone functions

timezone

This function returns the time zone, as defined on the computer where the Qlik engine is running.

```
timezone ( )
```

GMT

This function returns the current Greenwich Mean Time, as derived from the regional settings.

```
GMT ( )
```

UTC

Returns the current Coordinated Universal Time.

```
UTC ( )
```

daylightsaving

Returns the current adjustment for daylight saving time, as defined in Windows.

```
daylightsaving ( )
```

converttolocaltime

Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.

```
converttolocaltime (timestamp [, place [, ignore_dst=false]])
```

Set time functions

setdateyear

This function takes as input a **timestamp** and a **year** and updates the **timestamp** with the **year** specified in input.

```
setdateyear (timestamp, year)
```

setdateyearmonth

This function takes as input a **timestamp**, a **month** and a **year** and updates the **timestamp** with the **year** and the **month** specified in input.

```
setdateyearmonth (timestamp, year, month)
```

In... functions

inyear

This function returns True if **timestamp** lies inside the year containing **base_date**.

```
inyear (date, basedate , shift [, first_month_of_year = 1])
```

inyeartodate

This function returns True if **timestamp** lies inside the part of year containing **base_date** up until and including the last millisecond of **base_date**.

```
inyeartodate (date, basedate , shift [, first_month_of_year = 1])
```

inquarter

This function returns True if **timestamp** lies inside the quarter containing **base_date**.

```
inquarter (date, basedate , shift [, first_month_of_year = 1])
```

inquartertodate

This function returns True if **timestamp** lies inside the part of the quarter containing **base_date** up until and including the last millisecond of **base_date**.

```
inquartertodate (date, basedate , shift [, first_month_of_year = 1])
```

inmonth

This function returns True if **timestamp** lies inside the month containing **base_date**.

```
inmonth (date, basedate , shift)
```

inmonthtodate

Returns True if **date** lies inside the part of month containing **basedate** up until and including the last millisecond of **basedate**.

```
inmonthtodate (date, basedate , shift)
```

inmonths

This function finds if a timestamp falls within the same month, bi-month, quarter, four-month period, or half-year as a base date. It is also possible to find if the timestamp falls within a previous or following time period.

```
inmonths (n, date, basedate , shift [, first_month_of_year = 1])
```

inmonthstodate

This function finds if a timestamp falls within the part a period of the month, bi-month, quarter, four-month period, or half-year up to and including the last millisecond of **base_date**. It is also possible to find if the timestamp falls within a previous or following time period.

```
inmonthstodate (n, date, basedate , shift [, first_month_of_year = 1])
```

inweek

This function returns True if **timestamp** lies inside the week containing **base_date**.

```
inweek (date, basedate , shift [, weekstart])
```

inweektodate

This function returns True if **timestamp** lies inside the part of week containing **base_date** up until and including the last millisecond of **base_date**.

```
inweektodate (date, basedate , shift [, weekstart])
```

inlunarweek

This function determines if **timestamp** lies inside the lunar week containing **base_date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week., Apart from the final week of the year, each week will contain exactly seven days.

```
inlunarweek (date, basedate , shift [, weekstart])
```

inlunarweektodate

This function finds if **timestamp** lies inside the part of the lunar week up to and including the last millisecond of **base_date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

```
inlunarweektodate (date, basedate , shift [, weekstart])
```

inday

This function returns True if **timestamp** lies inside the day containing **base_timestamp**.

```
inday (timestamp, basetimestamp , shift [, daystart])
```

indaytotime

This function returns True if **timestamp** lies inside the part of day containing **base_timestamp** up until and including the exact millisecond of **base_timestamp**.

```
indaytotime (timestamp, basetimestamp , shift [, daystart])
```

Start ... end functions

yearstart

This function returns a timestamp corresponding to the start of the first day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

```
yearstart ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

```
yearend ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearname

This function returns a four-digit year as display value with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**.

```
yearname (date [, shift = 0 [, first_month_of_year = 1]])
```

quarterstart

This function returns a value corresponding to a timestamp of the first millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

```
quarterstart (date [, shift = 0 [, first_month_of_year = 1]])
```

quarterend

This function returns a value corresponding to a timestamp of the last millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

```
quarterend (date [, shift = 0 [, first_month_of_year = 1]])
```

quartername

This function returns a display value showing the months of the quarter (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the quarter.

```
quartername (date [, shift = 0 [, first_month_of_year = 1]])
```

monthstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthstart (date [, shift = 0])
```

monthend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

```
monthend (date [, shift = 0])
```

monthname

This function returns a display value showing the month (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the month.

```
monthname (date [, shift = 0])
```

monthsstart

This function returns a value corresponding to the timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for a previous or following time period. The default output format is the **DateFormat** set in the script.

```
monthsstart (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsend

This function returns a value corresponding to a timestamp of the last millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for a previous or following time period.

```
monthsend (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsname

This function returns a display value representing the range of the months of the period (formatted according to the **MonthNames** script variable) as well as the year. The underlying numeric value corresponds to a timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date.

```
monthsname (n, date [, shift = 0 [, first_month_of_year = 1]])
```

weekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the calendar week containing **date**. The default output format is the **DateFormat** set in the script.

```
weekstart (date [, shift = 0 [, weekoffset = 0]])
```

weekend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the calendar week containing **date**. The default output format will be the **DateFormat** set in the script.

```
weekend (date [, shift = 0 [, weekoffset = 0]])
```

weekname

This function returns a value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the week containing **date**.

```
weekname (date [, shift = 0 [, weekoffset = 0]])
```

lunarweekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the lunar week containing **date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

```
lunarweekstart (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the lunar week containing **date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

```
lunarweekend (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekname

This function returns a display value showing the year and lunar week number corresponding to a timestamp of the first millisecond of the first day of the lunar week containing **date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

```
lunarweekname (date [, shift = 0 [,weekoffset = 0]])
```

daystart

This function returns a value corresponding to a timestamp with the first millisecond of the day contained in the **time** argument. The default output format will be the **TimestampFormat** set in the script.

```
daystart (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayend

This function returns a value corresponding to a timestamp of the final millisecond of the day contained in **time**. The default output format will be the **TimestampFormat** set in the script.

```
dayend (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayname

This function returns a value showing the date with an underlying numeric value corresponding to a timestamp of the first millisecond of the day containing **time**.

```
dayname (timestamp [, shift = 0 [, dayoffset = 0]])
```

Day numbering functions

age

The **age** function returns the age at the time of **timestamp** (in completed years) of somebody born on **date_of_birth**.

```
age (timestamp, date_of_birth)
```

networkdays

The **networkdays** function returns the number of working days (Monday-Friday) between and including **start_date** and **end_date** taking into account any optionally listed **holiday**.

```
networkdays (start:date, end_date {, holiday})
```

firstworkdate

The **firstworkdate** function returns the latest starting date to achieve **no_of_workdays** (Monday-Friday) ending no later than **end_date** taking into account any optionally listed holidays. **end_date** and **holiday** should be valid dates or timestamps.

```
firstworkdate (end_date, no_of_workdays {, holiday} )
```

lastworkdate

The **lastworkdate** function returns the earliest ending date to achieve **no_of_workdays** (Monday-Friday) if starting at **start_date** taking into account any optionally listed **holiday**. **start_date** and **holiday** should be valid dates or timestamps.

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

daynumberofyear

This function calculates the day number of the year in which a timestamp falls. The calculation is made from the first millisecond of the first day of the year, but the first month can be offset.

```
daynumberofyear (date[,firstmonth])
```

daynumberofquarter

This function calculates the day number of the quarter in which a timestamp falls. This function is used when creating a Master Calendar.

```
daynumberofquarter (date[,firstmonth])
```

addmonths

This function returns the date occurring **n** months after **startdate** or, if **n** is negative, the date occurring **n** months before **startdate**.

Syntax:

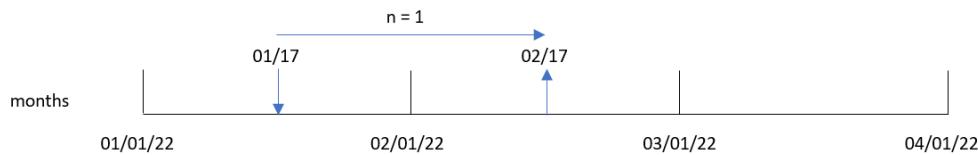
```
AddMonths (startdate, n , [ , mode])
```

Return data type: dual

The **addmonths()** function adds or subtracts a defined number of months, **n**, from a **startdate** and returns the resultant date.

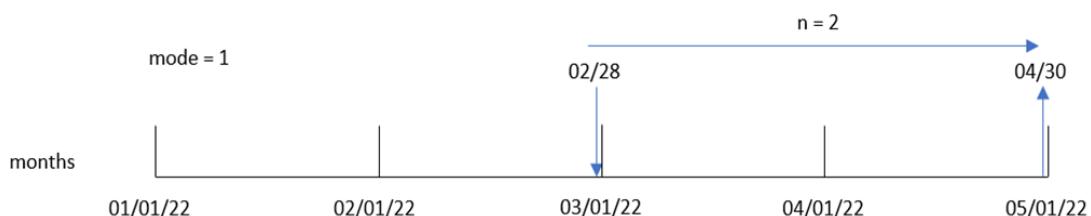
The **mode** argument will impact **startdate** values on or after the 28th of the month. By setting the **mode** argument to 1, the **addmonths()** function returns a date that is equal in relative distance to the end of the month as the **startdate**.

Example diagram of addmonths() function



For example, February 28 is the last day of the month. If the `addmonths()` function, with a `mode` of 1, is used to return the date two months later, the function will return the last date of April, April 30.

Example diagram of addmonths() function, with mode=1



Arguments

Argument	Description
startdate	The start date as a time stamp, for example '2012-10-12'.
n	Number of months as a positive or negative integer.
mode	Specifies if the month is added relative to the beginning or to the end of the month. Default mode is 0 for additions relative to the beginning of the month. Set mode to 1 for additions relative to the end of the month. When mode is set to 1 and the input date is the 28th or above, the function checks how many days are left to reach the end of the month on the startdate. The same number of days to reach the end of the month are set on the date returned.

When to use it

The `addmonths()` function will commonly be used in an expression to find a date a given number of months before or after a period of time.

For example, the `addmonths()` function can be used to identify the end date of mobile phone contracts.

Function examples

Example	Result
<code>addmonths ('01/29/2003' ,3)</code>	Returns '04/29/2003'.
<code>addmonths ('01/29/2003' ,3,0)</code>	Returns '04/29/2003'.
<code>addmonths ('01/29/2003' ,3,1)</code>	Returns '04/28/2003'.

Example	Result
<code>addmonths ('01/29/2003',1,0)</code>	Returns '02/28/2003'.
<code>addmonths ('01/29/2003',1,1)</code>	Returns '02/26/2003'.
<code>addmonths ('02/28/2003',1,0)</code>	Returns '03/28/2003'.
<code>addmonths ('02/28/2003',1,1)</code>	Returns '03/31/2003'.
<code>addmonths ('01/29/2003',-3)</code>	Returns '10/29/2002'.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `two_months_later`, that returns the date for two months after the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  addmonths(date,2) as two_months_later
;
Load
```

```
*  
Inline  
[  
id,date,amount  
8188,'01/10/2020',37.23  
8189,'02/28/2020',17.17  
8190,'04/09/2020',88.27  
8191,'04/16/2020',57.42  
8192,'05/21/2020',53.80  
8193,'08/14/2020',82.06  
8194,'10/07/2020',40.39  
8195,'12/05/2020',87.21  
8196,'01/22/2021',95.93  
8197,'02/03/2021',45.89  
8198,'03/17/2021',36.23  
8199,'04/23/2021',25.66  
8200,'05/04/2021',82.77  
8201,'06/30/2021',69.98  
8202,'07/26/2021',76.11  
8203,'12/27/2021',25.12  
8204,'02/02/2022',46.23  
8205,'02/26/2022',84.21  
8206,'03/07/2022',96.24  
8207,'03/11/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- two_months_later

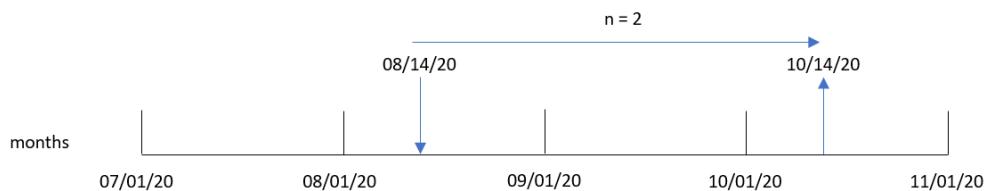
Results table

date	two_months_later
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021

date	two_months_later
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

The `two_months_later` field is created in the preceding load statement by using the `addmonths()` function. The first argument provided identifies which date is being evaluated. The second argument is the number of months to add or subtract from the `startdate`. In this instance, the value of 2 is provided.

Diagram of `addmonths()` function, example with no additional arguments



Transaction 8193 took place on August 14. Therefore, the `addmonths()` function returns October 14, 2020 for the `two_months_later` field.

Example 2 – Relative month end

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of month-end transactions in 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `dateFormat` system variable (MM/DD/YYYY) format.

- The creation of a field, `relative_two_months_prior`, that returns the relative month-end date for two months before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    addmonths(date,-2,1) as relative_two_months_prior
  ;
Load
*
Inline
[
id,date,amount
8188,'01/28/2022',37.23
8189,'01/31/2022',57.54
8190,'02/28/2022',17.17
8191,'04/29/2022',88.27
8192,'04/30/2022',57.42
8193,'05/31/2022',53.80
8194,'08/14/2022',82.06
8195,'10/07/2022',40.39
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

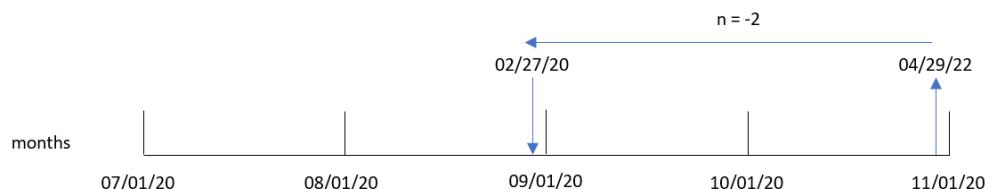
- `date`
- `relative_two_months_prior`

Results table

date	relative_two_months_prior
01/28/2022	11/27/2021
01/31/2022	11/30/2021
02/28/2022	12/31/2021
04/29/2022	02/27/2022
04/30/2022	02/28/2022
05/31/2022	03/31/2022
08/14/2022	06/14/2022
10/07/2022	08/07/2022

The `relative_two_months_prior` field is created in the preceding load statement by using the `addmonths()` function. The first argument provided identifies which date is being evaluated. The second argument is the number of months to add or subtract from the `startdate`. In this instance, the value of -2 is provided. The final argument is the mode, with a value of 1, which forces the function to calculate the relative month-end date for all dates greater than or equal to 28.

Diagram of `addmonths()` function, example with `n=-2`



Transaction 8191 takes place on April 29, 2022. Initially, two months prior would set the month to February. Then, due to the third argument of the function setting the mode to 1 and the day value being later than the 27th, the function calculates the relative month-end value. The function identifies that the 29th is the second last day of April and therefore returns the second last day of February, the 27th.

Example 3– Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns the date for two months after the transaction took place is created as a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
```

```
8196, '01/22/2021', 95.93  
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '12/27/2021', 25.12  
8204, '02/02/2022', 46.23  
8205, '02/26/2022', 84.21  
8206, '03/07/2022', 96.24  
8207, '03/11/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=addmonths(date,2)
```

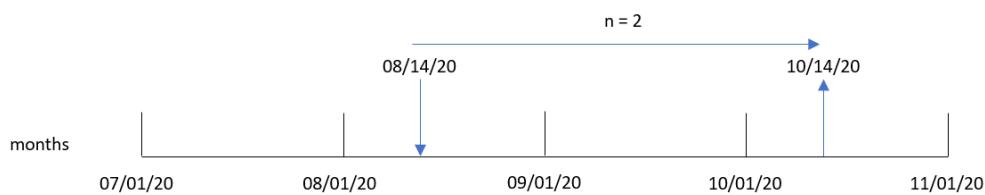
Results table

date	=addmonths(date,2)
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022

date	=addmonths(date,2)
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

The two_months_later measure is created in the chart object by using the addmonths() function. The first argument provided identifies which date is being evaluated. The second argument is the number of months to add or subtract from the startdate. In this instance, the value of 2 is provided.

Diagram of addmonths() function, chart object example



Transaction 8193 took place on August 14. Therefore, the addmonths() function returns the October 14, 2020 for the two_months_later field.

Example 4 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called Mobile_Plans.
- Information with the contract ID, start date, contract length, and monthly fee.

The end user would like a chart object that displays, by contract ID, the termination date of each phone contract.

Load script

```
Mobile_Plans:
Load
*
Inline
[
contract_id,start_date,contract_length,monthly_fee
8188,'01/13/2020',18,37.23
8189,'02/26/2020',24,17.17
8190,'03/27/2020',36,88.27
8191,'04/16/2020',24,57.42
```

```

8192,'05/21/2020',24,53.80
8193,'08/14/2020',12,82.06
8194,'10/07/2020',18,40.39
8195,'12/05/2020',12,87.21
8196,'01/22/2021',12,95.93
8197,'02/03/2021',18,45.89
8198,'03/17/2021',24,36.23
8199,'04/23/2021',24,25.66
8200,'05/04/2021',12,82.77
8201,'06/30/2021',12,69.98
8202,'07/26/2021',12,76.11
8203,'12/27/2021',36,25.12
8204,'06/06/2022',24,46.23
8205,'07/18/2022',12,84.21
8206,'11/14/2022',12,96.24
8207,'12/12/2022',18,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `contract_id`
- `start_date`
- `contract_length`

Create the following measure to calculate the end date of each contract:

```
=addmonths(start_date,contract_length, 0)
```

Results table

<code>contract_id</code>	<code>start_date</code>	<code>contract_length</code>	<code>=addmonths(start_date,contract_length,0)</code>
8188	01/13/2020	18	07/13/2021
8189	02/26/2020	24	02/26/2022
8190	03/27/2020	36	03/27/2023
8191	04/16/2020	24	04/16/2022
8192	05/21/2020	24	05/21/2022
8193	08/14/2020	12	08/14/2021
8194	10/07/2020	18	04/07/2022
8195	12/05/2020	12	12/05/2021
8196	01/22/2021	12	01/22/2022
8197	02/03/2021	18	08/03/2022
8198	03/17/2021	24	03/17/2023

contract_id	start_date	contract_length	=addmonths(start_date,contract_length,0)
8199	04/23/2021	24	04/23/2023
8200	05/04/2021	12	05/04/2022
8201	06/30/2021	12	06/30/2022
8202	07/26/2021	12	07/26/2022
8203	12/27/2021	36	12/27/2024
8204	06/06/2022	24	06/06/2024
8205	07/18/2022	12	07/18/2023
8206	11/14/2022	12	11/14/2023
8207	12/12/2022	18	06/12/2024

addyears

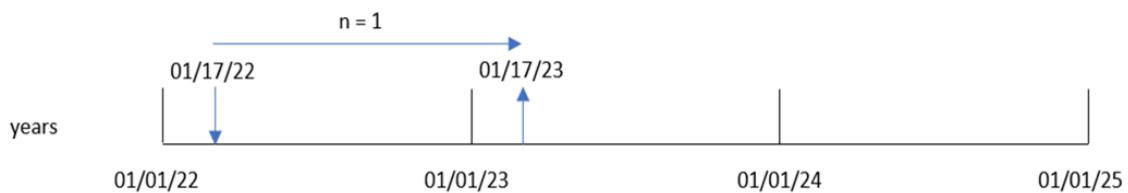
This function returns the date occurring **n** years after **startdate** or, if **n** is negative, the date occurring **n** years before **startdate**.

Syntax:

```
AddYears (startdate, n)
```

Return data type: dual

Example diagram of addyears() function



The addyears() function adds or subtracts a defined number of years, **n**, from a **startdate**. It then returns the resulting date.

Arguments

Argument	Description
startdate	The start date as a time stamp, for example '2012-10-12'.
n	Number of years as a positive or negative integer.

Function examples

Example	Result
<code>addyears ('01/29/2010',3)</code>	Returns '01/29/2013'.
<code>addyears ('01/29/2010',-1)</code>	Returns '01/29/2009'.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Simple example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `two_years_later`, that returns the date for two years after the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    addyears(date,2) as two_years_later
  ;
Load
*
Inline
[
id,date,amount
```

```
8188, '01/10/2020', 37.23  
8189, '02/28/2020', 17.17  
8190, '04/09/2020', 88.27  
8191, '04/16/2020', 57.42  
8192, '05/21/2020', 53.80  
8193, '08/14/2020', 82.06  
8194, '10/07/2020', 40.39  
8195, '12/05/2020', 87.21  
8196, '01/22/2021', 95.93  
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '12/27/2021', 25.12  
8204, '02/02/2022', 46.23  
8205, '02/26/2022', 84.21  
8206, '03/07/2022', 96.24  
8207, '03/11/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- two_years_later

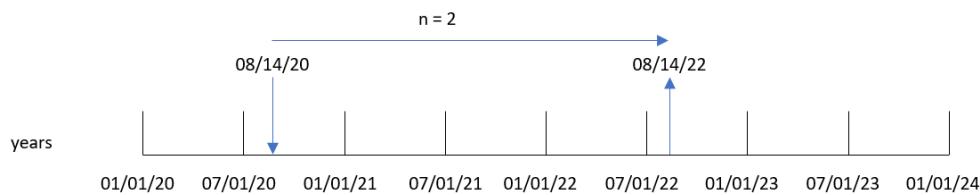
Results table

date	two_years_later
01/10/2020	01/10/2022
02/28/2020	02/28/2022
04/09/2020	04/09/2022
04/16/2020	04/16/2022
05/21/2020	05/21/2022
08/14/2020	08/14/2022
10/07/2020	10/07/2022
12/05/2020	12/05/2022
01/22/2021	01/22/2023
02/03/2021	02/03/2023
03/17/2021	03/17/2023
04/23/2021	04/23/2023

date	two_years_later
05/04/2021	05/04/2023
06/30/2021	06/30/2023
07/26/2021	07/26/2023
12/27/2021	12/27/2023
02/02/2022	02/02/2024
02/26/2022	02/26/2024
03/07/2022	03/07/2024
03/11/2022	03/11/2024

The `two_years_later` field is created in the preceding load statement by using the `addyears()` function. The first argument provided identifies which date is being evaluated. The second argument is the number of years to add or subtract from the start date. In this instance, the value of 2 is provided.

Diagram of `addyears()` function, basic example



Transaction 8193 took place on August 14, 2020. Therefore, the `addyears()` function returns August 14, 2022 for the `two_years_later` field.

Example 2 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

In a chart object, create a measure, `prior_year_date`, that returns the date one year prior to when the transaction takes place.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure to calculate the date one year prior to each transaction:

```
=addyears(date,-1)
```

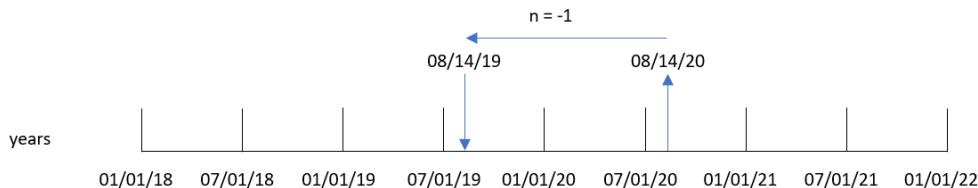
Results table

date	=addyears(date,-1)
01/10/2020	01/10/2019
02/28/2020	02/28/2019
04/09/2020	04/09/2019
04/16/2020	04/16/2019
05/21/2020	05/21/2019
08/14/2020	08/14/2019

date	=addyears(date,-1)
10/07/2020	10/07/2019
12/05/2020	12/05/2019
01/22/2021	01/22/2020
02/03/2021	02/03/2020
03/17/2021	03/17/2020
04/23/2021	04/23/2020
05/04/2021	05/04/2020
06/30/2021	06/30/2020
07/26/2021	07/26/2020
12/27/2021	12/27/2020
02/02/2022	02/02/2021
02/26/2022	02/26/2021
03/07/2022	03/07/2021
03/11/2022	03/11/2021

The `one_year_prior` measure is created in the chart object by using the `addyears()` function. The first argument provided identifies which date is being evaluated. The second argument is the number of years to add or subtract from the `startdate`. In this instance, the value of `-1` is provided.

Diagram of addyears() function, chart object example



Transaction 8193 took place on August 14. Therefore, the `addyears()` function returns August 14, 2019 for the `one_year_prior` field.

Example 3 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `warranties`.
- Information with the product ID, purchase date, warranty length, and purchase price.

The end user would like a chart object that displays, by product ID, the warranty termination date of each product.

Load script

```
warranties:  
Load  
*  
Inline  
[  
product_id,purchase_date,warranty_length,purchase_price  
8188,'01/13/2020',4,32000  
8189,'02/26/2020',2,28000  
8190,'03/27/2020',3,41000  
8191,'04/16/2020',4,17000  
8192,'05/21/2020',2,25000  
8193,'08/14/2020',1,59000  
8194,'10/07/2020',2,12000  
8195,'12/05/2020',3,12000  
8196,'01/22/2021',4,24000  
8197,'02/03/2021',1,50000  
8198,'03/17/2021',2,80000  
8199,'04/23/2021',3,10000  
8200,'05/04/2021',4,30000  
8201,'06/30/2021',3,30000  
8202,'07/26/2021',4,20000  
8203,'12/27/2021',4,10000  
8204,'06/06/2022',2,25000  
8205,'07/18/2022',1,32000  
8206,'11/14/2022',1,30000  
8207,'12/12/2022',4,22000  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `product_id`
- `purchase_date`
- `warranty_length`

Create the following measure to calculate the end date of each product's warranty:

```
=addyears(purchase_date,warranty_length)
```

Results table

product_id	purchase_date	warranty_length	=addyears(purchase_date,warranty_length)
8188	01/13/2020	4	01/13/2024
8189	02/26/2020	2	02/26/2022
8190	03/27/2020	3	03/27/2023
8191	04/16/2020	4	04/16/2024
8192	05/21/2020	2	05/21/2022
8193	08/14/2020	1	08/14/2021
8194	10/07/2020	2	10/07/2022
8195	12/05/2020	3	12/05/2023
8196	01/22/2021	4	01/22/2025
8197	02/03/2021	1	02/03/2022
8198	03/17/2021	2	03/17/2023
8199	04/23/2021	3	04/23/2024
8200	05/04/2021	4	05/04/2025
8201	06/30/2021	3	06/30/2024
8202	07/26/2021	4	07/26/2025
8203	12/27/2021	4	12/27/2025
8204	06/06/2022	2	06/06/2024
8205	07/18/2022	1	07/18/2023
8206	11/14/2022	1	11/14/2023
8207	12/12/2022	4	12/12/2026

age

The **age** function returns the age at the time of **timestamp** (in completed years) of somebody born on **date_of_birth**.

Syntax:

```
age(timestamp, date_of_birth)
```

Can be an expression.

Return data type: numeric

Arguments:

Arguments

Argument	Description
timestamp	The timestamp, or expression resolving to a timestamp, up to which to calculate the completed number of years.
date_of_birth	Date of birth of the person whose age is being calculated. Can be an expression.

Examples and results:

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Scripting examples

Example	Result
age('25/01/2014', '29/10/2012')	Returns 1.
age('29/10/2014', '29/10/2012')	Returns 2.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

Employees:

```
LOAD * INLINE [
Member|DateofBirth
John|28/03/1989
Linda|10/12/1990
Steve|5/2/1992
Birg|31/3/1993
Raj|19/5/1994
Prita|15/9/1994
Su|11/12/1994
Goran|2/3/1995
Sunny|14/5/1996
Ajoa|13/6/1996
Daphne|7/7/1998
Biffy|4/8/2000
] (delimiter is |);
AgeTable:
Load *,
age('20/08/2015', DateOfBirth) As Age
Resident Employees;
Drop table Employees;
```

The resulting table shows the returned values of age for each of the records in the table.

Results table

Member	DateOfBirth	Age
John	28/03/1989	26
Linda	10/12/1990	24
Steve	5/2/1992	23
Birg	31/3/1993	22
Raj	19/5/1994	21
Prita	15/9/1994	20
Su	11/12/1994	20
Goran	2/3/1995	20
Sunny	14/5/1996	19
Ajoa	13/6/1996	19
Daphne	7/7/1998	17
Biffy	4/8/2000	15

converttolocaltime

Converts a UTC or GMT timestamp to local time as a dual value. The place can be any of a number of cities, places and time zones around the world.

Syntax:

```
ConvertToLocalTime(timestamp [, place [, ignore_dst=false]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
timestamp	The timestamp, or expression resolving to a timestamp, to convert.

Argument	Description
place	<p>A place or timezone from the table of valid places and timezones below. Alternatively, you can use GMT or UTC to define the local time. The following values and time offset ranges are valid:</p> <ul style="list-style-type: none"> • GMT • GMT-12:00 - GMT-01:00 • GMT+01:00 - GMT+14:00 • UTC • UTC-12:00 - UTC-01:00 • UTC+01:00 - UTC+14:00 <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>You can only use standard time offsets. It's not possible to use an arbitrary time offset, for example, GMT-04:27.</i> </div>
ignore_dst	<p>Set to True to ignore DST (daylight saving time).</p> <p>Set to False to adjust for daylight saving time.</p>

Valid places and time zones

A-C	D-K	L-R	S-Z
Abu Dhabi	Darwin	La Paz	Samoa
Adelaide	Dhaka	Lima	Santiago
Alaska	Eastern Time (US & Canada)	Lisbon	Sapporo
Amsterdam	Edinburgh	Ljubljana	Sarajevo
Arizona	Ekaterinburg	London	Saskatchewan
Astana	Fiji	Madrid	Seoul
Athens	Georgetown	Magadan	Singapore
Atlantic Time (Canada)	Greenland	Mazatlan	Skopje
Auckland	Greenwich Mean Time : Dublin	Melbourne	Sofia
Azores	Guadalajara	Mexico City	Solomon Is.
Baghdad	Guam	Mid-Atlantic	Sri Jayawardenepura
Baku	Hanoi	Minsk	St. Petersburg
Bangkok	Harare	Monrovia	Stockholm

5 Script and chart functions

A-C	D-K	L-R	S-Z
Beijing	Hawaii	Monterrey	Sydney
Belgrade	Helsinki	Moscow	Taipei
Berlin	Hobart	Mountain Time (US & Canada)	Tallinn
Bern	Hong Kong	Mumbai	Tashkent
Bogota	Indiana (East)	Muscat	Tbilisi
Brasilia	International Date Line West	Nairobi	Tehran
Bratislava	Irkutsk	New Caledonia	Tokyo
Brisbane	Islamabad	New Delhi	Urumqi
Brussels	Istanbul	Newfoundland	Warsaw
Bucharest	Jakarta	Novosibirsk	Wellington
Budapest	Jerusalem	Nuku'alofa	West Central Africa
Buenos Aires	Kabul	Osaka	Vienna
Cairo	Kamchatka	Pacific Time (US & Canada)	Vilnius
Canberra	Karachi	Paris	Vladivostok
Cape Verde Is.	Kathmandu	Perth	Volgograd
Caracas	Kolkata	Port Moresby	Yakutsk
Casablanca	Krasnoyarsk	Prague	Yerevan
Central America	Kuala Lumpur	Pretoria	Zagreb
Central Time (US & Canada)	Kuwait	Quito	-
Chennai	Kyiv	Riga	-
Chihuahua	-	Riyadh	-
Chongqing	-	Rome	-
Copenhagen	-	-	-

Examples and results:

Scripting examples

Example	Result
ConvertToLocalTime('2007-11-10 23:59:00', 'Paris')	Returns '2007-11-11 00:59:00' and the corresponding internal timestamp representation.
ConvertToLocalTime(UTC(), 'GMT-05:00')	Returns the time for the North American east coast, for example, New York.
ConvertToLocalTime(UTC(), 'GMT-05:00', True)	Returns the time for the North American east coast, for example, New York, without daylight-saving time adjustment.

day

This function returns an integer representing the day when the fraction of the **expression** is interpreted as a date according to the standard number interpretation.

The function returns the day of the month for a particular date. It is commonly used to derive a day field as part of a calendar dimension.

Syntax:

day(expression)

Return data type: integer

Function examples

Example	Result
day(1971-10-12)	returns 12
day(35648)	returns 6, because 35648 = 1997-08-06

Example 1 – DateFormat dataset (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named `Master_Calendar`. The `DateFormat` system variable is set to `DD/MM/YYYY`.
- A preceding load that creates an additional field, named `day_of_month`, using the `day()` function.
- An additional field, named `long_date`, using the `date()` function to express the full month name.

Load script

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
    date,
    date(date,'dd-MMMM-YYYY') as long_date,
    day(date) as day_of_month
Inline
[
date
03/11/2022
03/12/2022
03/13/2022
03/14/2022
03/15/2022
03/16/2022
03/17/2022
03/18/2022
03/19/2022
03/20/2022
03/21/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- long_date
- day_of_month

Results table

date	long_date	day_of_month
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19

date	long_date	day_of_month
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

The day of the month is correctly evaluated by the day() function in the script.

Example 2 – ANSI dates (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named Master_Calendar. The DateFormat system variable DD/MM/YYYY is used. However, the dates that are included in the dataset are in ANSI standard date format.
- A preceding load that creates an additional field, named day_of_month, using the date() function.
- An additional field, named long_date, using the date() function to express the date with the full month name.

Load script

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
    date,
    date(date,'dd-MMMM-YYYY') as long_date,
    day(date) as day_of_month

Inline
[
date
2022-03-11
2022-03-12
2022-03-13
2022-03-14
2022-03-15
2022-03-16
2022-03-17
2022-03-18
2022-03-19
2022-03-20
2022-03-21
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- long_date
- day_of_month

Results table

date	long_date	day_of_month
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

The day of the month is correctly evaluated by the day() function in the script.

Example 3 – Unformatted dates (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named Master_Calendar. The DateFormat system variable DD/MM/YYYY is used.
- A preceding load that creates an additional field, named day_of_month, using the day() function.
- The original unformatted date, named unformatted_date.
- An additional field, named long_date, using the date() is used to convert the numerical date into a formatted date field.

Load script

```
SET DateFormat='DD/MM/YYYY';
```

```
Master_Calendar:  
Load
```

```
unformatted_date,  
date(unformatted_date,'dd-MMMM-YYYY') as long_date,  
day(date) as day_of_month  
  
Inline  
[  
unformatted_date  
44868  
44898  
44928  
44958  
44988  
45018  
45048  
45078  
45008  
45038  
45068  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- unformatted_date
- long_date
- day_of_month

Results table

unformatted_date	long_date	day_of_month
44868	03-November- 2022	3
44898	03-December- 2022	3
44928	02-January- 2023	2
44958	01-February- 2023	1
44988	03-March- 2023	3
45008	23-March- 2023	23
45018	02-April- 2023	2
45038	22-April- 2023	22
45048	02-May- 2023	2
45068	22-May- 2023	22
45078	01-June- 2023	1

The day of the month is correctly evaluated by the day() function in the script.

Example 4 – Calculating expiry month (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of orders placed in March named orders. The table contains three fields:
 - id
 - order_date
 - amount

Load script

```
Orders:  
Load  
    id,  
    order_date,  
    amount  
Inline  
[  
id,order_date,amount  
1,03/01/2022,231.24  
2,03/02/2022,567.28  
3,03/03/2022,364.28  
4,03/04/2022,575.76  
5,03/05/2022,638.68  
6,03/06/2022,785.38  
7,03/07/2022,967.46  
8,03/08/2022,287.67  
9,03/09/2022,764.45  
10,03/10/2022,875.43  
11,03/11/2022,957.35  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: order_date.

To calculate the delivery date, create this measure: =day(order_date+5).

Results table

order_date	=day(order_date+5)
03/11/2022	16

order_date	=day(order_date+5)
03/12/2022	17
03/13/2022	18
03/14/2022	19
03/15/2022	20
03/16/2022	21
03/17/2022	22
03/18/2022	23
03/19/2022	24
03/20/2022	25
03/21/2022	26

The `day()` function correctly determines that an order placed on the 11th of March would be delivered on the 16th based on a 5 day delivery period.

dayend

This function returns a value corresponding to a timestamp of the final millisecond of the day contained in **time**. The default output format will be the **TimestampFormat** set in the script.

Syntax:

```
DayEnd(time[, [period_no[, day_start]])
```

When to use it

The `dayend()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the day that has not yet occurred. For example, to calculate the total expenses still to be incurred during the day.

Return data type: dual

Arguments

Argument	Description
time	The timestamp to evaluate.
period_no	period_no is an integer, or expression that resolves to an integer, where the value 0 indicates the day that contains time . Negative values in period_no indicate preceding days and positive values indicate succeeding days.
day_start	To specify that days do not start at midnight, indicate an offset as a fraction of a day in day_start . For example, 0.125 to denote 3:00 AM. In other words, to create the offset, divide the start time by 24 hours. For example, for a day to begin at 7:00 AM, use the fraction 7/24.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>dayend('01/25/2013 16:45:00')</code>	Returns 01/25/2013 23:59:59. PM
<code>dayend('01/25/2013 16:45:00', -1)</code>	Returns 01/24/2013 23:59:59. PM
<code>dayend('01/25/2013 16:45:00', 0, 0.5)</code>	Returns 01/26/2013 11:59:59. PM

Example 1 - Basic script

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a list of dates is loaded into a table named "Calendar".
- The default `DateFormat` system variable (MM/DD/YYYY).
- A preceding load to create an additional field, 'EOD_timestamp', using the `dayend()` function.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Calendar:
  Load
    date,
    dayend(date) as EOD_timestamp
  ;
Load
date
Inline
[
date
03/11/2022 1:47:15 AM
```

```
03/12/2022 4:34:58 AM  
03/13/2022 5:15:55 AM  
03/14/2022 9:25:14 AM  
03/15/2022 10:06:54 AM  
03/16/2022 10:44:42 AM  
03/17/2022 11:33:30 AM  
03/18/2022 12:58:14 PM  
03/19/2022 4:23:12 PM  
03/20/2022 6:42:15 PM  
03/21/2022 7:41:16 PM  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- EOD_timestamp

Results table

date	EOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 11:59:59 PM
03/12/2022 4:34:58 AM	3/12/2022 11:59:59 PM
03/13/2022 5:15:55 AM	3/13/2022 11:59:59 PM
03/14/2022 9:25:14 AM	3/14/2022 11:59:59 PM
03/15/2022 10:06:54 AM	3/15/2022 11:59:59 PM
03/16/2022 10:44:42 AM	3/16/2022 11:59:59 PM
03/17/2022 11:33:30 AM	3/17/2022 11:59:59 PM
03/18/2022 12:58:14 PM	3/18/2022 11:59:59 PM
03/19/2022 4:23:12 PM	3/19/2022 11:59:59 PM
03/20/2022 6:42:15 PM	3/20/2022 11:59:59 PM
03/21/2022 7:41:16 PM	3/21/2022 11:59:59 PM

As you can see in the table above, the end of day timestamp is generated for each date in our dataset. The timestamp is in the format of the system variable `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT`.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

You will load a dataset containing service bookings into a table named 'Services'.

The dataset includes the following fields:

- service_id
- service_date
- amount

You will create two new fields in the table:

- deposit_due_date: The date when the deposit should be received. This is the end of the day three days before the service_date.
- final_payment_due_date: The date when the final payment should be received. This is the end of the day seven days after the service_date.

The two fields above are created in a preceding load using the dayend() function and they supply the first two parameters, time and period_no.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Services:
Load
  *,
  dayend(service_date,-3) as deposit_due_date,
  dayend(service_date,7) as final_payment_due_date
;

Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `service_date`
- `deposit_due_date`
- `final_payment_due_date`

Results table

<code>service_date</code>	<code>deposit_due_date</code>	<code>final_payment_due_date</code>
03/11/2022 9:25:14 AM	3/8/2022 11:59:59 PM	3/18/2022 11:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 11:59:59 PM	3/19/2022 11:59:59 PM
03/13/2022 10:44:42 AM	3/10/2022 11:59:59 PM	3/20/2022 11:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 11:59:59 PM	3/21/2022 11:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 11:59:59 PM	3/22/2022 11:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 11:59:59 PM	3/23/2022 11:59:59 PM
03/17/2022 6:42:15 PM	3/14/2022 11:59:59 PM	3/24/2022 11:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 11:59:59 PM	3/25/2022 11:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 11:59:59 PM	3/26/2022 11:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 11:59:59 PM	3/27/2022 11:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 11:59:59 PM	3/28/2022 11:59:59 PM

The values of the new fields are in the `TimestampFormat M/D/YYYY h:mm:ss[.fff]`. Because the function `dayend()` was used, the timestamp values are all the last millisecond of the day.

The deposit due date values are three days before the service date because the second argument passed in the `dayend()` function is negative.

The final payment due date values are seven days after the service date because the second argument passed in the `dayend()` function is positive.

Example 3 – day_start script

Load script and results

Overview

Open the Data load editor and add the load script below in a new tab.

The dataset and scenario used in this example is the same as in the previous example.

As in the previous example, you will create two new fields:

- `deposit_due_date`: The date when the deposit should be received. This is the end of the day three days before the `service_date`.
- `final_payment_due_date`: The date when the final payment should be received. This is the end of the day seven days after the `service_date`.

However, your company would like to operate under a policy where the working day begins at 5 PM and ends at 5 PM the following day. Your company can then monitor transactions that occur in those working hours.

To achieve these requirements, the two fields above are created in a preceding load using the `dayend()` function and use all three arguments, `time`, `period_no`, and `day_start`.

Load Script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Services:
  Load
    *,
    dayend(service_date,-3,17/24) as deposit_due_date,
    dayend(service_date,7,17/24) as final_payment_due_date
  ;
  Load
  service_id,
  service_date,
  amount
  Inline
  [
  service_id, service_date,amount
  1,03/11/2022 9:25:14 AM,231.24
  2,03/12/2022 10:06:54 AM,567.28
  3,03/13/2022 10:44:42 AM,364.28
  4,03/14/2022 11:33:30 AM,575.76
  5,03/15/2022 12:58:14 PM,638.68
  6,03/16/2022 4:23:12 PM,785.38
  7,03/17/2022 6:42:15 PM,967.46
  8,03/18/2022 7:41:16 PM,287.67
  9,03/19/2022 8:14:15 PM,764.45
  10,03/20/2022 9:23:51 PM,875.43
  11,03/21/2022 10:04:41 PM,957.35
  ];
;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `service_date`
- `deposit_due_date`
- `final_payment_due_date`

Results table

service_date	deposit_due_date	final_payment_due_date
03/11/2022 9:25:14 AM	3/8/2022 4:59:59 PM	3/18/2022 4:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 4:59:59 PM	3/19/2022 4:59:59 PM

service_date	deposit_due_date	final_payment_due_date
03/13/2022 10:44:42 AM	3/10/2022 4:59:59 PM	3/20/2022 4:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 4:59:59 PM	3/21/2022 4:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 4:59:59 PM	3/22/2022 4:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 4:59:59 PM	3/23/2022 4:59:59 PM
03/17/2022 6:42:15 PM	3/14/2022 4:59:59 PM	3/24/2022 4:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 4:59:59 PM	3/25/2022 4:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 4:59:59 PM	3/26/2022 4:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 4:59:59 PM	3/27/2022 4:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 4:59:59 PM	3/28/2022 4:59:59 PM

While the dates remain the same as in Example 2, the dates now have a timestamp of the last millisecond before 5:00 PM because the value of the third argument, day_start, passed into the dayend() function is 17/24.

Example 4 – Chart example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The dataset and scenario used in this example is the same as in the previous two examples. The company would like to operate under a policy where the working day begins at 5:00 PM and ends at 5:00 PM the following day.

As in the previous example, you will create two new fields:

- **deposit_due_date**: The date when the deposit should be received. This is the end of the day three days before the **service_date**.
- **final_payment_due_date**: The date when the final payment should be received. This is the end of the day seven days after the **service_date**.

Load Script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Services:
Load
service_id,
service_date,
amount
InLine
[
service_id, service_date,amount
```

```
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

`service_date`.

To create the `deposit_due_date` field, create this measure:

`=dayend(service_date,-3,17/24)`.

Then, to create the `final_payment_due_date` field, create this measure:

`=dayend(service_date,7,17/24)`.

Results table

<code>service_date</code>	<code>=dayend(service_date,-3,17/24)</code>	<code>=dayend(service_date,7,17/24)</code>
03/11/2022	3/8/2022 16:59:59 PM	3/18/2022 16:59:59 PM
03/12/2022	3/9/2022 16:59:59 PM	3/19/2022 16:59:59 PM
03/13/2022	3/10/2022 16:59:59 PM	3/20/2022 16:59:59 PM
03/14/2022	3/11/2022 16:59:59 PM	3/21/2022 16:59:59 PM
03/15/2022	3/12/2022 16:59:59 PM	3/22/2022 16:59:59 PM
03/16/2022	3/13/2022 16:59:59 PM	3/23/2022 16:59:59 PM
03/17/2022	3/14/2022 16:59:59 PM	3/24/2022 16:59:59 PM
03/18/2022	3/15/2022 16:59:59 PM	3/25/2022 16:59:59 PM
03/19/2022	3/16/2022 16:59:59 PM	3/26/2022 16:59:59 PM
03/20/2022	3/17/2022 16:59:59 PM	3/27/2022 16:59:59 PM
03/21/2022	3/18/2022 16:59:59 PM	3/28/2022 16:59:59 PM

The values of the new fields are in the `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT`. Because the function `dayend()` was used, the timestamp values are all the last millisecond of the day.

The payment due date values are three days before the service date because the second argument passed in the dayend() function is negative.

The final payment due date values are seven days after the service date because the second argument passed in the dayend() function is positive.

The dates have a timestamp of the last millisecond before 5:00 PM because the value of the third argument, day_start, that passed into the dayend() function is 17/24.

Arguments

Argument	Description
time	The timestamp to evaluate.
period_no	period_no is an integer, or expression that resolves to an integer, where the value 0 indicates the day that contains time . Negative values in period_no indicate preceding days and positive values indicate succeeding days.
day_start	To specify that days do not start at midnight, indicate an offset as a fraction of a day in day_start . For example, 0.125 to denote 3:00 AM.

daylightsaving

Returns the current adjustment for daylight saving time, as defined in Windows.

Syntax:

```
DaylightSaving( )
```

Return data type: dual

Example:

```
daylightsaving( )
```

dayname

This function returns a value showing the date with an underlying numeric value corresponding to a timestamp of the first millisecond of the day containing **time**.

Syntax:

```
DayName(time[, period_no [, day_start]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
time	The timestamp to evaluate.

Argument	Description
period_no	period_no is an integer, or expression that resolves to an integer, where the value 0 indicates the day that contains time . Negative values in period_no indicate preceding days and positive values indicate succeeding days.
day_start	To specify that days do not start at midnight, indicate an offset as a fraction of a day in day_start . For example, 0.125 to denote 3:00 AM.

Examples and results:

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Scripting examples

Example	Result
<code>dayname('25/01/2013 16:45:00')</code>	Returns 25/01/2013.
<code>dayname('25/01/2013 16:45:00', -1)</code>	Returns 24/01/2013.
<code>dayname('25/01/2013 16:45:00', 0, 0.5)</code>	Returns 25/01/2013. Displaying the full timestamp shows the underlying numeric value corresponds to '25/01/2013 12:00:00.000'.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

In this example, the day name is created from the timestamp that marks the beginning of the day after each invoice date in the table.

```
TempTable:  
LOAD RecNo() as InvID, * Inline [  
InvDate  
28/03/2012  
10/12/2012  
5/2/2013  
31/3/2013  
19/5/2013  
15/9/2013  
11/12/2013  
2/3/2014  
14/5/2014  
13/6/2014  
7/7/2014  
4/8/2014  
];
```

```
InvoiceData:  
LOAD *,
```

```
DayName(InvDate, 1) AS DName
Resident TempTable;
Drop table TempTable;
```

The resulting table contains the original dates and a column with the return value of the dayname() function. You can display the full timestamp by specifying the formatting in the properties panel.

Results table

InvDate	DName
28/03/2012	29/03/2012 00:00:00
10/12/2012	11/12/2012 00:00:00
5/2/2013	07/02/2013 00:00:00
31/3/2013	01/04/2013 00:00:00
19/5/2013	20/05/2013 00:00:00
15/9/2013	16/09/2013 00:00:00
11/12/2013	12/12/2013 00:00:00
2/3/2014	03/03/2014 00:00:00
14/5/2014	15/05/2014 00:00:00
13/6/2014	14/06/2014 00:00:00
7/7/2014	08/07/2014 00:00:00
4/8/2014	05/08/2014 00:00:00

daynumberofquarter

This function calculates the day number of the quarter in which a timestamp falls. This function is used when creating a Master Calendar.

Syntax:

```
DayNumberOfQuarter(timestamp[, start_month])
```

Return data type: integer

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
start_month	By specifying a start_month between 2 and 12 (1, if omitted), the beginning of the year may be moved forward to the first day of any month. For example, if you want to work with a fiscal year starting March 1, specify start_month = 3.

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>DayNumberofQuarter('12/09/2014')</code>	Returns 74, the day number of the current quarter.
<code>DayNumberofQuarter('12/09/2014', 3)</code>	Returns 12, the day number of the current quarter. In this case, the first quarter starts with March (because start_month is specified as 3). This means that the current quarter is the third quarter, which started on September 1.

Example 1 – January start of year (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A simple dataset containing a list of dates, which is loaded into a table named calendar. The default `DateFormat` system variable MM/DD/YYYY is used.
- A preceding load that creates an additional field, named `DayNrQtr`, using the `DayNumberofQuarter()` function.

Aside from the date, no additional parameters are provided to the function.

Load script

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
    date,
    DayNumberofQuarter(date) as DayNrQtr
;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- daynrqtr

Results table

date	daynrqtr
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

The first day of the year is January 1 because no second argument was passed into the DayNumberOfQuarter() function.

January 1st is the 1st day of the quarter whilst February 1st is the 32nd day of the quarter. The 31st of March is the 91st and final day of the quarter, whilst the 1st of April is the 1st day of the 2nd Quarter.

Example 2 – February start of year (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default DateFormat system variable MM/DD/YYYY is used.
- A start_month argument beginning on February 1. This sets the financial year to February 1.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Calendar:

```
Load
    date,
    DayNumberOfQuarter(date,2) as DayNrQtr
;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- daynrqtr

Results table

date	daynrqtr
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

The first day of the year is the 1st of February because the second argument passed into the `DayNumberOfQuarter()` function was 2.

The first quarter of the year operates between February and April whilst the fourth quarter operates between November and January. This is shown in the results table where February 1st is the 1st day of the quarter whilst January 31st is the 92nd and last day of the quarter.

Example 3 – January start of year (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default `DateFormat` system variable MM/DD/YYYY is used.

However, in this example, the unchanged dataset is loaded into the application. The value of the day of the quarter is calculated via a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Calendar:

```
Load  
date  
Inline  
[  
date  
01/01/2022  
01/10/2022  
01/31/2022  
02/01/2022  
02/10/2022  
02/28/2022  
03/01/2022  
03/31/2022  
04/01/2022  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=daynumberofquarter(date)
```

Results table

date	=daynumberofquarter(date)
01/01/2022	1
01/10/2022	10

date	=daynumberofquarter(date)
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

The first day of the year is the 1st of January because no second argument passed into the DayNumberOfQuarter() function.

January 1st is the 1st day of the quarter whilst February 1st is the 32nd day of the quarter. The 31st of March is the 91st and final day of the quarter, whilst the 1st of April is the 1st day of the 2nd Quarter.

Example 4 – February start of year (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default DateFormat system variable MM/DD/YYYY is used.
- The financial year runs from the 1st of February to the 31st of January.

However, in this example, the unchanged dataset is loaded into the application. The value of the day of the quarter is calculated via a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Calendar:

```
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
```

```
02/10/2022  
02/28/2022  
03/01/2022  
03/31/2022  
04/01/2022  
];
```

Chart object

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=daynumberofquarter(date,2)
```

Results

Results table

date	=daynumberofquarter(date,2)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

The first day of the year is the 1st of January because the second argument passed into the DayNumberOfQuarter() function was 2.

The first quarter of the year operates between February and April whilst the fourth quarter operates between November and January. This is evidenced in the results table where February 1st is the 1st day of the quarter whilst January 31st is the 92nd and last day of the quarter.

daynumberofyear

This function calculates the day number of the year in which a timestamp falls. The calculation is made from the first millisecond of the first day of the year, but the first month can be offset.

Syntax:

```
DayNumberOfYear(timestamp[, start_month])
```

Return data type: integer

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
start_month	By specifying a start_month between 2 and 12 (1, if omitted), the beginning of the year may be moved forward to the first day of any month. For example, if you want to work with a fiscal year starting March 1, specify start_month = 3.

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>DayNumberofYear('12/09/2014')</code>	Returns 256, the day number counted from the first of the year.
<code>DayNumberofYear('12/09/2014', 3)</code>	Returns 196, the number of the day, as counted from 1 March.

Example 1 – January start of year (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A simple dataset containing a list of dates, which is loaded into a table named `calendar`. The default `DateFormat` system variable `MM/DD/YYYY` is used.
- A preceding load that creates an additional field, named `daynryear`, using the `DayNumberofYear()` function.

Aside from the date, no additional parameters are provided to the function.

Load script

```
SET DateFormat='MM/DD/YYYY';

Calendar:
Load
    date,
    DayNumberofYear(date) as daynryear
;
Load
date
Inline
```

```
[  
date  
01/01/2022  
01/10/2022  
01/31/2022  
02/01/2022  
02/10/2022  
06/30/2022  
07/26/2022  
10/31/2022  
11/01/2022  
12/31/2022  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- daynryear

Results table

date	daynryear
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

The first day of the year is the 1st of January because no second argument was passed into the DayNumberOfYear() function.

January 1st is the 1st day of the quarter whilst February 1st is the 32nd day of the year. The 30th of June is the 182nd whilst the 31st of December is the 366th and final day of the year.

Example 2 – November start of year (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default `DateFormat` system variable MM/DD/YYYY is used
- A `start_month` argument beginning on November 1. This sets the financial year to November 1.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Calendar:

```
Load
    date,
    DayNumberOfYear(date,11) as daynryear
;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `daynryear`

Results table

date	daynryear
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269
10/31/2022	366
11/01/2022	1
12/31/2022	61

The first day of the year is the 1st of November because the second argument passed into the `DayNumberofYear()` function was 11.

January 1st is the 1st day of the quarter whilst February 1st is the 32nd day of the year. The 30th of June is the 182nd whilst the 31st of December is the 366th and final day of the year.

Example 3 – January start of year (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default `DateFormat` system variable `MM/DD/YYYY` is used.

However, in this example, the unchanged dataset is loaded into the application. The value of the day of the quarter is calculated via a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Calendar:  
Load  
date  
Inline  
[  
date
```

```
01/01/2022  
01/10/2022  
01/31/2022  
02/01/2022  
02/10/2022  
06/30/2022  
07/26/2022  
10/31/2022  
11/01/2022  
12/31/2022  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=daynumberofyear(date)
```

Results table

date	=daynumberofyear(date)
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

The first day of the year is the 1st of January because no second argument was passed into the DayNumberOfYear() function.

January 1st is the 1st day of the year whilst February 1st is the 32nd day of the year. The 30th of June is the 182nd whilst the 31st of December is the 366th and final day of the year.

Example 4 – November start of year (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default `DateFormat` system variable MM/DD/YYYY is used.
- The financial year runs from the 1st of November to the 31st of October.

However, in this example, the unchanged dataset is loaded into the application. The value of the day of the year is calculated via a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=daynumberofyear(date)
```

Results table

date	=daynumberofyear(date,11)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269

date	=daynumberofyear(date,11)
10/31/2022	366
11/01/2022	1
12/31/2022	61

The first day of the year is the 1st of November because the second argument passed into the `DayNumberofYear()` function was 11.

The financial year operates between November and October. This is shown in the results table where November 1st is the 1st day of the year whilst October 31st is the 366th and last day of the year.

daystart

This function returns a value corresponding to a timestamp with the first millisecond of the day contained in the **time** argument. The default output format will be the **TimestampFormat** set in the script.

Syntax:

```
DayStart(time[, [period_no[, day_start]])
```

Return data type: dual

Arguments

Argument	Description
time	The timestamp to evaluate.
period_no	period_no is an integer, or expression that resolves to an integer, where the value 0 indicates the day that contains time . Negative values in period_no indicate preceding days and positive values indicate succeeding days.
day_start	To specify that days do not start at midnight, indicate an offset as a fraction of a day in day_start . For example, 0.125 to denote 3:00 AM. In other words, to create the offset, divide the start time by 24 hours. For example, for a day to begin at 7:00 AM, use the fraction 7/24.

When to use it

The `daystart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the day that has elapsed thus far. For example, it could be used to calculate the total wages earned by employees in the day so far.

These examples use the timestamp format '`M/D/YYYY h:mm:ss[.fff] TT`'. The timestamp format is specified in the `SET Timestamp` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
daystart('01/25/2013 4:45:00 PM')	Returns 1/25/2013 12:00:00 AM.
daystart('1/25/2013 4:45:00 PM', -1)	Returns 1/24/2013 12:00:00 AM.
daystart('1/25/2013 16:45:00', 0, 0.5)	Returns 1/25/2013 12:00:00 PM.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - Simpleexample

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A simple dataset containing a list of dates, which is loaded into a table named `calendar`.
- The default `TimeStampFormat` system variable (`M/D/YYYY h:mm:ss[.fff] TT`) is used.
- A preceding load which creates an additional field, named `SOD_timestamp`, using the `daystart()` function.

Aside from the date, no additional parameters are provided to the function.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Calendar:
  Load
    date,
    daystart(date) as SOD_timestamp
  ;
Load
date
```

```
Inline
[
date
03/11/2022 1:47:15 AM
03/12/2022 4:34:58 AM
03/13/2022 5:15:55 AM
03/14/2022 9:25:14 AM
03/15/2022 10:06:54 AM
03/16/2022 10:44:42 AM
03/17/2022 11:33:30 AM
03/18/2022 12:58:14 PM
03/19/2022 4:23:12 PM
03/20/2022 6:42:15 PM
03/21/2022 7:41:16 PM
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- SOD_timestamp

Results table

date	SOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 12:00:00 AM
03/12/2022 4:34:58 AM	3/12/2022 12:00:00 AM
03/13/2022 5:15:55 AM	3/13/2022 12:00:00 AM
03/14/2022 9:25:14 AM	3/14/2022 12:00:00 AM
03/15/2022 10:06:54 AM	3/15/2022 12:00:00 AM
03/16/2022 10:44:42 AM	3/16/2022 12:00:00 AM
03/17/2022 11:33:30 AM	3/17/2022 12:00:00 AM
03/18/2022 12:58:14 PM	3/18/2022 12:00:00 AM
03/19/2022 4:23:12 PM	3/19/2022 12:00:00 AM
03/20/2022 6:42:15 PM	3/20/2022 12:00:00 AM
03/21/2022 7:41:16 PM	3/21/2022 12:00:00 AM

As can be seen in the table above, the end of day timestamp is generated for each date in our dataset. The timestamp is in the format of the system variable `TimestampFormat M/D/YYYY h:mm:ss[.fff] tt`.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing parking fines, which is loaded into a table named `Fines`. The dataset includes the following fields:
 - `id`
 - `due_date`
 - `number_plate`
 - `amount`
- A preceding load using the `daystart()` function and supplying all three parameters: `time`, `period_no`, and `day_start`. This preceding load creates the following two new date fields:
 - An `early_repayment_period` date field, beginning seven days before the payment is due.
 - A `late_penalty_period` date field, beginning 14 days after the payment is due.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Fines:
Load
  *,
  daystart(due_date,-7) as early_repayment_period,
  daystart(due_date,14) as late_penalty_period
;
Load
*
Inline
[
id, due_date, number_plate,amount
1,02/11/2022, 573RJG,50.00
2,03/25/2022, SC41854,50.00
3,04/14/2022, 8EHZ378,50.00
4,06/28/2022, 8HSS198,50.00
5,08/15/2022, 1221665,50.00
6,11/16/2022, EAK473,50.00
7,01/17/2023, KD6822,50.00
8,03/22/2023, 1GGLB,50.00
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- due_date
- early_repayment_period
- late_penalty_period

Results table

due_date	early_repayment_period	late_penalty_period
02/11/2022 9:25:14 AM	2/4/2022 12:00:00 AM	2/25/2022 12:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 12:00:00 AM	4/8/2022 12:00:00 AM
04/14/2022 10:44:42 AM	4/7/2022 12:00:00 AM	4/28/2022 12:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 12:00:00 AM	7/12/2022 12:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 12:00:00 AM	8/29/2022 12:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 12:00:00 AM	11/30/2022 12:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 12:00:00 AM	1/31/2023 12:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 12:00:00 AM	4/5/2023 12:00:00 AM

The values of the new fields are in the `TimestampFormat M/DD/YYYY tt`. Because the function `daystart()` was used, the timestamp values are all the first millisecond of the day.

The early repayment period values are seven days before the due date, as a result of the second argument being passed in the `daystart()` function being negative.

The late repayment period values are 14 days after the due date, as a result of the second argument being passed in the `daystart()` function being positive.

Example 3 - day_start

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the previous example.
- The same preceding load as the previous example.

In this example, we set the working day to begin and end at 7:00 AM each day.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Fines:

Load

```

*,  

daystart(due_date,-7,7/24) as early_repayment_period,  

daystart(due_date,14, 7/24) as late_penalty_period  

;  

Load  

*  

Inline  

[  

id, due_date, number_plate,amount  

1,02/11/2022, 573RJG,50.00  

2,03/25/2022, SC41854,50.00  

3,04/14/2022, 8EHZ378,50.00  

4,06/28/2022, 8HSS198,50.00  

5,08/15/2022, 1221665,50.00  

6,11/16/2022, EAK473,50.00  

7,01/17/2023, KD6822,50.00  

8,03/22/2023, 1GGLB,50.00  

];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- due_date
- early_repayment_period
- late_penalty_period

Results table

due_date	early_repayment_period	late_penalty_period
02/11/2022	2/3/2022 7:00:00 AM	2/24/2022 7:00:00 AM
03/25/2022	3/17/2022 7:00:00 AM	4/7/2022 7:00:00 AM
04/14/2022	4/6/2022 7:00:00 AM	4/27/2022 7:00:00 AM
06/28/2022	6/20/2022 7:00:00 AM	7/11/2022 7:00:00 AM
08/15/2022	8/7/2022 7:00:00 AM	8/28/2022 7:00:00 AM
11/16/2022	11/8/2022 7:00:00 AM	11/29/2022 7:00:00 AM
01/17/2023	1/9/2023 7:00:00 AM	1/30/2023 7:00:00 AM
03/22/2023	3/14/2023 7:00:00 AM	4/4/2023 7:00:00 AM

The dates now have a timestamp of 7:00 AM because the value of the day_start argument which was passed into the daystart() function was 7/24. This sets the beginning of the day to 7:00 AM.

Because the due_date field does not have a timestamp, it is treated as 12:00 AM, which is thus still part of the previous day, since the days start and end at 7:00 AM. Therefore, the early repayment period for a fine due on February 11 begins on February 3 at 7:00 AM.

Example 4 - Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

This example uses the same dataset and scenario as the previous example.

However, only the original `Fines` table is loaded into the application, with the two additional due dates values being calculated in a chart object.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Fines:
    Load
*
Inline
[
id, due_date, numer_plate,amount
1,02/11/2022 9:25:14 AM, 573RJG,50.00
2,03/25/2022 10:06:54 AM, SC41854,50.00
3,04/14/2022 10:44:42 AM, 8EHZ378,50.00
4,06/28/2022 11:33:30 AM, 8HSS198,50.00
5,08/15/2022 12:58:14 PM, 1221665,50.00
6,11/16/2022 4:23:12 PM, EAK473,50.00
7,01/17/2023 6:42:15 PM, KD6822,50.00
8,03/22/2023 7:41:16 PM, 1GGLB,50.00
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add this field as a dimension: `due_date`.
2. To create the `early_repayment_period` field, create the following measure:
`=daystart(due_date,-7,7/24)`
3. To create the `late_penalty_period` field, create the following measure:
`=daystart(due_date,14,7/24)`

Results table

<code>due_date</code>	<code>=daystart(due_date,-7,7/24)</code>	<code>=daystart(due_date,14,7/24)</code>
02/11/2022 9:25:14 AM	2/4/2022 7:00:00 AM	2/25/2022 7:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 7:00:00 AM	4/8/2022 7:00:00 AM

due_date	=daystart(due_date,-7,7/24)	=daystart(due_date,14,7/24)
04/14/2022 10:44:42 AM	4/7/2022 7:00:00 AM	4/28/2022 7:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 7:00:00 AM	7/12/2022 7:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 7:00:00 AM	8/29/2022 7:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 7:00:00 AM	11/30/2022 7:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 7:00:00 AM	1/31/2023 7:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 7:00:00 AM	4/5/2023 7:00:00 AM

The values of the new fields are in the `TimestampFormat M/D/YYYY h:mm:ss[.ffff] TT`. Because the `daystart()` function was used, the timestamp values correspond to the first millisecond of the day.

The early repayment period values are seven days before the due date, since the second argument passed in the `daystart()` function was negative.

The late repayment period values are 14 days after the due date, since the second argument passed in the `daystart()` function was positive.

The dates have a timestamp of 7:00 AM because the value of the third argument passed into the `daystart()` function, `day_start`, was 7/24.

firstworkdate

The **firstworkdate** function returns the latest starting date to achieve **no_of_workdays** (Monday-Friday) ending no later than **end_date** taking into account any optionally listed holidays. **end_date** and **holiday** should be valid dates or timestamps.

Syntax:

```
firstworkdate(end_date, no_of_workdays {, holiday} )
```

Return data type: integer

Arguments:

Arguments

Argument	Description
end_date	The timestamp of end date to evaluate.
no_of_workdays	The number of working days to achieve.
holiday	Holiday periods to exclude from working days. A holiday is stated as a string constant date. You can specify multiple holiday dates, separated by commas. Example: '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'

5 Script and chart functions

Examples and results:

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Scripting examples

Example	Result
<code>firstworkdate ('29/12/2014', 9)</code>	Returns '17/12/2014'.
<code>firstworkdate ('29/12/2014', 9, '25/12/2014', '26/12/2014')</code>	Returns 15/12/2014 because a holiday period of two days is taken into account.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
ProjectTable:  
LOAD *, recno() as InvID, INLINE [  
EndDate  
28/03/2015  
10/12/2015  
5/2/2016  
31/3/2016  
19/5/2016  
15/9/2016  
] ;  
NrDays:  
Load *,  
FirstWorkDate(EndDate,120) As StartDate  
Resident ProjectTable;  
Drop table ProjectTable;
```

The resulting table shows the returned values of FirstWorkDate for each of the records in the table.

Results table

InvID	EndDate	StartDate
1	28/03/2015	13/10/2014
2	10/12/2015	26/06/2015
3	5/2/2016	24/08/2015
4	31/3/2016	16/10/2015
5	19/5/2016	04/12/2015
6	15/9/2016	01/04/2016

GMT

This function returns the current Greenwich Mean Time, as derived from the regional settings. The function returns values in the `TimestampFormat` system variable format.

Whenever the app is reloaded, any load script table, variable, or chart object that uses the `GMT` function will be adjusted to the latest current Greenwich Mean Time as derived from the system clock.

Syntax:

```
GMT( )
```

Return data type: dual

These examples use the timestamp format `M/D/YYYY h:mm:ss[.fff] TT`. The date format is specified in the `SET TimestampFormat` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>GMT()</code>	3/28/2022 2:47:36 PM

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: `MM/DD/YYYY`. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - Variable (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab. This example will set the current Greenwich Mean Time as a variable in the load script using the `GMT` function.

Load script

```
LET vGMT = GMT();
```

Results

Load the data and create a sheet. Create a text box using the **Text & image** chart object.

Add this measure to the text box:

```
=vGMT
```

The text box should contain a line of text with a date and time, similar to the one shown below:

```
3/28/2022 2:47:36 PM
```

Example 2 - November start of year (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing overdue library books, which is loaded into a table named `overdue`. The default `DateFormat` system variable `MM/DD/YYYY` is used.
- The creation of a new field called `days_overdue`, which calculates how many day overdue each book is.

Load script

```
SET DateFormat='MM/DD/YYYY';

overdue:
  Load
    *,
    Floor(GMT()-due_date) as days_overdue
  ;
Load
*
Inline
[
  cust_id,book_id,due_date
  1,4,01/01/2021,
  2,24,01/10/2021,
  6,173,01/31/2021,
  31,281,02/01/2021,
  86,265,02/10/2021,
  52,465,06/30/2021,
  26,537,07/26/2021,
  92,275,10/31/2021,
  27,455,11/01/2021,
  27,46,12/31/2021
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- due_date
- book_id
- days_overdue

Results table

due_date	book_id	days_overdue
01/01/2021	4	455
01/10/2021	24	446
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

The values in the days_overdue field are calculated by finding the difference between the current Greenwich Mean Time, using the `GMT()` function, and the original due date. In order to calculate only the days, the results are rounded off to the nearest whole number using the `Floor()` function.

Example 3 - chart object (chart)

Load script and chart expression

Overview

Open the Data load editor, and add the load script below to a new tab. The load script contains the same dataset as the previous example. The default `DateFormat` system variable `MM/DD/YYYY` is used.

However, in this example, the unchanged dataset is loaded into the application. The value of the number of days overdue is calculated via a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Overdue:
Load

```
*  
Inline  
[  
cust_id,book_id,due_date  
1,4,01/01/2021,  
2,24,01/10/2021,  
6,173,01/31/2021,  
31,281,02/01/2021,  
86,265,02/10/2021,  
52,465,06/30/2021,  
26,537,07/26/2021,  
92,275,10/31/2021,  
27,455,11/01/2021,  
27,46,12/31/2021  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- due_date
- book_id

Create the following measure:

```
=Floor(GMT() - due_date)
```

Results table

due_date	book_id	=Floor(GMT()-due_date)
01/01/2021	4	455
01/10/2021	24	446
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

The values in the days_overdue field are calculated by finding the difference between the current Greenwich Mean Time, using the `GMT()` function, and the original due date. In order to calculate only the days, the results are rounded off to the nearest whole number using the `Floor()` function.

hour

This function returns an integer representing the hour when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
hour(expression)
```

Return data type: integer

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>hour('09:14:36')</code>	The text string supplied is implicitly converted to a timestamp as it matches the timestamp format defined in the <code>TimestampFormat</code> variable. The expression returns 9.
<code>hour('0.5555')</code>	The expression returns 13 (Because $0.5555 = 13:19:55$).

Example 1 – Variable (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing transactions by timestamp
- The default `Timestamp` system variable (M/D/YYYY h:mm:ss[.fff] TT)

Create a field, 'hour', calculating when purchases took place.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Transactions:
Load
  *,
  hour(date) as hour
;
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- hour

Results table

date	hour
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

The values in the hour field are created by using the `hour()` function and passing the date as the expression in the preceding load statement.

Example 2 – Chart object (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The default `Timestamp` system variable (`M/D/YYYY h:mm:ss[.fff] TT`).

However, in this example, the dataset, unchanged, is loaded into the application. The ‘hour’ values are calculated via a measure in a chart object.

Load Script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate the ‘hour’, create the following measure:

```
=hour(date)
```

Results table

due_date	=hour(date)
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

The values for ‘hour’ are created by using the hour() function and passing the date as the expression in a measure for the chart object.

inday

This function returns True if **timestamp** lies inside the day containing **base_timestamp**.

Syntax:

InDay (timestamp, base_timestamp, period_no[, day_start])

Diagram of inday function



The **inday()** function uses the **base_timestamp** argument to identify which day the timestamp falls into. The start time of the day is, by default, midnight; but you can change the start time of the day by using the **day_start** argument of the **inday()** function. Once this day is defined, the function will return Boolean results when comparing the prescribed timestamp values to that day.

When to use it

The **inday()** function returns a Boolean result. Typically, this type of function will be used as a condition in an **if** expression. This returns an aggregation or calculation dependent on whether a date evaluated occurred in the day of the timestamp in question.

For example, the `inday()` function can be used to identify all equipment manufactured in a given day.

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Arguments

Argument	Description
<code>timestamp</code>	The date and time that you want to compare with <code>base_timestamp</code> .
<code>base_timestamp</code>	Date and time that is used to evaluate the timestamp.
<code>period_no</code>	The day can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the day which contains <code>base_timestamp</code> . Negative values in <code>period_no</code> indicate preceding days and positive values indicate succeeding days.
<code>day_start</code>	If you want to work with days not starting midnight, indicate an offset as a fraction of a day in <code>day_start</code> , For example, 0.125 to denote 3 AM.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)</code>	Returns True
<code>inday ('01/12/2006 12:23:00 PM', '01/13/2006 12:00:00 AM', 0)</code>	Returns False
<code>inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)</code>	Returns False
<code>inday ('01/11/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)</code>	Returns True
<code>inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0, 0.5)</code>	Returns False
<code>inday ('01/12/2006 11:23:00 AM', '01/12/2006 12:00:00 AM', 0, 0.5)</code>	Returns True

Example 1 – Load statement (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing transactions by timestamp which is loaded into a table called `Transactions`.
- A date field which is provided in the `Timestamp` system variable (`M/D/YYYY h:mm:ss[.fff] TT`) format.
- A preceding load which contains the `inDay()` function which is set as the `in_day` field.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
  *,
  inDay(date, '01/05/2022 12:00:00 AM', 0) as in_day
;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_day`

Results table	
date	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0

date	in_day
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

The `in_day` field is created in the preceding load statement by using the `inday()` function and passing the `date` field, a hard-coded timestamp for January 5 and a `period_no` of 0 as the function's arguments.

Example 2 – period_no

Load script and results

Overview

The load script uses the same dataset and scenario that were used in the first example.

However, in this example, the task is to calculate whether the transaction date occurred two days before January 5.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Transactions:
Load
  *,
  inday(date, '01/05/2022 12:00:00 AM', -2) as in_day
;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
```

```
9506, '01/06/2022 8:49:09 AM', 74.23  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_day

Results table

date	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	-1
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	0
01/05/2022 11:29:38 AM	0
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

In this instance, because a period_no of -2 is used as the offset argument in the inday() function, the function determines whether each transaction date took place on January 3. This can be verified in the output table where one transaction returns a Boolean result of TRUE.

Example 3 – day_start

Load script and results

Overview

The load script uses the same dataset and scenario that were used in the previous examples.

However, in this example, the company policy is that the workday begins and ends at 7 AM.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';  
  
Transactions:  
    Load  
        *,  
        inday(date, '01/05/2022 12:00:00 AM', 0, 7/24) as in_day
```

```

;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_day

Results table

date	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	-1
01/04/2022 10:58:34 PM	-1
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	0
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

Because the `start_day` argument of `7/24`, which is 7 AM, is used in the `inday()` function, the function determines whether each transaction date took place on January 4 from 7 AM and January 5 before 7 AM.

This can be verified in the output table where transactions that take place after 7 AM on January 4 return a Boolean result of TRUE whilst transactions that take place after 7 AM on January 5 return a Boolean result of FALSE.

Example 4 – Chart object

Load script and chart expression

Overview

The load script uses the same dataset and scenario that were used in the previous examples.

However, in this example, the dataset is unchanged and loaded into the application. You will calculate to determine if a transaction takes place on January 5 by creating a measure in a chart object.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- date

To calculate whether a transaction takes place on January 5, create the following measure:

```
=inday(date,'01/05/2022 12:00:00 AM',0)
```

Results table

date	inday(date,'01/05/2022 12:00:00 AM',0)
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0

date	inday(date,'01/05/2022 12:00:00 AM',0)
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

Example 5 – Scenario

Load script and results

Overview

In this example, it has been identified that due to equipment error, products that were manufactured on January 5 were defective. The end user would like a chart object that displays, by date, the status of which products that were manufactured were ‘defective’ or ‘faultless’ and the cost of the products manufactured on January 5.

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called ‘Products’.
- The table contains the following fields:
 - product ID
 - manufacture time
 - cost price

Load script

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
```

```
9506, '01/06/2022 8:49:09 AM', 74.23
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

```
=dayname(manufacture_date)
```

Create the following measures:

- =if(only(InDay(manufacture_date,makedate(2022,01,05),0)),'Defective','Faultless')
- =sum(cost_price)

Set the measure's **Number Formatting** to **Money**.

Under **Appearance**, turn off **Totals**.

Results table

dayname (manufacture_date)	=if(only(InDay(manufacture_date,makedate (2022,01,05),0)),'Defective','Faultless')	=sum(cost_ price)
01/01/2022	Faultless	44.67
01/02/2022	Faultless	36.34
01/03/2022	Faultless	51.75
01/04/2022	Faultless	89.69
01/05/2022	Defective	170.78
01/06/2022	Faultless	74.23

The `inday()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured on January 5, the `inday()` function returns a Boolean value of TRUE and marks the products as 'Defective'. For any product returning a value of FALSE, and therefore not manufactured on that day, it marks the products as 'Faultless'.

indaytotime

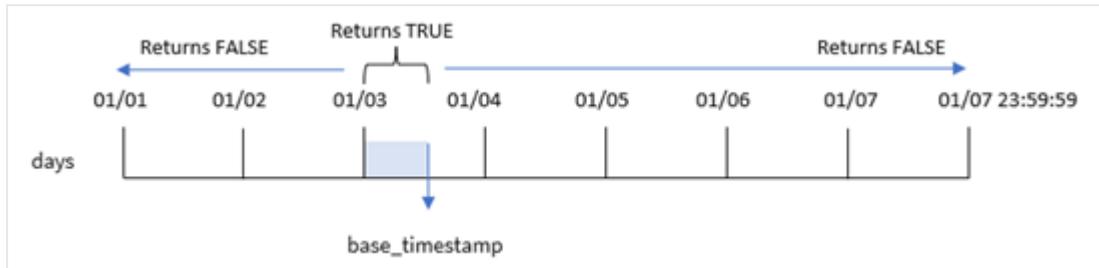
This function returns True if **timestamp** lies inside the part of day containing **base_timestamp** up until and including the exact millisecond of **base_timestamp**.

Syntax:

```
InDayToTime (timestamp, base_timestamp, period_no[, day_start])
```

The `indaytotime()` function returns a Boolean result depending on when a timestamp value occurs during the segment of the day. The start boundary of this segment is the start of the day, which is set as midnight by default; the start of the day can be modified by the `day_start` argument of the `indaytotime()` function. The end boundary of the day segment is determined by a `base_timestamp` argument of the function.

Diagram of `indaytotime` function.



When to use it

The `indaytotime()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. The `indaytotime()` function returns an aggregation or calculation depending on if a timestamp occurred in the segment of the day up to and including the time of the base timestamp.

For example, the `indaytotime()` function can be used to show the sum of ticket sales for shows that have taken place so far today.

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Arguments

Argument	Description
<code>timestamp</code>	The date and time that you want to compare with <code>base_timestamp</code> .
<code>base_timestamp</code>	Date and time that is used to evaluate the timestamp.
<code>period_no</code>	The day can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the day which contains <code>base_timestamp</code> . Negative values in <code>period_no</code> indicate preceding days and positive values indicate succeeding days.
<code>day_start</code>	(optional) If you want to work with days not starting midnight, indicate an offset as a fraction of a day in <code>day_start</code> . For example, use 0.125 to denote 3 AM

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', 0)</code>	Returns True
<code>indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)</code>	Returns False
<code>indaytotime '01/11/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', -1)</code>	Returns True

Example 1 – no additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the period between January 4 and 5 is loaded into a table called 'Transactions'.
- A date field which is provided in the `TimeStamp` system variable (`M/D/YYYY h:mm:ss[.fff] TT`) format.
- A preceding load which contains the `indaytotime()` function which is set as the '`in_day_to_time`', field that determines whether each of the transactions take place before 9:00 AM.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
  *,
  indaytotime(date, '01/05/2022 9:00:00 AM', 0) as in_day_to_time
;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
```

```
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

Results

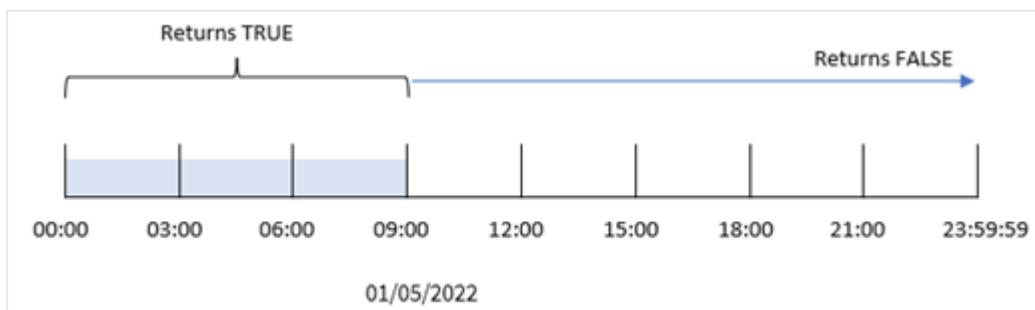
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_day_to_time

Results table

date	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

Example 1 diagram of *indaytotime* function with 9:00 AM limit..



The `in_day_to_time` field is created in the preceding load statement by using the `indaytotime()` function and passing the date field, a hard-coded timestamp for 9:00 AM January 5 and an offset of 0 as the function's arguments. Any transactions that occur between midnight and 9:00 AM on January 5 return TRUE.

Example 2 – period_no

Load script and results

Overview

The load script uses the same dataset and scenario that were used in the first example.

However, in this example, you will calculate whether the transaction date occurred one day before 9:00 AM on January 5.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Transactions:
Load
  *,
  indaytotime(date,'01/05/2022 9:00:00 AM', -1) as in_day_to_time
;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

Results

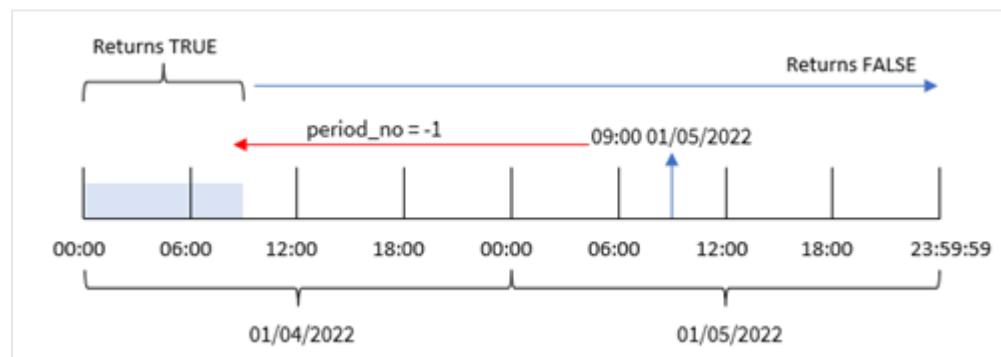
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_day_to_time`

Results table

date	in_day_to_time
01/04/2022 3:41:54 AM	-1
01/04/2022 4:19:43 AM	-1
01/04/2022 04:53:47 AM	-1
01/04/2022 8:38:53 AM	-1
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	0
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

Example 2 diagram of *indaytotime* function with transactions from January 4.



In this example, because an offset of -1 was used as the offset argument in the *indaytotime()* function, the function determines whether each transaction date took place before 9:00 AM on January 4. This can be verified in the output table where a transaction returns a Boolean result of TRUE.

Example 3 – day_start

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the company policy is that the workday begins and ends at 8AM.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Transactions:
Load
  *,
  indaytotime(date, '01/05/2022 9:00:00 AM', 0,8/24) as in_day_to_time
;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

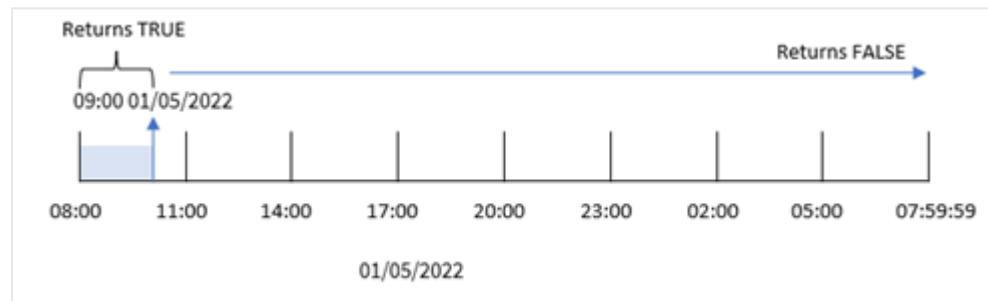
- date
- in_day_to_time

Results table

date	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0

date	in_day_to_time
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

Example 3 diagram of `indaytotime` function with transactions from 8:00 AM to 9:00 AM.,



Because the `start_day` argument of 8/24, which equates to 8:00 AM, is used in the `indaytotime()` function, each day begins and ends at 8:00 AM. Therefore, the `indaytotime()` function will return a Boolean result of TRUE for any transaction that took place between 8:00 AM and 9:00 AM on January 5.

Example 4 – Chart object

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. You will calculate to determine if a transaction takes place on January 5 before 9:00 AM by creating a measure in a chart object.

Load script

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
```

```
8189, '01/04/2022 4:19:43 AM', 87.21  
8190, '01/04/2022 4:53:47 AM', 53.80  
8191, '01/04/2022 8:38:53 AM', 69.98  
8192, '01/04/2022 10:37:52 AM', 57.42  
8193, '01/04/2022 1:54:10 PM', 45.89  
8194, '01/04/2022 5:53:23 PM', 82.77  
8195, '01/04/2022 8:13:26 PM', 36.23  
8196, '01/04/2022 10:00:49 PM', 76.11  
8197, '01/05/2022 7:45:37 AM', 82.06  
8198, '01/05/2022 8:44:36 AM', 17.17  
8199, '01/05/2022 11:26:08 AM', 40.39  
8200, '01/05/2022 6:43:08 PM', 37.23  
8201, '01/05/2022 10:54:10 PM', 88.27  
8202, '01/05/2022 11:09:09 PM', 95.93  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

date.

To determine if a transaction takes place on January 5 before 9:00 AM, create the following measure:

```
=indaytotime(date, '01/05/2022 9:00:00 AM', 0)
```

Results table

date	=indaytotime(date,'01/05/2022 9:00:00 AM',0)
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

The `in_day_to_time` measure is created in the chart object by using the `indaytotime()` function and passing the date field, a hard-coded timestamp for 9:00 AM on January 5 and an offset of 0 as the function's arguments. Any transactions that occur between midnight and 9:00 AM on January 5 return TRUE. This is validated in the results table.

Example 5 – Scenario

Load script and results

Overview

In this example, a dataset containing ticket sales for a local cinema is loaded into a table called `Ticket_Sales`. Today is May 3, 2022 and it is 11:00 AM.

The user would like a KPI chart object to show the revenue earned from all shows that have taken place so far today.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Ticket_Sales:
Load
*
Inline
[
sale ID, show time, ticket price
1,05/01/2022 09:30:00 AM,10.50
2,05/03/2022 05:30:00 PM,21.00
3,05/03/2022 09:30:00 AM,10.50
4,05/03/2022 09:30:00 AM,31.50
5,05/03/2022 09:30:00 AM,10.50
6,05/03/2022 12:00:00 PM,42.00
7,05/03/2022 12:00:00 PM,10.50
8,05/03/2022 05:30:00 PM,42.00
9,05/03/2022 08:00:00 PM,31.50
10,05/04/2022 10:30:00 AM,31.50
11,05/04/2022 12:00:00 PM,10.50
12,05/04/2022 05:30:00 PM,10.50
13,05/05/2022 05:30:00 PM,21.00
14,05/06/2022 12:00:00 PM,21.00
15,05/07/2022 09:30:00 AM,42.00
16,05/07/2022 10:30:00 AM,42.00
17,05/07/2022 10:30:00 AM,10.50
18,05/07/2022 05:30:00 PM,10.50
19,05/08/2022 05:30:00 PM,21.00
20,05/11/2022 09:30:00 AM,10.50
];
```

Results

Do the following:

1. Create a KPI object.
2. Create a measure that will show the sum of all ticket sales for shows that have taken place today so far using the `indaytotime()` function:

```
=sum(if(indaytotime([show time], '05/03/2022 11:00:00 AM',0),[ticket price],0))
```

3. Create a label for the KPI object, ‘Current Revenue’.
4. Set the measure’s **Number Formatting** to **Money**.

The sum total of ticket sales up to 11:00 AM on May 3, 2022 is \$52.50.

The `indaytotime()` function returns a Boolean value when comparing the show times of each of the ticket sales to the current time ('05/03/2022 11:00:00 AM'). For any show on May 3 before 11:00 AM, the `indaytotime()` function returns a Boolean value of TRUE and its ticket price will be included in the sum total.

inlunarweek

This function determines if **timestamp** lies inside the lunar week containing **base_date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week., Apart from the final week of the year, each week will contain exactly seven days.

Syntax:

```
InLunarWeek (timestamp, base_date, period_no[, first_week_day])
```

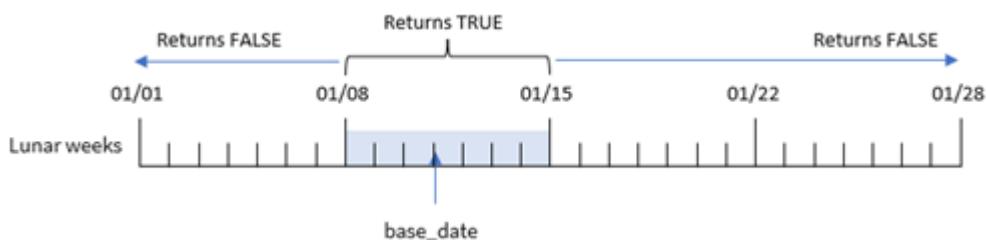
Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

The `inlunarweek()` function determines which lunar week the `base_date` falls into. It then returns a Boolean result once it has determined whether each timestamp value occurs during the same lunar week as the `base_date`.

Diagram of `inlunarweek()` function



When to use it

The `inlunarweek()` function returns a Boolean result. Typically, this type of function will be used as a condition in an IF expression. This would return an aggregation or calculation dependent on whether the date evaluated occurred during the lunar week in question.

5 Script and chart functions

For example, the `inlunarweek()` function can be used to identify all equipment manufactured in a particular lunar week.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the lunar week.
period_no	The lunar week can be offset by period_no . period_no is an integer, where the value 0 indicates the lunar week which contains base_date . Negative values in period_no indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
first_week_day	An offset that may be greater than or less than zero. This changes the beginning of the year by the specified number of days and/or fractions of a day.

Function examples

Example	Result
<code>inlunarweek('01/12/2013', '01/14/2013', 0)</code>	Returns TRUE, since the value of <code>timestamp</code> , 01/12/2013, falls in the week 01/08/2013 to 01/14/2013.
<code>inlunarweek('01/12/2013', '01/07/2013', 0)</code>	Returns FALSE, since the <code>base_date</code> 01/07/2013 is in the lunar week defined as 01/01/2013 to 01/07/2013.
<code>inlunarweek('01/12/2013', '01/14/2013', -1)</code>	Returns FALSE. Specifying a value of <code>period_no</code> as -1 shifts the week to the previous week, 01/01/2013 to 01/07/2013.
<code>inlunarweek('01/07/2013', '01/14/2013', -1)</code>	Returns TRUE. In comparison with the previous example, the <code>timestamp</code> is in the following week, after into account the shift backwards.
<code>inlunarweek('01/11/2006', '01/08/2006', 0, 3)</code>	Returns FALSE. Specifying a value of 3 for <code>first_week_day</code> means that the start of the year is calculated from 01/04/2013. Therefore, the value of <code>base_date</code> falls in the first week, and the value of <code>timestamp</code> falls in the week 01/11/2013 to 01/17/2013.

The `inlunarweek()` function is often used in combination with the following functions:

Related functions

Function	Interaction
<code>lunarweekname</code> (page 831)	This function is used to determine the lunar week number of the year in which an input date occurs.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of transactions for the month of January, which is loaded into a table called `Transactions`.
- The date field has been provided in the `DateFormat` system variable (MM/DD/YYYY) format.

Create a field, `in_lunar_week`, that determines whether the transactions took place in the same lunar week as January 10.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inlunarweek(date,'01/10/2022', 0) as in_lunar_week
  ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
```

```
8190, '1/12/2022', 88.27  
8191, '1/13/2022', 57.42  
8192, '1/14/2022', 53.80  
8193, '1/15/2022', 82.06  
8194, '1/16/2022', 87.21  
8195, '1/17/2022', 95.93  
8196, '1/18/2022', 45.89  
8197, '1/19/2022', 36.23  
8198, '1/20/2022', 25.66  
8199, '1/21/2022', 82.77  
8200, '1/22/2022', 69.98  
8201, '1/23/2022', 76.11  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

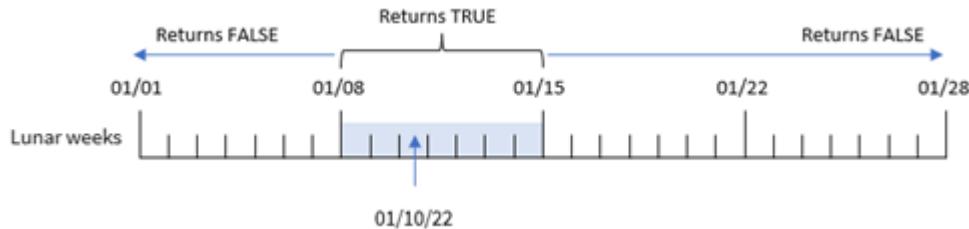
- date
- in_lunar_week

Results table

date	in_lunar_week
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0

date	in_lunar_week
1/22/2022	0
1/23/2022	0

inlunarweek() function, basic example



The `in_lunar_week` field is created in the preceding load statement by using the `inlunarweek()` function, then passing the following as the function's arguments:

- The date field
- A hard-coded date for January 10 as the `base_date`
- A `aperiod_no` of 0

Because lunar weeks begin on January 1, January 10 would fall in the lunar week that begins on January 8 and ends on January 14. Therefore, any transactions that occur between those two dates in January would return a Boolean value of TRUE. This is validated in the results table.

Example 2 - period_no

Examples and results:

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The date field has been provided in the `DateFormat` system variable (MM/DD/YYYY) format.

However, in this example, the task is to create a field, `2_lunar_weeks_later`, that determines whether or not the transactions took place two lunar weeks after January 10.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inlunarweek(date,'01/10/2022', 2) as [2_lunar_weeks_later]
```

```
;  
Load  
*  
Inline  
[  
id,date,amount  
8183,'1/5/2022',42.32  
8184,'1/6/2022',68.22  
8185,'1/7/2022',15.25  
8186,'1/8/2022',25.26  
8187,'1/9/2022',37.23  
8188,'1/10/2022',37.23  
8189,'1/11/2022',17.17  
8190,'1/12/2022',88.27  
8191,'1/13/2022',57.42  
8192,'1/14/2022',53.80  
8193,'1/15/2022',82.06  
8194,'1/16/2022',87.21  
8195,'1/17/2022',95.93  
8196,'1/18/2022',45.89  
8197,'1/19/2022',36.23  
8198,'1/20/2022',25.66  
8199,'1/21/2022',82.77  
8200,'1/22/2022',69.98  
8201,'1/23/2022',76.11  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

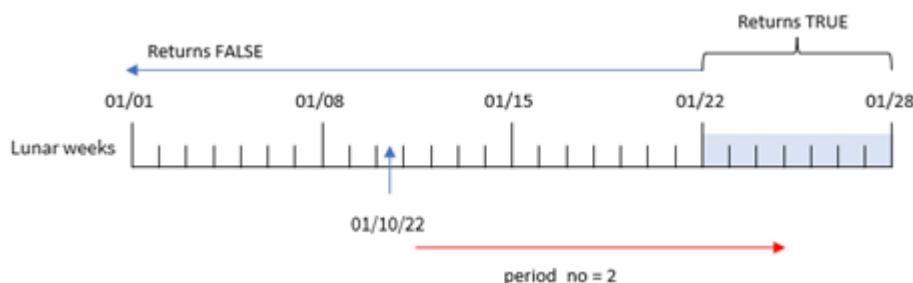
- date
- 2_lunar_weeks_later

Results table

date	2_lunar_weeks_later
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	0
1/9/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/13/2022	0

date	2_lunar_weeks_later
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	-1
1/23/2022	-1

inlunarweek() function, period_no example



In this instance, because a `period_no` of 2 was used as the offset argument in the `inlunarweek()` function, the function defines the week beginning on January 22 as the lunar week to validate transactions against. Therefore, any transaction that takes place between the January 22 and January 28 will return a Boolean result of TRUE.

Example 3 - first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script uses the same dataset and scenario as the first example. However, in the example, we set lunar weeks to begin on January 6.

- The same dataset and scenario as the first example.
- The default `dateFormat` system variable MM/DD/YYYY is used.
- A `first_week_day` argument of 5. This sets lunar weeks to begin on January 5.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inlunarweek(date,'01/10/2022', 0,5) as in_lunar_week
;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

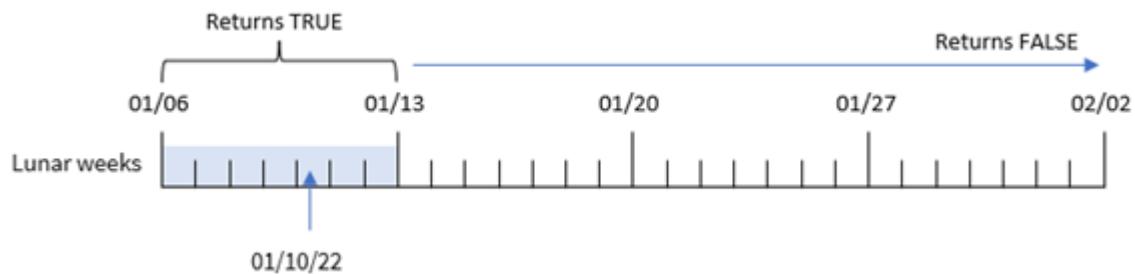
- date
- in_lunar_week

Results table

date	in_lunar_week
1/5/2022	0
1/6/2022	-1
1/7/2022	-1
1/8/2022	-1

date	in_lunar_week
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	0
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

inlunarweek() function, first_week_day example



In this instance, because the `first_week_date` argument of 5 is used in the `inlunarweek()` function, it offsets the start of the lunar week calendar to January 6. Therefore, January 10 falls in the lunar week beginning on January 6 and ending on January 12. Any transaction that falls between these two dates will return a Boolean value of `TRUE`.

Example 4 - Chart object

Load script and chart expression:

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The date field has been provided in the `DateFormat` system variable (MM/DD/YYYY) format.

However, in this example, the unchanged dataset is loaded into the application. The calculation that determines whether the transactions took place in the same lunar week as January 10 is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
```

```
*
```

```
Inline
```

```
[
```

```
id,date,amount
```

```
8183,'1/5/2022',42.32  
8184,'1/6/2022',68.22  
8185,'1/7/2022',15.25  
8186,'1/8/2022',25.26  
8187,'1/9/2022',37.23  
8188,'1/10/2022',37.23  
8189,'1/11/2022',17.17  
8190,'1/12/2022',88.27  
8191,'1/13/2022',57.42  
8192,'1/14/2022',53.80  
8193,'1/15/2022',82.06  
8194,'1/16/2022',87.21  
8195,'1/17/2022',95.93  
8196,'1/18/2022',45.89  
8197,'1/19/2022',36.23  
8198,'1/20/2022',25.66  
8199,'1/21/2022',82.77  
8200,'1/22/2022',69.98  
8201,'1/23/2022',76.11
```

```
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate whether a transaction takes place in the lunar week that contains January 10, create the following measure:

```
= inlunarweek(date,'01/10/2022', 0)
```

Results table

date	=inlunarweek(date,'01/10/2022', 0)
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

Example 5 - Scenario

Load script and chart expression:

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Products`.
- Information consisting of product ID, manufacture date, and cost price.

It has been identified that due to equipment error, products that were manufactured in the lunar week that included January 12 were defective. The end user would like a chart object that displays, by lunar week name, the status of whether the products manufactured were ‘defective’ or ‘faultless’ and the cost of the products manufactured in that month.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create a dimension to show the month names:
`=lunarweekname(manufacture_date)`
3. Create a measure to identify which of the products are defective and which are faultless using the `inlunarweek()` function:
`=if(only(inlunarweek(manufacture_date,makedate(2022,01,12),0)), 'Defective', 'Faultless')`
4. Create a measure to sum the `cost_price` of the products:
`=sum(cost_price)`
5. Set the measure's **Number formatting** to **Money**.
6. Under **Appearance**, turn off **Totals**.

Results table

lunarweekname (manufacture_date)	=if(only(inlunarweek(manufacture_date,makedate (2022,01,12),0)), 'Defective','Faultless')	sum(cost_ price)
2022/01	Faultless	\$125.79
2022/02	Defective	\$316.38
2022/03	Faultless	\$455.75
2022/04	Faultless	\$146.09

The **inlunarweek()** function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in the lunar week that contains January 10, the **inlunarweek()** function returns a Boolean value of **TRUE** and marks the products as ‘Defective’. For any product returning a value of **FALSE**, and therefore not manufactured in that week, it marks the products as ‘Faultless’.

inlunarweektodate

This function finds if **timestamp** lies inside the part of the lunar week up to and including the last millisecond of **base_date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

Syntax:

```
InLunarWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Example diagram of **inlunarweektodate()** function



The **inlunarweektodate()** function acts as the end point of the lunar week. In contrast, the **inlunarweek()** function, determines which lunar week the **base_date** falls into. For example, if the **base_date** were January 5, any timestamp between January 1 and January 5 would return a Boolean result of **TRUE**, while dates on January 6 and 7, and later, would return a Boolean result of **FALSE**.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the lunar week.
period_no	The lunar week can be offset by period_no . period_no is an integer, where the value 0 indicates the lunar week which contains base_date . Negative values in period_no indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
first_week_day	An offset that may be greater than or less than zero. This changes the beginning of the year by the specified number of days and/or fractions of a day.

When to use it

The `inlunarweektodate()` function returns a Boolean result. Typically, this type of function will be used as a condition in an IF expression. The `inlunarweektodate()` function would be used when the user would like the calculation to return an aggregation or calculation, dependent on whether the evaluated date occurred during a particular segment of the week in question.

For example, the `inlunarweektodate()` function can be used to identify all equipment manufactured in a particular week up to and including a particular date.

Function examples

Example	Result
<code>inlunarweektodate('01/12/2013', '01/13/2013', 0)</code>	Returns TRUE, since the value of the <code>timestamp</code> , 01/12/2013, falls in the part of the week 01/08/2013 to 01/13/2013.
<code>inlunarweektodate('01/12/2013', '01/11/2013', 0)</code>	Returns FALSE, since the value of the <code>timestamp</code> is later than the value of <code>base_date</code> , even though the two dates are in the same lunar week before 01/12/2012.
<code>inlunarweektodate('01/12/2006', '01/05/2006', 1)</code>	Returns TRUE. Specifying a value of 1 for <code>period_no</code> shifts the <code>base_date</code> forward one week, so the value of <code>timestamp</code> falls in the part of the lunar week.

The `inlunarweektodate()` function is often used in combination with the following functions:

Related functions

Function	Interaction
<code>lunarweekname</code> (page 831)	This function is used to determine the lunar week number of the year in which an input date occurs.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the

examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 - No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the month of January, which is loaded into a table called **Transactions**. The default **DateFormat** system variable MM/DD/YYYY is used.
- Create a field, **in_lunar_week_to_date**, that determines which transactions took place in lunar week to date of January 10.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inlunarweektodate(date, '01/10/2022', 0) as in_lunar_week_to_date
  ;
  Load
  *
  Inline
  [
  id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
```

```
8202,'1/4/2022',76.11
];
```

Results

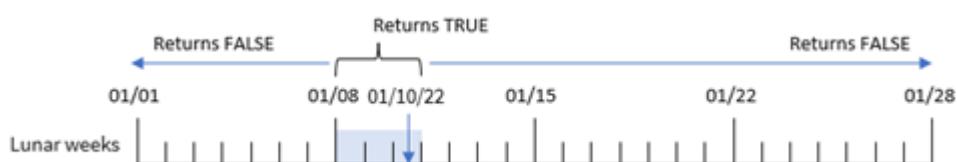
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_lunar_week_to_date

Results table

date	in_lunar_week_to_date
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() function, no additional arguments



The `in_lunar_week_to_date` field is created in the preceding load statement by using the `inlunarweektodate()` function and passing the date field, a hard-coded date for January 10 as our `base_date`, and an offset of 0 as the function's arguments.

Because lunar weeks begin on January 1, January 10 would fall in the lunar week that begins on January 8; and because we are using the `inlunarweektodate()` function, that lunar week would then end on the 10th. Therefore, any transactions that occur between those two dates in January would return a Boolean value of TRUE. This is validated in the results table.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor, and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the task is to create a field, `2_lunar_weeks_later`, that determines whether or not the transactions took place two weeks after the lunar week to date of January 1.

Load script

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inlunarweektodate(date, '01/10/2022', 2) as [2_lunar_weeks_later]
    ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

Results

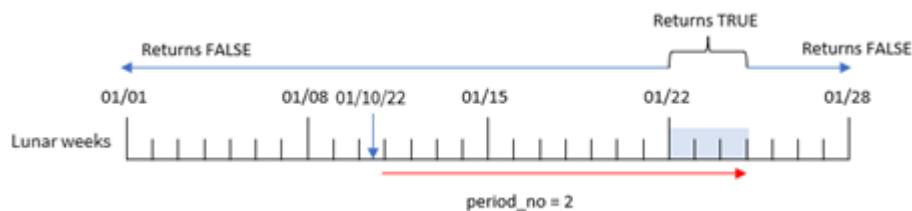
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- 2_lunar_weeks_later

Results table

date	2_lunar_weeks_later
1/1/2022	0
1/4/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	-1
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() function, period_no example



In this instance, the `inlunarweektodate()` function determines that the lunar week up to January 10 equates to three days (January 8, 9, 10). Since a `period_no` of 2 was used as the offset argument, this lunar week is shifted by 14 days. Therefore, this defines that three-day lunar week to include January 22, 23, and 24. Any transaction that takes place between January 22 and January 24 will return a Boolean result of TRUE.

Example 3 - first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The default `DateFormat` system variable MM/DD/YYYY is used.
- A `first_week_date` argument of 3. This sets lunar weeks to begin on January 3.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inlunarweek(date, '01/10/2022', 0,3) as in_lunar_week_to_date
;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

Results

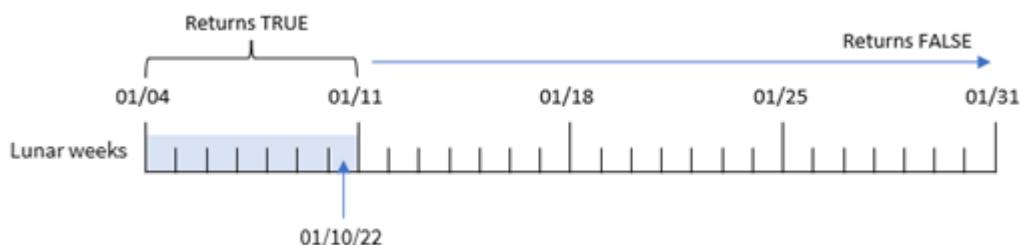
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_lunar_week_to_date`

Results table

date	in_lunar_week_to_date
1/1/2022	0
1/4/2022	-1
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() function, first_week_day example



In this instance, because the `first_week_date` argument of 3 is used in the `inlunarweek()` function, the first lunar week will be from January 3 to January 10. Because January 10 is also the `base_date`, any transaction that falls between these two dates will return a Boolean value of `TRUE`.

Example 4 - Chart object example

Load script and chart expression

Overview

Open the Data load editor, and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that determines whether the transactions took place in the lunar week up to January 10 is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load  
*  
Inline  
[  
id,date,amount  
8188,'1/10/2022',37.23  
8189,'1/17/2022',17.17  
8190,'1/26/2022',88.27  
8191,'1/12/2022',57.42  
8192,'1/19/2022',53.80  
8193,'1/21/2022',82.06  
8194,'1/1/2022',40.39  
8195,'1/27/2022',87.21  
8196,'1/11/2022',95.93  
8197,'1/29/2022',45.89  
8198,'1/31/2022',36.23  
8199,'1/18/2022',25.66  
8200,'1/23/2022',82.77  
8201,'1/15/2022',69.98  
8202,'1/4/2022',76.11  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

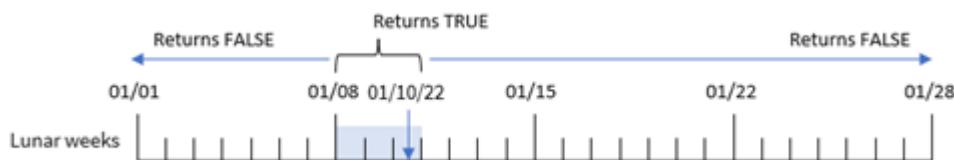
```
=inlunarweektodate(date,'01/10/2022', 0)
```

Results table

date	=inlunarweektodate(date,'01/10/2022', 0)
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0

date	=inlunarweektodate(date,'01/10/2022', 0)
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() function, chart object example



The `in_lunar_week_to_date` measure is created in the chart object by using the `inlunarweektodate()` function and passing the date field, a hard-coded date for January 10 as our `base_date`, and an offset of 0 as the function's arguments.

Because lunar weeks begin on January 1, January 10 would fall in the lunar week that begins on January 8. Additionally, since we are using the `inlunarweektodate()` function, that lunar week would then terminate on the 10th. Therefore, any transactions that occur between those two dates in January would return a Boolean value of TRUE. This is validated in the results table.

Example 5 - Scenario

Load script and chart expressions

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Products`.
- Information consisting of product ID, manufacture date, and cost price.

It has been identified that due to equipment error, products that were manufactured in the lunar week of January 12 were defective. The issue was resolved on January 13. The end user would like a chart object that displays, by week, the status of whether the products manufactured ‘defective’ or ‘faultless’ and the cost of the products manufactured in that week.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff]';

Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'01/02/2022 12:22:06',37.23
8189,'01/05/2022 01:02:30',17.17
8190,'01/06/2022 15:36:20',88.27
8191,'01/08/2022 10:58:35',57.42
8192,'01/09/2022 08:53:32',53.80
8193,'01/10/2022 21:13:01',82.06
8194,'01/11/2022 00:57:13',40.39
8195,'01/12/2022 09:26:02',87.21
8196,'01/13/2022 15:05:09',95.93
8197,'01/14/2022 18:44:57',45.89
8198,'01/15/2022 06:10:46',36.23
8199,'01/16/2022 06:39:27',25.66
8200,'01/17/2022 10:44:16',82.77
8201,'01/18/2022 18:48:17',69.98
8202,'01/26/2022 04:36:03',76.11
8203,'01/27/2022 08:07:49',25.12
8204,'01/28/2022 12:24:29',46.23
8205,'01/30/2022 11:56:56',84.21
8206,'01/30/2022 14:40:19',96.24
8207,'01/31/2022 05:28:21',67.67
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create a dimension to show the week names:
`=weekname(manufacture_date)`
3. Next, create a dimension which uses the `inlunarweektodate()` function to identify which of the products are defective and which are faultless:
`=if(inlunarweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')`
4. Create a measure to sum the `cost_price` of the products:
`=sum(cost_price)`
5. Set the measure's **Number formatting** to **Money**.

Results table

=lunarweekname (manufacture_date)	=if(InLunarWeekToDate(manufacture_date,makedate (2022,01,12),0),'Defective','Faultless')	=Sum(cost_ price)
2022/01	Faultless	\$142.67
2022/02	Defective	\$320.88
2022/02	Faultless	\$141.82
2022/03	Faultless	\$214.64
2022/04	Faultless	\$147.46
2022/05	Faultless	\$248.12

The `inlunarweektodate()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For those that return a Boolean value of `TRUE`, it marks the products as ‘Defective’. For any product returning a value of `FALSE`, and therefore not made in the lunar week up to January 12, it marks the products as ‘Faultless’.

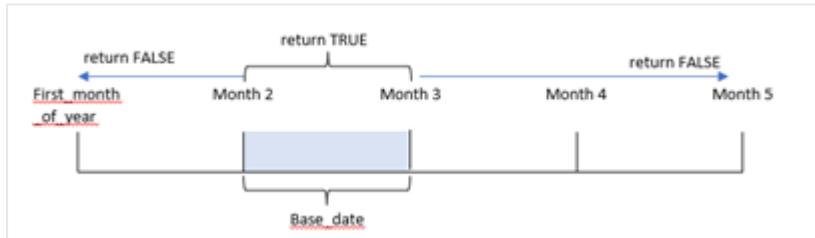
inmonth

This function returns True if **timestamp** lies inside the month containing **base_date**.

Syntax:

InMonth (`timestamp, base_date, period_no`)

Diagram of indaytotime function.



In other words, the `inmonth()` function determines if a set of dates fall into this month and returns a Boolean value based on a `base_date` that identifies the month.

When to use it

The `inmonth()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. This returns an aggregation or calculation depending on whether a date occurred in the month, including the date in question.

For example, the `inmonth()` function can be used to identify all equipment manufactured in a specific month.

Return data type:

Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date.
base_date	Date that is used to evaluate the month. It is important to note that the base_date can be any day within a month.
period_no	The month can be offset by period_no. period_no is an integer, where the value 0 indicates the month which contains base_date. Negative values in period_no indicate preceding months and positive values indicate succeeding months.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inmonth ('25/01/2013', '01/01/2013', 0)</code>	Returns True
<code>inmonth('25/01/2013', '23/04/2013', 0)</code>	Returns False
<code>inmonth ('25/01/2013', '01/01/2013', -1)</code>	Returns False
<code>inmonth ('25/12/2012', '17/01/2013', -1)</code>	Returns True

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the first half of 2022.
- A preceding load with an additional variable, ‘in_month’, that determines whether transactions took place in April.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inmonth(date,'04/01/2022', 0) as in_month
;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_month

Function examples

date	in_month
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0

date	in_month
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

The ‘in_month’ field is created in the preceding load statement by using the `inmonth()` function and passing the date field, a hard-coded date of April 1, as our `base_date` and a `period_no` of 0 as the function’s arguments.

The `base_date` identifies the month that will return a Boolean result of TRUE. Therefore, all transactions that occurred in April return TRUE which is validated in the results table.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario from the first example are used.

However, in this example, you will create a field, ‘2_months_prior’, that determines whether the transactions took place two months before April.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inmonth(date,'04/01/2022', -2) as [2_months_prior]
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
```

```
8197,'4/15/2022',45.89  
8198,'4/25/2022',36.23  
8199,'5/20/2022',25.66  
8200,'5/22/2022',82.77  
8201,'6/19/2022',69.98  
8202,'6/22/2022',76.11  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- 2_months_prior

Function examples

date	2_months_prior
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	-1
2/2/2022	-1
2/20/2022	-1
4/11/2022	0
4/13/2022	0
4/15/2022	0
4/25/2022	0
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

Using -2 as the period_no argument in the `inmonth()` function shifts the month defined by the `base_date` argument two months prior. In this example it changes the defined month from April to February.

Therefore, any transaction that takes place in February will return a Boolean result of TRUE.

Example 3 – Chart object

Load script and chart expression

Overview

The same dataset and scenario from the previous examples are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that determines whether transactions took place in April is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
```

```
*
```

```
Inline
```

```
[
```

```
id,date,amount
```

```
8188,'1/10/2022',37.23  
8189,'1/14/2022',17.17  
8190,'1/20/2022',88.27  
8191,'1/22/2022',57.42  
8192,'2/1/2022',53.80  
8193,'2/2/2022',82.06  
8194,'2/20/2022',40.39  
8195,'4/11/2022',87.21  
8196,'4/13/2022',95.93  
8197,'4/15/2022',45.89  
8198,'4/25/2022',36.23  
8199,'5/20/2022',25.66  
8200,'5/22/2022',82.77  
8201,'6/19/2022',69.98  
8202,'6/22/2022',76.11
```

```
];
```

Chart object

Load the data and open a sheet. Create a new table and add this field as a dimension:

```
date
```

To calculate whether a transaction takes place in April, create the following measure:

```
=inmonth(date,'04/01/2022', 0)
```

Results

Function examples

date	=inmonth(date,'04/01/2022', 0)
-------------	---------------------------------------

1/10/2022	0
-----------	---

1/14/2022	0
-----------	---

date	=inmonth(date,'04/01/2022', 0)
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

Example 4 – Scenario

Load script and results

Overview

In this example, a dataset is loaded into a table called ‘Products’. The table contains the following fields:

- Product ID
- Manufacture date
- Cost price

Due to equipment error, products that were manufactured in the month of July 2022 were defective. The issue was resolved on July 27, 2022.

The end user would like a chart that displays, by month, the status of products that were manufactured as ‘defective’ (Boolean TRUE) or ‘faultless’ (Boolean FALSE) and the cost of the products manufactured in that month.

Load script

```
Products:  
Load  
*  
Inline  
[  
product_id,manufacture_date,cost_price  
8188,'1/19/2022',37.23  
8189,'1/7/2022',17.17  
8190,'2/28/2022',88.27
```

```

8191, '2/5/2022', 57.42
8192, '3/16/2022', 53.80
8193, '4/1/2022', 82.06
8194, '5/7/2022', 40.39
8195, '5/16/2022', 87.21
8196, '6/15/2022', 95.93
8197, '6/26/2022', 45.89
8198, '7/9/2022', 36.23
8199, '7/22/2022', 25.66
8200, '7/23/2022', 82.77
8201, '7/27/2022', 69.98
8202, '8/2/2022', 76.11
8203, '8/8/2022', 25.12
8204, '8/19/2022', 46.23
8205, '9/26/2022', 84.21
8206, '10/14/2022', 96.24
8207, '10/29/2022', 67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

```
=monthname(manufacture_date)
```

Create the following measures:

- =sum(cost_price)
 - =if(only(inmonth(manufacture_date,makedate(2022,07,01),0)),'Defective','Faultless')
1. Set the measure's **Number Formatting to Money**.
 2. Under **Appearance**, turn off **Totals**.

Results table

monthname (manufacture_date)	=if(only(inmonth(manufacture_date,makedate (2022,07,01),0)),'Defective','Faultless')	sum(cost_ price)
Jan 2022	Faultless	\$54.40
Feb 2022	Faultless	\$145.69
Mar 2022	Faultless	\$53.80
Apr 2022	Faultless	\$82.06
May 2022	Faultless	\$127.60
Jun 2022	Faultless	\$141.82
Jul 2022	Defective	\$214.64
Aug 2022	Faultless	\$147.46
Sep 2022	Faultless	\$84.21
Oct 2022	Faultless	\$163.91

The `inmonth()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in July 2022, the `inmonth()` function returns a Boolean value of True and marks the products as ‘Defective’. For any product returning a value of False, and therefore not manufactured in July, it marks the products as ‘Faultless’.

inmonths

This function finds if a timestamp falls within the same month, bi-month, quarter, four-month period, or half-year as a base date. It is also possible to find if the timestamp falls within a previous or following time period.

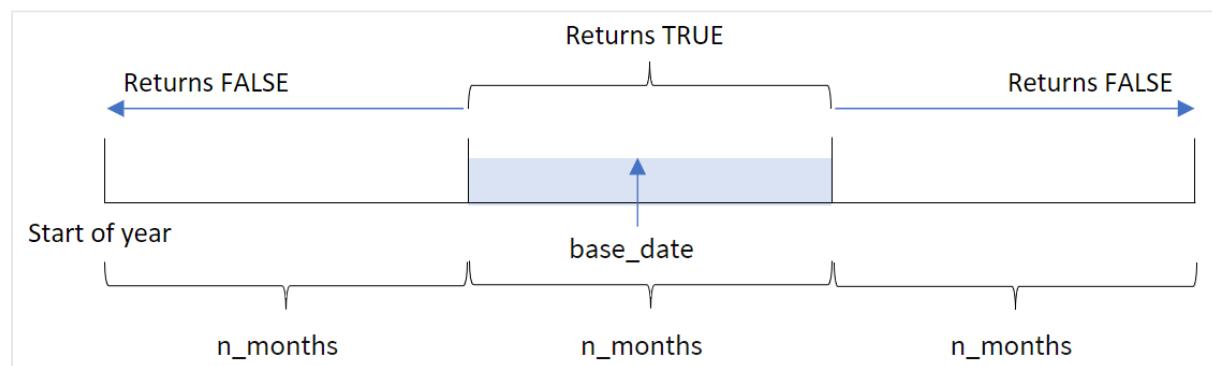
Syntax:

```
InMonths(n_months, timestamp, base_date, period_no [, first_month_of_year])
```

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of `inmonths()` function



The `inmonths()` function divides the year into segments based on the `n_months` argument provided. It then determines whether each timestamp evaluated falls into the same segment as the `base_date` argument. If, however, a `period_no` argument is provided, the function determines whether the timestamps fall into a previous or following period from the `base_date`.

The following segments of the year are available in the function as `n_month` arguments.

`n_month` arguments

Period	Number of months
month	1
bi-month	2
quarter	3
four months	4
half-year	6

When to use it

The `inmonths()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. By using the `inmonths()` function, you can select the period that you want to be evaluated. For example, letting the user identify products manufactured in the month, quarter, or half-year of a certain period.

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Arguments

Argument	Description
n_months	The number of months that defines the period. An integer or expression that resolves to an integer that must be one of: 1 (equivalent to the <code>inmonth()</code> function), 2 (bi-month), 3 (equivalent to the <code>inquarter()</code> function), 4 (four-month period), or 6 (half year).
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the period.
period_no	The period can be offset by period_no , an integer, or expression resolving to an integer, where the value 0 indicates the period that contains base_date . Negative values in period_no indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

You can use the following values to set the first month of year in the `first_month_of_year` argument:

`first_month_of_year` values

Month	Value
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inmonths(4, '01/25/2013', '04/25/2013', 0)</code>	Returns TRUE. Because the value of timestamp, 01/25/2013, lies within the four-month period 01/01/2013 to 04/30/2013, in which the value of <code>base_date</code> , 04/25/2013 lies.
<code>inmonths(4, '05/25/2013', '04/25/2013', 0)</code>	Returns FALSE. Because 05/25/2013 is outside the same period as the previous example.
<code>inmonths(4, '11/25/2012', '02/01/2013', -1)</code>	Returns TRUE. Because the value of <code>period_no</code> , -1, shifts the search period back one period of four months (the value of <code>n-months</code>), which makes the search period 09/01/2012 to 12/31/2012.
<code>inmonths(4, '05/25/2006', '03/01/2006', 0, 3)</code>	Returns TRUE. Because the value of <code>first_month_of_year</code> is set to 3, which makes the search period 03/01/2006 to 07/30/2006 instead of 01/01/2006 to 04/30/2006.

Example 1 - No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- A preceding load with an additional variable, ‘`in_months`’, that determines which transactions took place in the same quarter as May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inmonths(3,date,'05/15/2022', 0) as in_months
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,'2/19/2022',37.23
  8189,'3/7/2022',17.17
  8190,'3/30/2022',88.27
  8191,'4/5/2022',57.42
  8192,'4/16/2022',53.80
  8193,'5/1/2022',82.06
  8194,'5/7/2022',40.39
  8195,'5/22/2022',87.21
  8196,'6/15/2022',95.93
  8197,'6/26/2022',45.89
  8198,'7/9/2022',36.23
  8199,'7/22/2022',25.66
  8200,'7/23/2022',82.77
  8201,'7/27/2022',69.98
  8202,'8/2/2022',76.11
  8203,'8/8/2022',25.12
  8204,'8/19/2022',46.23
  8205,'9/26/2022',84.21
  8206,'10/14/2022',96.24
  8207,'10/29/2022',67.67
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_months

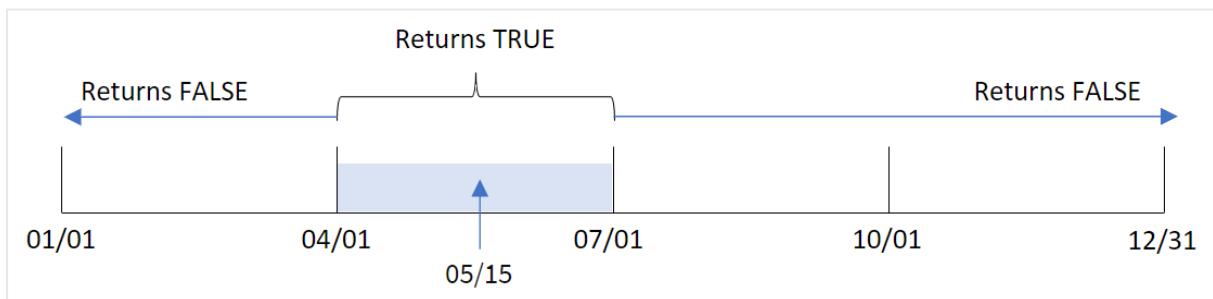
Results table

date	in_months
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1

date	in_months
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_months’ field is created in the preceding load statement by using the `inmonths()` function. The first argument provided is 3 which divides the year into quarter segments. The second argument identifies which field is being evaluated, the date field in this example. The third argument is a hard-coded date for the May 15 which is the `base_date` and a `period_no` of 0 is the final argument.

Diagram of `inmonths()` function with quarter segments



May falls into the second quarter of the year. Therefore, any transaction that occurs between April 1 and June 30 will return a Boolean result of TRUE. This is validated in the results table.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- A preceding load with an additional variable, ‘previous_quarter’, that determines whether transactions took place in the quarter before May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inmonths(3,date,'05/15/2022', -1) as previous_quarter
;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

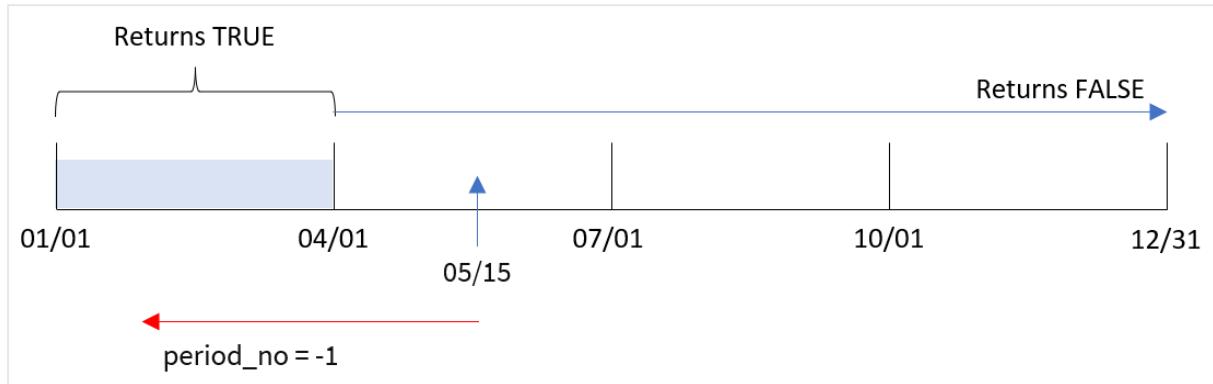
- date
- previous_quarter

Results table

date	previous quarter
2/19/2022	-1
3/7/2022	-1
3/30/2022	-1
4/5/2022	0
4/16/2022	0
5/1/2022	0
5/7/2022	0
5/22/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The function evaluates whether transactions occurred in the first quarter of the year by using -1 as the period_no argument in the `inmonths()` function. May 15 is the base_date and falls into the second quarter of the year (April-June).

Diagram of inmonths() function with quarter segments and the period_no set to -1



Therefore, any transaction that occurs between January and March will return a Boolean result of TRUE.

Example 3 - first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- A preceding load with an additional variable, ‘in_months’, that determines which transactions took place in the same quarter as May 15, 2022.

In this example, the organizational policy is for March to be the first month of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inmonths(3,date,'05/15/2022', 0, 3) as in_months
;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195,'5/22/2022',87.21  
8196,'6/15/2022',95.93  
8197,'6/26/2022',45.89  
8198,'7/9/2022',36.23  
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_months

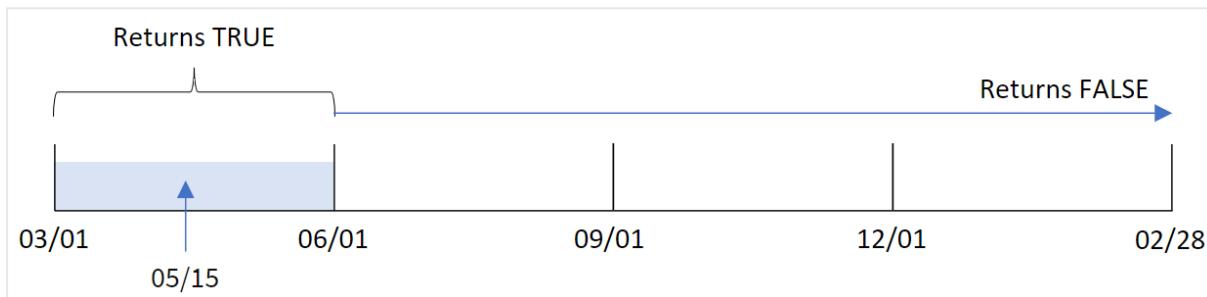
Results table

date	in_months
2/19/2022	0
3/7/2022	-1
3/30/2022	-1
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

date	in_months
9/26/2022	0
10/14/2022	0
10/29/2022	0

By using 3 as the `first_month_of_year` argument in the `inmonths()` function, the function begins the year on March 1. The `inmonths()` function then divides the year into quarters: Mar-May, Jun-Aug, Sep-Nov, Dec-Feb. Therefore, May 15 falls into the first quarter of the year (March-May).

Diagram of `inmonths()` function with March set as first month of the year



Any transaction that occurs in these months will return a Boolean result of TRUE.

Example 4 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario from the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that determines whether transactions took place in the same quarter as May 15, 2022 is created as a measure in a chart in the app.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
```

```
8193, '5/1/2022', 82.06  
8194, '5/7/2022', 40.39  
8195, '5/22/2022', 87.21  
8196, '6/15/2022', 95.93  
8197, '6/26/2022', 45.89  
8198, '7/9/2022', 36.23  
8199, '7/22/2022', 25.66  
8200, '7/23/2022', 82.77  
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- date

To calculate whether transactions took place in the same quarter as May 15, create the following measure:

```
=inmonths(3,date,'05/15/2022', 0)
```

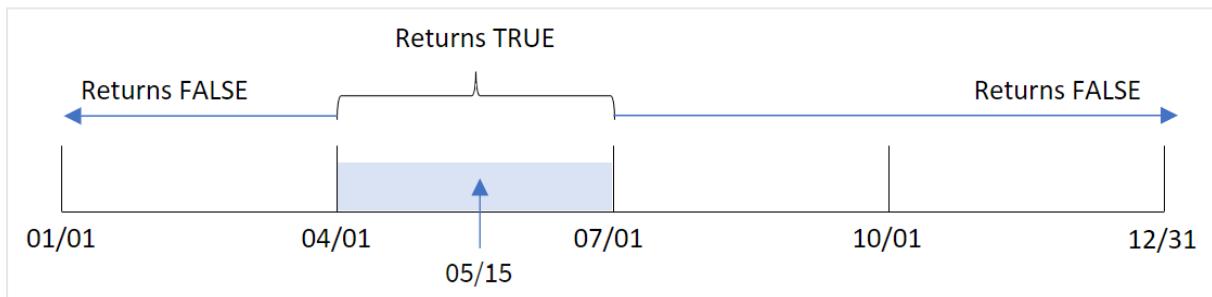
Results table

date	=inmonths(3,date,'05/15/2022', 0)
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0

date	=inmonths(3,date,'05/15/2022', 0)
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_months’ field is created in the chart by using the `inmonths()` function. The first argument provided is 3 which divides the year into quarter segments. The second argument identifies which field is being evaluated, the date field in this example. The third argument is a hard-coded date for the for May 15 which is the base_date and a period_no of 0 is the final argument.

Diagram of `inmonths()` function with quarter segments



May falls into the second quarter of the year. Therefore, any transaction that occurs between April 1 and June 30 will return a Boolean result of TRUE. This is validated in the results table.

Example 5 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called ‘products’.
- The table contains the following fields:
 - product ID
 - product type
 - manufacture date
 - cost price

The end user would like a chart that displays, by product type, the cost of products manufactured in the first segment of 2021. The user would like to be able to define the length of this segment.

Load script

```
SET vPeriod = 1;

Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'2/19/2022',37.23
8189,product D,'3/7/2022',17.17
8190,product C,'3/30/2022',88.27
8191,product B,'4/5/2022',57.42
8192,product D,'4/16/2022',53.80
8193,product D,'5/1/2022',82.06
8194,product A,'5/7/2022',40.39
8195,product B,'5/22/2022',87.21
8196,product C,'6/15/2022',95.93
8197,product B,'6/26/2022',45.89
8198,product C,'7/9/2022',36.23
8199,product D,'7/22/2022',25.66
8200,product D,'7/23/2022',82.77
8201,product A,'7/27/2022',69.98
8202,product A,'8/2/2022',76.11
8203,product B,'8/8/2022',25.12
8204,product B,'8/19/2022',46.23
8205,product B,'9/26/2022',84.21
8206,product C,'10/14/2022',96.24
8207,product D,'10/29/2022',67.67
];

```

Results

Load the data and open a sheet.

At the start of the load script, a variable, **vPeriod**, is created that is tied to the variable input control.

Do the following:

1. In the assets panel, click **Custom objects**.
2. Select **Qlik Dashboard bundle**, create a **Variable input** object.
3. Enter a title for the chart object.
4. Under **Variable**, select **vPeriod** as the name and set the object to show as a **Drop down**.
5. Under **Values**, click **Dynamic** values. Enter the following:
='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.
6. Add a new table to the sheet.
7. Under **Data** in the properties panel, add **product_type** as a dimension.

8. Add the following expression as a measure:

```
=sum(if(inmonths($(vPeriod),manufacture_date,makedate(2022,01,01),0),cost_price,0))
```

9. Set the measure's **Number formatting** to **Money**.

Results table

product_type	=sum(if(inmonths(\$(vPeriod),manufacture_date,makedate(2022,01,01),0),cost_price,0))
product A	\$88.27
product B	\$37.23
product C	\$17.17
product D	\$0.00

The `inmonths()` function uses the user input as its argument to define the size of the starting segment of the year. The function passes in the manufacture date of each of the products as the `inmonths()` function's second argument. By using January 1 as the third argument in the `inmonths()` function, products with manufacture dates that fall in the opening segment of the year will return a Boolean value of TRUE and therefore the sum function will add the costs of those products.

inmonthstodate

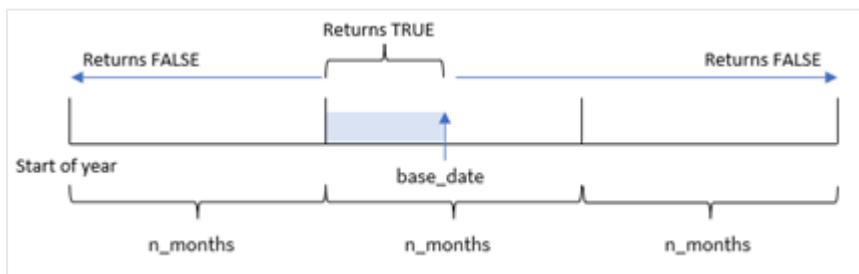
This function finds if a timestamp falls within the part a period of the month, bi-month, quarter, four-month period, or half-year up to and including the last millisecond of `base_date`. It is also possible to find if the timestamp falls within a previous or following time period.

Syntax:

```
InMonths (n_months, timestamp, base_date, period_no[, first_month_of_year ])
```

Return data type: Boolean

Diagram of inmonthstodate function.



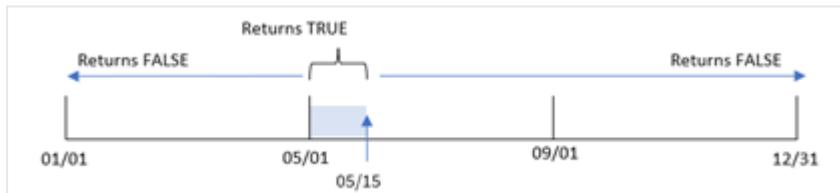
Arguments

Argument	Description
n_months	The number of months that defines the period. An integer or expression that resolves to an integer that must be one of: 1 (equivalent to the <code>inmonth()</code> function), 2 (bi-month), 3 (equivalent to the <code>inquarter()</code> function), 4 (four-month period), or 6 (half year).
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the period.
period_no	The period can be offset by period_no , an integer, or expression resolving to an integer, where the value 0 indicates the period that contains base_date . Negative values in period_no indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

In the `inmonthstodate()` function, the `base_date` acts as the end point of the particular year segment that it is part of.

For example, if the year was broken into tertial segments, and the `base_date` was May 15, any timestamp between the start of January and end of April would return a Boolean result of FALSE. Dates between May 1 and May 15 would return TRUE. The rest of the year would return FALSE.

Diagram of Boolean results range of `inmonthstodate` function.



The following segments of the year are available in the function as `n_month` arguments.

`n_month` arguments

Period	Number of months
month	1
bi-month	2
quarter	3
tertial	4
half-year	6

When to use it

The `inmonthstodate()` function returns a Boolean result. Typically, this type of function is used as a condition in an `if` expression. By using the `inmonthstodate()` function, you can select the period you want to be evaluated. For example, providing an input variable that lets the user identify the products manufactured in the month, quarter, or half-year of a period, up to a certain date.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inmonthstodate(4, '01/25/2013', '04/25/2013', 0)</code>	Returns True, because the value of timestamp, 01/25/2013, lies within the four-month period 01/01/2013 up to the end of 04/25/2013, in which the value of <code>base_date</code> , 04/25/2013 lies.
<code>inmonthstodate(4, '04/26/2013', '04/25/2006', 0)</code>	Returns False, because 04/26/2013 is outside the same period as the previous example.
<code>inmonthstodate(4, '09/25/2005', '02/01/2006', -1)</code>	Returns True, because the value of <code>period_no</code> , -1, shifts the search period back one period of four months (the value of <code>n-months</code>), which makes the search period 01/09/2005 to 02/01/2006.
<code>inmonthstodate(4, '04/25/2006', '06/01/2006', 0, 3)</code>	Returns True, because the value of <code>first_month_of_year</code> is set to 3, which makes the search period 03/01/2006 to 06/01/2006 instead of 05/01/2006 to 06/01/2006.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 that is loaded into a table called ‘Transactions’.
- A date field in the `DateFormat` system variable (MM/DD/YYYY) format.
- A preceding load statement containing:
 - The `inmonthstodate()` function that is set as the field, ‘`in_months_to_date`’. This determines which transactions took place in the quarter up until May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
  *,
  inmonthstodate(3,date,'05/15/2022', 0) as in_months_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_months_to_date`

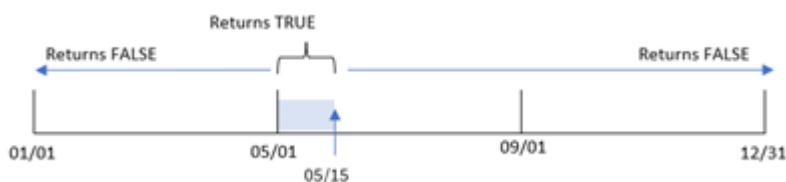
Results table

date	in_months_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_months_to_date’ field is created in the preceding load statement by using the `inmonthstodate()` function.

The first argument provided is 3, dividing the year into quarter segments. The second argument identifies which field is being evaluated. The third argument is a hard-coded date for May 15, which is the `base_date` that defines the end boundary of the segment. A `period_no` of 0 is the final argument.

Diagram of inmonthstodate function with no additional arguments.



Any transaction that occurs between April 1 and May 15 returns a Boolean result of TRUE. Transaction dates outside of that period return FALSE.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the task is to create a field, ‘previous_qtr_to_date’, that determines if the transactions took place a quarter before May 15.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inmonthstodate(3,date,'05/15/2022', -1) as previous_qtr_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_qtr_to_date

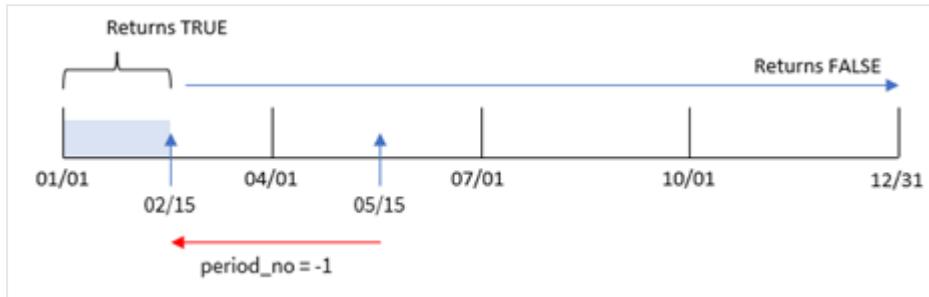
Results table

date	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

By using -1 as the period_no argument in the `inmonthstodate()` function, the function shifts the boundaries of the comparator year segment by a quarter.

May 15 falls into the second quarter of the year and therefore the segment initially equates to between April 1 and May 15. The period_no argument offsets this segment by a negative three months. The date boundaries become January 1 to February 15.

Diagram of inmonthstodate function with period_no set to -1.



Therefore, any transaction that occurs between January 1 and February 15 will return a Boolean result of `TRUE`.

Example 3 – first_month_of_year

Load script and results

Overview

The same dataset and scenario as the first example are used.

In this example, the organizational policy is for March to be the first month of the financial year.

Create a field, ‘`in_months_to_date`’, that determines which transactions took place in the same quarter up to May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
  *,
  inmonthstodate(3,date,'05/15/2022', 0,3) as in_months_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
```

```
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_months_to_date

Results table

date	previous_qtr_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

By using 3 as the `first_month_of_year` argument in the `inmonthstodate()` function, the function begins the year on March 1 and then divides the year into quarters based on the first argument provided. Therefore, the quarter segments are:

- Mar-May
- Jun-Aug
- Sep-Nov
- Dec-Feb

The `base_date` of May 15 then segments the Mar-May quarter by setting its end boundary as May 15.

Diagram of inmonthstodate function with March set as first month of the year.



Therefore, any transaction that occurs between March 1 and May 15 will return a Boolean result of TRUE, and transactions with dates outside these boundaries will return a value of FALSE.

Example 4 – Chart example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

In this example, the dataset is unchanged and loaded into the app. The task is to create a calculation that determines whether transactions took place in the same quarter as May 15 as a measure in a chart of the app.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

Load

*

Inline

[

id,date,amount

```
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195, '5/16/2022', 87.21  
8196, '6/15/2022', 95.93  
8197, '6/26/2022', 45.89  
8198, '7/9/2022', 36.23  
8199, '7/22/2022', 25.66  
8200, '7/23/2022', 82.77  
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

date

To calculate whether transactions took place in the same quarter as May 15, create the following measure:

```
=inmonthstodate(3,date,'05/15/2022', 0)
```

Results table

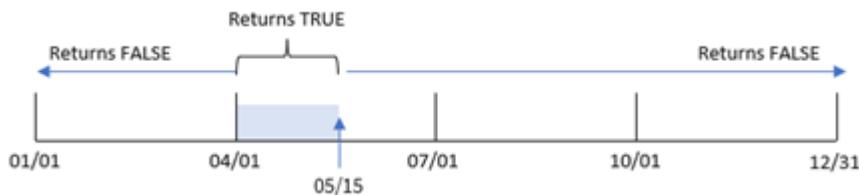
date	=inmonthstodate(3,date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0

date	=inmonthstodate(3,date,'05/15/2022', 0)
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_months_to_date’ measure is created in the chart by using the `inmonthstodate()` function.

The first argument provided is 3, dividing the year into quarter segments. The second argument identifies which field is being evaluated. The third argument is a hard-coded date May 15 which is the base_date that defines the end boundary of the segment. A period_no of 0 is the final argument.

Diagram of inmonthstodate function with quarter segments.



Any transaction that occurs between April 1 and May 15 will return a Boolean result of TRUE. Transaction dates outside of that segment will return FALSE.

Example 5 – Scenario

Load script and results

Overview

In this example, a dataset is loaded into a table called ‘Sales’. The table contains the following fields:

- Product ID
- Product type
- Sales date
- Sales price

The end user would like a chart that displays, by product type, the sales of products sold in the period leading up to December 24, 2022. The user would like to be able to define the length of this period.

Load script

```
SET vPeriod = 1;
```

```
Products:  
Load
```

```
*  
Inline  
[  
product_id,product_type,sales_date,sales_price  
8188,product A,'9/19/2022',37.23  
8189,product D,'10/27/2022',17.17  
8190,product C,'10/30/2022',88.27  
8191,product B,'10/31/2022',57.42  
8192,product D,'11/16/2022',53.80  
8193,product D,'11/28/2022',82.06  
8194,product A,'12/2/2022',40.39  
8195,product B,'12/5/2022',87.21  
8196,product C,'12/15/2022',95.93  
8197,product B,'12/16/2022',45.89  
8198,product C,'12/19/2022',36.23  
8199,product D,'12/22/2022',25.66  
8200,product D,'12/23/2022',82.77  
8201,product A,'12/24/2022',69.98  
8202,product A,'12/24/2022',76.11  
8203,product B,'12/26/2022',25.12  
8204,product B,'12/27/2022',46.23  
8205,product B,'12/27/2022',84.21  
8206,product C,'12/28/2022',96.24  
8207,product D,'12/29/2022',67.67  
];
```

Results

Load the data and open a sheet.

At the start of the load script, a variable, vPeriod, is created that is tied to the variable input control.

Do the following:

1. In the assets panel, click **Custom objects**.
2. Select **Qlik Dashboard bundle** and add a **Variable input** to your sheet.
3. Enter a title for the chart.
4. Under **Variable**, select **vPeriod** as the name and set the object to show as a **Drop down**.
5. Under **Values**, click **Dynamic** values. Enter the following:
`'1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`
6. Add a new table to the sheet.
7. Under **Data** in the properties panel, add product_type as a dimension.
8. Add the following expression as a measure:
`=sum(if(inmonthstodate($(vPeriod),sales_date,makedate(2022,12,24),0),sales_price,0))`
9. Set the measure's **Number formatting** to **Money**.

Results table

product_type	=sum(if(inmonthstodate(\${vPeriod}),sales_date,makedate(2022,12,24),0),sales_price,0))
product A	\$186.48
product B	\$190.52
product C	\$220.43
product D	\$261.46

The `inmonthstodate()` function uses the user input as its argument to define the size of the starting segment of the year.

The function passes in the sales date of each of the products as the `inmonthstodate()` function's second argument. By using December 24 as the third argument in the `inmonthstodate()` function, products with sales dates that occur in the defined period up to and including December 24 return a Boolean value of TRUE. The sum function adds the sales of these products.

inmonthtodate

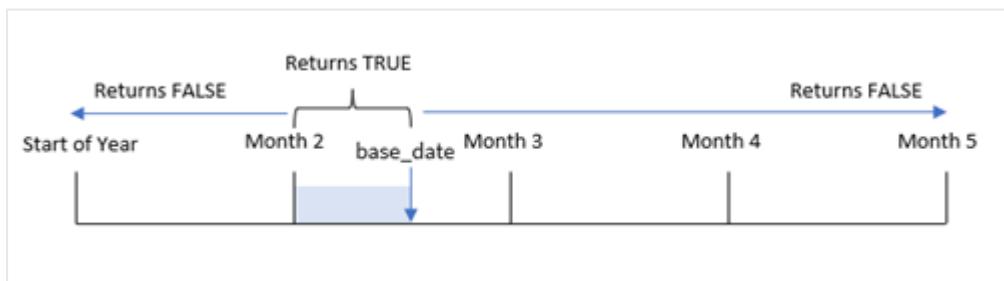
Returns True if **date** lies inside the part of month containing **basedate** up until and including the last millisecond of **basedate**.

Syntax:

```
InMonthToDate (timestamp, base_date, period_no)
```

Return data type: Boolean

Diagram of inmonthtodate function.



The `inmonthtodate()` function identifies a selected month as a segment. The start boundary is the beginning of the month. The end boundary can be set as a later date in the month. It then determines whether a set of dates fall into this segment or not, returning a TRUE or FALSE Boolean value.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date .

Argument	Description
base_date	Date that is used to evaluate the month.
period_no	The month can be offset by period_no . period_no is an integer, where the value 0 indicates the month which contains base_date . Negative values in period_no indicate preceding months and positive values indicate succeeding months.

When to use it

The `inmonthtodate()` function returns a Boolean result. Typically, this type of function is used as a condition in an `if` expression. The `inmonthtodate()` function returns an aggregation or calculation that depends on whether a date occurred in the month up to and including the date in question.

For example, the `inmonthtodate()` function can be used to identify all equipment manufactured in a month up to a specific date.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inmonthtodate ('01/25/2013', '25/01/2013', 0)</code>	Returns True
<code>inmonthtodate ('01/25/2013', '24/01/2013', 0)</code>	Returns False
<code>inmonthtodate ('01/25/2013', '28/02/2013', -1)</code>	Returns True

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- A date field is provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- A preceding load statement containing:
 - The `inmonthtodate()` function which is set as the field, ‘`in_month_to_date`’. This determines which transactions took place between July 1 and July 26, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
  *,
  inmonthtodate(date, '07/26/2022', 0) as in_month_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_month_to_date`

Results table

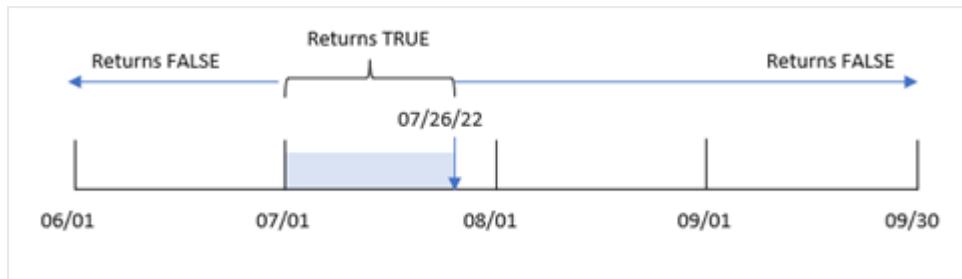
date	in_month_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_month_to_date’ field is created in the preceding load statement by using the `inmonthtodate()` function.

The first argument identifies which field is being evaluated. The second argument is a hard-coded date, July 26, which is the `base_date`. This `base_date` argument identifies which month is segmented and the end boundary of that segment.

A `period_no` of 0 is the final argument meaning that the function is not comparing months preceding or following the segmented month.

Diagram of inmonthtodate function with no additional arguments.



As a result, any transaction that occurs between July 1 and July 26 returns a Boolean result of TRUE. Any transaction that occurs in July after July 26 returns a Boolean result of FALSE as will any transaction in any other month of the year.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

In this example, the task is to create a field, 'six_months_prior', that determines which transactions took place a full six months before July 1 and July 26.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
 *,
 inmonthtodate(date, '07/26/2022', -6) as six_months_prior
;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
```

```
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

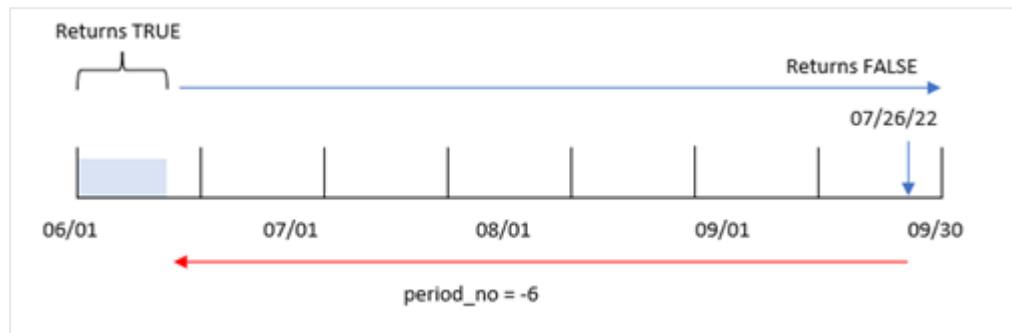
- date
- six_months_prior

Results table

date	six_months_prior
1/7/2022	-1
1/19/2022	-1
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

By using -6 as the period_no argument in the `inmonthtodate()` function, the boundaries of the comparator month segment shift by six months. Initially the month segment equates to between July 1 and July 26. The period_no then offsets this segment by a negative six months and the date boundaries are shifted and fall between January 1 and January 26.

Diagram of `inmonthtodate` function with `period_no` set to -6.



As a result, any transaction that occurs between January 1 and January 26 will return a Boolean result of `TRUE`.

Example 3 – Chart example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

In this example, the dataset is unchanged and loaded into the app. The task is to create a calculation that determines whether transactions took place between July 1 and July 26 as a measure in a chart of the app.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
```

```
8199, '7/22/2022', 25.66  
8200, '7/23/2022', 82.77  
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

date

To calculate whether transactions took place between July 1 and July 26, create the following measure:

```
=inmonthtodate(date, '07/26/2022', 0)
```

Results table

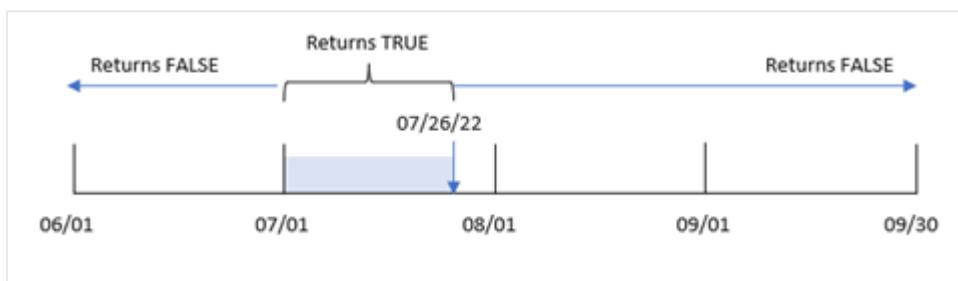
date	=inmonthtodate(date,'07/26/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

date	=inmonthtodate(date,'07/26/2022', 0)
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_month_to_date’ field measure is created in the chart by using the `inmonthtodate()` function.

The first argument identifies which field is being evaluated. The second argument is a hard-coded date, July 26, which is the `base_date`. This `base_date` argument identifies which month is segmented and the end boundary of that segment. A `period_no` of 0 is the final argument. This means that the function is not comparing months preceding or following the segmented month.

Diagram of inmonthtodate function with no additional arguments.



As a result, any transaction that occurs between July 1 and July 26 returns a Boolean result of TRUE. Any transaction that occurs in July after July 26 returns a Boolean result of FALSE as will any transaction in any other month of the year.

Example 4 – Scenario

Load script and results

Overview

In this example, a dataset is loaded into a table called ‘Products’. The table contains the following fields:

- Product ID
- Manufacture date
- Cost price

Due to equipment error, products that were manufactured in the month of July 2022 were defective. The issue was resolved on July 27, 2022.

The end user would like a chart that displays, by month, the status of products that were manufactured as ‘defective’ (Boolean TRUE) or ‘faultless’ (Boolean FALSE) and the cost of the products manufactured in that month.

Load script

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- =monthname(manufacture_date)
- =if(Inmonthtodate(manufacture_date,makedate(2022,07,26),0),'Defective','Faultless')

To calculate the sum cost of the products, create this measure:

=sum(cost_price)

Set the measure's **Number Formatting** to **Money**.

Results table

monthname (manufacture_date)	if(Inmonthtodate(manufacture_date,makedate (2022,07,26),0),'Defective','Faultless')	Sum(cost_ price)
Jan 2022	Faultless	\$54.40
Feb 2022	Faultless	\$145.69
Mar 2022	Faultless	\$53.80

monthname (manufacture_date)	if(Inmonthtodate(manufacture_date,makedate (2022,07,26),0),'Defective','Faultless')	Sum(cost_ price)
Apr 2022	Faultless	\$82.06
May 2022	Faultless	\$127.60
Jun 2022	Faultless	\$141.82
Jul 2022	Defective	\$144.66
Jul 2022	Faultless	\$69.98
Aug 2022	Faultless	\$147.46
Sep 2022	Faultless	\$84.21
Oct 2022	Faultless	\$163.91

The `inmonthtodate()` function returns a Boolean value when evaluating the manufacturing dates of each of the products.

For the dates that return a Boolean value of TRUE, the product is marked as ‘Defective’. For any product returning a value of FALSE, and therefore not made in the month up to and including July 26, it marks the products as ‘Faultless’.

inquarter

This function returns True if **timestamp** lies inside the quarter containing **base_date**.

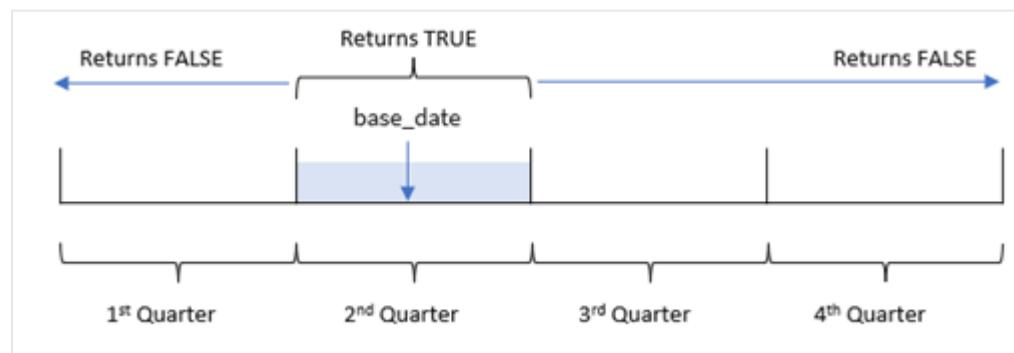
Syntax:

```
InQuarter (timestamp, base_date, period_no[, first_month_of_year])
```

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of `inquarter()` function's range



In other words, the `inquarter()` function divides the year into four equal quarters between January 1 and December 31. You can use the `first_month_of_year` argument to change what month is considered the first in your app, and the quarters will change based on that argument. The `base_date`, the function identifies which quarter should be used as the comparator for the function. Finally, the function returns a Boolean result when comparing date values to that quarter segment.

When to use it

The `inquarter()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. This returns an aggregation or calculation that depends on whether a date occurred in the selected quarter.

For example, the `inquarter()` function can be used to identify all equipment manufactured in a quarter segment based on the dates when the equipment was manufactured.

Arguments

Argument	Description
<code>timestamp</code>	The date that you want to compare with <code>base_date</code> .
<code>base_date</code>	Date that is used to evaluate the quarter.
<code>period_no</code>	The quarter can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the quarter which contains <code>base_date</code> . Negative values in <code>period_no</code> indicate preceding quarters and positive values indicate succeeding quarters.
<code>first_month_of_year</code>	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in <code>first_month_of_year</code> .

You can use the following values to set the first month of year in the `first_month_of_year` argument:

`first_month_of_year` values

Month	Value
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inquarter ('01/25/2013', '01/01/2013', 0)</code>	Returns TRUE
<code>inquarter ('01/25/2013', '04/01/2013', 0)</code>	Returns FALSE
<code>inquarter ('01/25/2013', '01/01/2013', -1)</code>	Returns FALSE
<code>inquarter ('12/25/2012', '01/01/2013', -1)</code>	Returns TRUE
<code>inquarter ('01/25/2013', '03/01/2013', 0, 3)</code>	Returns FALSE
<code>inquarter ('03/25/2013', '03/01/2013', 0, 3)</code>	Returns TRUE

Example 1 - No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inquarter()` function that is set as the ‘in_quarter’ field, and determines which transactions took place in the same quarter as May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inquarter (date, '05/15/2022', 0) as in_quarter
  ;
```

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_quarter

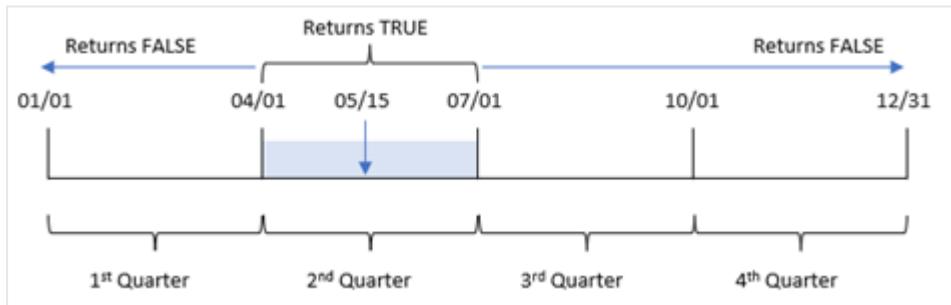
Results table

date	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1

date	in_quarter
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_quarter’ field is created in the preceding load statement by using the `inquarter()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for May 15 that identifies which quarter to define as the comparator. A `period_no` of 0 is the final argument and ensures the `inquarter()` function does not compare quarters preceding or following the segmented quarter.

Diagram of `inquarter()` function with May 15 as the base date



Any transaction that occurs between April 1 and the end of June 30 returns a Boolean result of TRUE.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inquarter()` function that is set as the ‘previous_quarter’ field, and determines which transactions took place in the quarter preceding the quarter of May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inquarter (date,'05/15/2022', -1) as previous_qtr
;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_qtr

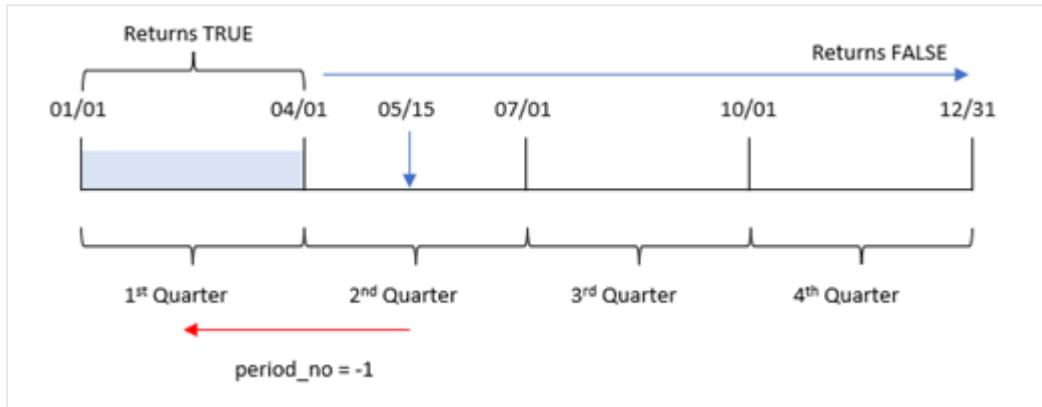
Results table

date	previous_qtr
1/7/2022	-1

date	previous_qtr
1/19/2022	-1
2/5/2022	-1
2/28/2022	-1
3/16/2022	-1
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

Using -1 as the period_no argument in the inquarter() function shifts the boundaries of the comparator quarter back by a full quarter. May 15 falls into the second quarter of the year and therefore the segment initially equates to the quarter of April 1 to June 30. The period_no offsets this segment by a negative three months and causes the date boundaries to become January 1 to March 30.

Diagram of `inquarter()` function with May 15 as the base date



Therefore, any transaction that occurs between January 1 and March 30 will return a Boolean result of TRUE.

Example 3 - first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inquarter()` function that is set as the ‘`in_quarter`’ field, and determines which transactions took place in the same quarter as May 15, 2022.

However, in this example, the organizational policy is for March to be the first month of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inquarter(date,'05/15/2022', 0, 3) as in_quarter
;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
```

```
8194, '5/7/2022', 40.39  
8195, '5/16/2022', 87.21  
8196, '6/15/2022', 95.93  
8197, '6/26/2022', 45.89  
8198, '7/9/2022', 36.23  
8199, '7/22/2022', 25.66  
8200, '7/23/2022', 82.77  
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_qtr

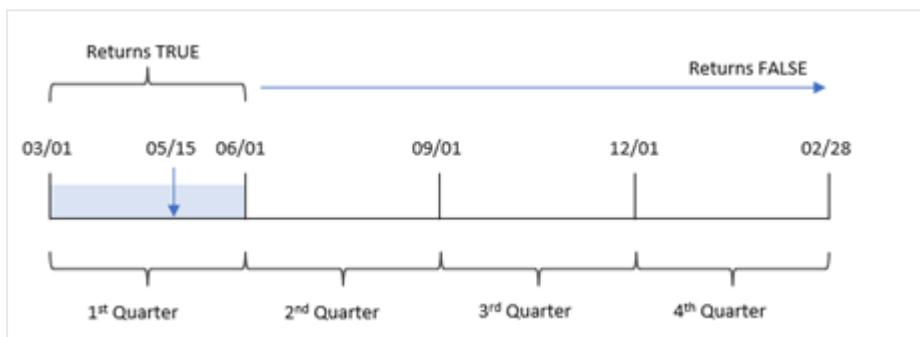
Results table

date	previous_qtr
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0

date	previous_qtr
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

Using 3 as the `first_month_of_year` argument in the `inquarter()` function sets March 1 as the start of the year and then divides the year into quarters. Therefore, the quarter segments are Mar-May, Jun-Aug, Sep-Nov, Dec-Feb. The `base_date` of May 15 sets the Mar-May quarter as the comparator quarter for the function.

Diagram of `inquarter()` function with March set as the first month of the year



Therefore, any transaction that occurs between March 1 and May 31 will return a Boolean result of TRUE.

Example 4 - Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inquarter()` function that is set as the ‘in_quarter’ field, and determines which transactions took place in the same quarter as May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
Load
*
Inline
[
id,date,amount
```

```
8188,'1/19/2022',37.23  
8189,'1/7/2022',17.17  
8190,'2/28/2022',88.27  
8191,'2/5/2022',57.42  
8192,'3/16/2022',53.80  
8193,'4/1/2022',82.06  
8194,'5/7/2022',40.39  
8195,'5/16/2022',87.21  
8196,'6/15/2022',95.93  
8197,'6/26/2022',45.89  
8198,'7/9/2022',36.23  
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- date

Create the following measure to calculate whether transactions took place in the same quarter as May 15:

```
=inquarter(date,'05/15/2022', 0)
```

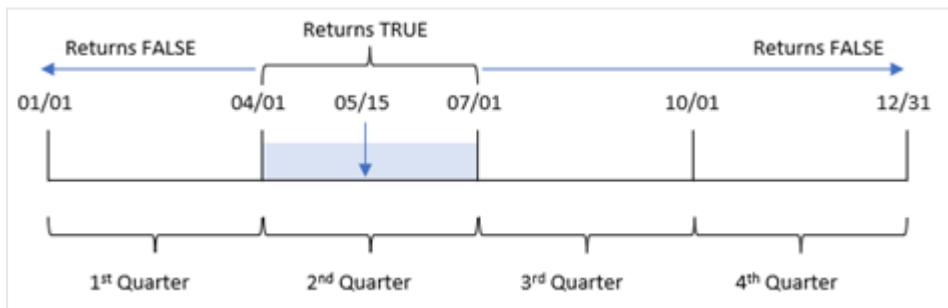
Results table

date	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0

date	in_quarter
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The ‘in_quarter’ measure is created in the chart by using the `inquarter()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for May 15 that identifies which quarter to define as the comparator. A period_no of 0 is the final argument and ensures the `inquarter()` function does not compare quarters preceding or following the segmented quarter.

Diagram of `inquarter()` function with May 15 as the base date



Any transaction that occurs between April 1 and the end of June 30 returns a Boolean result of TRUE.

Example 5 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called ‘Products’.
- The table contains the following fields:
 - product ID
 - product type

- manufacture date
- cost price

It has been identified that due to equipment error, products that were manufactured in the quarter of May 15, 2022 were defective. The end user would like a chart that displays, by quarter name, the status of which products manufactured were ‘defective’ or ‘faultless’ and the cost of the products manufactured in that quarter.

Load script

```
Products:  
Load  
*  
Inline  
[  
product_id,manufacture_date,cost_price  
8188,'1/19/2022',37.23  
8189,'1/7/2022',17.17  
8190,'2/28/2022',88.27  
8191,'2/5/2022',57.42  
8192,'3/16/2022',53.80  
8193,'4/1/2022',82.06  
8194,'5/7/2022',40.39  
8195,'5/16/2022',87.21  
8196,'6/15/2022',95.93  
8197,'6/26/2022',45.89  
8198,'7/9/2022',36.23  
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

```
=quartername(manufacture_date)
```

Create the following measures:

- =if(only(InQuarter(manufacture_date,makedate(2022,05,15),0)),'Defective','Faultless'), to identify which of the products are defective and which are faultless using the inquarter() function.
- =sum(cost_price), to show the sum of the cost of each product.

Do the following:

1. Set the measure's **Number Formatting** to **Money**.
2. Under **Appearance**, turn off **Totals**.

Results table

quartername (manufacture_date)	=if(only(InQuarter(manufacture_date,makedate (2022,05,15),0)),'Defective','Faultless')	Sum(cost_ price)
Jan-Mar 2022	Faultless	253.89
Apr-Jun 2022	Defective	351.48
Jul-Sep 2022	Faultless	446.31
Oct-Dec 2022	Faultless	163.91

The `inquarter()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in the quarter that contains May 15, the `inquarter()` function returns a Boolean value of TRUE and marks the products as 'Defective'. For any product returning a value of FALSE, and therefore not manufactured in that quarter, it marks the products as 'Faultless'.

inquarterToDate

This function returns True if **timestamp** lies inside the part of the quarter containing **base_date** up until and including the last millisecond of **base_date**.

Syntax:

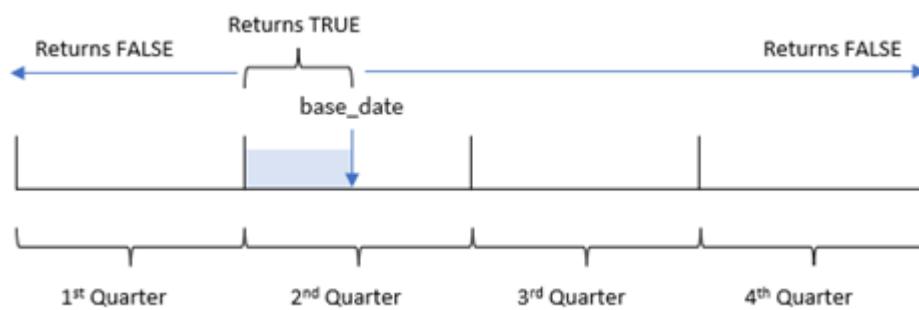
```
InQuarterToDate (timestamp, base_date, period_no [, first_month_of_year])
```

Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of inquarterToDate function



The `inquartertodate()` function divides the year into four equal quarters between January 1 and December 31 (or the user-defined start of year and its corresponding end date). Using the `base_date`, the function will then segment a particular quarter, with the `base_date` identifying both which quarter and the maximum allowed date for that quarter segment. Finally, the function returns a Boolean result when comparing the prescribed date values to that segment.

Arguments

Argument	Description
<code>timestamp</code>	The date that you want to compare with <code>base_date</code> .
<code>base_date</code>	Date that is used to evaluate the quarter.
<code>period_no</code>	The quarter can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the quarter which contains <code>base_date</code> . Negative values in <code>period_no</code> indicate preceding quarters and positive values indicate succeeding quarters.
<code>first_month_of_year</code>	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in <code>first_month_of_year</code> .

When to use it

The `inquartertodate()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. The `inquartertodate()` function would be used to return an aggregation or calculation dependent on whether a date evaluated occurred in the quarter up to and including the date in question.

For example, the `inquartertodate()` function can be used to identify all equipment manufactured in a quarter up to a specific date.

Function examples

Example	Result
<code>inquartertodate('01/25/2013', '03/25/2013', 0)</code>	Returns TRUE, since the value of <code>timestamp</code> , 01/25/2013, lies within the three-month period from 01/01/2013 to 03/25/2013, in which the value of <code>base_date</code> , 03/25/2013, lies.
<code>inquartertodate('04/26/2013', '03/25/2013', 0)</code>	Returns FALSE, since 04/26/2013 is outside the same period as the previous example.
<code>inquartertodate('02/25/2013', '06/09/2013', -1)</code>	Returns TRUE, since the value of <code>period_no</code> , -1, shifts the search period back one period of three months (one quarter of the year). This makes the search period 01/01/2013 to 03/09/2013.
<code>inquartertodate('03/25/2006', '04/15/2006', 0, 2)</code>	Returns TRUE, since the value of <code>first_month_of_year</code> is set to 2, which makes the search period 02/01/2006 to 04/15/2006 instead of 04/01/2006 to 04/15/2006.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `in_quarter_to_date`, that determines which transactions took place in the quarter up until May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inquartertodate(date, '05/15/2022', 0) as in_quarter_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
```

```
8196,'6/15/2022',95.93  
8197,'6/26/2022',45.89  
8198,'7/9/2022',36.23  
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_quarter_to_date

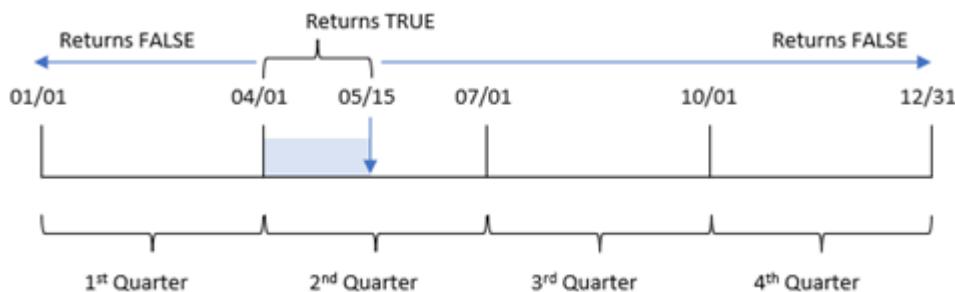
Results table

date	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

date	in_quarter_to_date
9/26/2022	0
10/14/2022	0
10/29/2022	0

The `in_quarter_to_date` field is created in the preceding load statement by using the `inquartertodate()` function. The first argument provided identifies which field is being evaluated. The second argument is a hard-coded date for May 15, which is the `base_date` that identifies which quarter to segment and defines the end boundary of that segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing quarters preceding or following the segmented quarter.

Diagram of inquartertodate function, no additional arguments



Any transaction that occurs in between April 1 and May 15 returns a Boolean result of TRUE. Transactions dates of May 16 and later will return FALSE, as do any transactions before April 1.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_qtr_to_date`, that determines which transactions took place a full quarter before the quarter segment ending on May 15, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inquartertodate(date, '05/15/2022', -1) as previous_qtr_to_date
```

```
;  
Load  
*  
Inline  
[  
id,date,amount  
8188,'1/19/2022',37.23  
8189,'1/7/2022',17.17  
8190,'2/28/2022',88.27  
8191,'2/5/2022',57.42  
8192,'3/16/2022',53.80  
8193,'4/1/2022',82.06  
8194,'5/7/2022',40.39  
8195,'5/16/2022',87.21  
8196,'6/15/2022',95.93  
8197,'6/26/2022',45.89  
8198,'7/9/2022',36.23  
8199,'7/22/2022',25.66  
8200,'7/23/2022',82.77  
8201,'7/27/2022',69.98  
8202,'8/2/2022',76.11  
8203,'8/8/2022',25.12  
8204,'8/19/2022',46.23  
8205,'9/26/2022',84.21  
8206,'10/14/2022',96.24  
8207,'10/29/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_qtr_to_date

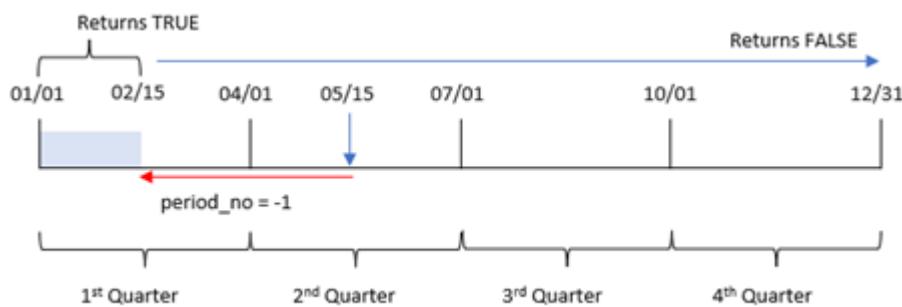
Results table

date	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0

date	previous_qtr_to_date
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

A period_no value of -1 indicates that the `inquartertodate` () function compares the input quarter segment to the preceding quarter. May 15 falls into the second quarter of the year, so the segment initially equates to between April 1 and May 15. The period_no then offsets this segment by three months earlier, causing the date boundaries to become January 1 to February 15.

Diagram of inquartertodate function, period_no example



Therefore, any transaction that occurs between January 1 and February 15 will return a Boolean result of TRUE.

Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `in_quarter_to_date`, that determines which transactions took place in the same quarter up to May 15, 2022.

In this example, we set March as the first month of the fiscal year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inquartertodate(date, '05/15/2022', 0,3) as in_quarter_to_date
;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_quarter_to_date`

Results table

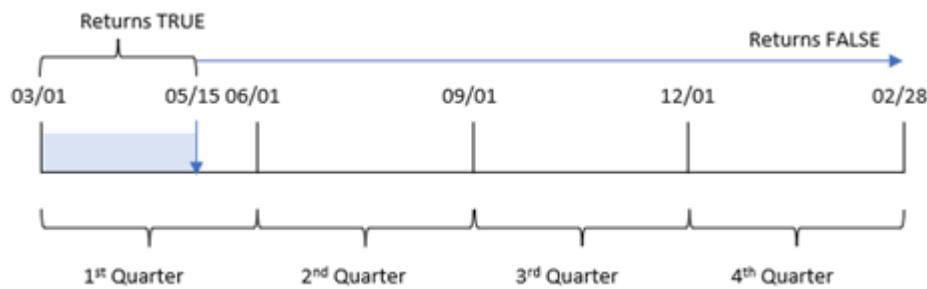
date	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

By using 3 as the `first_month_of_year` argument in the `inquartertodate()` function, the function begins the year on March 1, and then divides the year into quarters. Therefore, the quarter segments are:

- March to May
- June to August
- September to November
- December to February

The `base_date` of May 15 then segments the March to May quarter by setting its end boundary as May 15.

Diagram of inquarterToDate function, first_month_of_year example



Therefore, any transaction that occurs in between the March 1 and May 15 will return a Boolean result of TRUE, while transactions with dates outside these boundaries will return a value of FALSE.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that determines which transactions took place in the same quarter as May 15 is created as a measure in the chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
```

```
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:date.

Create the following measure:

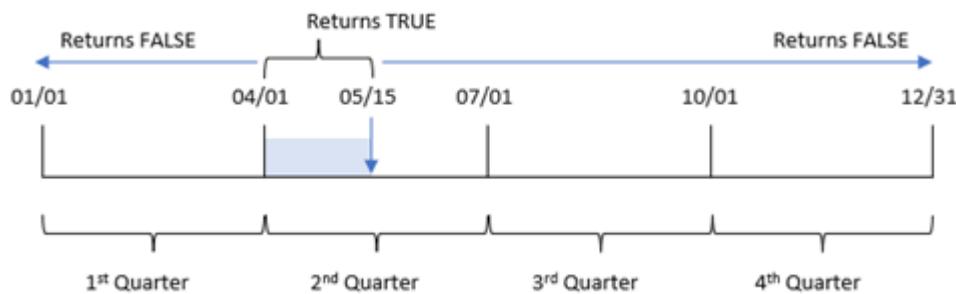
```
=inquartertodate(date, '05/15/2022', 0)
```

Results table

date	=inquartertodate(date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

The `in_quarter_to_date` measure is created in a chart object by using the `inquartertodate()` function. The first argument is the date field being evaluated. The second argument is a hard-coded date for May 15, which is the `base_date` that identifies which quarter to segment and defines the end boundary of that segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing quarters preceding or following the segmented quarter.

Diagram of inquartertodate function, chart object example



Any transaction that occurs between April 1 and May 15 returns a Boolean result of `TRUE`. Transactions on May 16 and later will return `FALSE`, as do any transactions before April 1.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Products`.
- Information concerning product ID, manufacture date, and cost price.

On May 15, 2022, a piece of equipment error was identified in the manufacturing process and resolved. Products that were manufactured in that quarter up to this date will be defective. The end user would like a chart object that displays, by quarter name, the status of whether the product is ‘defective’ or ‘faultless’ and the cost of the products manufactured in that quarter to date.

Load script

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
```

```

8192, '3/16/2022', 53.80
8193, '4/1/2022', 82.06
8194, '5/7/2022', 40.39
8195, '5/16/2022', 87.21
8196, '6/15/2022', 95.93
8197, '6/26/2022', 45.89
8198, '7/9/2022', 36.23
8199, '7/22/2022', 25.66
8200, '7/23/2022', 82.77
8201, '7/27/2022', 69.98
8202, '8/2/2022', 76.11
8203, '8/8/2022', 25.12
8204, '8/19/2022', 46.23
8205, '9/26/2022', 84.21
8206, '10/14/2022', 96.24
8207, '10/29/2022', 67.67
];

```

Results

Do the following:

1. Load the data and open a sheet. Create a new table. Create a dimension to show the quarter names:
`=quartername(manufacture_date)`
2. Next, create a dimension to identify which of the products are defective and which are faultless:
`=if(inquartertodate(manufacture_date,makedate(2022,05,15),0),'Defective','Faultless')`
3. Create a measure to sum the `cost_price` of the products:
`=sum(cost_price)`
4. Set the measure's **Number formatting** to **Money**.

Results table

quartername (manufacture_date)	if(inquartertodate(manufacture_date,makedate (2022,05,15),0),'Defective','Faultless')	Sum(cost_ price)
Jan-Mar 2022	Faultless	\$253.89
Apr-Jun 2022	Faultless	\$229.03
Apr-Jun 2022	Defective	\$122.45
Jul-Sep 2022	Faultless	\$446.31
Oct-Dec 2022	Faultless	\$163.91

The `inquartertodate()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For those that return a Boolean value of TRUE, it marks the products as 'Defective'. For any product returning a value of FALSE, and therefore not made in the quarter up to and including May 15, it marks the products as 'Faultless'.

inweek

This function returns True if **timestamp** lies inside the week containing **base_date**.

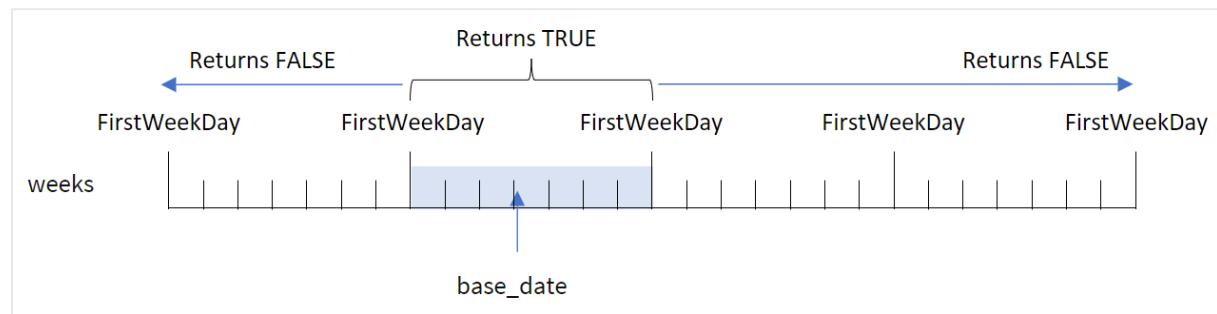
Syntax:

```
InWeek (timestamp, base_date, period_no[, first_week_day])
```

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of inweek() function's range



The inweek() function uses the `base_date` argument to identify which seven-day period the date falls into. The start day of the week is based on the `FirstWeekDay` system variable. However, you can also change the first day of the week by using the `first_week_day` argument of the inweek() function.

After the selected week has been defined, the function will return Boolean results when comparing the prescribed date values to that week segment.

When to use it

The inweek() function returns a Boolean result. Typically, this type of function will be used as a condition in an if expression. The inweek() function returns an aggregation or calculation which depends on whether a date evaluated occurred in the week with the selected date of the `base_date` argument.

For example, the inweek() function can be used to identify all equipment manufactured in a specific week.

Arguments

Argument	Description
timestamp	The date that you want to compare with <code>base_date</code> .
base_date	Date that is used to evaluate the week.
period_no	The week can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the week which contains <code>base_date</code> . Negative values in <code>period_no</code> indicate preceding weeks and positive values indicate succeeding weeks.
first_week_day	By default, the first day of the week is Sunday (as determined by the <code>FirstWeekDay</code> system variable), starting at midnight between Saturday and Sunday. The <code>first_week_day</code> parameter supersedes the <code>FirstWeekDay</code> variable. To indicate the week starting on another day, specify a flag between 0 and 6.

first_week_day values

Day	Value
Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inweek('01/12/2006', '01/14/2006', 0)</code>	Returns TRUE
<code>inweek('01/12/2006', '01/20/2006', 0)</code>	Returns FALSE
<code>inweek('01/12/2006', '01/14/2006', -1)</code>	Returns FALSE
<code>inweek('01/07/2006', '01/14/2006', -1)</code>	Returns TRUE
<code>inweek('01/12/2006', '01/09/2006', 0, 3)</code>	Returns FALSE because <code>first_week_day</code> is specified as 3 (Thursday), which makes 01/12/2006 the first day of the week following the week containing 01/09/2006.

These topics may help you work with this function:

Related topics

Topic	Default Flag / Value	Description
<i>FirstWeekDay</i> (page 215)	6 / Sunday	Defines the start day of each week.

Example 1 - No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the month of January 2022 which is loaded into a table called ‘Transactions’.
- The `FirstWeekday` system variable which is set to 6 (Sunday).
- A preceding load which contains the following:
 - The `inweek()` function, set as the field ‘in_week’ that determines which transactions took place in the week of January 14, 2022.
 - The `weekday()` function, set as the field ‘week_day’ that shows which day of the week corresponds to each date.

Load script

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    weekday(date) as week_day,
    inweek(date,'01/14/2022', 0) as in_week
  ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
```

```
8199, '01/16/2022', 25.66  
8200, '01/17/2022', 82.77  
8201, '01/18/2022', 69.98  
8202, '01/26/2022', 76.11  
8203, '01/27/2022', 25.12  
8204, '01/28/2022', 46.23  
8205, '01/29/2022', 84.21  
8206, '01/30/2022', 96.24  
8207, '01/31/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week_day
- in_week

Results table

date	week_day	in_week
01/02/2022	Sun	0
01/05/2022	Wed	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	-1
01/10/2022	Mon	-1
01/11/2022	Tue	-1
01/12/2022	Wed	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0

date	week_day	in_week
01/30/2022	Sun	0
01/31/2022	Mon	0

The ‘in_week’ field is created in the preceding load statement by using the `inweek()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for January 14 which is the `base_date`. The `base_date` argument works in with the `Firstweekday` system variable to identify the comparator week. A `period_no` of 0 — meaning that the function is not comparing weeks preceding or following the segmented week — is the final argument.

The `Firstweekday` system variable determines that weeks begin on a Sunday and end on a Saturday. Therefore, January would be broken into weeks according to the diagram below, with the dates between January 9 and 15 providing the valid period for the `inweek()` calculation:

Diagram of calendar with the `inweek()` function's range highlighted

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Any transaction that occurs between January 9 and the 15 of January returns a Boolean result of TRUE.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- The `Firstweekday` system variable which is set to 6 (Sunday).
- A preceding load which contains the following:
 - The `inweek()` function, set as the field ‘`prev_week`’ that determines which transactions took place a full week before the week of January 14, 2022.
 - The `weekday()` function, set as the field ‘`week_day`’ that shows which day of the week corresponds to each date.

Load script

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  weekday(date) as week_day,
  inweek(date, '01/14/2022', -1) as prev_week
;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
```

```
8206, '01/30/2022', 96.24  
8207, '01/31/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week_day
- prev_week

Results table

date	week_day	prev_week
01/02/2022	Sun	-1
01/05/2022	Wed	-1
01/06/2022	Thu	-1
01/08/2022	Sat	-1
01/09/2022	Sun	0
01/10/2022	Mon	0
01/11/2022	Tue	0
01/12/2022	Wed	0
01/13/2022	Thu	0
01/14/2022	Fri	0
01/15/2022	Sat	0
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

Using -1 as the period_no argument in the inweek() function shifts the boundaries of the comparator week back by a full seven days. With a period_no of 0 the week would be between January 9 and 15. But in this example, the period_no of -1 shifts the start and end boundary of this segment backwards by one week. The date boundaries become January 2 to January 8.

Diagram of calendar with the inweek() function's range highlighted

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Therefore, any transaction that occurs between January 2 and January 8 will return a Boolean result of TRUE.

Example 3 - first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- The FirstWeekDay system variable which is set to 6 (Sunday).

- A preceding load which contains the following:
 - The `inweek()` function, set as the field ‘`in_week`’ that determines which transactions took place in the week of January 14, 2022.
 - The `weekday()` function, set as the field ‘`week_day`’ that shows which day of the week corresponds to each date.

Load script

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  weekday(date) as week_day,
  inweek(date, '01/14/2022', 0, 0) as in_week
;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `week_day`
- `in_week`

Results table

date	week_day	in_week
01/02/2022	Sun	0
01/05/2022	Wed	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	0
01/10/2022	Mon	-1
01/11/2022	Tue	-1
01/12/2022	Wed	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	Sun	-1
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

Using 0 as the `first_week_day` argument in the `inweek()` function supersedes the `Firstweekday` system variable and sets Monday as the first day of the week.

Diagram of calendar with the `inweek()` function's range highlighted

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Therefore, any transaction that occurs between January 10 and 16 will return a Boolean result of TRUE.

Example 4 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. Create a measure in the results table to determine which transactions took place in the week of January 14, 2022.

Load script

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
```

```
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- date

Create the following measures:

- =inweek (date, '01/14/2022',0), to calculate whether transactions took place in the same week as January 14.
- =weekday(date), to show which day of the week corresponds to each date.

Results table

date	week_day	=inweek (date,'01/14/2022',0)
01/02/2022	Sun	0
01/05/2022	Wed	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	Sun	-1
01/10/2022	Mon	-1
01/11/2022	Tue	-1

date	week_day	=inweek (date,'01/14/2022',0)
01/12/2022	Wed	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	Sun	0
01/17/2022	Mon	0
01/18/2022	Tue	0
01/26/2022	Wed	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	Sun	0
01/31/2022	Mon	0

The ‘in_week’ measure is created in chart by using the `inweek()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for January 14 which is the `base_date`. The `base_date` argument works in with the `Firstweekday` system variable to identify the comparator week. A `period_no` of 0 is the final argument.

The `Firstweekday` system variable determines that weeks begin on a Sunday and end on a Saturday. Therefore, January would be broken into weeks according to the diagram below, with the dates between January 9 and 15 providing the valid period for the `inweek()` calculation:

Diagram of calendar with the `inweek()` function's range highlighted

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Any transaction that occurs between January 9 and the 15 of January returns a Boolean result of TRUE.

Example 5 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called ‘Products’.
- The table contains the following fields:
 - product ID
 - product type
 - manufacture date
 - cost price

It has been identified that due to equipment error, products that were manufactured in the week of January 12 were defective. The end user would like a chart that displays, by week, the status of which products manufactured were ‘defective’ or ‘faultless’ and the cost of the products manufactured in that week.

Load script

```
Products:  
Load  
*  
Inline  
[  
product_id,manufacture_date,cost_price  
8188,'01/02/2022',37.23  
8189,'01/05/2022',17.17  
8190,'01/06/2022',88.27  
8191,'01/08/2022',57.42  
8192,'01/09/2022',53.80  
8193,'01/10/2022',82.06  
8194,'01/11/2022',40.39  
8195,'01/12/2022',87.21  
8196,'01/13/2022',95.93  
8197,'01/14/2022',45.89  
8198,'01/15/2022',36.23  
8199,'01/16/2022',25.66  
8200,'01/17/2022',82.77  
8201,'01/18/2022',69.98  
8202,'01/26/2022',76.11  
8203,'01/27/2022',25.12  
8204,'01/28/2022',46.23  
8205,'01/29/2022',84.21  
8206,'01/30/2022',96.24  
8207,'01/31/2022',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- =weekname(manufacture_date)

Create the following measures:

- =if(only(inweek(manufacture_date,makedate(2022,01,12),0)),'Defective','Faultless'), to identify which of the products are defective and which are faultless using the inweek() function.
- =sum(cost_price), to show the sum of the cost of each product.

Do the following:

1. Set the measure’s **Number Formatting** to **Money**.
2. Under **Appearance**, turn off **Totals**.

Results table

weekname (manufacture_date)	=if(only(inweek(manufacture_date,makedate(2022,01,12),0)), 'Defective','Faultless')	Sum(cost_price)
2022/02	Faultless	200.09
2022/03	Defective	441.51
2022/04	Faultless	178.41
2022/05	Faultless	231.67
2022/06	Faultless	163.91

The `inweek()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in the week of January 12, the `inweek()` function returns a Boolean value of TRUE and marks the products as ‘Defective’. For any product returning a value of FALSE, and therefore not manufactured in that week, it marks the products as ‘Faultless’.

inweektodate

This function returns True if **timestamp** lies inside the part of week containing **base_date** up until and including the last millisecond of **base_date**.

Syntax:

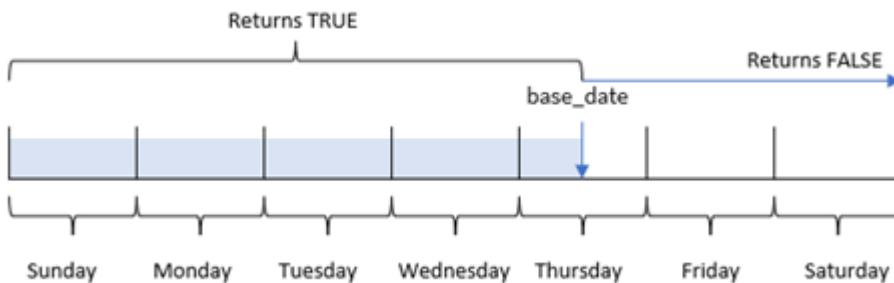
```
InWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of `inweektodate` function



The `inweektodate()` function uses the `base_date` parameter to identify a maximum boundary date of a week segment, as well as its corresponding date for the start of the week, which is based on the `Firstweekday` system variable (or user-defined `first_week_day` parameter). Once this week segment has been defined, the function will then return Boolean results when comparing the prescribed date values to that segment.

When to use it

The `inweektodate()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. This will return an aggregation or calculation dependent on whether a date evaluated occurred during the week in question up to and including a particular date.

For example, the `inweektodate()` function can be used to calculate all sales made during a specified week up to a particular date.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the week.
period_no	The week can be offset by period_no . period_no is an integer, where the value 0 indicates the week which contains base_date . Negative values in period_no indicate preceding weeks and positive values indicate succeeding weeks.
first_week_day	<p>By default, the first day of the week is Sunday (as determined by the <code>FirstWeekDay</code> system variable), starting at midnight between Saturday and Sunday. The first_week_day parameter supersedes the <code>FirstWeekDay</code> variable. To indicate the week starting on another day, specify a flag between 0 and 6.</p> <p>For a week starting on Monday and ending on Sunday, use a flag of 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday.</p>

Function examples

Example	Interaction
<code>inweektodate('01/12/2006', '01/12/2006', 0)</code>	Returns TRUE.
<code>inweektodate('01/12/2006', '01/11/2006', 0)</code>	Returns FALSE.
<code>inweektodate('01/12/2006', '01/18/2006', -1)</code>	<p>Returns FALSE.</p> <p>Because <code>period_no</code> is specified as -1, the effective date that <code>timestamp</code> is measured against is 01/11/2006.</p>
<code>inweektodate('01/11/2006', '01/12/2006', 0, 3)</code>	Returns FALSE, since <code>first_week_day</code> is specified as 3 (Thursday), which makes 01/12/2006 the first day of the week following the week containing 01/12/2006.

These topics may help you work with this function:

Related topics

Topic	Default Flag / Value	Description
<i>FirstWeekDay</i> (page 215)	6 / Sunday	Defines the start day of each week.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the month of January 2022, which is loaded into a table called `Transactions`.
- The data field provided in the `TimestampFormat='M/D/YYYY h:mm:ss[.ffff]'` format.
- The creation of a field, `in_week_to_date`, which determines which transactions took place in the week up until January 14, 2022.
- The creation of an additional field, named `weekday`, using the `weekday()` function. This new field is created to show which day of the week corresponds to each date.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff]';
SET FirstWeekDay=6;
Transactions:
  Load
    *,
    weekday(date) as week_day,
    inweektodate(date, '01/14/2022', 0) as in_week_to_date
  ;
Load
*
Inline
```

```
[  
id,date,amount  
8188,'2022-01-02 12:22:06',37.23  
8189,'2022-01-05 01:02:30',17.17  
8190,'2022-01-06 15:36:20',88.27  
8191,'2022-01-08 10:58:35',57.42  
8192,'2022-01-09 08:53:32',53.80  
8193,'2022-01-10 21:13:01',82.06  
8194,'2022-01-11 00:57:13',40.39  
8195,'2022-01-12 09:26:02',87.21  
8196,'2022-01-13 15:05:09',95.93  
8197,'2022-01-14 18:44:57',45.89  
8198,'2022-01-15 06:10:46',36.23  
8199,'2022-01-16 06:39:27',25.66  
8200,'2022-01-17 10:44:16',82.77  
8201,'2022-01-18 18:48:17',69.98  
8202,'2022-01-26 04:36:03',76.11  
8203,'2022-01-27 08:07:49',25.12  
8204,'2022-01-28 12:24:29',46.23  
8205,'2022-01-30 11:56:56',84.21  
8206,'2022-01-30 14:40:19',96.24  
8207,'2022-01-31 05:28:21',67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week_day
- in_week_to_date

Results table

date	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	-1
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1

date	week_day	in_week_to_date
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

The `in_week_to_date` field is created in the preceding load statement by using the `inweektodate()` function. The first argument provided identifies which field is being evaluated. The second argument is a hard-coded date for January 14, which is the `base_date` that identifies which week to segment and defines the end boundary of that segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing weeks preceding or following the segmented week.

The `Firstweekday` system variable determines that weeks begin on a Sunday and end on a Saturday. Therefore, January would be broken into weeks according to the diagram below, with the dates between January 9 and 14 providing the valid period for the `inweekdodate()` calculation:

Calendar diagram showing transaction dates which would return a Boolean result of TRUE

Sun	Mon	Tue	Wed	Thur	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Any transaction that occurs in between January 9 and 14 returns a Boolean result of TRUE. Transactions before and after the dates return a Boolean result of FALSE.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, prev_week_to_date, that determines which transactions took place a full week before the week segment ending on January 14, 2022.
- The creation of an additional field, named weekday, using the weekday() function. This is to show which day of the week corresponds to each date.

Load script

```
SET FirstWeekDay=6;
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff]';
Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date, '01/14/2022', -1) as prev_week_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week_day
- prev_week_to_date

Results table

date	week_day	prev_week_to_date
2022-01-02 12:22:06	Sun	-1
2022-01-05 01:02:30	Wed	-1
2022-01-06 15:36:20	Thu	-1
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	0
2022-01-10 21:13:01	Mon	0
2022-01-11 00:57:13	Tue	0
2022-01-12 09:26:02	Wed	0
2022-01-13 15:05:09	Thu	0
2022-01-14 18:44:57	Fri	0
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

A period_no value of -1 indicates that the `iweektodate ()` function compares the input quarter segment to the preceding week. The week segment initially equates to between January 9 and January 14. The period_no then offsets both the start and end boundary of this segment to one week earlier, causing the date boundaries to become January 2 to January 7.

Calendar diagram showing transaction dates which would return a Boolean result of TRUE

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Therefore, any transaction that occurs between January 2 and 8 (not including January 8 itself) will return a Boolean result of TRUE.

Example 3 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `in_week_to_date`, that determines which transactions took place in the week up until January 14, 2022.
- The creation of an additional field, named `weekday`, using the `weekday()` function. This is to show which day of the week corresponds to each date.

In this example, we consider Monday as the first day of the week.

Load script

```
SET FirstWeekDay=6;  
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff]';
```

Transactions:

```
Load  
*,  
    weekday(date) as week_day,  
    inweektodate(date, '01/14/2022', 0, 0) as in_week_to_date  
;
```

Load

*

```
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- week_day
- in_week_to_date

Results table

date	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	0
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1

date	week_day	in_week_to_date
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

By using 0 as the `first_week_day` argument in the `inweektodate()` function, the function argument supersedes the `FirstWeekDay` system variable and sets Monday as the first day of the week.

Calendar diagram showing transaction dates which would return a Boolean result of TRUE

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	17
24	25	26	27	28	29	30
31						

Therefore, any transaction that occurs in between January 10 and 14 will return a Boolean result of `TRUE`, while transactions with dates outside these boundaries will return a value of `FALSE`.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that determines which transactions took place in the week up until January 14, 2022 is created as a measure in the chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add this field as a dimension: date.
2. To calculate whether transactions took place in the same week up until the 14th of January, create the following measure:
`=inweektodate(date, '01/14/2022', 0)`
3. To show which day of the week corresponds to each date, create an additional measure:
`=weekday(date)`

Results table

date	week_day	in_week_to_date
2022-01-02 12:22:06	Sun	0
2022-01-05 01:02:30	Wed	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	Sun	-1
2022-01-10 21:13:01	Mon	-1
2022-01-11 00:57:13	Tue	-1
2022-01-12 09:26:02	Wed	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	Sun	0
2022-01-17 10:44:16	Mon	0
2022-01-18 18:48:17	Tue	0
2022-01-26 04:36:03	Wed	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	Sun	0
2022-01-30 14:40:19	Sun	0
2022-01-31 05:28:21	Mon	0

The `in_week_to_date` field is created as a measure in the chart object using the `inweektodate()` function. The first argument provided identifies which field is being evaluated. The second argument is a hard-coded date for January 14, which is the `base_date` that identifies which week to segment and defines the end boundary of that segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing weeks preceding or following the segmented week.

The `FirstWeekday` system variable determines that weeks begin on a Sunday and end on a Saturday. Therefore, January would be broken into weeks according to the diagram below, with the dates between January 9 and 14 providing the valid period for the `inweekdodate()` calculation:

Calendar diagram showing transaction dates which would return a Boolean result of TRUE

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Any transaction that occurs in between January 9 and 14 returns a Boolean result of `TRUE`. Transactions before and after the dates return a Boolean result of `FALSE`.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Products`.
- Information concerning product ID, manufacture date, and cost price.

It has been identified that due to equipment error, products that were manufactured in the week of January 12 were defective. The issue was resolved on January 13. The end user would like a chart object that displays, by week, the status of whether the products manufactured are ‘defective’ or ‘faultless’, and the cost of the products manufactured in that week.

Load script

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];

```

Results

Do the following:

1. Load the data and open a sheet. Create a new table. Create a dimension to show the week names:
=weekname(manufacture_date)
2. Next, create a dimension to identify which of the products are defective and which are faultless:
=if(inweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')
3. Create a measure to sum the cost_price of the products:
=sum(cost_price)
4. Set the measure's **Number formatting** to **Money**.

Results table

weekname(manufacture_date)	if(inweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')	Sum(cost_price)
2022/02	Faultless	\$200.09
2022/03	Defective	\$263.46
2022/03	Faultless	\$178.05

weekname(manufacture_date)	if(inweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')	Sum(cost_price)
2022/04	Faultless	\$178.41
2022/05	Faultless	\$147.46
2022/06	Faultless	\$248.12

The `inweektodate()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For those that return a Boolean value of `TRUE`, it marks the products as 'Defective'. For any product returning a value of `FALSE`, and therefore not made in the week up to January 12, it marks the products as 'Faultless'.

inyear

This function returns True if **timestamp** lies inside the year containing **base_date**.

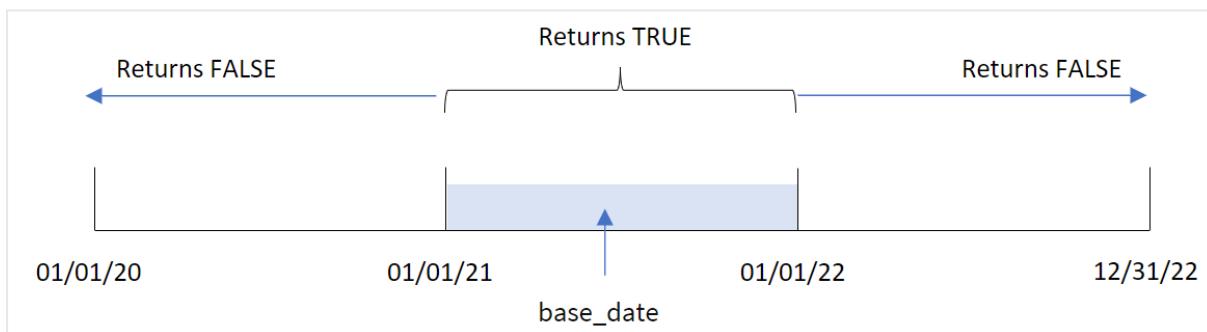
Syntax:

```
InYear (timestamp, base_date, period_no [, first_month_of_year])
```

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of `inyear()` function's range



The `inyear()` function returns a Boolean result when comparing the selected date values to a year defined by the `base_date`.

When to use it

The `inyear()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. This returns an aggregation or calculation dependent on whether a date evaluated occurred in the year in question. For example, the `inyear()` function can be used to identify all sales that occurred in a defined year.

Arguments

Argument	Description
timestamp	The date that you want to compare with base_date .
base_date	Date that is used to evaluate the year.
period_no	The year can be offset by period_no . period_no is an integer, where the value 0 indicates the year that contains base_date . Negative values in period_no indicate preceding years, and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

You can use the following values to set the first month of year in the **first_month_of_year** argument:

first_month_of_year values

Month	Value
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>inyear ('01/25/2013', '01/01/2013', 0)</code>	Returns TRUE
<code>inyear ('01/25/2012', '01/01/2013', 0)</code>	Returns FALSE
<code>inyear ('01/25/2013', '01/01/2013', -1)</code>	Returns FALSE
<code>inyear ('01/25/2012', '01/01/2013', -1)</code>	Returns TRUE
<code>inyear ('01/25/2013', '01/01/2013', 0, 3)</code>	Returns TRUE The value of <code>base_date</code> and <code>first_month_of_year</code> specify that timestamp must fall within 01/03/2012 and 02/28/2013
<code>inyear ('03/25/2013', '07/01/2013', 0, 3)</code>	Returns TRUE

Example 1 - Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inyear()` function that is set as the ‘`in_year`’ field, and determines which transactions took place in the same year as July 26, 2021.

Load script

```
SET DateFormat='MM/DD/YYYY';
Transactions:
Load
  *,
  inyear(date,'07/26/2021', 0) as in_year
;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
```

```
8191, '04/16/2020', 57.42
8192, '05/21/2020', 53.80
8193, '08/14/2020', 82.06
8194, '10/07/2020', 40.39
8195, '12/05/2020', 87.21
8196, '01/22/2021', 95.93
8197, '02/03/2021', 45.89
8198, '03/17/2021', 36.23
8199, '04/23/2021', 25.66
8200, '05/04/2021', 82.77
8201, '06/30/2021', 69.98
8202, '07/26/2021', 76.11
8203, '12/27/2021', 25.12
8204, '06/06/2022', 46.23
8205, '07/18/2022', 84.21
8206, '11/14/2022', 96.24
8207, '12/12/2022', 67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_year

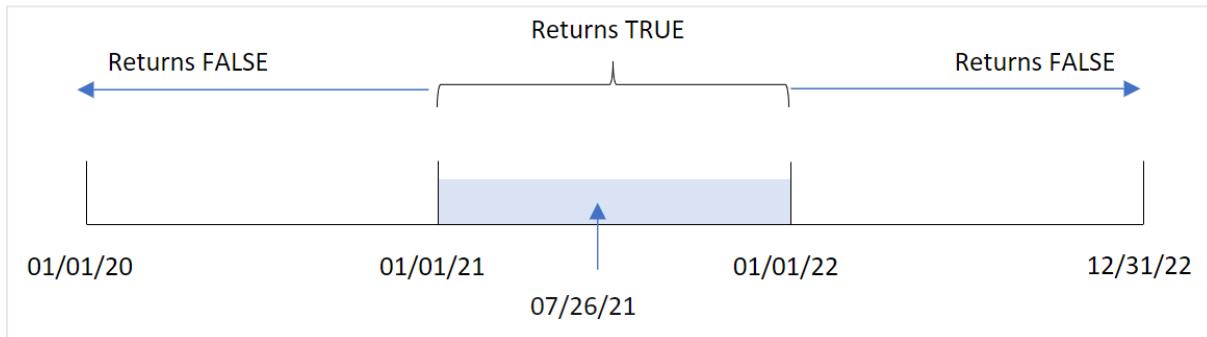
Results table

date	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1

date	in_year
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

The ‘in_year’ field is created in the preceding load statement by using the `inyear()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for July 26, 2021 which is the `base_date` that determines the comparator year. A `period_no` of 0 is the final argument meaning that the `inyear()` function does not compare years preceding or following the year.

Diagram of `inyear()` function's range with July 26 as the base date



Any transaction that occurs in 2021 returns a Boolean result of TRUE.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inyear()` function that is set as the ‘previous_year’ field, and determines which transactions took place in the year before the year containing July 26, 2021.

Load script

```
SET DateFormat='MM/DD/YYYY';
Transactions:
```

```
Load
  *,
  inyear(date, '07/26/2021', -1) as previous_year
;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_year

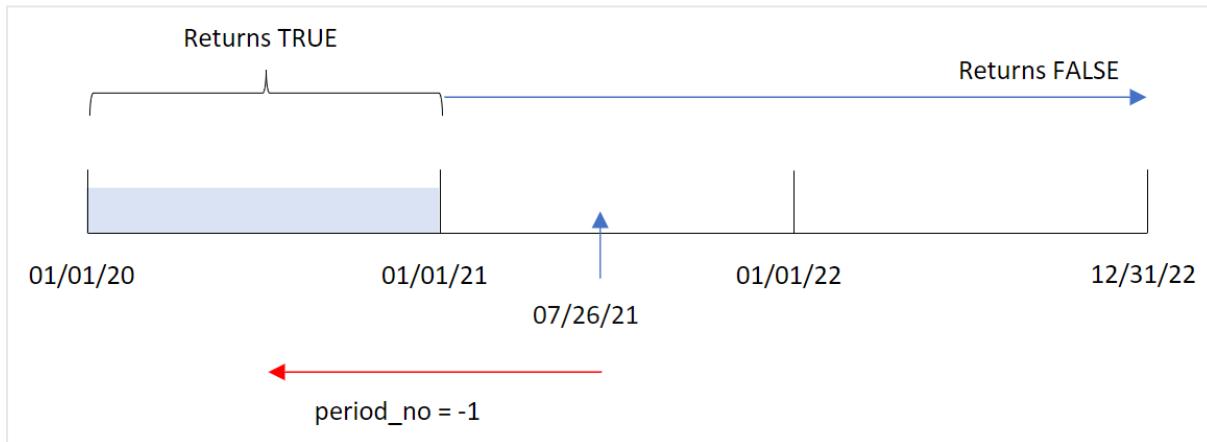
Results table

date	previous_year
01/13/2020	-1
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
08/14/2020	-1
10/07/2020	-1

date	previous_year
12/05/2020	-1
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

Using -1 as the period_no argument in the inyear() function shifts the boundaries of the comparator year back by a full year. 2021 is initially identified as the comparator year. The period_no offsets the comparator year by one, making 2020 the comparator year.

Diagram of inyear() function's range with the period_no argument set to -1



Therefore, any transaction that occurs in 2020 returns a Boolean result of TRUE.

Example 3 - first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the `inyear()` function that is set as the ‘`in_year`’ field, and determines which transactions took place in the same year as July 26, 2021.

However, in this example, the organizational policy is for March to be the first month of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';
Transactions:
  Load
    *,
    inyear(date, '07/26/2021', 0, 3) as in_year
  ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

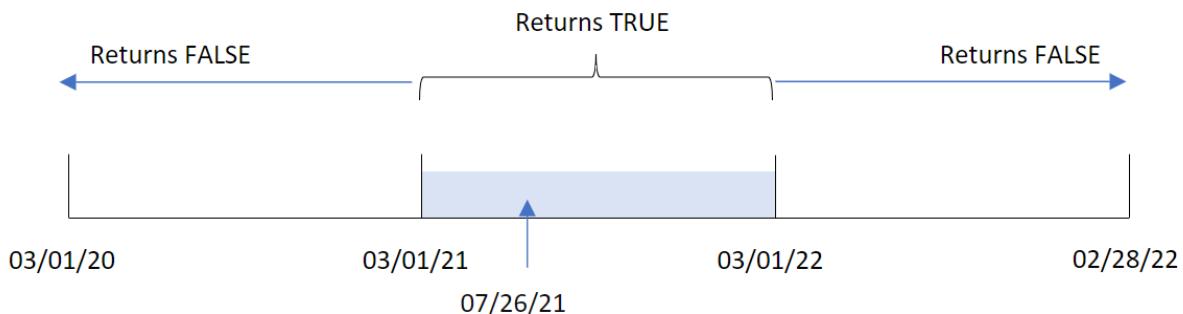
- date
- in_year

Results table

date	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

Using 3 as the first_month_of_year argument in the inyear() function begins the year on March 1 and ends the year at the end of February.

Diagram of `inyear()` function's range with March set as the first month of the year



Therefore, any transaction that occurs between March 1, 2021 and March 1, 2022 will return a Boolean result of TRUE.

Example 4 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that determines whether transactions took place in the same year as July 26, 2021 is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
```

```
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- date

To calculate whether transactions took place in the same year as July 26, 2021, create the following measure:

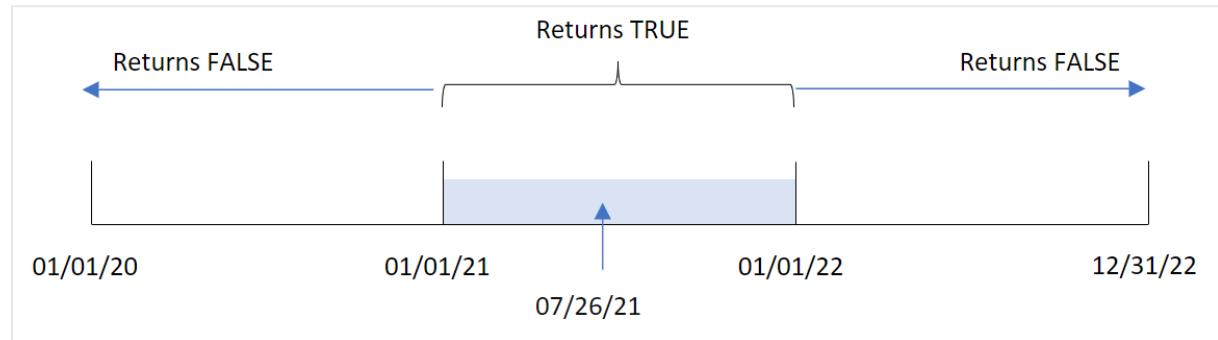
- =inyear(date, '07/26/2021', 0)

Results table

date	=inyear(date,'07/26/2021',0)
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

The ‘in_year’ field is created in the chart by using the `inyear()` function. The first argument identifies which field is being evaluated. The second argument is a hard-coded date for July 26, 2021 which is the `base_date` that determines the comparator year. A `period_no` of 0 is the final argument meaning that the `inyear()` function does not compare years preceding or following the year.

Diagram of `inyear()` function's range with July 27 as the base date



Any transaction that occurs in 2021 returns a Boolean result of TRUE.

Example 5 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called ‘Products’.
- The table contains the following fields:
 - product ID
 - product type
 - manufacture date
 - cost price

The end user would like a chart object that displays, by product type, the cost of the products manufactured in 2021.

Load script

```
Products:  
Load  
*  
Inline  
[  
product_id,product_type,manufacture_date,cost_price  
8188,product A,'01/13/2020',37.23  
8189,product B,'02/26/2020',17.17  
8190,product B,'03/27/2020',88.27
```

```

8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

- product_type

To calculate the sum of each product that was manufactured in 2021, create the following measure:

- =sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))

Do the following:

1. Set the measure's **Number Formatting** to **Money**.
2. Under **Appearance**, turn off **Totals**.

Results table

product_type	=sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))
product A	\$95.93
product B	\$128.66
product C	\$61.89
product D	\$171.21

The `inyear()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in 2021, the `inyear()` function returns a Boolean value of TRUE and shows the sum of the `cost_price`.

inyearToDate

This function returns True if **timestamp** lies inside the part of year containing **base_date** up until and including the last millisecond of **base_date**.

Syntax:

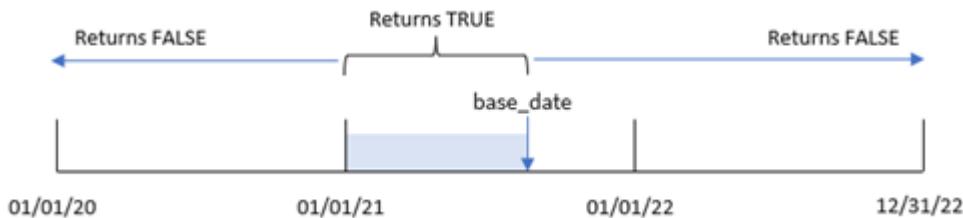
```
InYearToDate (timestamp, base_date, period_no[, first_month_of_year])
```

Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Diagram of inyeartodate function



The `inyearstodate()` function will segment a particular portion of the year with the `base_date`, identifying the maximum allowed date for that year segment. The function then evaluates whether a date field or value falls into this segment and returns a Boolean result.

Arguments

Argument	Description
<code>timestamp</code>	The date that you want to compare with <code>base_date</code> .
<code>base_date</code>	Date that is used to evaluate the year.
<code>period_no</code>	The year can be offset by <code>period_no</code> . <code>period_no</code> is an integer, where the value 0 indicates the year that contains <code>base_date</code> . Negative values in <code>period_no</code> indicate preceding years, and positive values indicate succeeding years.
<code>first_month_of_year</code>	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in <code>first_month_of_year</code> .

When to use it

The `inyearstodate()` function returns a Boolean result. Typically, this type of function will be used as a condition in an `if` expression. This would return an aggregation or calculation dependent on whether a date evaluated occurred in the year up to and including the date in question.

For example, the `inyearstodate()` function can be used to identify all equipment manufactured in a year up to a specific date.

These examples use the date format MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>inyearstodate ('01/25/2013', '02/01/2013', 0)</code>	Returns TRUE.
<code>inyearstodate ('01/25/2012', '01/01/2013', 0)</code>	Returns FALSE.
<code>inyearstodate ('01/25/2012', '02/01/2013', -1)</code>	Returns TRUE.
<code>inyearstodate ('11/25/2012', '01/31/2013', 0, 4)</code>	Returns TRUE. The value of timestamp falls inside the fiscal year beginning in the fourth month and before the value of base_date.
<code>inyearstodate ('3/31/2013', '01/31/2013', 0, 4)</code>	Returns FALSE. Compared with the previous example, the value of timestamp is still inside the fiscal year, but it is after the value of base_date, so it falls outside the part of the year.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

- The creation of a field, `in_year_to_date`, that determines which transactions took place in the year up until July 26, 2021.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inyeartodate(date, '07/26/2021', 0) as in_year_to_date
  ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `in_year_to_date`

Results table

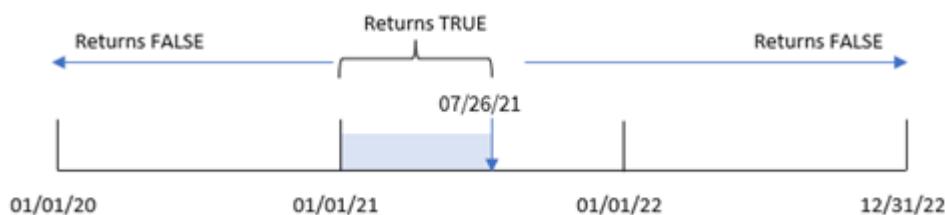
date	in_year_to_date
01/13/2020	0
02/26/2020	0

date	in_year_to_date
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

The `in_year_to_date` field is created in the preceding load statement by using the `inyeartodate()` function. The first argument provided identifies which field is being evaluated.

The second argument is a hard-coded date for the July 26, 2021, which is the `base_date` that identifies the end boundary of the year segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing years preceding or following the segmented year.

Diagram of inyeartodate function, no additional arguments



Any transaction that occurs in between January 1 and July 26 returns a Boolean result of TRUE. Transactions dates before 2021 and beyond July 26, 2021 return FALSE.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, previous_year_to_date, that determines which transactions took place a full year before the year segment ending on July 26, 2021.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  inyeartodate(date, '07/26/2021', -1) as previous_year_to_date
;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

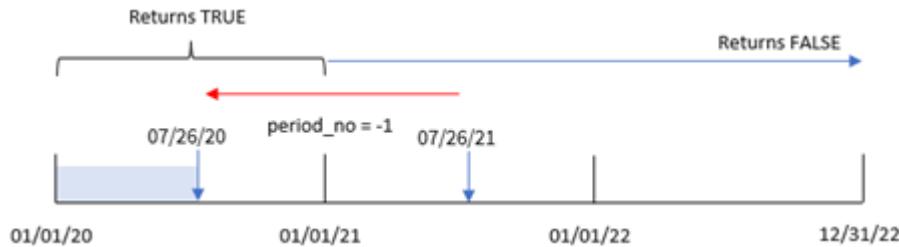
- date
- previous_year_to_date

Results table

date	previous_year_to_date
01/13/2020	-1
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
06/14/2020	-1
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

A period_no value of -1 indicates that the inyeartodate () function compares the input quarter segment to the preceding year. With an input date of July 26, 2021, the segment from January 1, 2021 to July 26, 2021 was initially identified as the year-to-date. The period_no then offsets this segment by a full year earlier, causing the date boundaries to become January 1 to July 26, 2020.

Diagram of inyeartodate function, period_no example



Therefore, any transaction that occurs between January 1 and July 26, 2020 will return a Boolean result of TRUE.

Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `in_year_to_date`, that determines which transactions took place in the same year up to July 26, 2021.

In this example, we set March as the first month of the fiscal year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    inyeartodate(date, '07/26/2021', 0, 3) as in_year_to_date
  ;
  Load
  *
  Inline
  [
  id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
```

```
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '07/27/2021', 25.12  
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- in_year_to_date

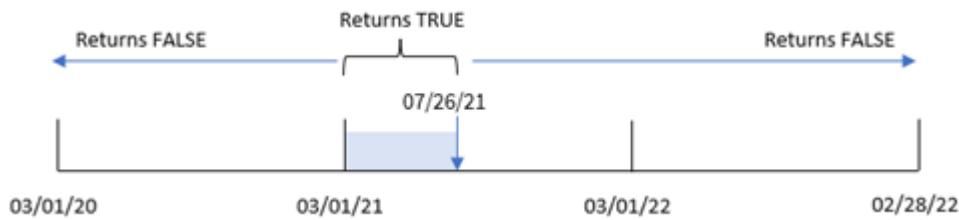
Results table

date	in_year_to_date
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0

date	in_year_to_date
07/18/2022	0
11/14/2022	0
12/12/2022	0

By using 3 as the `first_month_of_year` argument in the `inyearstodate()` function, the function begins the year on March 1. The `base_date` of July 26, 2021 then sets the end date for that year segment.

Diagram of inyearstodate function, first_month_of_year example



Therefore, any transaction that occurs between March 1 and July 26, 2021 will return a Boolean result of TRUE, while transactions with dates outside these boundaries will return a value of FALSE.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that determines which transactions took place in the same year up to July 26, 2021 is created as a measure in a chart object in the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
```

```
8194, '08/07/2020', 40.39  
8195, '09/05/2020', 87.21  
8196, '01/22/2021', 95.93  
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '07/27/2021', 25.12  
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:date.

Create the following measure:

```
=inyeartodate(date, '07/26/2021', 0)
```

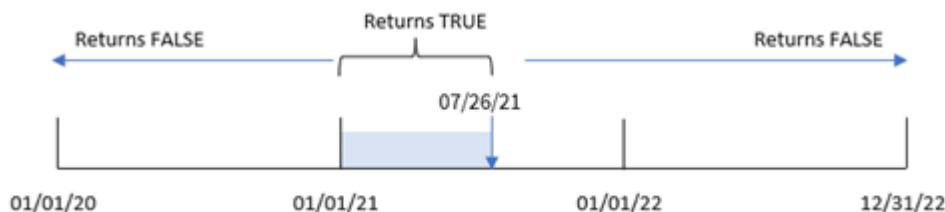
Results table

date	=inyeartodate(date,'07/26/2021', 0)
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1

date	=inyeartodate(date,'07/26/2021', 0)
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

The `in_year_to_date` measure is created in the chart object by using the `inyeartodate()` function. The first argument provided identifies which field is being evaluated. The second argument is a hard-coded date for July 26, 2021, which is the `base_date` that identifies the end boundary of the comparator year segment. A `period_no` of 0 is the final argument, meaning that the function is not comparing years preceding or following the segmented year.

Diagram of inyeartodate function, chart object example



Any transaction that occurs between January 1 and July 26, 2021 returns a Boolean result of `TRUE`. Transaction dates before 2021 and after July 26, 2021 return `FALSE`.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Products`.
- Information concerning product ID, product type, manufacture date, and cost price.

The end user would like a chart object that displays, by product type, the cost of the products manufactured in 2021 up to July 26.

Load script

```
Products:
Load
*
```

```
InLine
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'01/13/2020',37.23
8189,product B,'02/26/2020',17.17
8190,product B,'03/27/2020',88.27
8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:product_type.

Create a measure that calculates the sum of each product that was manufactured in 2021 before July 27:

```
=sum(if(inyeartodate(manufacture_date,makedate(2021,07,26),0),cost_price,0))
```

Set the measure's **Number formatting** to **Money**.

Results table

product_type	<code>=sum(if(inyeartodate(manufacture_date,makedate(2021,07,26),0),cost_price,0))</code>
product A	\$95.93
product B	\$128.66
product C	\$61.89
product D	\$146.09

The `inyeartodate()` function returns a Boolean value when evaluating the manufacturing dates of each of the products. For any product manufactured in 2021 before July 27, the `inyeartodate()` function returns a Boolean value of `TRUE` and sums the `cost_price`.

Product D is the only product that was also manufactured after July 26th in 2021. The entry with `product_ID` 8203 was manufactured on December 27 and cost \$25.12. Therefore, this cost was not included in the total for Product D in the chart object.

lastworkdate

The **lastworkdate** function returns the earliest ending date to achieve **no_of_workdays** (Monday-Friday) if starting at **start_date** taking into account any optionally listed **holiday**. **start_date** and **holiday** should be valid dates or timestamps.

Syntax:

```
lastworkdate(start_date, no_of_workdays {, holiday})
```

Return data type: integer

A calendar that shows how the `Lastworkdate()` function is used

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10 start_date	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26 end_date	27
28	29	30	31			

Limitations

There is no method to modify the `Lastworkdate()` function for regions or scenarios that involve anything other than a work week that begins on Monday and ends on Friday.

The **holiday** parameter must be a string constant. It does not accept an expression.

When to use it

The `Lastworkdate()` function is commonly used as part of an expression when the user would like to calculate the proposed end date of a project or assignment, based on when the project begins and the holidays that will occur in that period.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Arguments

Argument	Description
<code>start_date</code>	The start date to evaluate.
<code>no_of_workdays</code>	The number of working days to achieve.
<code>holiday</code>	Holiday periods to exclude from working days. A holiday is stated as a string constant date. You can specify multiple holiday dates, separated by commas. Example: '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'

Example 1 - Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing project IDs, project start dates, and the estimated effort, in days, required for the projects. The dataset is loaded into a table called ‘Projects’.
- A preceding load which contains the `Lastworkdate()` function which is set as the field ‘end_date’ and identifies when each project is scheduled to end.

Load script

```
SET DateFormat='MM/DD/YYYY';

Projects:
  Load
    *,
    LastWorkDate(start_date,effort) as end_date
```

```
;  
Load  
id,  
start_date,  
effort  
Inline  
[  
id,start_date,effort  
1,01/01/2022,14  
2,02/10/2022,17  
3,05/17/2022,5  
4,06/01/2022,12  
5,08/10/2022,26  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- effort
- end_date

Results table

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

Because there are no scheduled holidays, the function adds the defined number of working days, Monday to Friday, to the start date to find the earliest possible end date.

The following calendar shows the start and end date for project 3, with the working days highlighted in green.

A calendar that shows the start and end date of project 3

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19	20	21
22	23 End Date	24	25	26	27	28
29	30	31				

Example 2 - Single holiday

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing project IDs, project start dates, and the estimated effort, in days, required for the projects. The dataset is loaded into a table called ‘Projects’.
- A preceding load which contains the `lastworkdate()` function which is set as the field ‘end_date’ and identifies when each project is scheduled to end.

However, there is one holiday scheduled on May 18, 2022. The `lastworkdate()` function in the preceding load includes the holiday in its third argument to identify when each project is scheduled to end.

Load script

```
SET DateFormat='MM/DD/YYYY';

Projects:
  Load
    *,
    LastWorkDate(start_date,effort, '05/18/2022') as end_date
  ;
  Load
    id,
    start_date,
    effort
  Inline
  [
    id,start_date,effort
    1,01/01/2022,14
    2,02/10/2022,17
    3,05/17/2022,5
    4,06/01/2022,12
    5,08/10/2022,26
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `start_date`
- `effort`
- `end_date`

Results table

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/24/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

The single scheduled holiday is entered as the third argument in the `Lastworkdate()` function. As a result, the end date for project 3 is shifted one day later because the holiday takes place on one of the working days before the end date.

The following calendar shows the start and end date for project 3 and shows that the holiday changes the end date of the project by one day.

A calendar that shows the start and end date of project 3 with a holiday on May 18

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

Example 3 - Multiple holidays

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing project IDs, project start dates, and the estimated effort, in days, required for the projects. The dataset is loaded into a table called ‘Projects’.
- A preceding load which contains the `lastworkdate()` function which is set as the field ‘end_date’ and identifies when each project is scheduled to end.

However, there are three holidays scheduled for May 19, 20, 21, and 22. The `lastworkdate()` function in the preceding load includes each of the holidays in its third argument to identify when each project is scheduled to end.

Load script

```
SET DateFormat='MM/DD/YYYY';

Projects:
  Load
    *,
    LastWorkDate(start_date,effort, '05/19/2022','05/20/2022','05/21/2022','05/22/2022') as
end_date
  ;
  Load
  id,
  start_date,
  effort
  Inline
  [
  id,start_date,effort
  1,01/01/2022,14
  2,02/10/2022,17
  3,05/17/2022,5
  4,06/01/2022,12
  5,08/10/2022,26
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `start_date`
- `effort`
- `end_date`

Results table

id	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/25/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

The four holidays are entered as a list of arguments in the `Lastworkdate()` function after the start date and number of working days.

The following calendar shows the start and end date for project 3 and shows that the holidays change the end date of the project by three days.

A calendar that shows the start and end date of project 3 with holidays from May 19 to 22

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19 Holiday	20 Holiday	21 Holiday
22 Holiday	23	24	25 End Date	26	27	28
29	30	31				

Example 4 - Single holiday (chart)

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the app. The `end_date` field is calculated as a measure in a chart.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Projects:

```
Load
id,
start_date,
effort
Inline
[
```

```
id,start_date,effort  
1,01/01/2022,14  
2,02/10/2022,17  
3,05/17/2022,5  
4,06/01/2022,12  
5,08/10/2022,26  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- effort

To calculate the end_date, create the following measure:

- =LastworkDate(start_date,effort,'05/18/2022')

Results table

id	start_date	effort	=LastWorkDate(start_date,effort,'05/18/2022')
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

The single scheduled holiday is entered as a measure in the chart. As a result, the end date for project 3 is shifted one day later because the holiday takes place on one of the working days before the end date.

The following calendar shows the start and end date for project 3 and shows that the holiday changes the end date of the project by one day.

A calendar that shows the start and end date of project 3 with a holiday on May 18

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

localtime

This function returns a timestamp of the current time for a specified time zone.

Syntax:

```
LocalTime([timezone [, ignoreDST ]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
timezone	The timezone is specified as a string containing any of the geographical places listed under Time Zone in the Windows Control Panel for Date and Time or as a string in the form 'GMT+hh:mm'. If no time zone is specified the local time will be returned.
ignoreDST	If ignoreDST is -1 (True) daylight savings time will be ignored.

Examples and results:

The examples below are based on the function being called on 2014-10-22 12:54:47 local time, with the local time zone being GMT+01:00.

Scripting examples

Example	Result
<code>localtime()</code>	Returns the local time 2014-10-22 12:54:47.
<code>localtime('London')</code>	Returns the local time in London, 2014-10-22 11:54:47.
<code>localtime('GMT+02:00')</code>	Returns the local time in the timezone of GMT+02:00, 2014-10-22 13:54:47.
<code>localtime('Paris', '-1')</code>	Returns the local time in Paris with daylight savings time ignored, 2014-10-22 11:54:47.

lunarweekend

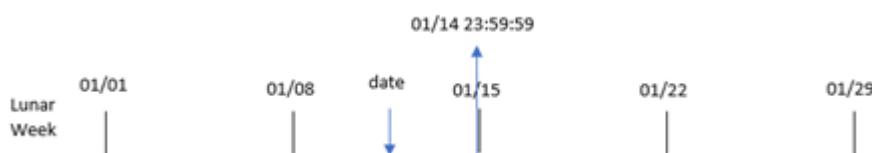
This function returns a value corresponding to a timestamp of the last millisecond of the last day of the lunar week containing **date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

Syntax:

```
LunarweekEnd(date[, period_no[, first_week_day]])
```

Return data type: dual

Example diagram of `lunarweekend()` function



The `lunarweekend()` function determines which lunar week the date falls into. It then returns a timestamp, in date format, for the last millisecond of that week.

Arguments

Argument	Description
date	The date or timestamp to evaluate.

Argument	Description
period_no	period_no is an integer or expression resolving to an integer, where the value 0 indicates the lunar week which contains date . Negative values in period_no indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
first_week_day	An offset that may be greater than or less than zero. This changes the beginning of the year by the specified number of days and/or fractions of a day.

When to use it

The `lunarweekend()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the week that has not yet occurred. Unlike the `weekend()` function, the final lunar week of each calendar year will end on December 31. For example, the `lunarweekend()` function can be used to calculate interest not yet incurred during the week.

Function examples

Example	Result
<code>lunarweekend('01/12/2013')</code>	Returns 01/14/2013 23:59:59.
<code>lunarweekend('01/12/2013', -1)</code>	Returns 01/07/2013 23:59:59.
<code>lunarweekend('01/12/2013', 0, 1)</code>	Returns 01/15/2013 23:59:59.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `end_of_week`, that returns a timestamp for the end of the lunar week in which the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekend(date) as end_of_week,
  timestamp(lunarweekend(date)) as end_of_week_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `end_of_week`
- `end_of_week_timestamp`

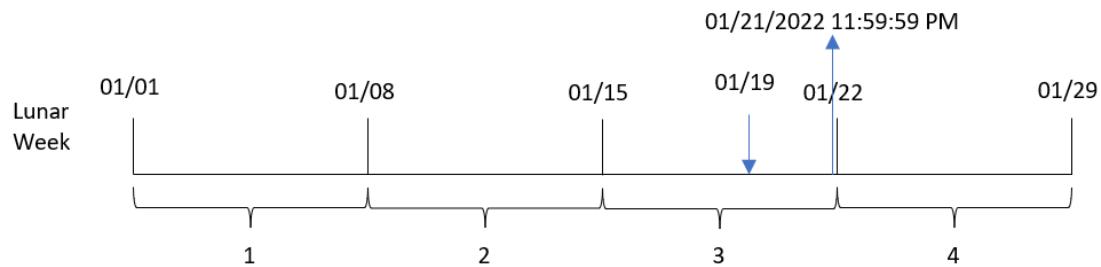
Results table

date	end_of_week	end_of_week_timestamp
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

The `end_of_week` field is created in the preceding load statement by using the `lunarweekend()` function, and passing the `date` field as the function's argument.

The `lunarweekend()` function identifies which lunar week the date value falls into, returning a timestamp for the last millisecond of that week.

Diagram of `lunarweekend()` function, example with no additional arguments



Transaction 8189 took place on January 19. The `lunarweekend()` function identifies that the lunar week begins on January 15. Therefore, the `end_of_week` value for that transaction returns the last millisecond of the lunar week, which is January 21 at 11:59:59 PM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_lunar_week_end`, that returns the timestamp for the end of the lunar week before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekend(date,-1) as previous_lunar_week_end,
  timestamp(lunarweekend(date,-1)) as previous_lunar_week_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
```

```
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_lunar_week_end
- previous_lunar_week_end_timestamp

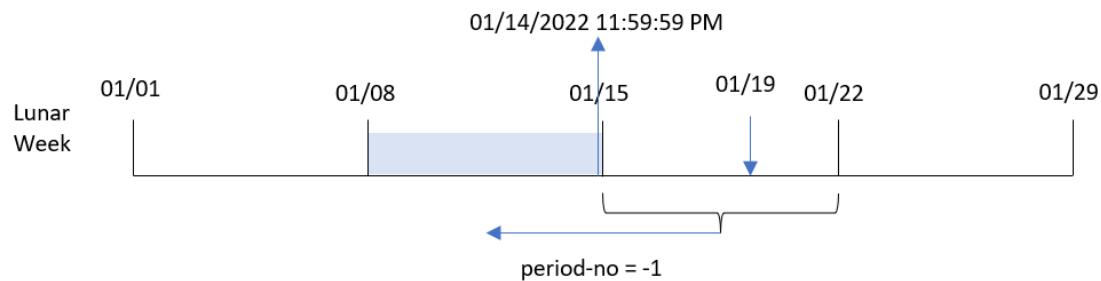
Results table

date	previous_lunar_week_end	previous_lunar_week_end_timestamp
1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
1/19/2022	01/14/2022	1/14/2022 11:59:59 PM
2/5/2022	02/04/2022	2/4/2022 11:59:59 PM
2/28/2022	02/25/2022	2/25/2022 11:59:59 PM
3/16/2022	03/11/2022	3/18/2022 11:59:59 PM
4/1/2022	03/25/2022	3/25/2022 11:59:59 PM
5/7/2022	05/06/2022	5/6/2022 11:59:59 PM
5/16/2022	05/13/2022	5/13/2022 11:59:59 PM
6/15/2022	06/10/2022	6/10/2022 11:59:59 PM
6/26/2022	06/24/2022	6/24/2022 11:59:59 PM
7/9/2022	07/08/2022	7/8/2022 11:59:59 PM
7/22/2022	07/15/2022	7/15/2022 11:59:59 PM
7/23/2022	07/22/2022	7/22/2022 11:59:59 PM
7/27/2022	07/22/2022	7/22/2022 11:59:59 PM
8/2/2022	07/29/2022	7/29/2022 11:59:59 PM

date	previous_lunar_week_end	previous_lunar_week_end_timestamp
8/8/2022	08/05/2022	8/5/2022 11:59:59 PM
8/19/2022	08/12/2022	8/12/2022 11:59:59 PM
9/26/2022	09/23/2022	9/23/2022 11:59:59 PM
10/14/2022	10/07/2022	10/7/2022 11:59:59 PM
10/29/2022	10/28/2022	10/28/2022 11:59:59 PM

In this instance, because a period_no of -1 was used as the offset argument in the `lunarweekend()` function, the function first identifies the lunar week in which the transactions took place. It then shifts one week prior and identifies the final millisecond of that lunar week.

Diagram of `lunarweekend()` function, period_no example



Transaction 8189 took place on January 19. The `lunarweekend()` function identifies that the lunar week begins on January 15. Therefore, the previous lunar week began on the January 8 and ended on January 14 at 11:59:59 PM; this is the value that is returned for the `previous_lunar_week_end` field.

Example 3 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. In this example, we set lunar weeks to begin on January 5.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekend(date,0,4) as end_of_week,
  timestamp(lunarweekend(date,0,4)) as end_of_week_timestamp
```

```
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- end_of_week
- end_of_week_timestamp

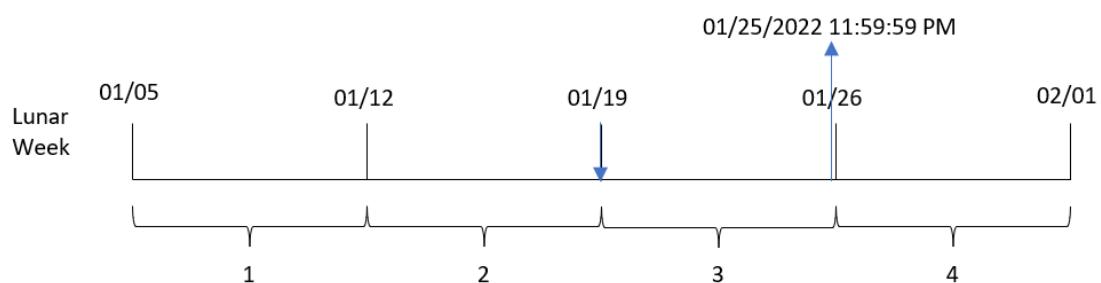
Results table

date	end_of_week	end_of_week_timestamp
1/7/2022	01/11/2022	1/11/2022 11:59:59 PM
1/19/2022	01/25/2022	1/25/2022 11:59:59 PM
2/5/2022	02/08/2022	2/8/2022 11:59:59 PM
2/28/2022	03/01/2022	3/1/2022 11:59:59 PM
3/16/2022	03/22/2022	3/22/2022 11:59:59 PM
4/1/2022	04/05/2022	4/5/2022 11:59:59 PM
5/7/2022	05/10/2022	5/10/2022 11:59:59 PM
5/16/2022	05/17/2022	5/17/2022 11:59:59 PM

date	end_of_week	end_of_week_timestamp
6/15/2022	06/21/2022	6/21/2022 11:59:59 PM
6/26/2022	06/28/2022	6/28/2022 11:59:59 PM
7/9/2022	07/12/2022	7/12/2022 11:59:59 PM
7/22/2022	07/26/2022	7/26/2022 11:59:59 PM
7/23/2022	07/26/2022	7/26/2022 11:59:59 PM
7/27/2022	08/02/2022	8/2/2022 11:59:59 PM
8/2/2022	08/02/2022	8/2/2022 11:59:59 PM
8/8/2022	08/09/2022	8/9/2022 11:59:59 PM
8/19/2022	08/23/2022	8/23/2022 11:59:59 PM
9/26/2022	09/27/2022	9/27/2022 11:59:59 PM
10/14/2022	10/18/2022	10/18/2022 11:59:59 PM
10/29/2022	11/01/2022	11/1/2022 11:59:59 PM

In this instance, because the `first_week_date` argument of 4 is used in the `lunarweekend()` function, it offsets the start of the year from January 1 to January 5.

Diagram of `lunarweekend()` function, `first_week_day` example



Transaction 8189 took place on January 19. Due to lunar weeks beginning on January 5, the `lunarweekend()` function identifies that the lunar week containing January 19 also begins on January 19. Therefore, the end of that lunar week occurs on January 25 at 11:59:59 PM; this is the value returned for the `end_of_week` field.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the end of the lunar week in which the transactions took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Add the following measures:

```
=lunarweekend(date)
=timestamp(lunarweekend(date))
```

Results table

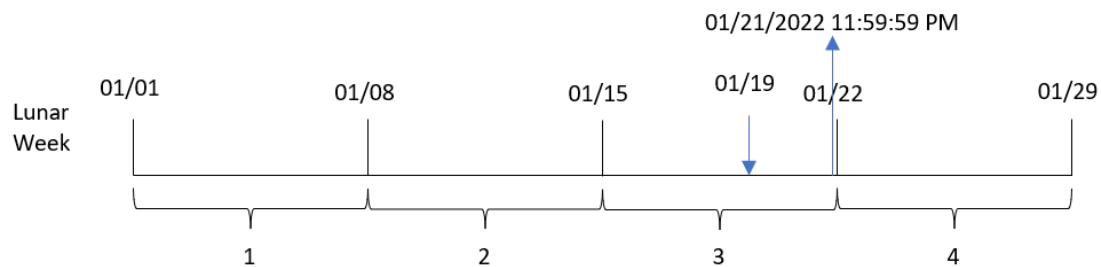
date	=lunarweekend(date)	=timestamp(lunarweekend(date))
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM

date	=lunarweekend(date)	=timestamp(lunarweekend(date))
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

The `end_of_week` measure is created in the chart object by using the `lunarweekend()` function, and passing the date field as the function's argument.

The `lunarweekend()` function identifies which lunar week the date value falls into, returning a timestamp for the last millisecond of that week.

Diagram of `lunarweekend()` function, chart object example



Transaction 8189 took place on January 19. The `lunarweekend()` function identifies that the lunar week begins on January 15. Therefore, the `end_of_week` value for that transaction returns the last millisecond of the lunar week, which is January 21 at 11:59:59 PM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called Employee_Expenses.
- The employee IDs, employee name and the average daily expense claims of each employee.

The end user would like a chart object that displays, by employee ID and employee name, the estimated expense claims still to be incurred for the remainder of the lunar week.

Load script

```
Employee_Expenses:  
Load  
*  
Inline  
[  
employee_id,employee_name,avg_daily_claim  
182,Mark, $15  
183,Deryck, $12.5  
184,Dexter, $12.5  
185,Sydney,$27  
186,Agatha,$18  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Add the following fields as dimensions:
 - employee_id
 - employee_name
3. Next, create the following measure to calculate the accumulated interest:
$$=(\text{lunarweekend}(\text{today}(1))-\text{today}(1))*\text{avg_daily_claim}$$
4. Set the measure's **Number formatting** to **Money**.

Results table

employee_id	employee_name	=lunarweekend(today(1))-today(1)*avg_daily_claim
182	Mark	\$75.00

employee_id	employee_name	=lunarweekend(today(1))-today(1)*avg_daily_claim
183	Deryck	\$62.50
184	Dexter	\$62.50
185	Sydney	\$135.00
186	Agatha	\$90.00

The `lunarkweekend()` function, by using today's date as its only argument, returns the end date of the current lunar week. Then, by subtracting today's date from the lunar week end date, the expression returns the number of days that remain this week.

This value is then multiplied by the average daily expense claim by each employee to calculate the estimated value of claims each employee is expected to make in the remaining lunar week.

lunarweekname

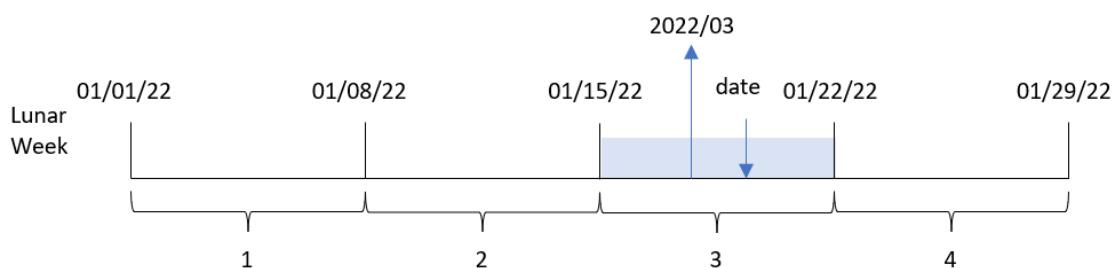
This function returns a display value showing the year and lunar week number corresponding to a timestamp of the first millisecond of the first day of the lunar week containing `date`. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

Syntax:

```
LunarWeekName(date [, period_no[, first_week_day]])
```

Return data type: dual

Example diagram of `Lunarweekname()` function



The `Lunarweekname()` function determines which lunar week the date falls into, beginning a week count from January 1. It then returns a value comprised of year/weekcount.

Arguments

Argument	Description
date	The date or timestamp to evaluate.

Argument	Description
period_no	period_no is an integer or expression resolving to an integer, where the value 0 indicates the lunar week which contains date . Negative values in period_no indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
first_week_day	An offset that may be greater than or less than zero. This changes the beginning of the year by the specified number of days and/or fractions of a day.

When to use it

The `lunarweekname()` function is useful when you would like to compare aggregations by lunar weeks. For example, the function could be used to determine the total sales of products by lunar week. Lunar weeks are useful when you would like to ensure that all values contained in the first week of the year contain only values from January 1 at the earliest.

These dimensions can be created in the load script by using the function to create a field in a Master Calendar table. The function can also be used directly in a chart as a calculated dimension.

Function examples

Example	Result
<code>lunarweekname('01/12/2013')</code>	Returns 2006/02.
<code>lunarweekname('01/12/2013', -1)</code>	Returns 2006/01.
<code>lunarweekname('01/12/2013', 0, 1)</code>	Returns 2006/02.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – date with no additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `Lunar_week_name`, that returns the year and week number for the lunar week in which the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekname(date) as Lunar_week_name
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `Lunar_week_name`

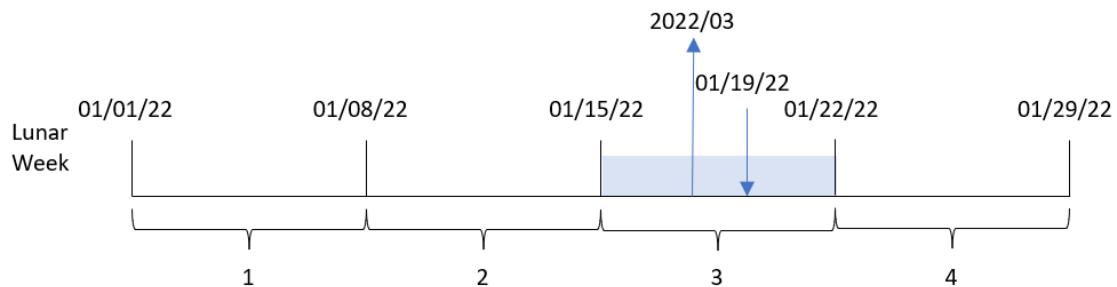
Results table

date	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

The `lunar_week_name` field is created in the preceding load statement by using the `lunarweekname()` function, and passing the `date` field as the function's argument.

The `lunarweekname()` function identifies which lunar week the date value falls into, returning the year and week number of that date.

Diagram of `lunarweekname()` function, example with no additional arguments



Transaction 8189 took place on January 19. The `lunarweekname()` function identifies that this date falls into the lunar week beginning on January 15; this is the third lunar week of the year. Therefore, the `lunar_week_name` value returned for that transaction is 2022/03.

Example 2 – date with period_no argument

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_lunar_week_name`, that returns the year and week number for the lunar week prior to when the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekname(date,-1) as previous_lunar_week_name
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
```

```
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_lunar_week_name

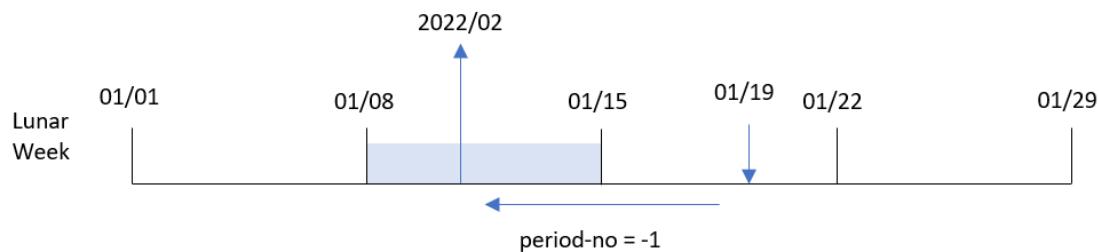
Results table

date	previous_lunar_week_name
1/7/2022	2021/52
1/19/2022	2022/02
2/5/2022	2022/05
2/28/2022	2022/08
3/16/2022	2022/10
4/1/2022	2022/12
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/23
6/26/2022	2022/25
7/9/2022	2022/27
7/22/2022	2022/28
7/23/2022	2022/29
7/27/2022	2022/29
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/32

date	previous_lunar_week_name
9/26/2022	2022/38
10/14/2022	2022/40
10/29/2022	2022/43

In this instance, because a period_no of -1 was used as the offset argument in the `lunarweekname()` function, the function first identifies the lunar week in which the transactions took place. It then returns the year and number of one week prior.

Diagram of `lunarweekname()` function, period_no example



Transaction 8189 took place on January 19. The `lunarweekname()` function identifies that this transaction took place in the third lunar week of the year, so it then returns the year and value for one week prior, 2022/02, for the `previous_lunar_week_name` field.

Example 3 – date with `first_week_day` argument

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. In this example, we set lunar weeks to begin on January 5.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    lunarweekname(date,0,4) as lunar_week_name
  ;
  Load
  *
  Inline
[
```

```
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

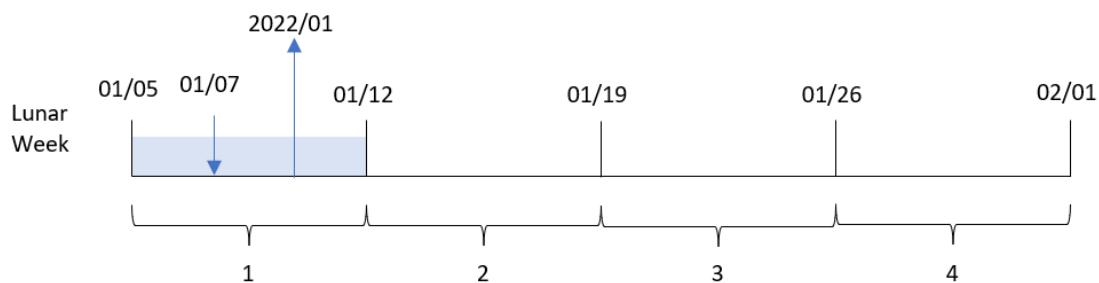
- date
- lunar_week_name

Results table

date	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/05
2/28/2022	2022/08
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/24
6/26/2022	2022/25
7/9/2022	2022/27

date	lunar_week_name
7/22/2022	2022/29
7/23/2022	2022/29
7/27/2022	2022/30
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/33
9/26/2022	2022/38
10/14/2022	2022/41
10/29/2022	2022/43

Diagram of `lunarweekname()` function, `first_week_day` example



In this instance, because the `first_week_date` argument of 4 is used in the `lunarweekname()` function, it offsets the start of lunar weeks from January 1 to January 5.

Transaction 8188 took place on January 7. Due to lunar weeks beginning on January 5, the `lunarweekname()` function identifies that the lunar week containing January 7 is the first lunar week of the year. Therefore, the returned `lunar_week_name` value for that transaction is 2022/01.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns the lunar week number and year in which the transactions took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate the start date of the lunar week in which a transaction takes place, create the following measure:

```
=lunarweekname(date)
```

Results table

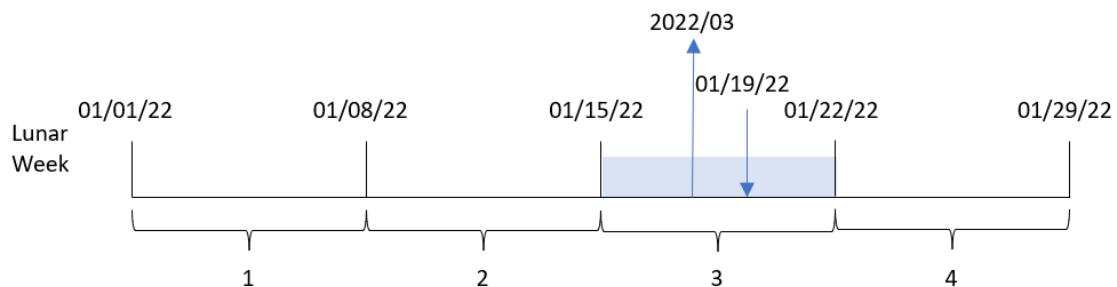
date	=lunarweekname(date)
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19

date	=lunarweekname(date)
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

The `lunar_week_name` measure is created in the chart object by using the `lunarweekname()` function and passing the date field as the function's argument.

The `lunarweekname()` function identifies which lunar week the date value falls into, returning the year and week number of that date.

Diagram of `Lunarweekname()` function, chart object example



Transaction 8189 took place on January 19. The `lunarweekname()` function identifies that this date falls into the lunar week beginning on January 15; this is the third lunar week of the year. Therefore, the `lunar_week_name` value for that transaction is 2022/03.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

The end user would like a chart object that presents the total sales by week for the current year. Week 1, with a length of seven days, should begin on January 1. This could be achieved even when this dimension is not available in the data model by using the `1unarweekname()` function as a calculated dimension in the chart.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create a calculated dimension using the following expression:
`=lunarweekname(date)`
3. Calculate total sales using the following aggregation measure:
`=sum(amount)`
4. Set the measure's **Number formatting** to **Money**.

Results table

<code>=lunarweekname(date)</code>	<code>=sum(amount)</code>
2022/01	\$17.17
2022/03	\$37.23
2022/06	\$57.42
2022/09	\$88.27
2022/11	\$53.80
2022/13	\$82.06
2022/19	\$40.39
2022/20	\$87.21
2022/24	\$95.93
2022/26	\$45.89
2022/28	\$36.23
2022/29	\$25.66
2022/30	\$152.75
2022/31	\$76.11
2022/32	\$25.12
2022/33	\$46.23
2022/39	\$84.21
2022/41	\$96.24
2022/44	\$67.67

lunarweekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the lunar week containing **date**. Lunar weeks in Qlik Sense are defined by counting January 1 as the first day of the week and, apart from the final week of the year, will contain exactly seven days.

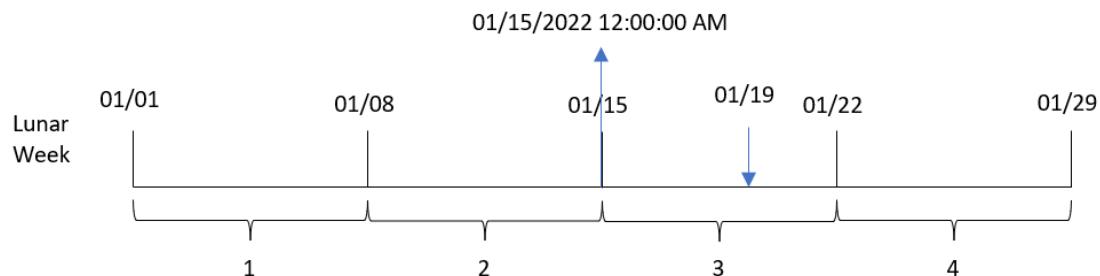
Syntax:

```
LunarweekStart(date[, period_no[, first_week_day]])
```

Return data type: dual

The `lunarweekstart()` function determines which lunar week the date falls into. It then returns a timestamp, in date format, for the first millisecond of that week.

Example diagram of `lunarweekstart()` function



Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer or expression resolving to an integer, where the value 0 indicates the lunar week which contains date . Negative values in period_no indicate preceding lunar weeks and positive values indicate succeeding lunar weeks.
first_week_day	An offset that may be greater than or less than zero. This changes the beginning of the year by the specified number of days and/or fractions of a day.

When to use it

The `lunarweekstart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the week that has elapsed thus far. Unlike the `weekstart()` function, at the start of each new calendar year, week's begin on January 1 and each subsequent week begins seven days later. The `lunarweekstart()` function is not affected by the `Firstweekday` system variable.

For example, the `lunarweekstart()` can be used to calculate the interest that has been accumulated in a week to date.

Function examples

Example	Result
<code>lunarweekstart('01/12/2013')</code>	Returns 01/08/2013.
<code>lunarweekstart('01/12/2013', -1)</code>	Returns 01/01/2013.
<code>lunarweekstart('01/12/2013', 0, 1)</code>	Returns 01/09/2013, because setting first_week_day to 1 means the beginning of the year is changed to 01/02/2013.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `start_of_week`, that returns a timestamp for the start of the lunar week in which the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    lunarweekstart(date) as start_of_week,
    timestamp(lunarweekstart(date)) as start_of_week_timestamp
  ;
```

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- start_of_week
- start_of_week_timestamp

Results table

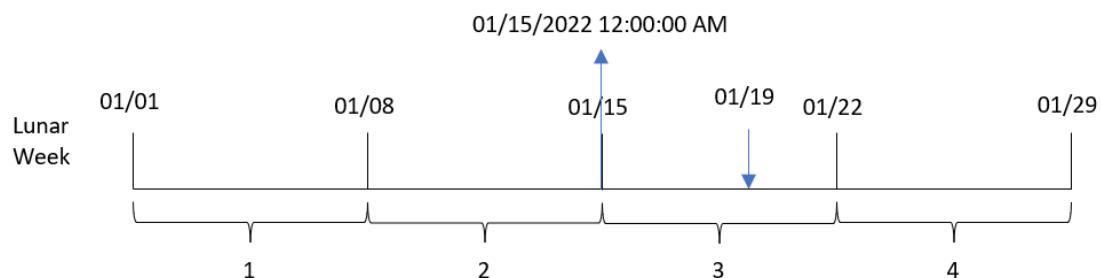
date	start_of_week	start_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM

date	start_of_week	start_of_week_timestamp
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

The `start_of_week` field is created in the preceding load statement by using the `lunarweekstart()` function and passing the `date` field as the function's argument.

The `lunarweekstart()` function identifies the lunar week into which the date falls, returning a timestamp for the first millisecond of that week.

Diagram of `lunarweekstart()` function, example with no additional arguments



Transaction 8189 took place on January 19. The `lunarweekstart()` function identifies that the lunar week begins on January 15. Therefore, the `start_of_week` value for that transaction returns the first millisecond of that day, which is January 15 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_lunar_week_start`, that returns the timestamp for the start of the lunar week before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    lunarweekstart(date,-1) as previous_lunar_week_start,
    timestamp(lunarweekstart(date,-1)) as previous_lunar_week_start_timestamp
  ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

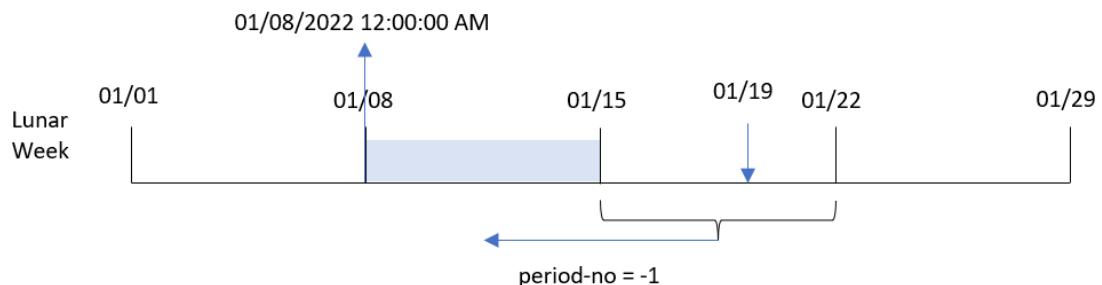
Results table

date	previous_lunar_week_start	previous_lunar_week_start_timestamp
1/7/2022	12/24/2021	12/24/2021 12:00:00 AM
1/19/2022	01/08/2022	1/8/2022 12:00:00 AM
2/5/2022	01/29/2022	1/29/2022 12:00:00 AM

date	previous_lunar_week_start	previous_lunar_week_start_timestamp
2/28/2022	02/19/2022	2/19/2022 12:00:00 AM
3/16/2022	03/05/2022	3/5/2022 12:00:00 AM
4/1/2022	03/19/2022	3/19/2022 12:00:00 AM
5/7/2022	04/30/2022	4/30/2022 12:00:00 AM
5/16/2022	05/07/2022	5/7/2022 12:00:00 AM
6/15/2022	06/04/2022	6/4/2022 12:00:00 AM
6/26/2022	06/18/2022	6/18/2022 12:00:00 AM
7/9/2022	07/02/2022	7/2/2022 12:00:00 AM
7/22/2022	07/09/2022	7/9/2022 12:00:00 AM
7/23/2022	07/16/2022	7/16/2022 12:00:00 AM
7/27/2022	07/16/2022	7/16/2022 12:00:00 AM
8/2/2022	07/23/2022	7/23/2022 12:00:00 AM
8/8/2022	07/30/2022	7/30/2022 12:00:00 AM
8/19/2022	08/06/2022	8/6/2022 12:00:00 AM
9/26/2022	09/17/2022	9/17/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/22/2022	10/22/2022 12:00:00 AM

In this instance, because a period_no of -1 was used as the offset argument in the `lunarweekstart()` function, the function first identifies the lunar week that the transactions take place in. It then shifts one week prior and identifies the first millisecond of that lunar week.

Diagram of `lunarweekstart()` function, period_no example



Transaction 8189 took place on January 19. The `lunarweekstart()` function identifies that the lunar week begins on January 15. Therefore, the previous lunar week began on January 8 at 12:00:00 AM; this is the value returned for the `previous_lunar_week_start` field.

Example 3 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. In this example, we set lunar weeks to begin on January 5.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  lunarweekstart(date,0,4) as start_of_week,
  timestamp(lunarweekstart(date,0,4)) as start_of_week_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

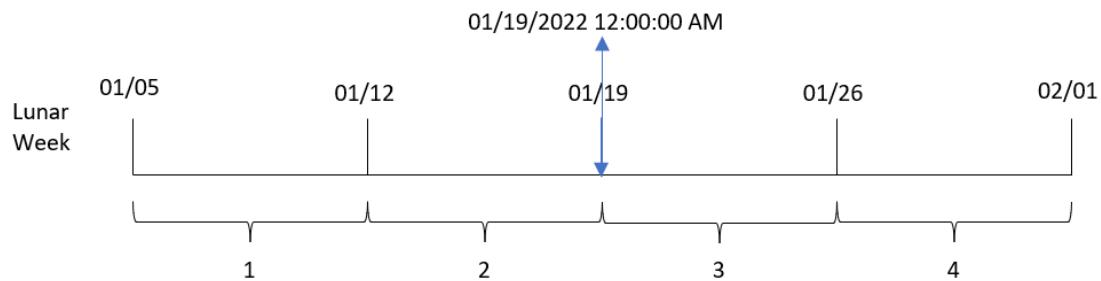
- `date`
- `start_of_week`
- `start_of_week_timestamp`

Results table

date	start_of_week	start_of_week_timestamp
1/7/2022	01/05/2022	1/5/2022 12:00:00 AM
1/19/2022	01/19/2022	1/19/2022 12:00:00 AM
2/5/2022	02/02/2022	2/2/2022 12:00:00 AM
2/28/2022	02/23/2022	2/23/2022 12:00:00 AM
3/16/2022	03/16/2022	3/16/2022 12:00:00 AM
4/1/2022	03/30/2022	3/30/2022 12:00:00 AM
5/7/2022	05/04/2022	5/4/2022 12:00:00 AM
5/16/2022	05/11/2022	5/11/2022 12:00:00 AM
6/15/2022	06/15/2022	6/15/2022 12:00:00 AM
6/26/2022	06/22/2022	6/22/2022 12:00:00 AM
7/9/2022	07/06/2022	7/6/2022 12:00:00 AM
7/22/2022	07/20/2022	7/20/2022 12:00:00 AM
7/23/2022	07/20/2022	7/20/2022 12:00:00 AM
7/27/2022	07/27/2022	7/27/2022 12:00:00 AM
8/2/2022	07/27/2022	7/27/2022 12:00:00 AM
8/8/2022	08/03/2022	8/3/2022 12:00:00 AM
8/19/2022	08/17/2022	8/17/2022 12:00:00 AM
9/26/2022	09/21/2022	9/21/2022 12:00:00 AM
10/14/2022	10/12/2022	10/12/2022 12:00:00 AM
10/29/2022	10/26/2022	10/26/2022 12:00:00 AM

In this instance, because the `first_week_date` argument of 4 is used in the `lunarweekstart()` function, it offsets the start of the year from January 1 to January 5.

Diagram of `lunarweekstart()` function, `first_week_day` example



Transaction 8189 took place on January 19. Due to lunar weeks beginning on January 5, the `lunarweekstart()` function identifies that the lunar week containing January 19 begins on January 19 at 12:00:00 AM as well. Therefore, that is the value returned for the `start_of_week` field.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the start of the lunar week in which the transactions took place is created as a measure in a chart object of the application.

Load script

```
Transactions:  
Load  
*  
Inline  
[  
id,date,amount  
8188,1/7/2022,17.17  
8189,1/19/2022,37.23  
8190,2/28/2022,88.27  
8191,2/5/2022,57.42  
8192,3/16/2022,53.80  
8193,4/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98
```

```
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Add the following measures:

```
=lunarweekstart(date)
=timestamp(lunarweekstart(date))
```

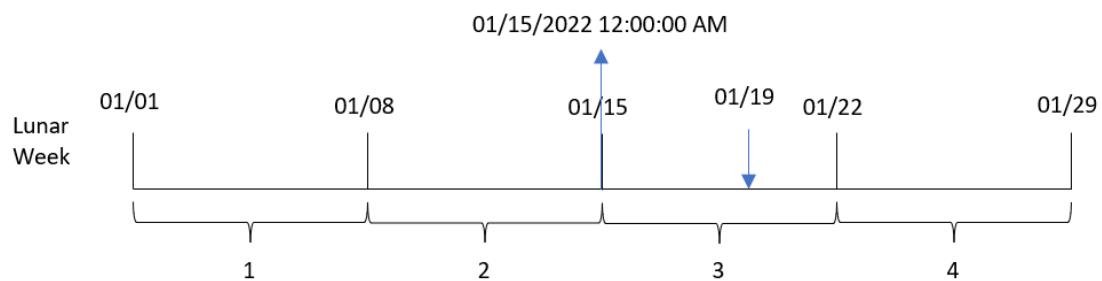
Results table

date	=lunarweekstart(date)	=timestamp(lunarweekstart(date))
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

The `start_of_week` measure is created in the chart object by using the `lunarweekstart()` function, and passing the date field as the function's argument.

The `lunarweekstart()` function identifies which lunar week the date value falls into, returning a timestamp for the last millisecond of that week.

Diagram of `lunarweekstart()` function, chart object example



Transaction 8189 took place on January 19. The `lunarweekstart()` function identifies that the lunar week begins on January 15. Therefore, the `start_of_week` value for that transaction is first millisecond of that day, which is the January 15 at 12:00:00 AM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of loan balances, which is loaded into a table called `Loans`.
- Data consisting of loan IDs, the balance at the beginning of the week, and the simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan ID, the current interest that has been accrued on each loan in the week to date.

Load script

```
Loans:  
Load  
*  
Inline  
[  
loan_id,start_balance,rate  
8188,$10000.00,0.024  
8189,$15000.00,0.057  
8190,$17500.00,0.024  
8191,$21000.00,0.034
```

```
8192,$90000.00,0.084
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Add the following fields as dimensions:
 - loan_id
 - start_balance
3. Next, create the following measure to calculate the accumulated interest:
 $=start_balance*(rate*(today(1)-lunarweekstart(today(1)))/365)$
4. Set the measure's **Number formatting** to **Money**.

Results table

loan_id	start_balance	$=start_balance*(rate*(today(1)-lunarweekstart(today(1)))/365)$
8188	\$10000.00	\$15.07
8189	\$15000.00	\$128.84
8190	\$17500.00	\$63.29
8191	\$21000.00	\$107.59
8192	\$90000.00	\$1139.18

The `lunarweekstart()` function, using today's date as its only argument, returns the start date of the current year. By subtracting that result from the current date, the expression returns the number of days that have elapsed so far this week.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate incurred for this period. The result is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this week.

makedate

This function returns a date calculated from the year **YYYY**, the month **MM** and the day **DD**.

Syntax:

```
MakeDate(YYYY [ , MM [ , DD ] ])
```

Return data type: dual

Arguments

Argument	Description
YYYY	The year as an integer.

Argument	Description
MM	The month as an integer. If no month is stated, 1 (January) is assumed.
DD	The day as an integer. If no day is stated, 1 (the 1st) is assumed.

When to use it

The `makedate()` function would commonly be used in the script for data generation to generate a calendar. This could also be used when the date field is not directly available as date, but needs some transformations to extract year, month and day components.

These examples use the date format MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>makedate(2012)</code>	Returns 01/01/2012.
<code>makedate(12)</code>	Returns 01/01/2012.
<code>makedate(2012,12)</code>	Returns 12/01/2012.
<code>makedate(2012,2,14)</code>	Returns 02/14/2012.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2018, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `transaction_date`, that returns a date in the format MM/DD/YYYY.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  makedate(transaction_year, transaction_month, transaction_day) as transaction_date
;
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_year`
- `transaction_month`
- `transaction_day`
- `transaction_date`

Results table

<code>transaction_year</code>	<code>transaction_month</code>	<code>transaction_day</code>	<code>transaction_date</code>
2018	08	30	08/30/2018
2018	09	07	09/07/2018
2018	09	16	09/16/2018
2018	09	22	09/22/2018
2018	09	23	09/23/2018

The `transaction_date` field is created in the preceding load statement by using the `makedate()` function and passing the year, month, day fields as function arguments.

The function then combines and converts these values into a date field, returning the results in the format of the `DateFormat` system variable.

Example 2 – Modified DateFormat

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `transaction_date`, in the format DD/MM/YYYY without modifying the `DateFormat` system variable.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
  *,
  date(makedate(transaction_year, transaction_month, transaction_day), 'DD/MM/YYYY') as
transaction_date
;
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_year`
- `transaction_month`
- `transaction_day`
- `transaction_date`

Results table

<code>transaction_year</code>	<code>transaction_month</code>	<code>transaction_day</code>	<code>transaction_date</code>
2018	08	30	30/08/2018

transaction_year	transaction_month	transaction_day	transaction_date
2018	09	07	07/09/2018
2018	09	16	16/09/2018
2018	09	22	22/09/2018
2018	09	23	23/09/2018

In this instance, the `makedate()` function is nested inside the `date()` function. The second argument of the `date()` function sets the format of the `makedate()` function results as the required DD/MM/YYYY.

Example 3 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2018, which is loaded into a table called `Transactions`.
- The transaction dates provided across two fields: `year` and `month`.

Create a chart object measure, `transaction_date`, that returns a date in the format MM/DD/YYYY.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_amount, transaction_quantity,
customer_id
3750, 2018, 08, 12423.56, 23, 2038593
3751, 2018, 09, 5356.31, 6, 203521
3752, 2018, 09, 15.75, 1, 5646471
3753, 2018, 09, 1251, 7, 3036491
3754, 2018, 09, 21484.21, 1356, 049681
3756, 2018, 09, -59.18, 2, 2038593
3757, 2018, 09, 3177.4, 21, 203521
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `year`
- `month`

To determine the `transaction_date`, create this measure:

```
=makedate(transaction_year ,transaction_month)
```

Results table

transaction_year	transaction_month	transaction_date
2018	08	08/01/2018
2018	09	09/01/2018

The `transaction_date` measure is created in the chart object by using the `makedate()` function, and passing the year and month fields as function arguments.

The function then combines these values, as well as the assumed day value of 01. These values are then converted into a date field, returning the results in the format of the `DateFormat` system variable.

Example 4 – Scenario

Load script and chart expression

Overview

Create a calendar dataset for the calendar year of 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Calendar:
  load
    *
  where year(date)=2022;
load
  date(recno()+makedate(2021,12,31)) as date
AutoGenerate 400;
```

Results

Results table

date
01/01/2022
01/02/2022
01/03/2022
01/04/2022
01/05/2022
01/06/2022

date
01/07/2022
01/08/2022
01/09/2022
01/10/2022
01/11/2022
01/12/2022
01/13/2022
01/14/2022
01/15/2022
01/16/2022
01/17/2022
01/18/2022
01/19/2022
01/20/2022
01/21/2022
01/22/2022
01/23/2022
01/24/2022
01/25/2022
+ 340 more rows

The `makedate()` function creates a date value for December 31, 2021. The `recno()` function provides the record number of the current record being loaded into the table, starting from 1. Therefore, the first record has the date January 1, 2022. Each successive `recno()` will then increment this date by 1. This expression is wrapped in a `date()` function to convert the value into a date. This process is repeated 400 times by the autogenerate function. Finally, by using a preceding load, a `where` condition can be used to only load dates from year 2022. This script generates a calendar containing every date in 2022.

maketime

This function returns a time calculated from the hour **hh**, the minute **mm**, and the second **ss**.

Syntax:

```
MakeTime(hh [ , mm [ , ss ] ])
```

Return data type: dual

Arguments

Argument	Description
hh	The hour as an integer.
mm	The minute as an integer. If no minute is stated, 00 is assumed.
ss	The second as an integer. If no second is stated, 00 is assumed.

When to use it

The `maketime()` function would commonly be used in the script for data generation to generate a time field. Sometimes, when the time field is derived from input text, this function could be used to construct the time using its components.

These examples use the time format `h:mm:ss`. The time format is specified in the `SET TimeFormat` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
<code>maketime(22)</code>	Returns 22:00:00.
<code>maketime(22, 17)</code>	Returns 22:17:00.
<code>maketime(22,17,52)</code>	Returns 22:17:52.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – maketime()

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions , which is loaded into a table called `Transactions`.
- Transaction times provided across three fields: `hours`, `minutes`, and `seconds`.
- The creation of a field, `transaction_time`, that returns the time in the format of the `TimeFormat` system variable.

Load script

```
SET TimeFormat='h:mm:ss TT';

Transactions:
  Load
    *,
    maketime(transaction_hour, transaction_minute, transaction_second) as transaction_time
  ;
  Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
3757, 9, 25, 23, 3177.4, 21, 203521
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_hour`
- `transaction_minute`
- `transaction_second`
- `transaction_time`

Results table

<code>transaction_hour</code>	<code>transaction_minute</code>	<code>transaction_second</code>	<code>transaction_time</code>
2	52	22	2:52:22 AM

transaction_hour	transaction_minute	transaction_second	transaction_time
6	32	07	6:32:07 AM
9	25	23	9:25:23 AM
12	09	16	12:09:16 PM
17	55	22	5:55:22 PM
18	43	30	6:43:30 PM
21	43	41	9:43:41 PM

The `transaction_time` field is created in the preceding load statement by using the `maketime()` function, and passing the hour, minute, and second fields as function arguments.

The function then combines and converts these values into a time field, returning the results in the time format of the `TimeFormat` system variable.

Example 2 – `time()` function

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `transaction_time`, which will allow us to show the results in 24-hour time format without modifying the `TimeFormat` system variable.

Load script

```
SET TimeFormat='h:mm:ss TT';

Transactions:
  Load
    *,
    time(maketime(transaction_hour, transaction_minute, transaction_second), 'h:mm:ss') as
    transaction_time
  ;
  Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
```

```
3757, 9, 25, 23, 3177.4, 21, 203521  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_hour`
- `transaction_minute`
- `transaction_second`
- `transaction_time`

Results table

<code>transaction_hour</code>	<code>transaction_minute</code>	<code>transaction_second</code>	<code>transaction_time</code>
2	52	22	2:52:22
6	32	07	6:32:07
9	25	23	9:25:23
12	09	16	12:09:16
17	55	22	17:55:22
18	43	30	18:43:30
21	43	41	21:43:41

In this instance, the `maketime()` function is nested inside the `time()` function. The second argument of the `time()` function sets the format of the `maketime()` function results as the required `h:mm:ss`.

Example 3 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions , which is loaded into a table called `Transactions`.
- Transaction times provided across two fields: `hours` and `minutes`.
- The creation of a field, `transaction_time`, that returns the time in the format of the `TimeFormat` system variable.

Create a chart object measure, `transaction_time`, that returns a time in the format `h:mm:ss TT`.

Load script

```
SET TimeFormat='h:mm:ss TT';

Transactions:
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_amount, transaction_
quantity, customer_id
3750, 18, 43, 12423.56, 23, 2038593
3751, 6, 32, 5356.31, 6, 203521
3752, 12, 09, 15.75, 1, 5646471
3753, 21, 43, 7, 3036491
3754, 17, 55, 21484.21, 1356, 049681
3756, 2, 52, -59.18, 2, 2038593
3757, 9, 25, 3177.4, 21, 203521
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_hour`
- `transaction_minute`

To calculate the `transaction_time`, create this measure:

```
=maketime(transaction_hour,transaction_minute)
```

Results table

<code>transaction_hour</code>	<code>transaction_minute</code>	<code>=maketime(transaction_hour,transaction_minute)</code>
2	52	2:52:00 AM
6	32	6:32:00 AM
9	25	9:25:00 AM
12	09	12:09:00 PM
17	55	5:55:00 PM
18	43	6:43:00 PM
21	43	9:43:00 PM

The `transaction_time` measure is created in the chart object by using the `maketime()` function, and passing the hour and minute fields as function arguments.

The function then combines these values, and seconds are assumed to be 00. These values are then converted into a time field, returning the results in the format of the `TimeFormat` system variable.

Example 4 – Scenario

Load script and chart expression

Overview

Create a calendar dataset for the month of January 2022, broken out into eight-hour increments.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

tmpCalendar:
  load
    *
  where year(date)=2022;
  load
    date(recno()+makedate(2021,12,31)) as date
  AutoGenerate 31;

  Left join(tmpCalendar)
  load
    maketime((recno()-1)*8,00,00) as time
  autogenerate 3;

Calendar:
  load
    timestamp(date + time) as timestamp
  resident tmpCalendar;

drop table tmpCalendar;
```

Results

Results table

timestamp
1/1/2022 12:00:00 AM
1/1/2022 8:00:00 AM
1/1/2022 4:00:00 PM
1/2/2022 12:00:00 AM
1/2/2022 8:00:00 AM
1/2/2022 4:00:00 PM
1/3/2022 12:00:00 AM
1/3/2022 8:00:00 AM

timestamp
1/3/2022 4:00:00 PM
1/4/2022 12:00:00 AM
1/4/2022 8:00:00 AM
1/4/2022 4:00:00 PM
1/5/2022 12:00:00 AM
1/5/2022 8:00:00 AM
1/5/2022 4:00:00 PM
1/6/2022 12:00:00 AM
1/6/2022 8:00:00 AM
1/6/2022 4:00:00 PM
1/7/2022 12:00:00 AM
1/7/2022 8:00:00 AM
1/7/2022 4:00:00 PM
1/8/2022 12:00:00 AM
1/8/2022 8:00:00 AM
1/8/2022 4:00:00 PM
1/9/2022 12:00:00 AM
+ 68 more rows

The initial autogenerate function creates a calendar containing all the dates in January in a table called `tmpCalendar`.

A second table, containing three records, is created. For each record, `recno() - 1` is taken (values 0, 1, 2) and the result is multiplied by 8. As a result, this generates the values 0, 8 16. These values are used as the hour parameter in a `maketime()` function, with minute and second values of 0. As a result, the table contains three time fields: 12:00:00 AM, 8:00:00 AM, and 4:00:00 PM.

This table is joined to the `tmpCalendar` table. Because there are no matching fields between the two tables for the join, the time rows are added to each date row. As a result, each date row is now repeated three times with each time value.

Finally, the Calendar table is created from a resident load of the `tmpCalendar` table. The date and time fields are concatenated and wrapped in the `timestamp()` function to create the `timestamp` field.

The `tmpcalendar` table is then dropped.

makeweekdate

This function returns a date calculated from the year, the week number, and the day of week .

Syntax:

```
MakeWeekDate(weekyear [, week [, weekday [, first_week_day [, broken_weeks [, reference_day]]]]])
```

Return data type: dual

The `makeweekdate()` function is available both as script and chart function. The function will calculate the date based on the parameters passed into the function.

Arguments

Argument	Description
weekyear	<p>The year as defined by the <code>weekyear()</code> function for the specific date, that is the year to which the week number belongs.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>The week year can in some cases be different from the calendar year, for example if week 1 starts already in December of the previous year.</i> </div>
week	<p>The week number as defined by the <code>week()</code> function for the specific date.</p> <p>If no week number is stated, 1 is assumed.</p>
weekday	<p>The day-of-week as defined by the <code>weekday()</code> function for the date in question. 0 is the first day of the week, and 6 is the last day of the week.</p> <p>If no day-of-week is stated, 0 is assumed.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>Even though 0 always means first day of the week and 6 is always the last, which weekdays that corresponds to is determined by the first_week_day parameter. If omitted, the value of variable FirstWeekDay is used.</i> </div> <p>If broken weeks are used, together with an impossible combination of parameters, this may lead to a result that does not belong to the chosen year.</p>
Example:	
<pre>Makeweekdate(2021,1,0,6,1)</pre> <p>Returns ‘Dec 27 2020’ since this day is the first day (the Sunday) of the specified week. Jan 1 2021 was a Friday.</p>	
first_week_day	<p>Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used.</p> <p>The possible values first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday.</p> <p>For more information about the system variable, see <i>FirstWeekDay</i> (page 215).</p>

Argument	Description
broken_weeks	If you don't specify broken_weeks , the value of variable BrokenWeeks is used to define whether weeks are broken or not.
reference_day	If you don't specify reference_day , the value of variable ReferenceDay is used to define which day in January to set as reference day to define week 1.

When to use it

The `makeweekdate()` function would commonly be used in the script for data generation to generate a list of dates, or to construct dates when the year, week and day-of-week are provided in the input data.

The following examples assume:

```
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;
```

Function examples

Example	Result
<code>makeweekdate(2014,6,6)</code>	returns 02/09/2014
<code>makeweekdate(2014,6,1)</code>	returns 02/04/2014
<code>makeweekdate(2014,6)</code>	returns 02/03/2014 (weekday 0 is assumed)

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – day included

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing weekly sales total for 2022 in a table called sales.
- Transaction dates provided across three fields: year, week, and sales.
- A preceding load, which is used to create a measure, end_of_week, using the makeweedate() function to return the date for the Friday of that week in the format MM/DD/YYYY.

To prove that the date returned is a Friday, the end_of_week expression is also wrapped in the weekday() function to show the day of the week.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
Load
  *,
  makeweedate(transaction_year, transaction_week,4) as end_of_week,
  weekday(makeweedate(transaction_year, transaction_week,4)) as week_day
;
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- transaction_year
- transaction_week
- end_of_week
- week_day

Results table

transaction_year	transaction_week	end_of_week	week_day
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri

transaction_year	transaction_week	end_of_week	week_day
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

The `end_of_week` field is created in the preceding load statement by using the `makeweekdate()` function. The `transaction_year`, `transaction_week` fields are passed through the function as the year and week arguments. A value of 4 is used for the day argument.

The function then combines and converts these values into a date field, returning the results in the format of the `DateFormat` system variable.

The `makeweekdate()` function, and its arguments are also wrapped in a `weekday()` function to return the `week_day` field; and as can be seen in the table above, the `week_day` field shows that these dates do occur on a Friday.

Example 2 – day excluded

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing weekly sales totals for 2022 in a table called `sales`.
- Transaction dates provided across three fields: `year`, `week`, and `sales`.
- A preceding load, which is used to create a measure, `first_day_of_week`, using the `makeweekdate()` function. This will return the date for the Monday of that week in the format `MM/DD/YYYY`.

To prove that the date returned is a Monday, the `first_day_of_week` expression is also wrapped in the `weekday()` function to show the day of the week.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
  Load
    *,
    makeweekdate(transaction_year, transaction_week) as first_day_of_week,
    weekday(makeweekdate(transaction_year, transaction_week)) as week_day
  ;
Load * InLine [
transaction_year, transaction_week, sales
```

```
2022, 01, 10000  
2022, 02, 11250  
2022, 03, 9830  
2022, 04, 14010  
2022, 05, 28402  
2022, 06, 9992  
2022, 07, 7292  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `transaction_year`
- `transaction_week`
- `first_day_of_week`
- `week_day`

Results table

<code>transaction_year</code>	<code>transaction_week</code>	<code>first_day_of_week</code>	<code>week_day</code>
2022	01	01/03/2022	Mon
2022	02	01/10/2022	Mon
2022	03	01/17/2022	Mon
2022	04	01/24/2022	Mon
2022	05	01/31/2022	Mon
2022	06	02/07/2022	Mon
2022	07	02/14/2022	Mon

The `first_day_of_week` field is created in the preceding load statement by using the `makeweedate()` function. The `transaction_year` and `transaction_week` parameters are passed as function arguments, and the `day` parameter is left blank.

The function then combines and converts these values into a date field, returning the results in the format of the `DateFormat` system variable.

The `makeweedate()` function and its arguments are also wrapped in a `weekday()` function to return the `week_day` field. As can be seen in the table above, the `week_day` field returns Monday in all cases since that parameter was left blank in the `makeweedate()` function, which defaults to 0 (first day of the week), and first day of the week is set to Monday by the `FirstWeekDay` system variable.

Example 3 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing weekly sales totals for 2022 in a table called sales.
- Transaction dates provided across three fields: year, week, and sales.

In this example, a chart object will be used to create a measure equivalent to the end_of_week calculation from the first example. This measure will use the makeweekdate() function to return the date for the Friday of that week in the format MM/DD/YYYY.

To prove that the date returned is a Friday, a second measure is created to return the day of the week.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Master_Calendar:
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add these fields as dimensions:
 - transaction_year
 - transaction_week
2. To perform the calculation equivalent to that of the end_of_week field from the first example, create the following measure:
`=makeweekdate(transaction_year,transaction_week,4)`

3. To calculate the day of the week for each transaction, create the following measure:
`=weekday(makeweedate(transaction_year,transaction_week,4))`

Results table

transaction_year	transaction_week	=makeweedate(transaction_year,transaction_week,4)	=weekday(makeweedate(transaction_year,transaction_week,4))
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

An equivalent field to `end_of_week` is created in the chart object as a measure by using the `makeweedate()` function. The `transaction_year` and `transaction_week` fields are passed as year and week arguments. A value of 4 is used for the day argument.

The function then combines and converts these values into a date field, returning the results in the format of the `DateFormat` system variable.

The `makeweedate()` function and its arguments are also wrapped in a `weekday()` function to return a calculation equivalent to that of the `week_day` field from the first example. As can be seen in the table above, the last column on the right shows that these dates do occur on a Friday.

Example 4 – Scenario

Load script and chart expression

Overview

In this example, create a list of dates containing all the Fridays for the year 2022.

Open the Data load editor and add the load script below to a new tab.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;
```

Calendar:

```
  load
    * ,
```

```
    weekday(date) as weekday
    where year(date)=2022;
load
    makeweekdate(2022,recno()-2,4) as date
AutoGenerate 60;
```

Results

Results table

date	weekday
01/07/2022	Fri
01/14/2022	Fri
01/21/2022	Fri
01/28/2022	Fri
02/04/2022	Fri
02/11/2022	Fri
02/18/2022	Fri
02/25/2022	Fri
03/04/2022	Fri
03/11/2022	Fri
03/18/2022	Fri
03/25/2022	Fri
04/01/2022	Fri
04/08/2022	Fri
04/15/2022	Fri
04/22/2022	Fri
04/29/2022	Fri
05/06/2022	Fri
05/13/2022	Fri
05/20/2022	Fri
05/27/2022	Fri
06/03/2022	Fri
06/10/2022	Fri
06/17/2022	Fri
+ 27 more rows	

The `makeweekdate()` function finds each Friday in 2022. Using a week parameter of -2 ensures that no dates are missed. Finally, a preceding load creates an additional weekday field for clarity, to show that each date value is a Friday.

minute

This function returns an integer representing the minute when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
minute (expression)
```

Return data type: integer

When to use it

The `minute()` function is useful when you would like to compare aggregations by minute. For example, you could use the function if you would like to see activity count distribution by minute.

These dimensions can be created either in the load script by using the function to create a field in a Master Calendar table. Alternatively, they can be used directly in a chart as a calculated dimension.

Function examples

Example	Result
<code>minute ('09:14:36')</code>	Returns 14.
<code>minute ('0.5555')</code>	Returns 19 (Because $0.5555 = 13:19:55$).

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Variable (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing transactions by timestamp, which is loaded into a table called `Transactions`.
- The default `Timestamp` system variable (`M/D/YYYY h:mm:ss[.ffff] TT`) is used.
- The creation of a field, `minute`, to calculate when transactions took place.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';
```

`Transactions:`

```
    Load
        *,
        minute(timestamp) as minute
    ;
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `timestamp`
- `minute`

Results table

timestamp	minute
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49

timestamp	minute
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

The values in the `minute` field are created by using the `minute()` function and passing the `timestamp` as the expression in the preceding load statement.

Example 2 – Chart object (chart)

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The default `Timestamp` system variable (`M/D/YYYY h:mm:ss[.ffff] TT`) is used.

However, in this example, the unchanged dataset is loaded into the application. The `minute` values are calculated via a measure in a chart object.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';
```

Transactions:

```
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: `timestamp`.

Create the following measure:

```
=minute(timestamp)
```

Results table

timestamp	minute
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

The values for `minute` are created by using the `minute()` function and passing the `timestamp` as the expression in a measure for the chart object.

Example 3 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of timestamps, which is generated to represent entries at a ticket barrier.
- Information with each `timestamp` and its corresponding `id`, which is loaded into a table called `Ticket_Barrier_Tracker`.
- The default `Timestamp` system variable (`M/D/YYYY h:mm:ss[.fff] TT`) is used.

The user would like a chart object that shows, by minute, the count of barrier entries.

Load script

```

SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

tmpTimeStampCreator:
  load
    *
  where year(date)=2022;
load
  date(recno()+makedate(2021,12,31)) as date
AutoGenerate 1;

join load
  maketime(floor(rand()*24),floor(rand()*59),floor(rand()*59)) as time
autogenerate 10000;

Ticket_Barrier_Tracker:
load
  recno() as id,
  timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;

```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create a calculated dimension using the following expression:
=minute(timestamp)
3. Add the following aggregation measure to calculate total count of entries:
=count(id)
4. Set the measure's **Number formatting** to **Money**.

Results table

minute(timestamp)	=count(id)
0	174
1	171
2	175
3	165
4	188
5	176
6	158
7	187

minute(timestamp)	=count(id)
8	178
9	178
10	197
11	161
12	166
13	184
14	159
15	161
16	152
17	160
18	176
19	164
20	170
21	170
22	142
23	145
24	155
+ 35 more rows	

month

This function returns a dual value: a month name as defined in the environment variable **MonthNames** and an integer between 1-12. The month is calculated from the date interpretation of the expression, according to the standard number interpretation.

The function returns the name of the month in the format of the MonthName system variable for a particular date. It is commonly used to create a day field as a dimension in a Master Calendar.

Syntax:

```
month( expression )
```

Return data type: integer

Function examples

Example	Result
month(2012-10-12)	returns Oct
month(35648)	returns Aug, because 35648 = 1997-08-06

Example 1 – DateFormat dataset (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named `Master_Calendar`. The `DateFormat` system variable is set to `DD/MM/YYYY`.
- A preceding load that creates an additional field, named `month_name`, using the `month()` function.
- An additional field, named `long_date`, using the `date()` function to express the full date.

Load script

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
    date,
    date(date,'dd-MMMM-YYYY') as long_date,
    month(date) as month_name
Inline
[
date
03/01/2022
03/02/2022
03/03/2022
03/04/2022
03/05/2022
03/06/2022
03/07/2022
03/08/2022
03/09/2022
03/10/2022
03/11/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `long_date`
- `month_name`

Results table

date	long_date	month_name
03/01/2022	03-January- 2022	Jan
03/02/2022	03-February- 2022	Feb
03/03/2022	03-March- 2022	Mar
03/04/2022	03-April- 2022	Apr
03/05/2022	03-May- 2022	May
03/06/2022	03-June- 2022	Jun
03/07/2022	03-July- 2022	Jul
03/08/2022	03-August- 2022	Aug
03/09/2022	03-September- 2022	Sep
03/10/2022	03-October- 2022	Oct
03/11/2022	03-November- 2022	Nov

The month name is correctly evaluated by the `month()` function in the script.

Example 2 – ANSI dates (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named `Master_Calendar`. The `DateFormat` system variable `DD/MM/YYYY` is used. However, the dates that are included in the dataset are in ANSI standard date format.
- A preceding load that creates an additional field, named `month_name`, using the `month()` function.
- An additional field, named `long_date`, using the `date()` function to express the full date.

Load script

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
    date,
    date(date,'dd-MMMM-YYYY') as long_date,
    month(date) as month_name

Inline
[
date
2022-01-11
```

```
2022-02-12  
2022-03-13  
2022-04-14  
2022-05-15  
2022-06-16  
2022-07-17  
2022-08-18  
2022-09-19  
2022-10-20  
2022-11-21  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- long_date
- month_name

Results table

date	long_date	month_name
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

The month name is correctly evaluated by the month() function in the script.

Example 3 – Unformatted dates (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates named `Master_Calendar`. The `DateFormat` system variable DD/MM/YYYY is used.
- A preceding load that creates an additional field, named `month_name`, using the `month()` function.
- The original unformatted date, named `unformatted_date`.
- An additional field, named `long_date`, using the `date()` function to express the full date.

Load script

```
SET DateFormat='DD/MM/YYYY';

Master_Calendar:
Load
    unformatted_date,
    date(unformatted_date,'dd-MMMM-YYYY') as long_date,
    month(unformatted_date) as month_name

Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
;
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `unformatted_date`
- `long_date`
- `month_name`

Results table

unformatted_date	long_date	month_name
44868	03-January- 2022	Jan
44898	03-February- 2022	Feb
44928	03-March- 2022	Mar
44958	03-April- 2022	Apr

unformatted_date	long_date	month_name
44988	03-May- 2022	May
45018	03-June- 2022	Jun
45048	03-July- 2022	Jul
45078	03-August- 2022	Aug
45008	03-September- 2022	Sep
45038	03-October- 2022	Oct
45068	03-November- 2022	Nov

The month name is correctly evaluated by the `month()` function in the script.

Example 4 – Calculating expiry month

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of orders placed in March named `subscriptions`. The table contains three fields:
 - `id`
 - `order_date`
 - `amount`

Load script

`subscriptions:`

```
Load
    id,
    order_date,
    amount
Inline
[
id,order_date,amount
1,03/01/2022,231.24
2,03/02/2022,567.28
3,03/03/2022,364.28
4,03/04/2022,575.76
5,03/05/2022,638.68
6,03/06/2022,785.38
7,03/07/2022,967.46
8,03/08/2022,287.67
9,03/09/2022,764.45
```

```
10,03/10/2022,875.43  
11,03/11/2022,957.35  
1;
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: `order_date`.

To calculate the month an order will expire, create this measure: `=month(order_date+180)`.

Results table

order_date	=month(order_date+180)
03/01/2022	Jul
03/02/2022	Aug
03/03/2022	Aug
03/04/2022	Sep
03/05/2022	Oct
03/06/2022	Nov
03/07/2022	Dec
03/08/2022	Jan
03/09/2022	Mar
03/10/2022	Apr
03/11/2022	May

The `month()` function correctly determines that an order placed on the 11th of March would expire in July.

monthend

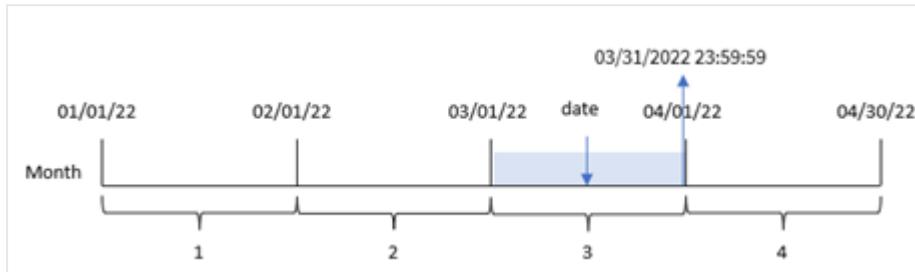
This function returns a value corresponding to a timestamp of the last millisecond of the last day of the month containing date. The default output format will be the `DateFormat` set in the script.

Syntax:

```
MonthEnd(date[, period_no])
```

In other words, the `monthend()` function determines which month the date falls into. It then returns a timestamp, in date format, for the last millisecond of that month.

Diagram of monthend function.



When to use it

The `monthend()` function is used as part of an expression when you would like the calculation to use the fraction of the month that has not yet occurred. For example, if you want to calculate the total interest not yet incurred during the month.

Return data type: dual

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, which, if 0 or omitted, indicates the month that contains date . Negative values in period_no indicate preceding months and positive values indicate succeeding months.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>monthend('02/19/2012')</code>	Returns 02/29/2012 23:59:59.
<code>monthend('02/19/2001', -1)</code>	Returns 01/31/2001 23:59:59.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 that is loaded into a table called ‘Transactions’.
- A date field in the `DateFormat` system variable (`MM/DD/YYYY`) format.
- A preceding load statement that contains:
 - The `monthend()` function which is set as the ‘`end_of_month`’ field.
 - The `timestamp` function which is set as the ‘`end_of_month_timestamp`’ field.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
  *,
  monthend(date) as end_of_month,
  timestamp(monthend(date)) as end_of_month_timestamp
  ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `end_of_month`
- `end_of_month_timestamp`

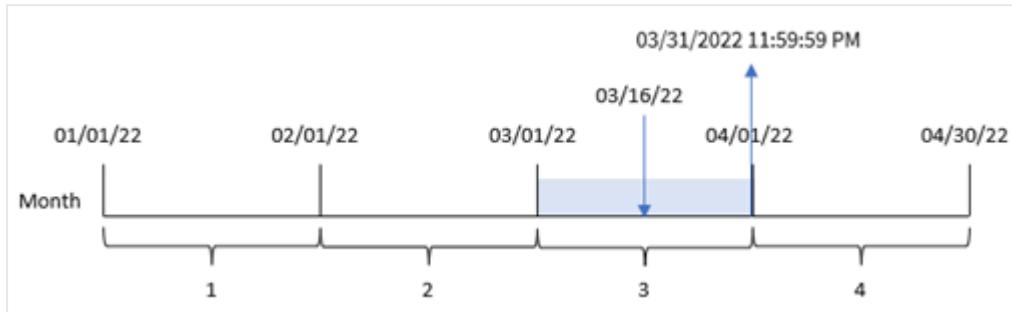
Results table

id	date	end_of_month	end_of_month_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

The ‘end_of_month’ field is created in the preceding load statement by using the `monthend()` function and passing the `date` field as the function’s argument.

The `monthend()` function identifies which month the date value falls into returning a timestamp for the last millisecond of that month.

Diagram of monthend function with March as the selected month.



Transaction 8192 took place on March 16. The `monthend()` function returns the last millisecond of that month, which is March 31 at 11:59:59 PM.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

In this example, the task is to create a field, ‘previous_month_end’, that returns the timestamp for the end of the month before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
 *,
 monthend(date,-1) as previous_month_end,
 timestamp(monthend(date,-1)) as previous_month_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **previous_month_end**
- **previous_month_end_timestamp**

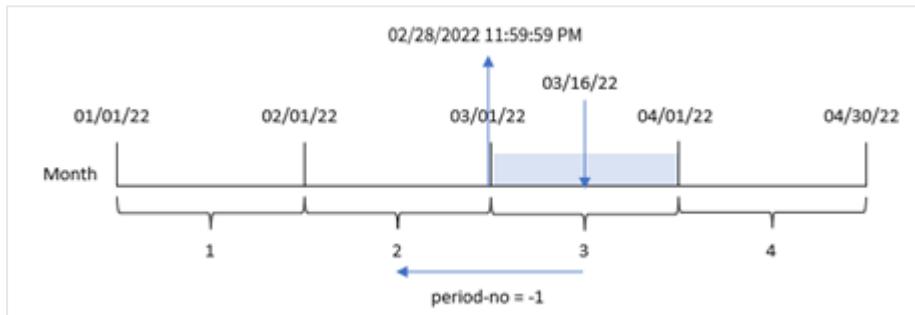
Results table

id	date	previous_month_end	previous_month_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	01/31/2022	1/31/2022 11:59:59 PM
8191	2/28/2022	01/31/2022	1/31/2022 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	05/31/2022	5/31/2022 11:59:59 PM
8197	6/26/2022	05/31/2022	5/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	07/31/2022	7/31/2022 11:59:59 PM
8203	8/8/2022	07/31/2022	7/31/2022 11:59:59 PM
8204	8/19/2022	07/31/2022	7/31/2022 11:59:59 PM

id	date	previous_month_end	previous_month_end_timestamp
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

The `monthend()` function first identifies the month that the transactions take place in as a `period_no` of -1 is used as the offset argument. It then shifts one month prior and identifies the final millisecond of that month.

Diagram of monthend function with the period_no variable.



Transaction 8192 took place on March 16. The `monthend()` function identifies that the month before the transaction took place in was February. It then returns the final millisecond of that month, February 28 at 11:59:59 PM.

Example 3 – Chart example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

In this example, the dataset is unchanged and loaded into the app. The task is to create a calculation that returns a timestamp for the end of the month when the transactions took place as a measure in a chart of the app.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
```

```

8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- id

To calculate the end date of the month that a transaction takes place in, create the following measures:

- =monthend(date)
- =timestamp(monthend(date))

Results table

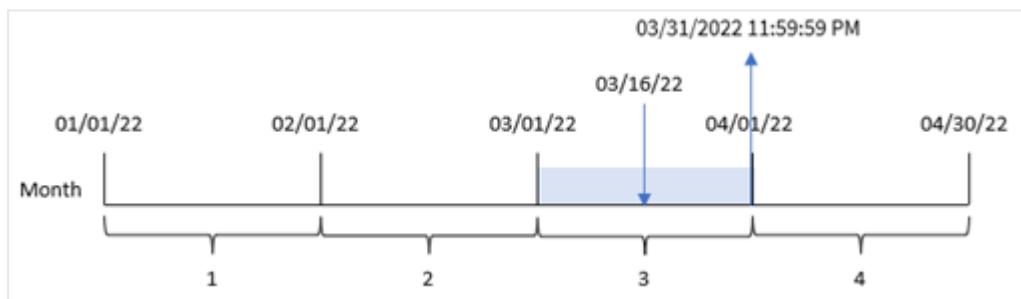
id	date	=monthend(date)	=timestamp(monthend(date))
8188	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8189	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM
8190	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8191	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8192	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8193	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8194	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8195	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8196	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM

id	date	=monthend(date)	=timestamp(monthend(date))
8200	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8201	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8202	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8203	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8204	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8205	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8206	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8207	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM

The ‘end_of_month’ measure is created in the chart by using the `monthend()` function and passing the date field as the function’s argument.

The `monthend()` function identifies which month the date value falls into and returns a timestamp for the last millisecond of that month.

Diagram of monthend function with the period_no variable.



Transaction 8192 took place on March 16. The `monthend()` function returns the last millisecond of that month, which is March 31 at 11:59:59 PM.

Example 4 – Scenario

Load script and results

Overview

In this example, a dataset is loaded into a table called ‘Employee_Expenses’. The table contains the following fields:

- Employee IDs
- Employee names
- The average daily expense claims of each employee.

The end user would like a chart that displays, by employee id and employee name, the estimated expense claim for the remainder of the month.

Load script

```
Employee_Expenses:  
Load  
*  
Inline  
[  
employee_id,employee_name,avg_daily_claim  
182,Mark, $15  
183,Deryck, $12.5  
184,Dexter, $12.5  
185,Sydney,$27  
186,Agatha,$18  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- employee_id
- employee_name

To calculate the accumulated interest, create this measure:

```
=floor(monthend(today(1),0)-today(1))*avg_daily_claim
```



This measure is dynamic and will produce different table results depending on the date when you load the data.

Set the measure's **Number formatting** to **Money**.

Results table

employee_id	employee_name	=floor(monthend(today(1),0)-today(1))*avg_daily_claim
182	Mark	\$30.00
183	Deryck	\$25.00
184	Dexter	\$25.00
185	Sydney	\$54.00
186	Agatha	\$36.00

The `monthend()` function returns the end date of the current month by using today's date as its only argument. The expression returns the number of days that remain this month by subtracting today's date from the month end date.

This value is then multiplied by the average daily expense claim by each employee to calculate the estimated value of claims each employee is expected to make in the remaining month.

monthname

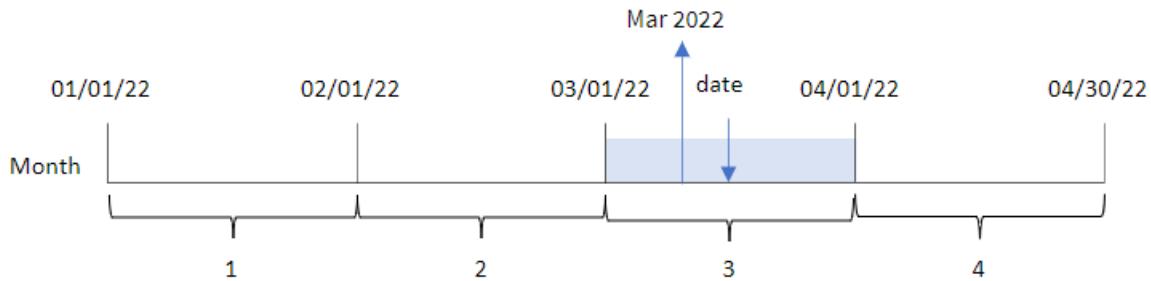
This function returns a display value showing the month (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the month.

Syntax:

```
MonthName(date[, period_no])
```

Return data type: dual

Diagram of monthname function



Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, which, if 0 or omitted, indicates the month that contains date . Negative values in period_no indicate preceding months and positive values indicate succeeding months.

Function examples

Example	Result
<code>monthname('10/19/2013')</code>	Returns Oct 2013
<code>monthname('10/19/2013', -1)</code>	Returns Sep 2013

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `transaction_month`, that returns the month in which the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
  *,
  monthname(date) as transaction_month
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
```

```
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

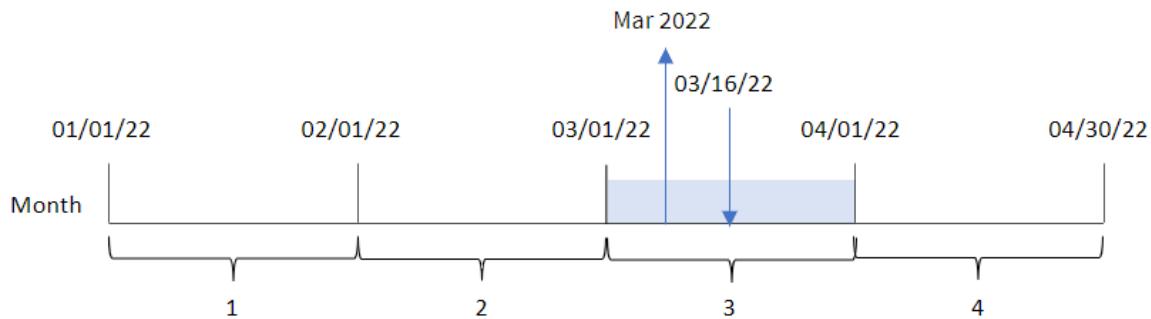
- date
- transaction_month

Results table

date	transaction_month
1/7/2022	Jan 2022
1/19/2022	Jan 2022
2/5/2022	Feb 2022
2/28/2022	Feb 2022
3/16/2022	Mar 2022
4/1/2022	Apr 2022
5/7/2022	May 2022
5/16/2022	May 2022
6/15/2022	Jun 2022
6/26/2022	Jun 2022
7/9/2022	Jul 2022
7/22/2022	Jul 2022
7/23/2022	Jul 2022
7/27/2022	Jul 2022
8/2/2022	Aug 2022
8/8/2022	Aug 2022
8/19/2022	Aug 2022
9/26/2022	Sep 2022
10/14/2022	Oct 2022
10/29/2022	Oct 2022

The `transaction_month` field is created in the preceding load statement by using the `monthname()` function and passing the `date` field as the function's argument.

Diagram of monthname function, basic example



The `monthname()` function identifies that transaction 8192 took place in March 2022, and returns this value using the `MonthNames` system variable.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same inline dataset and scenario as the first example.
- The creation of a field, `transaction_previous_month`, that returns the timestamp for the end of the month before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

Transactions:

```
Load
  *,
  monthname(date,-1) as transaction_previous_month
;
```

Load

*

Inline

[

id,date,amount

8188,1/7/2022,17.17

8189,1/19/2022,37.23

8190,2/28/2022,88.27

8191,2/5/2022,57.42

8192,3/16/2022,53.80

8193,4/1/2022,82.06

8194,5/7/2022,40.39

```
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- transaction_previous_month

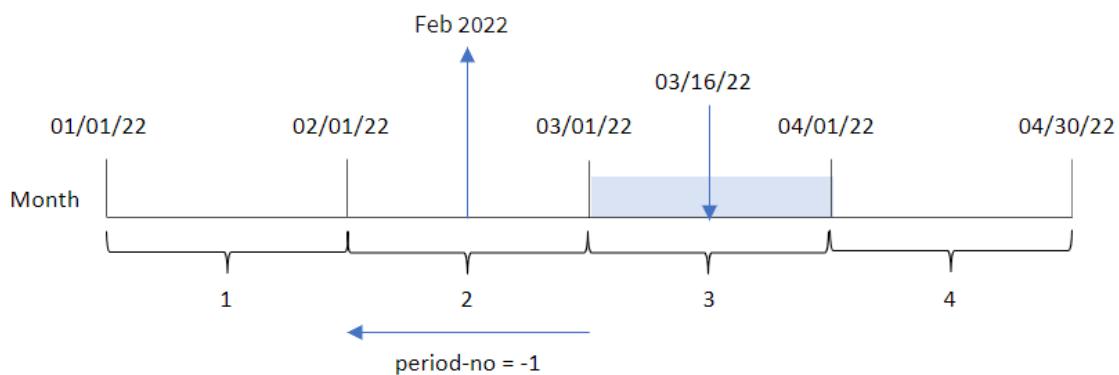
Results table

date	transaction_previous_month
1/7/2022	Dec 2021
1/19/2022	Dec 2021
2/5/2022	Jan 2022
2/28/2022	Jan 2022
3/16/2022	Feb 2022
4/1/2022	Mar 2022
5/7/2022	Apr 2022
5/16/2022	Apr 2022
6/15/2022	May 2022
6/26/2022	May 2022
7/9/2022	Jun 2022
7/22/2022	Jun 2022
7/23/2022	Jun 2022
7/27/2022	Jun 2022
8/2/2022	Jul 2022
8/8/2022	Jul 2022
8/19/2022	Jul 2022

date	transaction_previous_month
9/26/2022	Aug 2022
10/14/2022	Sep 2022
10/29/2022	Sep 2022

In this instance, because a period_no of -1 was used as the offset argument in the monthname() function, the function first identifies the month that the transactions take place in. It then shifts to one month prior and returns the month name and year.

Diagram of monthname function, period_no example



Transaction 8192 took place on March 16. The monthname() function identifies that the month before the transaction took place was February and returns the month, in the MonthNames system variable format, along with the year 2022.

Example 3 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same inline dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the end of the month when the transactions took place is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
```

Transactions:

```
Load
*
```

```
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:date.

Create the following measure:

```
=monthname(date)
```

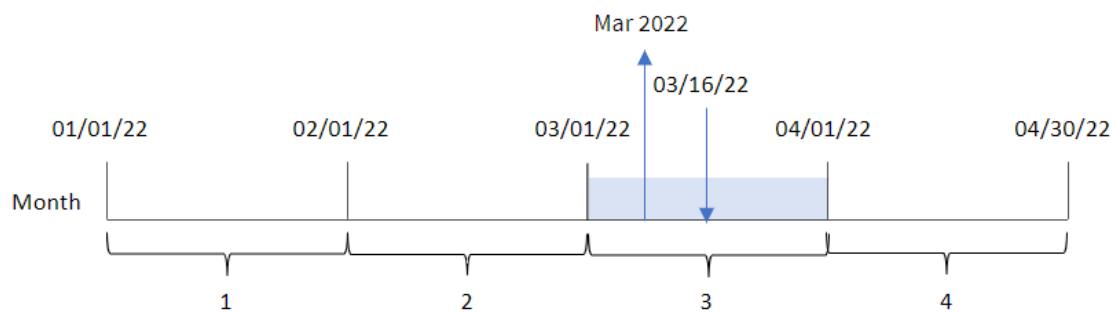
Results table

date	=monthname(date)
1/7/2022	Jan 2022
1/19/2022	Jan 2022
2/5/2022	Feb 2022
2/28/2022	Feb 2022
3/16/2022	Mar 2022
4/1/2022	Apr 2022
5/7/2022	May 2022
5/16/2022	May 2022
6/15/2022	Jun 2022
6/26/2022	Jun 2022

date	=monthname(date)
7/9/2022	Jul 2022
7/22/2022	Jul 2022
7/23/2022	Jul 2022
7/27/2022	Jul 2022
8/2/2022	Aug 2022
8/8/2022	Aug 2022
8/19/2022	Aug 2022
9/26/2022	Sep 2022
10/14/2022	Oct 2022
10/29/2022	Oct 2022

The `month_name` measure is created in the chart object by using the `monthname()` function and passing the date field as the function's argument.

Diagram of monthname function, chart object example



The `monthname()` function identifies that transaction 8192 took place in March 2022, and returns this value using the `MonthNames` system variable.

monthsend

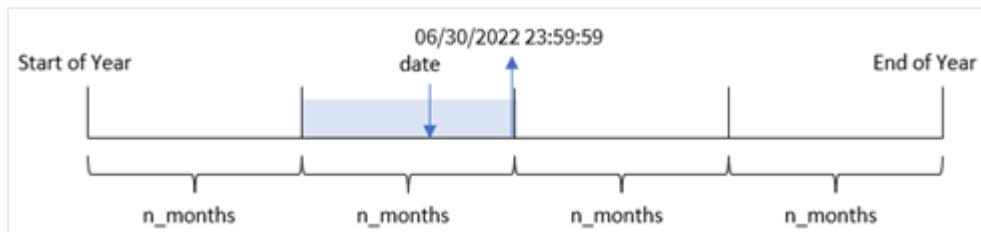
This function returns a value corresponding to the timestamp of the last millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for the end of a previous or following time period. The default output format is the `DateFormat` set in the script.

Syntax:

```
MonthsEnd(n_months, date[, period_no [, first_month_of_year]])
```

Return data type: dual

Diagram of monthsend function.



Arguments

Argument	Description
n_months	The number of months that defines the period. An integer or expression that resolves to an integer that must be one of: 1 (equivalent to the inmonth() function), 2 (bi-month), 3 (equivalent to the inquarter()function), 4 (four-month period), or 6 (half year).
date	The date or timestamp to evaluate.
period_no	The period can be offset by period_no , an integer, or expression resolving to an integer, where the value 0 indicates the period that contains base_date . Negative values in period_no indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

The `monthsend()` function divides the year into segments based on the `n_months` argument provided. It then evaluates what segment each date provided falls into and returns the last millisecond, in date format, of that segment. The function can return the end timestamp from preceding or following segments as well as redefine the first month of the year.

The following segments of the year are available in the function as `n_month` arguments.

`n_month` arguments

Period	Number of months
month	1
bi-month	2
quarter	3
four months	4
half-year	6

When to use it

The `monthsend()` function is used as part of an expression when the user would like the calculation to use the fraction of the month that has elapsed so far. The user has the opportunity, using a variable, to select the period of their choosing. For example, the `monthsend()` can provide an input variable to let the user calculate the total interest not yet incurred during the month, quarter, or half-year.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>monthsend(4, '07/19/2013')</code>	Returns 08/31/2013.
<code>monthsend(4, '10/19/2013', -1)</code>	Returns 08/31/2013.
<code>monthsend(4, '10/19/2013', 0, 2)</code>	Returns 01/31/2014. Because the start of the year becomes month 2.

Example 1 - Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 is loaded into a table called ‘Transactions’.
- A date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- A preceding load statement containing:
 - The `monthsend` function which is set as the field, ‘bi_monthly_end’. This groups transactions into bi-monthly segments.
 - The `timestamp` function which returns the starting timestamp of the segment for each transaction.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
  *,
  monthsend(2,date) as bi_monthly_end,
  timestamp(monthsend(2,date)) as bi_monthly_end_timestamp
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,1/7/2022,17.17
  8189,1/19/2022,37.23
  8190,2/28/2022,88.27
  8191,2/5/2022,57.42
  8192,3/16/2022,53.80
  8193,4/1/2022,82.06
  8194,5/7/2022,40.39
  8195,5/22/2022,87.21
  8196,6/15/2022,95.93
  8197,6/26/2022,45.89
  8198,7/9/2022,36.23
  8199,7/22/2022,25.66
  8200,7/23/2022,82.77
  8201,7/27/2022,69.98
  8202,8/2/2022,76.11
  8203,8/8/2022,25.12
  8204,8/19/2022,46.23
  8205,9/26/2022,84.21
  8206,10/14/2022,96.24
  8207,10/29/2022,67.67
  ];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `bi_monthly_end`
- `bi_monthly_end_timestamp`

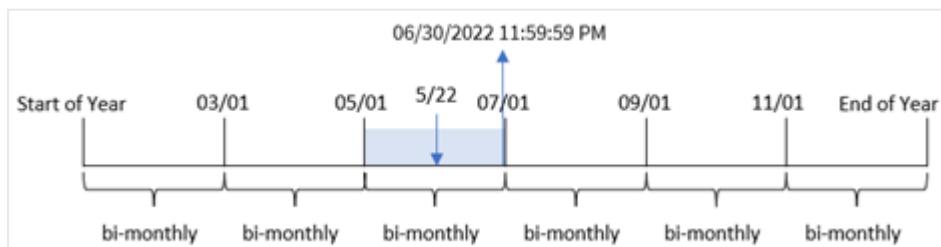
Results table

id	date	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM

id	date	bi_monthly_end	bi_monthly_end_timestamp
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

The ‘bi_monthly_end’ field is created in the preceding load statement by using the `monthsend()` function. The first argument provided is 2, dividing the year into bi-monthly segments. The second argument identifies which field is being evaluated.

Diagram of `monthsend` function with bi-monthly segments.



Transaction 8195 takes place on May 22. The `monthsend()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. As a result, the function returns the last millisecond of this segment, 06/30/2022 11:59:59 PM.

Example 2 - period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

In this example, the task is to create a field, ‘prev_bi_monthly_end’, that returns the first millisecond of the bi-monthly segment before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
 *,
 monthsend(2,date,-1) as prev_bi_monthly_end,
 timestamp(monthsend(2,date,-1)) as prev_bi_monthly_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

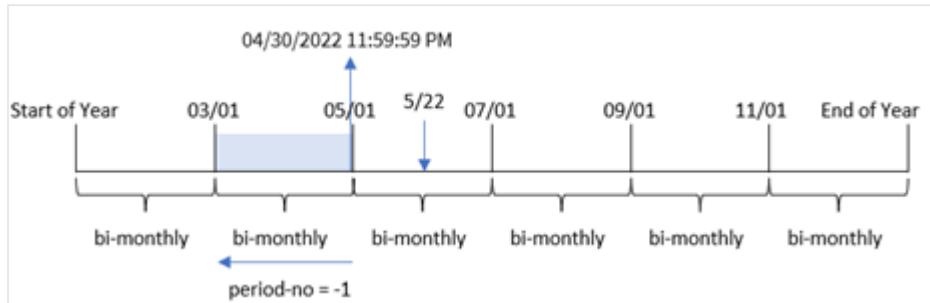
- `id`
- `date`
- `prev_bi_monthly_end`
- `prev_bi_monthly_end_timestamp`

Results table

id	date	prev_bi_monthly_end	prev_bi_monthly_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	02/28/2022	2/28/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/22/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	04/30/2022	4/30/2022 11:59:59 PM
8197	6/26/2022	04/30/2022	4/30/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	08/31/2022	8/31/2022 11:59:59 PM
8207	10/29/2022	08/31/2022	8/31/2022 11:59:59 PM

By using `-1` as the `period_no` argument in the `monthsEnd()` function, after initially dividing a year into bi-monthly segments, the function returns the last millisecond of the previous bi-monthly segment to when a transaction takes place.

Diagram of monthsend function that returns the previous bi-monthly segment.



Transaction 8195 occurs in the segment between May and June. As a result, the previous bi-monthly segment was between March 1 and April 30 and so the function returns the last millisecond of this segment, 04/30/2022 11:59:59 PM.

Example 3 – first_month_of_year

Load script and results

Overview

The same dataset and scenario as the first example are used.

In this example, the organizational policy is for April to be the first month of the financial year.

Create a field, ‘bi_monthly_end’, that groups transactions into bi-monthly segments and returns the last millisecond timestamp of the segment for each transaction.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
 *,
 monthsend(2,date,0,4) as bi_monthly_end,
 timestamp(monthsend(2,date,0,4)) as bi_monthly_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **bi_monthly_end**
- **bi_monthly_end_timestamp**

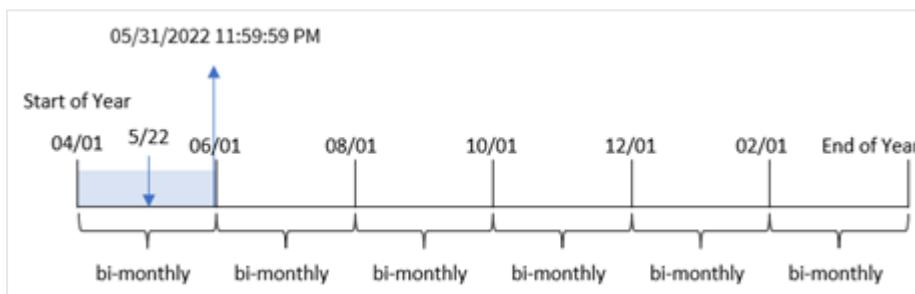
Results table

id	date	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/22/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	6/26/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM

id	date	bi_monthly_end	bi_monthly_end_timestamp
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

By using 4 as the `first_month_of_year` argument in the `monthsend()` function, the function begins the year on April 1. It then divides the year into bi-monthly segments: Apr-May, Jun-Jul, Aug-Sep, Oct-Nov, Dec-Jan, Feb-Mar.

Diagram of monthsend function with the first month of the year set as April



Transaction 8195 took place on May 22 and falls into the segment between April 1 and May 31. As a result, the function returns the last millisecond of this segment, 05/31/2022 11:59:59 PM.

Example 4 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used. However in this example, the dataset is unchanged, and loaded into the app.

In this example, the task is to create a calculation that groups transactions into bi-monthly segments and returns the last millisecond timestamp of the segment for each transaction as a measure in a chart object of an app.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
```

```
8189,3/7/2022,17.17  
8190,3/30/2022,88.27  
8191,4/5/2022,57.42  
8192,4/16/2022,53.80  
8193,5/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/22/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

date

To fetch the last millisecond timestamp of the bi-monthly segment when the transaction took place, create the following measures:

- `=monthsEnd(2,date)`
- `=timestamp(monthsend(2,date))`

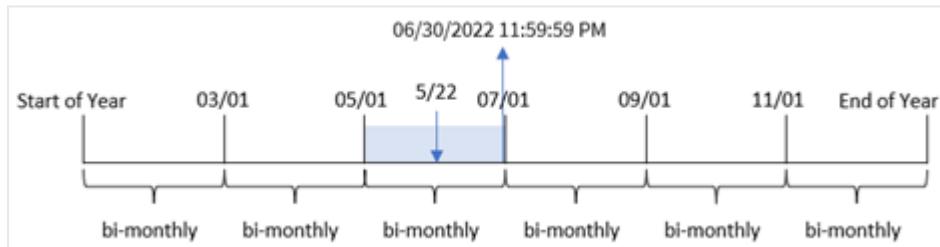
Results table

id	date	=monthsend(2,date)	=timestamp(monthsend(2,date))
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM

id	date	=monthsend(2,date)	=timestamp(monthsend(2,date))
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

The ‘bi_monthly_end’ field is created as a measure in the chart object by using the `monthsend()` function. The first argument provided is 2, which divides the year into bi-monthly segments. The second argument identifies which field is being evaluated.

Diagram of monthsend function with bi-monthly segments.



Transaction 8195 takes place on May 22. The `monthsend()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. As a result, the function returns the first millisecond of this segment, 06/30/2022 11:59:59 PM.

Example 5 – Scenario

Load script and results

Overview

Open the data load editor and add the load script below to a new tab.

In this example, a dataset is loaded into a table called ‘Employee_Expenses’. The table contains the following fields:

- Employee IDs
- Employee names

- The average daily expense claims of each employee.

The end user would like a chart that displays, by employee id and employee name, the estimated expense claim for the remainder of a period of their own choosing. The financial year begins in January.

Load script

```
SET vPeriod = 1;

Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

Results

Load the data and open a new sheet.

At the start of the load script, a variable, **vPeriod**, is created that is tied to the variable input control.

Do the following:

1. In the assets panel, click **Custom objects**.
2. Select **Qlik Dashboard bundle**, create a **Variable input** object.
3. Enter a title for the chart object.
4. Under **Variable**, select **vPeriod** as the name and set the object to show as a **Drop down**.
5. Under **Values**, click **Dynamic** values. Enter the following:
='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.

Create a new table and these fields as dimensions:

- **employee_id**
- **employee_name**

To calculate the accumulated interest, create this measure:

```
=floor(monthsend($(vPeriod),today(1))-today(1))*avg_daily_claim
```



This measure is dynamic and will produce different table results depending on the date when you load the data.

Set the measure's **Number formatting** to **Money**.

Results table

employee_id	employee_name	=floor(monthsend\$(vPeriod),today(1))-today(1)*avg_daily_claim
182	Mark	\$1410.00
183	Deryck	\$1175.00
184	Dexter	\$1175.00
185	Sydney	\$2538.00
186	Agatha	\$1692.00

The `monthsend()` function uses the user input as its first argument and today's date as its second argument. This returns the end date for the user selected period of time. Then, the expression returns the number of days that remain the selected period of time by subtracting today's date from this end date.

This value is then multiplied by the average daily expense claim by each employee to calculate the estimated value of claims each employee is expected to make in the remaining days of this period.

monthsname

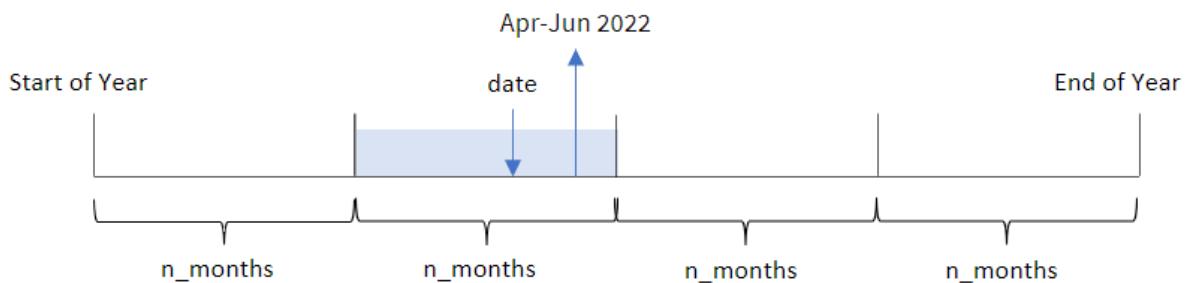
This function returns a display value representing the range of the months of the period (formatted according to the **MonthNames** script variable) as well as the year. The underlying numeric value corresponds to a timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date.

Syntax:

```
MonthsName(n_months, date[, period_no[, first_month_of_year]])
```

Return data type: dual

Diagram of monthsname function



The `monthsname()` function divides the year into segments based on the `n_months` argument provided. It then evaluates the segment to which each provided date belongs, and returns the start and end month names of that segment, as well as the year. The function also provides the ability to return these boundaries from preceding or following segments, as well as redefining which is the first month of the year.

The following segments of the year are available in the function as `n_month` arguments:

Possible n_month arguments

Periods	Number of months
month	1
bi-month	2
quarter	3
four months	4
half-year	6

Arguments

Argument	Description
n_months	The number of months that defines the period. An integer or expression that resolves to an integer that must be one of: 1 (equivalent to the inmonth() function), 2 (bi-month), 3 (equivalent to the inquarter()function), 4 (four-month period), or 6 (half year).
date	The date or timestamp to evaluate.
period_no	The period can be offset by period_no , an integer, or expression resolving to an integer, where the value 0 indicates the period that contains base_date . Negative values in period_no indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

When to use it

The `monthsname()` function is useful when you would like to provide the user with the functionality to compare aggregations by a period of their choosing. For example, you could provide an input variable to let the user see the total sales of products by month, quarter, or half-year.

These dimensions can be created either in the load script by adding the function as a field in a Master Calendar table, or alternatively, by creating the dimension directly in a chart as a calculated dimension.

Function examples

Example	Result
<code>monthsname(4, '10/19/2013')</code>	Returns 'Sep-Dec 2013.' In this and the other examples, the SET Monthnames statement is set to Jan;Feb;Mar, and so on.
<code>monthsname(4, '10/19/2013', -1)</code>	Returns 'May-Aug 2013'.
<code>monthsname(4, '10/19/2013', 0, 2)</code>	Returns 'Oct-Jan 2014', since the year is specified to begin in month 2. Therefore, the four-month period ends on the first month of the following year.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `bi_monthly_range`, that groups transactions into bi-monthly segments and returns the boundary names of that segment for each transaction.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    monthsname(2,date) as bi_monthly_range
  ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
```

```
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- bi_monthly_range

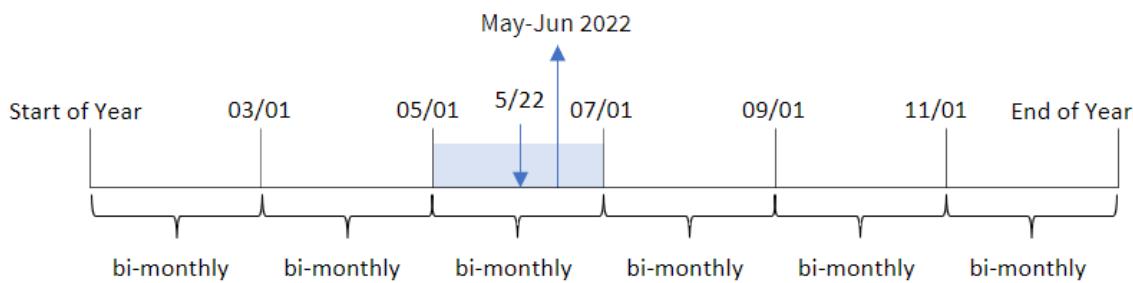
Results table

date	bi_monthly_range
2/19/2022	Jan-Feb 2022
3/7/2022	Mar-Apr 2022
3/30/2022	Mar-Apr 2022
4/5/2022	Mar-Apr 2022
4/16/2022	Mar-Apr 2022
5/1/2022	May-Jun 2022
5/7/2022	May-Jun 2022
5/22/2022	May-Jun 2022
6/15/2022	May-Jun 2022
6/26/2022	May-Jun 2022
7/9/2022	Jul-Aug 2022
7/22/2022	Jul-Aug 2022
7/23/2022	Jul-Aug 2022
7/27/2022	Jul-Aug 2022
8/2/2022	Jul-Aug 2022
8/8/2022	Jul-Aug 2022
8/19/2022	Jul-Aug 2022

date	bi_monthly_range
9/26/2022	Sep-Oct 2022
10/14/2022	Sep-Oct 2022
10/29/2022	Sep-Oct 2022

The `bi_monthly_range` field is created in the preceding load statement by using the `monthsname()` function. The first argument provided is 2, dividing the year into bi-monthly segments. The second argument identifies which field is being evaluated.

Diagram of monthsname function, basic example



Transaction 8195 takes place on May 22. The `monthsname()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. Therefore, the function returns these months in the `MonthNames` system variable format, as well as the year, May-Jun 2022.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same inline dataset and scenario as the first example.
- The creation of a field, `prev_bi_monthly_range`, that groups transactions into bi-monthly segments and returns the previous segment boundary names for each transaction.

Add your other text here, as needed, with lists etc.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

Load

*

```
MonthsName(2,date,-1) as prev_bi_monthly_range  
;  
Load  
*  
Inline  
[  
id,date,amount  
8188,2/19/2022,37.23  
8189,3/7/2022,17.17  
8190,3/30/2022,88.27  
8191,4/5/2022,57.42  
8192,4/16/2022,53.80  
8193,5/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/22/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- prev_bi_monthly_range

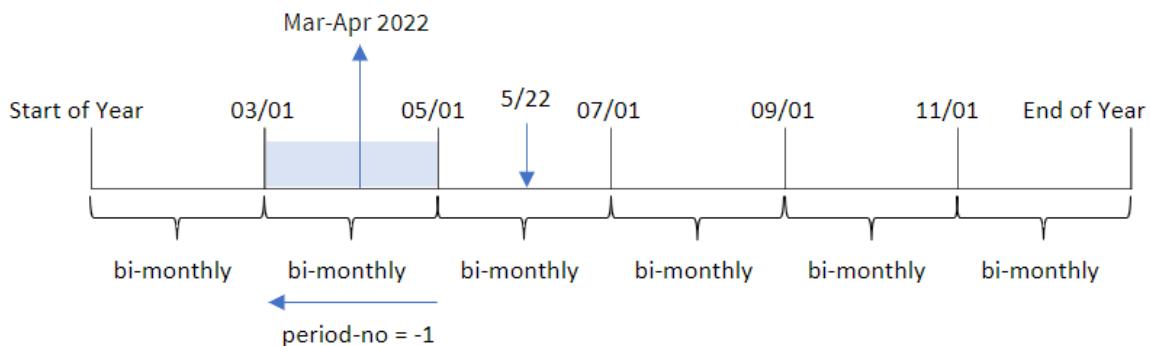
Results table

date	prev_bi_monthly_range
2/19/2022	Nov-Dec 2021
3/7/2022	Jan-Feb 2022
3/30/2022	Jan-Feb 2022
4/5/2022	Jan-Feb 2022
4/16/2022	Jan-Feb 2022
5/1/2022	Mar-Apr 2022
5/7/2022	Mar-Apr 2022
5/22/2022	Mar-Apr 2022

date	prev_bi_monthly_range
6/15/2022	Mar-Apr 2022
6/26/2022	Mar-Apr 2022
7/9/2022	May-Jun 2022
7/22/2022	May-Jun 2022
7/23/2022	May-Jun 2022
7/27/2022	May-Jun 2022
8/2/2022	May-Jun 2022
8/8/2022	May-Jun 2022
8/19/2022	May-Jun 2022
9/26/2022	Jul-Aug 2022
10/14/2022	Jul-Aug 2022
10/29/2022	Jul-Aug 2022

In this example, -1 is used as the period_no argument in the monthsname() function. After initially dividing a year into bi-monthly segments, the function then returns the previous segment boundaries for when a transaction takes place.

Diagram of monthsname function, period_no example



Transaction 8195 occurs in the segment between May and June. Therefore, the previous bi-monthly segment was between March 1 and April 30, and so the function returns Mar-Apr 2022.

Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same inline dataset and scenario as the first example.
- The creation of a different field, `bi_monthly_range`, that groups transactions into bi-monthly segments and returns the segment boundaries for each transaction.

However, in this example, we also need to set April as the first month of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  MonthsName(2,date,0,4) as bi_monthly_range
;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `date`
- `bi_monthly_range`

Results table

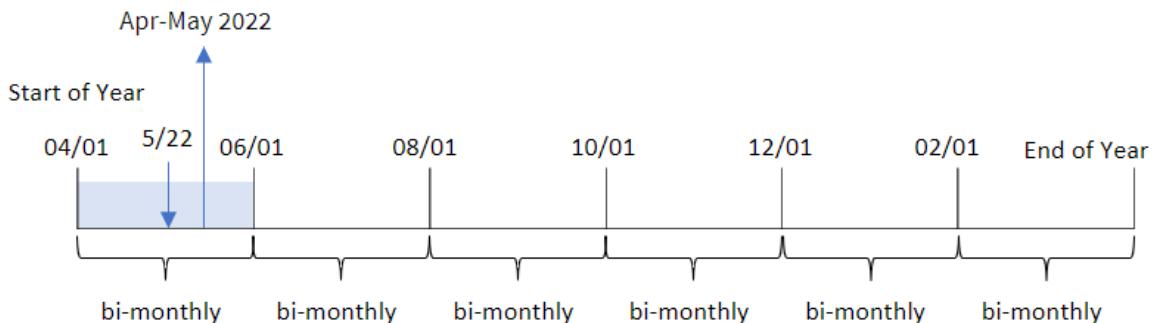
date	bi_monthly_range
2/19/2022	Feb-Mar 2021
3/7/2022	Feb-Mar 2021
3/30/2022	Feb-Mar 2021
4/5/2022	Apr-May 2022
4/16/2022	Apr-May 2022
5/1/2022	Apr-May 2022
5/7/2022	Apr-May 2022
5/22/2022	Apr-May 2022
6/15/2022	Jun-Jul 2022
6/26/2022	Jun-Jul 2022
7/9/2022	Jun-Jul 2022
7/22/2022	Jun-Jul 2022
7/23/2022	Jun-Jul 2022
7/27/2022	Jun-Jul 2022
8/2/2022	Aug-Sep 2022
8/8/2022	Aug-Sep 2022
8/19/2022	Aug-Sep 2022
9/26/2022	Aug-Sep 2022
10/14/2022	Oct-Nov 2022
10/29/2022	Oct-Nov 2022

By using 4 as the `first_month_of_year` argument in the `monthsname()` function, the function begins the year on April 1. It then divides the year into bi-monthly segments: Apr-May, Jun-Jul, Aug-Sep, Oct-Nov, Dec-Jan, Feb-Mar.

Paragraph text for Results.

Transaction 8195 took place on May 22 and falls into the segment between April 1 and May 31. Therefore, the function returns Apr-May 2022.

Diagram of monthsname function, first_month_of_year example



Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same inline dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that groups transactions into bi-monthly segments and returns the segment boundaries for each transaction is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
```

```
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:date.

Create the following measure:

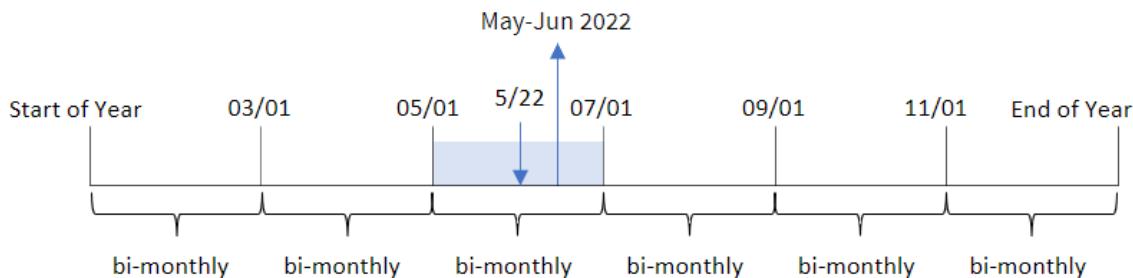
```
=monthsname(2,date)
```

Results table

date	=monthsname(2,date)
2/19/2022	Jan-Feb 2022
3/7/2022	Mar-Apr 2022
3/30/2022	Mar-Apr 2022
4/5/2022	Mar-Apr 2022
4/16/2022	Mar-Apr 2022
5/1/2022	May-Jun 2022
5/7/2022	May-Jun 2022
5/22/2022	May-Jun 2022
6/15/2022	May-Jun 2022
6/26/2022	May-Jun 2022
7/9/2022	Jul-Aug 2022
7/22/2022	Jul-Aug 2022
7/23/2022	Jul-Aug 2022
7/27/2022	Jul-Aug 2022
8/2/2022	Jul-Aug 2022
8/8/2022	Jul-Aug 2022
8/19/2022	Jul-Aug 2022
9/26/2022	Sep-Oct 2022
10/14/2022	Sep-Oct 2022
10/29/2022	Sep-Oct 2022

The `bi_monthly_range` field is created as a measure in the chart object by using the `monthsname()` function. The first argument provided is 2, dividing the year into bi-monthly segments. The second argument identifies which field is being evaluated.

Diagram of monthsname function, chart object example



Transaction 8195 takes place on May 22. The `monthsname()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. Therefore, the function returns these months in the `MonthNames` system variable format, as well as the year, May-Jun 2022.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

The end user would like a chart object that displays total sales by a period of their own choosing. This could be achieved even when this dimension is not available in the data model, using the `monthsname()` function as a calculated dimension that is dynamically modified by a variable input control.

Load script

```
SET vPeriod = 1;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
```

```
8190, '2/28/2022', 88.27  
8191, '2/5/2022', 57.42  
8192, '3/16/2022', 53.80  
8193, '4/1/2022', 82.06  
8194, '5/7/2022', 40.39  
8195, '5/16/2022', 87.21  
8196, '6/15/2022', 95.93  
8197, '6/26/2022', 45.89  
8198, '7/9/2022', 36.23  
8199, '7/22/2022', 25.66  
8200, '7/23/2022', 82.77  
8201, '7/27/2022', 69.98  
8202, '8/2/2022', 76.11  
8203, '8/8/2022', 25.12  
8204, '8/19/2022', 46.23  
8205, '9/26/2022', 84.21  
8206, '10/14/2022', 96.24  
8207, '10/29/2022', 67.67  
];
```

Results

Load the data and open a sheet.

At the start of the load script, a variable (**vPeriod**) has been created that will be tied to the variable input control. Next, configure the variable as a custom object in the sheet.

Do the following:

1. In the assets panel, click **Custom objects**.
2. Select **Qlik Dashboard bundle**, and create a **Variable input** object.
3. Enter a title for the chart object.
4. Under **Variable**, select **vPeriod** as the Name and set the object to show as a **Drop down**.
5. Under **Values**, configure the object to use dynamic values. Enter the following:
`= '1~month|2~bi-month|3~quarter|4~tertial|6~half-year'`

Next, create the results table.

Do the following:

1. Create a new table and add the following calculated dimension:
`=monthsname($(vPeriod),date)`
2. Add this measure to calculate the total sales:
`=sum(amount)`
3. Set the measure's **Number formatting** to **Money**. Click ✓ **Done editing**. You can now modify the data shown in the table by adjusting the time segment in the variable object.

This is what the results table will look like when the **tertial** option is selected:

Results table

monthsname\$(vPeriod),date)	=sum(amount)
Jan-Apr 2022	253.89
May-Aug 2022	713.58
Sep-Dec 2022	248.12

monthsstart

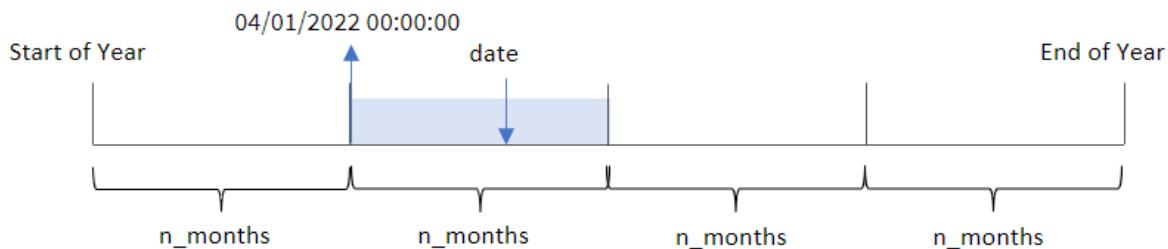
This function returns a value corresponding to the timestamp of the first millisecond of the month, bi-month, quarter, four-month period, or half-year containing a base date. It is also possible to find the timestamp for a previous or following time period. The default output format is the **DateFormat** set in the script.

Syntax:

```
MonthsStart(n_months, date[, period_no [, first_month_of_year]])
```

Return data type: dual

Diagram of monthsstart() function



The `monthsstart()` function divides the year into segments based on the `n_months` argument provided. It then evaluates what segment each date provided falls into and returns the first millisecond of that segment, in date format. The function also provides the ability to return the start timestamp from preceding or following segments, as well as redefining which is the first month of the year.

The following segments of the year are available in the function as `n_month` arguments:

Possible `n_month` arguments

Periods	Number of months
month	1
bi-month	2
quarter	3
four months	4
half-year	6

Arguments

Argument	Description
n_months	The number of months that defines the period. An integer or expression that resolves to an integer that must be one of: 1 (equivalent to the inmonth() function), 2 (bi-month), 3 (equivalent to the inquarter()function), 4 (four-month period), or 6 (half year).
date	The date or timestamp to evaluate.
period_no	The period can be offset by period_no , an integer, or expression resolving to an integer, where the value 0 indicates the period that contains base_date . Negative values in period_no indicate preceding periods and positive values indicate succeeding periods.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

When to use it

The `monthsstart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of a period that has not yet occurred. This could be used, for example, to provide an input variable to let the user calculate the total interest that has been accumulated so far in the month, quarter, or half-year.

Function examples

Example	Result
<code>monthsstart(4, '10/19/2013')</code>	Returns 09/01/2013.
<code>monthsstart(4, '10/19/2013, -1)</code>	Returns 05/01/2013.
<code>monthsstart(4, '10/19/2013', 0, 2)</code>	Returns 10/01/2013, because the start of the year becomes month 2.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `bi_monthly_start`, that groups transactions into bi-monthly segments and returns the starting timestamp of the segment for each transaction.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  monthsstart(2,date) as bi_monthly_start,
  timestamp(monthsstart(2,date)) as bi_monthly_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

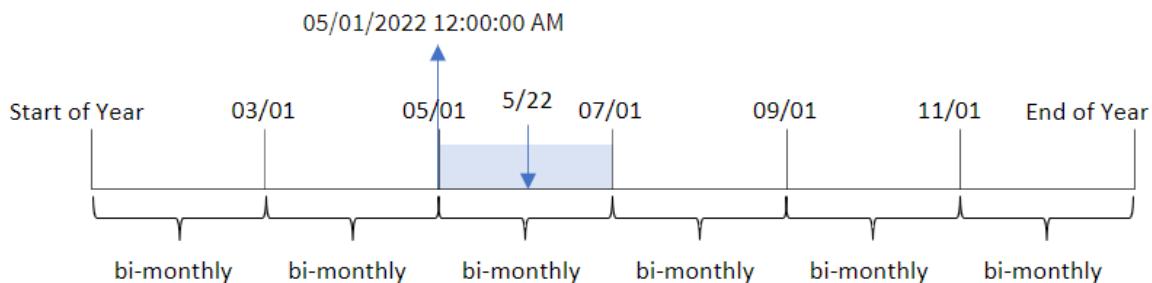
- date
- bi_monthly_start
- bi_monthly_start_timestamp

Results table

date	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	01/01/2022	1/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

The bi_monthly_start field is created in the preceding load statement by using the monthsstart() function. The first argument provided is 2, dividing the year into bi-monthly segments. The second argument identifies which field is being evaluated.

Diagram of monthsstart() function, example with no additional arguments



Transaction 8195 takes place on May 22. The `monthsstart()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. Therefore, the function returns the first millisecond of this segment, May 1, 2022 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `prev_bi_monthly_start`, that returns the first millisecond of the bi-monthly segment before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  monthsstart(2,date,-1) as prev_bi_monthly_start,
  timestamp(monthsstart(2,date,-1)) as prev_bi_monthly_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
```

```
8194,5/7/2022,40.39  
8195,5/22/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- prev_bi_monthly_start
- prev_bi_monthly_start_timestamp

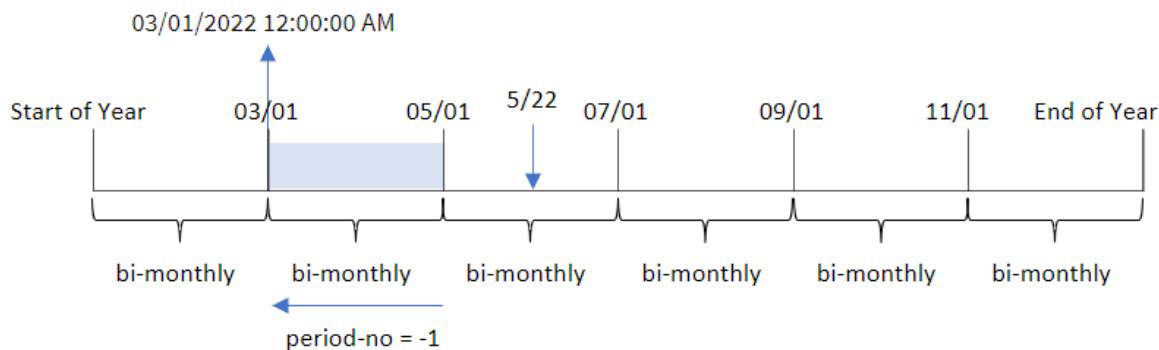
Results table

date	prev_bi_monthly_start	prev_bi_monthly_start_timestamp
2/19/2022	11/01/2021	11/1/2021 12:00:00 AM
3/7/2022	01/01/2022	1/1/2022 12:00:00 AM
3/30/2022	01/01/2022	1/1/2022 12:00:00 AM
4/5/2022	01/01/2022	1/1/2022 12:00:00 AM
4/16/2022	01/01/2022	1/1/2022 12:00:00 AM
5/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/22/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	03/01/2022	3/1/2022 12:00:00 AM
6/26/2022	03/01/2022	3/1/2022 12:00:00 AM
7/9/2022	05/01/2022	5/1/2022 12:00:00 AM
7/22/2022	05/01/2022	5/1/2022 12:00:00 AM
7/23/2022	05/01/2022	5/1/2022 12:00:00 AM
7/27/2022	05/01/2022	5/1/2022 12:00:00 AM
8/2/2022	05/01/2022	5/1/2022 12:00:00 AM

date	prev_bi_monthly_start	prev_bi_monthly_start_timestamp
8/8/2022	05/01/2022	5/1/2022 12:00:00 AM
8/19/2022	05/01/2022	5/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

By using -1 as the `period_no` argument in the `monthsstart()` function, after initially dividing a year into bi-monthly segments, the function then returns the first millisecond of the previous bi-monthly segment to when a transaction takes place.

Diagram of monthsstart() function, period_no example



Transaction 8195 occurs in the segment between May and June. Therefore, the previous bi-monthly segment was between March 1 and April 30, so the function returns the first millisecond of this segment, March 1, 2022 at 12:00:00 AM.

Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `bi_monthly_start`, that groups transactions into bi-monthly segments and returns the starting timestamp of the set for each transaction.

However, in this example, we also need to set April as the first month of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    monthsstart(2,date,0,4) as bi_monthly_start,
    timestamp(monthsstart(2,date,0,4)) as bi_monthly_start_timestamp
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,1/7/2022,17.17
  8189,1/19/2022,37.23
  8190,2/28/2022,88.27
  8191,2/5/2022,57.42
  8192,3/16/2022,53.80
  8193,4/1/2022,82.06
  8194,5/7/2022,40.39
  8195,5/16/2022,87.21
  8196,6/15/2022,95.93
  8197,6/26/2022,45.89
  8198,7/9/2022,36.23
  8199,7/22/2022,25.66
  8200,7/23/2022,82.77
  8201,7/27/2022,69.98
  8202,8/2/2022,76.11
  8203,8/8/2022,25.12
  8204,8/19/2022,46.23
  8205,9/26/2022,84.21
  8206,10/14/2022,96.24
  8207,10/29/2022,67.67
  ];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- bi_monthly_start
- bi_monthly_start_timestamp

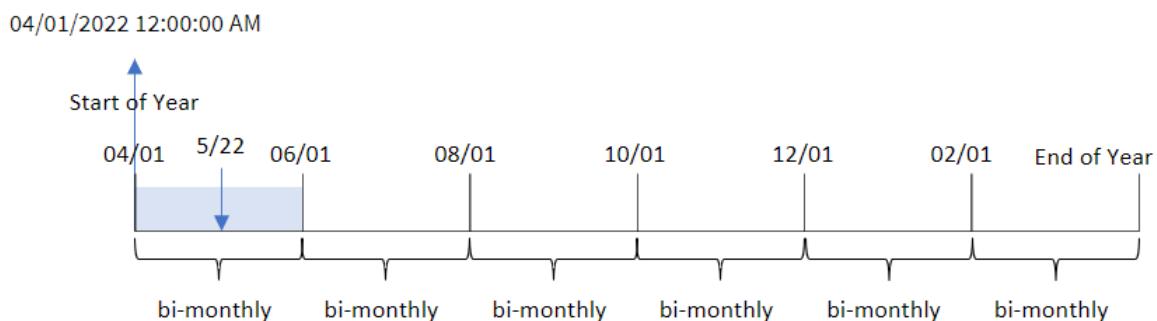
Results table

date	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	02/01/2022	2/1/2022 12:00:00 AM
3/7/2022	02/01/2022	2/1/2022 12:00:00 AM

date	bi_monthly_start	bi_monthly_start_timestamp
3/30/2022	02/01/2022	2/1/2022 12:00:00 AM
4/5/2022	04/01/2022	4/1/2022 12:00:00 AM
4/16/2022	04/01/2022	4/1/2022 12:00:00 AM
5/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/22/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

By using 4 as the `first_month_of_year` argument in the `monthsstart()` function, the function begins the year on April 1. It then divides the year into bi-monthly segments: Apr-May,Jun-Jul,Aug-Sep,Oct-Nov,Dec-Jan,Feb-Mar.

Diagram of `monthsstart()` function, `first_month_of_year` example



Transaction 8195 took place on May 22 and falls into the segment between April 1 and May 31. Therefore, the function returns the first millisecond of this segment, April 1, 2022 at 12:00:00 AM.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation which groups transactions into bi-monthly segments and returns the starting timestamp of the set for each transaction is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load  
*  
Inline  
[  
id,date,amount  
8188,2/19/2022,37.23  
8189,3/7/2022,17.17  
8190,3/30/2022,88.27  
8191,4/5/2022,57.42  
8192,4/16/2022,53.80  
8193,5/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/22/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measures:

```
=monthsstart(2,date)
```

5 Script and chart functions

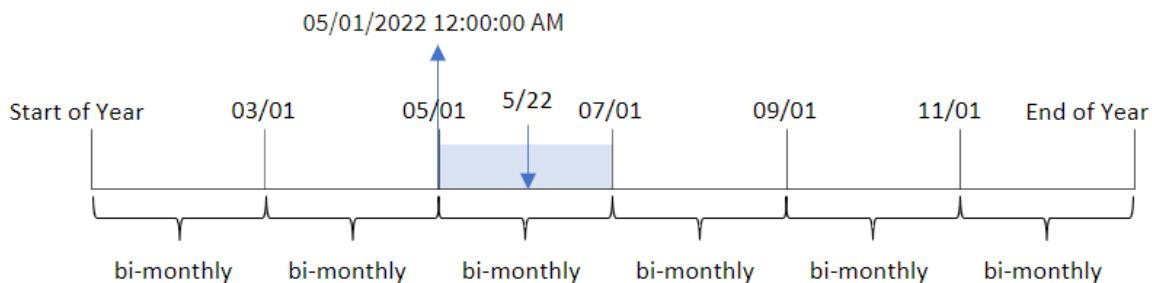
```
=timestamp(monthsstart(2,date))
```

These calculations will retrieve the starting timestamp of the bi-monthly segment in which each transaction took place.

Results table

date	=monthsstart(2,date)	=timestamp(monthsstart(2,date))
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/19/2022	01/01/2022	1/1/2021 12:00:00 AM

Diagram of `monthsstart()` function, chart object example



Transaction 8195 took place on May 22. The `monthsstart()` function initially divides the year into bi-monthly segments. Transaction 8195 falls into the segment between May 1 and June 30. Therefore, the function returns the first millisecond of this segment, 05/01/2022 12:00:00 AM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of loan balances, which is loaded into a table called `Loans`.
- Data consisting of loan IDs, the balance at the start of the month, and the simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan ID, the current interest that has been accrued on each loan for the period of their choosing. The financial year begins in January.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Loans:  
Load  
*  
Inline  
[  
loan_id,start_balance,rate  
8188,$10000.00,0.024  
8189,$15000.00,0.057  
8190,$17500.00,0.024  
8191,$21000.00,0.034  
8192,$90000.00,0.084  
];
```

Results

Load the data and open a sheet.

At the start of the load script, a variable (`vPeriod`) has been created that will be tied to the variable input control. Next, configure the variable as a custom object in the sheet.

Do the following:

1. In the assets panel, click **Custom objects**.
2. Select **Qlik Dashboard bundle**, and create a **Variable input** object.
3. Enter a title for the chart object.
4. Under **Variable**, select **vPeriod** as the Name and set the object to show as a **Drop down**.
5. Under **Values**, configure the object to use dynamic values. Enter the following:
`= '1~month|2~bi-month|3~quarter|4~tertial|6~half-year'`

Next, create the results table.

Do the following:

1. Create a new table. Add the following fields as dimensions:
 - `employee_id`
 - `employee_name`
2. Create a measure to calculate the accumulated interest:
`=start_balance*(rate*(today(1)-monthsstart($(vPeriod),today(1)))/365)`
3. Set the measure's **Number formatting** to **Money**. Click ✓ **Done editing**. You can now modify the data shown in the table by adjusting the time segment in the variable object.

This is what the results table will look like when the month period option is selected:

Results table

<code>loan_id</code>	<code>start_balance</code>	<code>=start_balance*(rate*(today(1)-monthsstart(\$(vPeriod),today(1)))/365)</code>
8188	\$10000.00	\$7.95
8189	\$15000.00	\$67.93
8190	\$17500.00	\$33.37
8191	\$21000.00	\$56.73
8192	\$90000.00	\$600.66

The `monthsstart()` function, using the user's input as its first argument and today's date as its second argument, returns the start date of the period of the user's choosing. By subtracting that result from the current date, the expression returns the number of days that have elapsed so far in this period.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate incurred for this period. The result is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this period.

monthstart

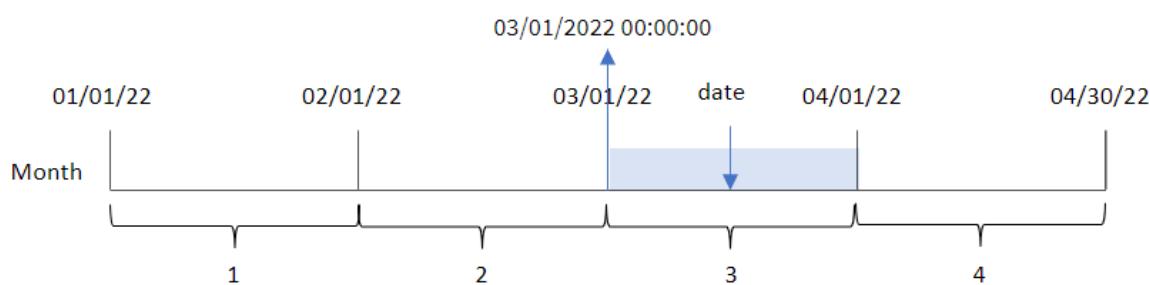
This function returns a value corresponding to a timestamp of the first millisecond of the first day of the month containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
MonthStart(date[, period_no])
```

Return data type: dual

Diagram of monthstart() function



The `monthstart()` function determines which month the date falls into. It then returns a timestamp, in date format, for the first millisecond of that month.

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, which, if 0 or omitted, indicates the month that contains date . Negative values in period_no indicate preceding months and positive values indicate succeeding months.

When to use it

The `monthstart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the month that has elapsed thus far. For example, it can be used to calculate the interest that has been accumulated in a month up to a certain date.

Function examples

Example	Result
<code>monthstart('10/19/2001')</code>	Returns 10/01/2001.
<code>monthstart('10/19/2001', -1)</code>	Returns 09/01/2001.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `start_of_month`, which returns a timestamp for the start of the month when the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    monthstart(date) as start_of_month,
    timestamp(monthstart(date)) as start_of_month_timestamp
  ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
```

```
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- start_of_month
- start_of_month_timestamp

Results table

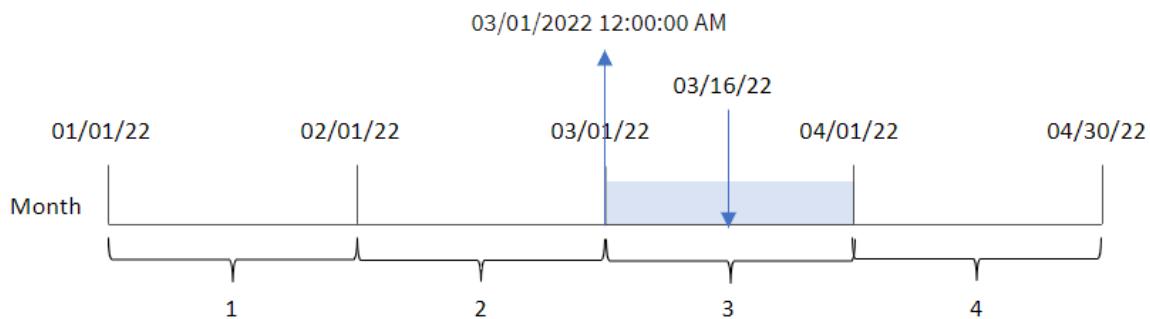
date	start_of_month	start_of_month_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	07/01/2022	6/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM

date	start_of_month	start_of_month_timestamp
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

The `start_of_month` field is created in the preceding load statement by using the `monthstart()` function and passing the `date` field as the function's argument.

The `monthstart()` function identifies which month the date value falls into, returning a timestamp for the first millisecond of that month.

Diagram of `monthstart()` function, example with no additional arguments



Transaction 8192 took place on March 16. The `monthstart()` function returns the first millisecond of that month, which is March 1 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_month_start`, which returns the timestamp for the start of the month before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```

Transactions:
Load
  *,
  monthstart(date,-1) as previous_month_start,
  timestamp(monthstart(date,-1)) as previous_month_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_month_start
- previous_month_start_timestamp

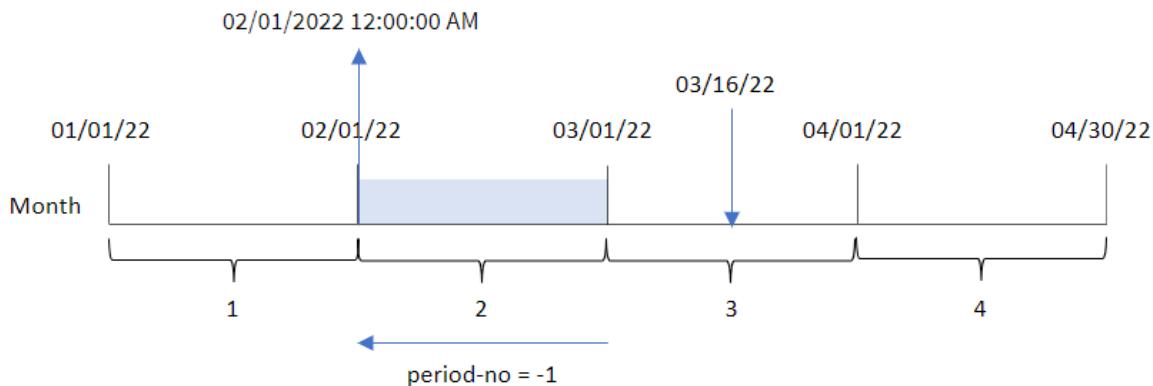
Results table

date	previous_month_start	previous_month_start_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	02/01/2022	2/1/2022 12:00:00 AM

date	previous_month_start	previous_month_start_timestamp
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

In this instance, because a period_no of -1 was used as the offset argument in the monthstart() function, the function first identifies the month that the transactions take place in. It then shifts one month prior and identifies the first millisecond of that month.

Diagram of monthstart() function, period_no example



Transaction 8192 took place on March 16. The monthstart() function identifies that the month before the transaction took place in was February. It then returns the first millisecond of that month, February 1 at 12:00:00 AM.

Example 3 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the start of the month when the transactions took place is created as a measure in a chart object of the application.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate the start date of the month that a transaction takes place in, create the following measures:

- `=monthstart(date)`
- `=timestamp(monthstart(date))`

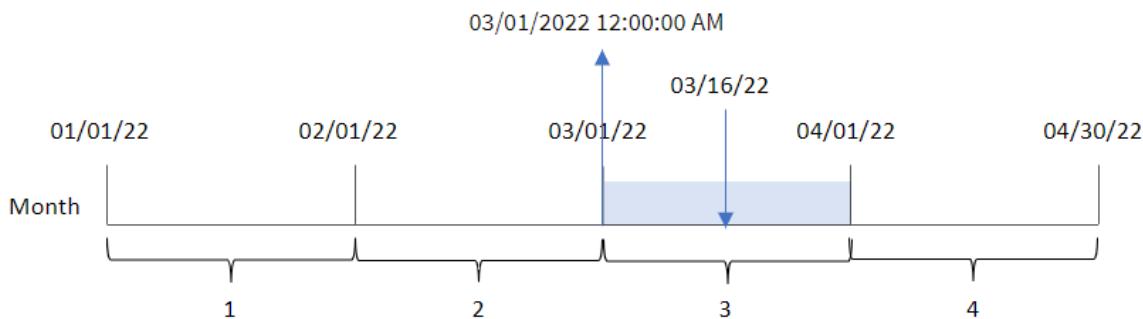
Results table

date	=monthstart(date)	=timestamp(monthstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM

The `start_of_month` measure is created in the chart object by using the `monthstart()` function and passing the date field as the function's argument.

The `monthstart()` function identifies which month the date value falls into returning a timestamp for the first millisecond of that month.

Diagram of `monthstart()` function, chart object example



Transaction 8192 took place on March 16. The `monthstart()` function identifies that the transaction took place in March and returns the first millisecond of that month, which is March 1 at 12:00:00 AM.

Example 4 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of loan balances, which is loaded into a table called `Loans`.
- Data consisting of loan IDs, the balance at the start of the month, and the simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan ID, the current interest that has been accrued on each loan in the month to date.

Load script

```
SET DateFormat='MM/DD/YYYY';

Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add these fields as dimensions:
 - loan_id
 - start_balance
2. Next, create a measure to calculate the accumulated interest:
 $=start_balance*(rate*(today(1)-monthstart(today(1)))/365)$
3. Set the measure's **Number formatting** to **Money**.

Results table

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	\$10000.00	\$16.44
8189	\$15000.00	\$58.56
8190	\$17500.00	\$28.77
8191	\$21000.00	\$48.90
8192	\$90000.00	\$517.81

The `monthstart()` function, using today's date as its only argument, returns the start date of the current month. By subtracting that result from the current date, the expression returns the number of days that have elapsed so far this month.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate incurred for this period. The result is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this month.

networkdays

The **networkdays** function returns the number of working days (Monday-Friday) between and including **start_date** and **end_date** taking into account any optionally listed **holiday**.

Syntax:

```
networkdays (start_date, end_date [, holiday])
```

Return data type: integer

Calendar diagram displaying date range returned by networkdays function

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10 start_date	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26 end_date	27
28	29	30	31			

The networkdays function has the following limitations:

- There is no method to modify workdays. In other words, there is no way to modify the function for regions or situations that involve anything other than working Monday to Friday.
- The **holiday** parameter must be a string constant. Expressions are not accepted.

Arguments

Argument	Description
start_date	The start date to evaluate.
end_date	The end date to evaluate.
holiday	Holiday periods to exclude from working days. A holiday is stated as a string constant date. You can specify multiple holiday dates, separated by commas. Example: '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'

When to use it

The networkdays() function is commonly used as part of an expression when the user would like the calculation to use the number of working week days that occur between two dates. For example, if a user would like to calculate the total wages that will be earned by an employee on a PAYE (pay-as-you-earn) contract.

Function examples

Example	Result
<code>networkdays ('12/19/2013', '01/07/2014')</code>	Returns 14. This example does not take holidays into account.
<code>networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013')</code>	Returns 12. This example takes the holiday 12/25/2013 to 12/26/2013 into account.
<code>networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014')</code>	Returns 10. This example takes two holiday periods into account.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing project IDs, their start dates, and their end dates. This information is loaded into a table called `Projects`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of an additional field, `net_work_days`, to calculate the number of working days involved in each project.

Load script

```
SET DateFormat='MM/DD/YYYY';

Projects:
  Load
    *,
    networkdays(start_date,end_date) as net_work_days
```

```
;  
Load  
id,  
start_date,  
end_date  
Inline  
[  
id,start_date,end_date  
1,01/01/2022,01/18/2022  
2,02/10/2022,02/17/2022  
3,05/17/2022,07/05/2022  
4,06/01/2022,06/12/2022  
5,08/10/2022,08/26/2022  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- end_date
- net_work_days

Results table

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	13

Because there are no scheduled holidays (this would have been present in the third argument of the networkdays() function), the function subtracts the start_date from the end_date, as well as all weekends, to calculate the number of working days between the two dates.

Calendar diagram highlighting work days for project 5 (no holidays)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

The calendar above visually outlines the project with id of 5. Project 5 begins on Wednesday, August 10, 2022 and ends on August 26, 2022. With all Saturdays and Sundays ignored, there are 13 working days between, and including, these two dates.

Example 2 – Single holiday

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario from the previous example.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of an additional field, `net_work_days`, to calculate the number of working days involved in each project.

In this example, there is a one-day holiday scheduled on August 19, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Projects:

```
Load
  *,
  networkdays(start_date,end_date,'08/19/2022') as net_work_days
;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- end_date
- net_work_days

Results table

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

The single scheduled holiday is entered as the third argument in the `networkdays()` function.

Calendar diagram highlighting work days for project 5 (single holiday)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

The calendar above visually outlines project 5, demonstrating this adjustment to include the holiday. This holiday occurs during project 5 on Friday, August 19, 2022. As a result, the total net_work_days value for project 5 decreases by one day, from 13 to 12 days.

Example 3 – Multiple holidays

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario from the first example.
- The date field provided in the dateFormat system variable (MM/DD/YYYY) format.
- The creation of an additional field, net_work_days, to calculate the number of working days involved in each project.

However, in this example, there are four holidays scheduled from August 18 to August 21, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Projects:
  Load
    *,
    networkdays(start_date,end_date,'08/18/2022','08/19/2022','08/20/2022','08/21/2022')
  as net_work_days
  ;
  Load
  id,
  start_date,
  end_date
  Inline
  [
  id,start_date,end_date
  1,01/01/2022,01/18/2022
  2,02/10/2022,02/17/2022
  3,05/17/2022,07/05/2022
  4,06/01/2022,06/12/2022
  5,08/10/2022,08/26/2022
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- end_date
- net_work_days

Results table

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	11

The four scheduled holidays are entered as a comma separated list, from the third argument onwards in the `networkdays()` function.

Calendar diagram highlighting work days for project 5 (multiple holidays)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18 Holiday	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

The calendar above visually outlines project 5, demonstrating this adjustment to include these holidays. This period of scheduled holidays occurs during project 5, with two of the days occurring on a Thursday and Friday. As a result, the total `net_work_days` value for project 5 decreases from 13 to 11 days.

Example 4 – Single holiday

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario from the first example.
- The date field provided in the `dateFormat` system variable (MM/DD/YYYY) format.

There is a one-day holiday scheduled on August 19, 2022.

However, in this example, the unchanged dataset is loaded into the application. The `net_work_days` field is calculated as a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Projects:  
Load  
id,  
start_date,  
end_date  
Inline  
[  
id,start_date,end_date  
1,01/01/2022,01/18/2022  
2,02/10/2022,02/17/2022  
3,05/17/2022,07/05/2022  
4,06/01/2022,06/12/2022  
5,08/10/2022,08/26/2022  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- start_date
- end_date

Create the following measure:

```
= networkdays(start_date,end_date,'08/19/2022')
```

Results table

id	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

The single scheduled holiday is entered as the third argument in the networkdays() function.

Calendar diagram showing net work days with single holiday (chart object)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

The calendar above visually outlines project 5, demonstrating this adjustment to include the holiday. This holiday occurs during project 5 on Friday, August 19, 2022. As a result, the total `net_work_days` value for project 5 decreases by one day, from 13 to 12 days.

NOW

This function returns a timestamp of the current time. The function returns values in the **TimeStamp** system variable format. The default **timer_mode** value is 1.

Syntax:

```
now([ timer_mode])
```

Return data type: dual

The `now()` function can be used either in the load script or in chart objects.

Arguments

Argument	Description
timer_mode	<p>Can have the following values:</p> <ul style="list-style-type: none"> 0 (time at last finished data load) 1 (time at function call) 2 (time when the app was opened) <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>If you use the function in a data load script, timer_mode=0 will result in the time of the last finished data load, while timer_mode=1 will give the time of the function call in the current data load.</i> </div>

When to use it

The now() function is commonly used as a component within an expression. For example, it can be used to calculate the time remaining in a product's lifecycle. The now() function would be used instead of the today() function when the expression requires the use of a fraction of a day.

The following table provides an explanation of the result returned by the now() function, given different values for the timer_mode argument:

Function examples

timer_mode value	Result if used in load script	Result if used in chart object
0	Returns a timestamp, in the <code>Timestamp</code> system variable format, of the last successful data reload prior to the latest data reload.	Returns a timestamp, in the <code>Timestamp</code> system variable format, for the latest data reload.
1	Returns a timestamp, in the <code>Timestamp</code> system variable format, for the latest data reload.	Returns a timestamp, in the <code>Timestamp</code> system variable format, of the function call.
2	Returns a timestamp, in the <code>Timestamp</code> system variable format, for when the user's session in the application began. This will not be updated unless the user reloads the script.	Returns the timestamp, in the <code>Timestamp</code> system variable format, for when the user's session in the application began. This will be refreshed once a new session begins or the data in the application is reloaded.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may

be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Generation of objects using load script

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

This example creates three variables using the now() function. Each variable uses one of the timer_mode options to demonstrate their effect.

For the variables to demonstrate their purpose, reload the script and then, after a short period of time, reload the script a second time. This will result in the now(0) and now(1) variables showing different values, thereby correctly demonstrating their purpose.

Load script

```
LET vPreviousDataLoad = now(0);  
LET vCurrentDataLoad = now(1);  
LET vApplicationOpened = now(2);
```

Results

Once the data has been loaded for a second time, create three textboxes using the directions below.

First, create a textbox for the data which has previously been loaded.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
`=vPreviousDataLoad`
3. Under **Appearance**, select **Show titles** and add the title 'Previous Reload Time' to the object.

Next, create a textbox for the data which is currently being loaded.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
=vCurrentDataLoad
3. Under **Appearance**, select **Show titles** and add the title 'Current Reload Time' to the object.

Create a final textbox to show when the user's session in the application was started.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
=vApplicationopened
3. Under **Appearance**, select **Show titles** and add the title 'User Session Started' to the object.

now() load script variables

Previous Reload Time	Current Reload Time	User Session Began
6/22/2022 8:54:03 AM	6/22/2022 9:02:08 AM	6/22/2022 8:40:40 AM

The above image shows example values for each of the created variables. For example, the values could be as follows:

- Previous Reload Time: 6/22/2022 8:54:03 AM
- Current Reload Time: 6/22/2022 9:02:08 AM
- User Session Began: 6/22/2022 8:40:40 AM

Example 2 – Generation of objects without load script

Load script and chart expression

Overview

In this example, you will create three chart objects using the now() function, without loading any variables or data into the application. Each chart object uses one of the timer_mode options to demonstrate their effect.

There is no load script for this example.

Do the following:

1. Open the Data load editor.
2. Without changing the existing load script, click **Load data**.
3. After a short period of time, load the script a second time.

Results

Once the data has been loaded for a second time, create three textboxes.

First, create a textbox for the latest data reload.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=now(0)`
3. Under **Appearance**, select **Show titles** and add the title 'Latest Data Reload' to the object.

Next, create a textbox to show the current time.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=now(1)`
3. Under **Appearance**, select **Show titles** and add the title 'Current Time' to the object.

Create a final textbox to show when the user's session in the application was started.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=now(2)`
3. Under **Appearance**, select **Show titles** and add the title 'User Session Began' to the object.

now() chart object examples

Latest Data Reload 6/22/2022 9:02:08 AM	Current Time 6/22/2022 9:25:16 AM	User Session Began 6/22/2022 8:40:40 AM
---	---	---

The above image shows example values for each of the created objects. For example, the values could be as follows:

- Latest Data Reload: 6/22/2022 9:02:08 AM
- Current Time: 6/22/2022 9:25:16 AM
- User Session Began: 6/22/2022 8:40:40 AM

The 'Latest Data Reload' chart object uses a `timer_mode` value of 0. This returns the timestamp for the last time the data was successfully reloaded.

The 'Current Time' chart object uses a `timer_mode` value of 1. This returns the current time according to the system clock. If the sheet or object is refreshed, this value will be updated.

The 'User Session Began' chart object uses a `timer_mode` value of 2. This returns the timestamp for when the application was opened, and the user's session began.

Example 3 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset consisting of inventory for a cryptocurrency mining operation, which is loaded into a table called `Inventory`.
- Data with the following fields: `id`, `purchase_date`, and `wph` (watts per hour).

The user would like a table that displays, by `id`, the total cost each mining rig has incurred in the month so far, in terms of power consumption.

This value should update whenever the chart object is refreshed. The current cost of electricity is \$0.0678 per kWh.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Inventory:  
Load  
*  
Inline  
[  
id,purchase_date,wph  
8188,1/7/2022,1123  
8189,1/19/2022,1432  
8190,2/28/2022,1227  
8191,2/5/2022,1322  
8192,3/16/2022,1273  
8193,4/1/2022,1123  
8194,5/7/2022,1342  
8195,5/16/2022,2342  
8196,6/15/2022,1231  
8197,6/26/2022,1231  
8198,7/9/2022,1123  
8199,7/22/2022,1212  
8200,7/23/2022,1223  
8201,7/27/2022,1232  
8202,8/2/2022,1232  
8203,8/8/2022,1211
```

```
8204,8/19/2022,1243  
8205,9/26/2022,1322  
8206,10/14/2022,1133  
8207,10/29/2022,1231  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: `id`.

Create the following measure:

```
= (now(1)-monthstart(now(1)))*24*wph/1000*0.0678
```

If the chart object was refreshed at 6/22/2022 10:39:05 AM, it would return the following results:

Results table

id	= (now(1)-monthstart(now(1)))*24*wph/1000*0.0678
8188	\$39.18
8189	\$49.97
8190	\$42.81
8191	\$46.13
8192	\$44.42
8193	\$39.18
8194	\$46.83
8195	\$81.72
8196	\$42.95
8197	\$42.95
8198	\$39.18
8199	\$42.29
8200	\$42.67
8201	\$42.99
8202	\$42.99
8203	\$42.25
8204	\$43.37
8205	\$46.13
8206	\$39.53

The user would like the object results to refresh every time the object is refreshed. Therefore, the `timer_mode` argument of supplied for instances of the `now()` function in the expression. The timestamp for the start of the month, identified by using the `now()` function as the timestamp argument in the `monthstart()` function, is subtracted from the current time which is identified by the `now()` function. This provides the total amount of time that has elapsed so far this month, in days.

This value is multiplied by 24 (the number of hours in a day) and then by the value in the `wph` field.

To convert from watts per hour to kilowatts per hour, the result is divided by 1000 before finally being multiplied by the `kWH` rate supplied.

quarterend

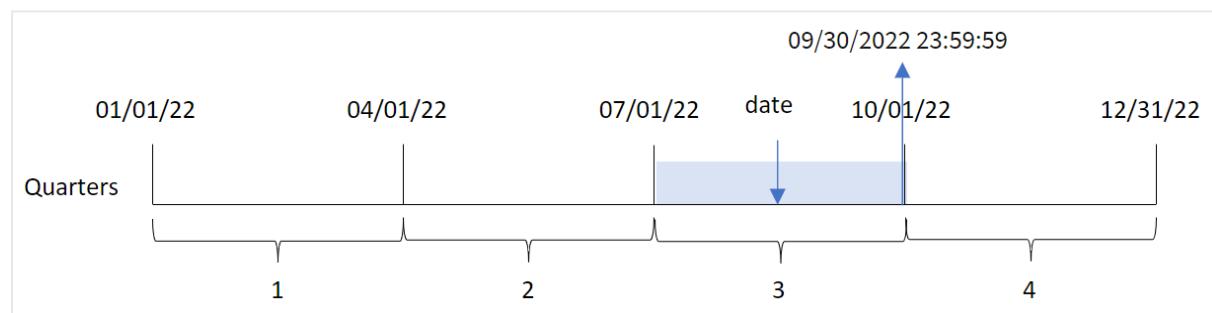
This function returns a value corresponding to a timestamp of the last millisecond of the quarter containing `date`. The default output format will be the **DateFormat** set in the script.

Syntax:

```
QuarterEnd(date[, period_no[, first_month_of_year]])
```

Return data type: dual

Diagram of the quarterend() function



The `quarterend()` function determines which quarter the date falls into. It then returns a timestamp, in date format, for the last millisecond of the last month of that quarter. The first month of the year is, by default, January. However, you can change which month is set as first by using the `first_month_of_year` argument in the `quarterend()` function.



The `quarterend()` function does not consider the `FirstMonthofYear` system variable. The year begins on January 1 unless the `first_month_of_year` argument is used to change it.

When to use it

The `quarterend()` function is commonly used as part of an expression when you would like the calculation to use the fraction of the quarter that has not yet occurred. For example, if you want to calculate the total interest not yet incurred during the quarter.

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the quarter which contains date . Negative values in period_no indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

You can use the following values to set the first month of year in the **first_month_of_year** argument:

first_month_of_year values

Month	Value
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
quarterend('10/29/2005')	Returns 12/31/2005 23:59:59.
quarterend('10/29/2005', -1)	Returns 09/30/2005 23:59:59.
quarterend('10/29/2005', 0, 3)	Returns 11/30/2005 23:59:59.

Example 1 - Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the following:
 - The `quarterend()` function that is set as the ‘end_of_quarter’ field and returns a timestamp for the end of the quarter when the transactions took place.
 - The `timestamp()` function that is set as the ‘end_of_quarter_timestamp’ field and returns the exact timestamp of the end of the selected quarter.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    quarterend(date) as end_of_quarter,
    timestamp(quarterend(date)) as end_of_quarter_timestamp
  ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
```

```
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `end_of_quarter`
- `end_of_quarter_timestamp`

Results table

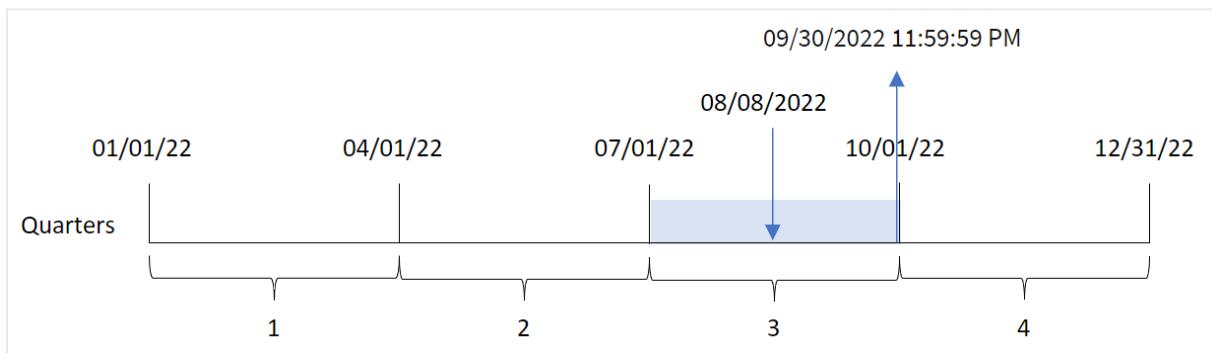
id	date	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM

id	date	end_of_quarter	end_of_quarter_timestamp
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

The ‘end_of_quarter’ field is created in the preceding load statement by using the `quarterend()` function and passing the date field as the function’s argument.

The `quarterend()` function initially identifies which quarter the date value falls into and then returns a timestamp for the last millisecond of that quarter.

Diagram of the `quarterend()` function with the quarter end of transaction 8203 identified



Transaction 8203 took place on August 8. The `quarterend()` function identifies that the transaction took place in the third quarter, and returns the last millisecond of that quarter, which is September 30 at 11:59:59 PM.

Example 2 - period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the following:
 - The `quarterend()` function that is set as the ‘previous_quarter_end’ field and returns a timestamp for the end of the quarter before the transaction took place.
 - The `timestamp()` function that is set as the ‘previous_end_of_quarter_timestamp’ field and returns the exact timestamp of the end of the quarter before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```

Load
  *,
  quarterend(date, -1) as previous_quarter_end,
  timestamp(quarterend(date, -1)) as previous_quarter_end_timestamp
;

Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- previous_quarter_end
- previous_quarter_end_timestamp

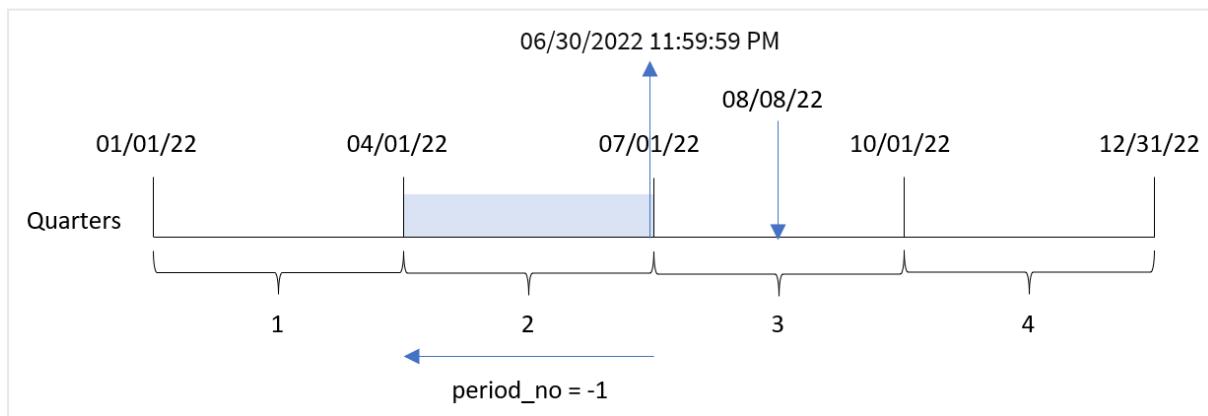
Results table

id	date	previous_quarter_end	previous_quarter_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM

id	date	previous_quarter_end	previous_quarter_end_timestamp
8192	3/16/2022	12/31/2021	12/31/2021 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8195	5/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8196	6/15/2022	03/31/2022	3/31/2022 11:59:59 PM
8197	6/26/2022	03/31/2022	3/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

Because a period_no of -1 is used as the offset argument in the quarterend() function, the function first identifies the quarter that the transactions take place in. It then shifts one quarter prior and identifies the final millisecond of that quarter.

Diagram of the quarterend() function with a period_no of -1



Transaction 8203 took place on August 8. The quarterend() function identifies that the quarter before the transaction took place was between April 1 and June 30. The function then returns the final millisecond of that quarter, June 30 at 11:59:59 PM.

Example 3 - first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions in 2022 which is loaded into a table called ‘Transactions’.
- A preceding load which contains the following:
 - The `quarterend()` function that is set as the ‘`end_of_quarter`’ field and returns a timestamp for the end of the quarter when the transactions took place.
 - The `timestamp()` function that is set as the ‘`end_of_quarter_timestamp`’ field and returns the exact timestamp of the end of the selected quarter.

However, in this example, the company policy is that the financial year begins on March 1.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  quarterend(date, 0, 3) as end_of_quarter,
  timestamp(quarterend(date, 0, 3)) as end_of_quarter_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
```

```
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

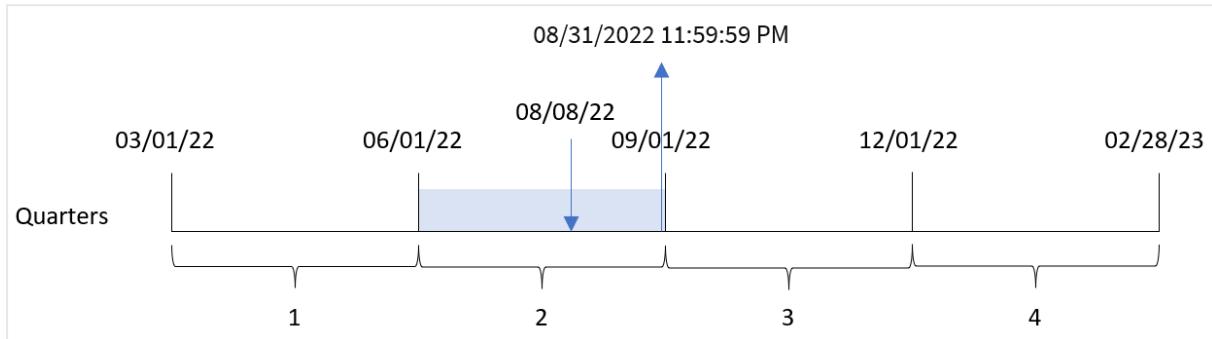
Results

Results table

id	date	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	08/31/2022	8/31/2022 11:59:59 PM
8197	6/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	11/30/2022	11/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

Because the `first_month_of_year` argument of 3 is used in the `quarterend()` function, the start of the year moves from January 1 to March 1.

Diagram of the quarterend() function with March as the first month of the year



Transaction 8203 took place on August 8. Because the beginning of the year is March 1, the quarters in the year occur between Mar-May, Jun-Aug, Sep-Nov, and Dec-Feb.

The quarterend() function identifies that the transaction took place in the quarter between the start of June and of August and returns the last millisecond of that quarter, which is August 31 at 11:59:59 PM.

Example 4 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that returns a timestamp for the end of the quarter when the transactions took place is created as a measure in a chart in the app.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**

To calculate the end date of the quarter that a transaction takes place in, create the following measures:

- **=quarterend(date)**
- **=timestamp(quarterend(date))**

Results table

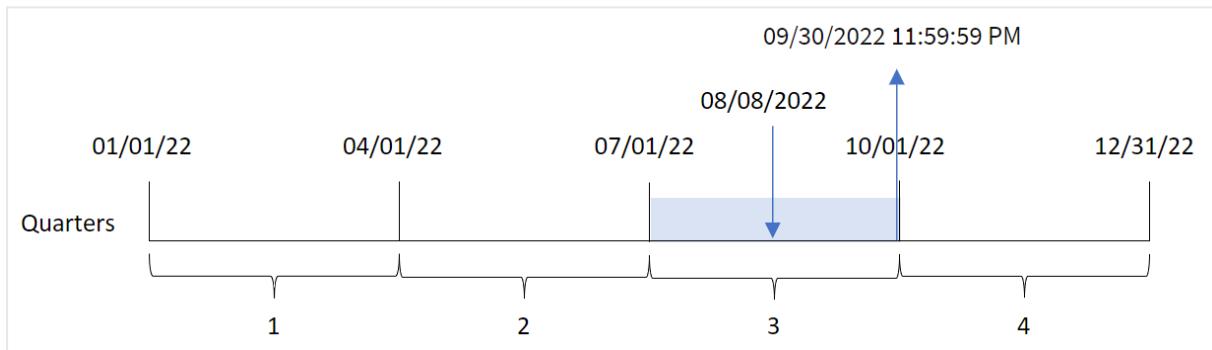
id	date	=quarterend(date)	=timestamp(quarterend(date))
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM

id	date	=quarterend(date)	=timestamp(quarterend(date))
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

The ‘end_of_quarter’ field is created in the preceding load statement by using the quarterend() function and passing the date field as the function’s argument.

The quarterend() function initially identifies which quarter the date value falls into and then returns a timestamp for the last millisecond of that quarter.

Diagram of the quarterend() function with the quarter end of transaction 8203 identified



Transaction 8203 took place on August 8. The quarterend() function identifies that the transaction took place in the third quarter, and returns the last millisecond of that quarter, which is September 30 at 11:59:59 PM.

Example 5 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset is loaded into a table called ‘Employee_Expenses’. The table contains the following fields:
 - Employee IDs
 - Employee names
 - The average daily expense claims of each employee.

The end user would like a chart object that displays, by employee id and employee name, the estimated expense claims still to be incurred for the remainder of the quarter. The financial year begins in January.

Load script

```
Employee_Expenses:  
Load  
*  
Inline  
[  
employee_id,employee_name,avg_daily_claim  
182,Mark, $15  
183,Deryck, $12.5  
184,Dexter, $12.5  
185,Sydney,$27  
186,Agatha,$18  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- employee_id
- employee_name

To calculate the accumulated interest, create the following measure:

- =(quarterend(today(1))-today(1))*avg_daily_claim

Set the measure's **Number Formatting** to **Money**.

Results table

employee_id	employee_name	=quarterend(today(1))-today(1))*avg_daily_claim
182	Mark	\$480.00
183	Deryck	\$400.00
184	Dexter	\$400.00
185	Sydney	\$864.00
186	Agatha	\$576.00

The quarterend() function uses today's date as its only argument and returns the end date of the current month. Then, it subtracts today's date from the year end date, and the expression returns the number of days that remain this month.

This value is then multiplied by the average daily expense claim of each employee to calculate the estimated value of claims each employee is expected to make in the remaining quarter.

quartername

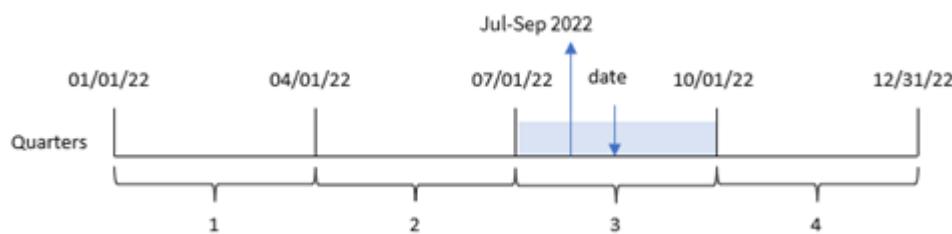
This function returns a display value showing the months of the quarter (formatted according to the **MonthNames** script variable) and year with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the quarter.

Syntax:

```
QuarterName(date[, period_no[, first_month_of_year]])
```

Return data type: dual

Diagram of quartername() function



The `quartername()` function determines which quarter the date falls into. It then returns a value showing the start-end months of this quarter as well as the year. The underlying numeric value of this result is the first millisecond of the quarter.

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the quarter which contains date . Negative values in period_no indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

When to use it

The `quartername()` function is useful when you would like to compare aggregations by quarter. For example, if you would like to see the total sales of products by quarter.

This function could be used in the load script to create a field in a Master Calendar table. Alternatively, it could be used directly in a chart as a calculated dimension.

These examples use the date format MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement at the top of your data load script. Change the format in the examples to suit your requirements.

Function examples

Example	Result
quartername('10/29/2013')	Returns Oct-Dec 2013.
quartername('10/29/2013', -1)	Returns Jul-Sep 2013.
quartername('10/29/2013', 0, 3)	Returns Sep-Nov 2013.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – date with no additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `transaction_quarter`, which returns the quarter in which the transactions took place.

Add your other text here, as needed, with lists etc.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
  *,
  quartername(date) as transaction_quarter
;
Load
```

```
*  
Inline  
[  
id,date,amount  
8188,1/7/2022,17.17  
8189,1/19/2022,37.23  
8190,2/28/2022,88.27  
8191,2/5/2022,57.42  
8192,3/16/2022,53.80  
8193,4/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- transaction_quarter

Results table

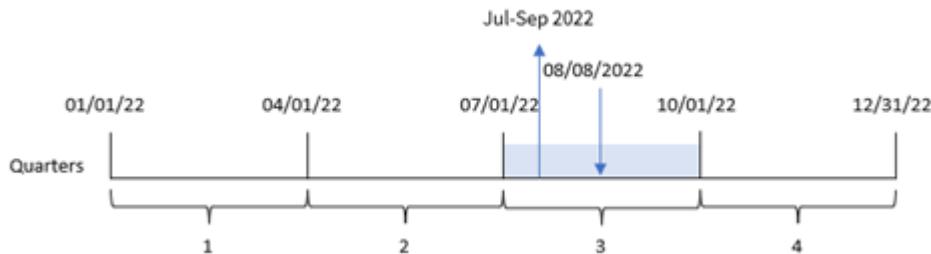
date	transaction_quarter
1/7/2022	Jan-Mar 2022
1/19/2022	Jan-Mar 2022
2/5/2022	Jan-Mar 2022
2/28/2022	Jan-Mar 2022
3/16/2022	Jan-Mar 2022
4/1/2022	Apr-Jun 2022
5/7/2022	Apr-Jun 2022
5/16/2022	Apr-Jun 2022
6/15/2022	Apr-Jun 2022
6/26/2022	Apr-Jun 2022

date	transaction_quarter
7/9/2022	Jul-Sep 2022
7/22/2022	Jul-Sep 2022
7/23/2022	Jul-Sep 2022
7/27/2022	Jul-Sep 2022
8/2/2022	Jul-Sep 2022
8/8/2022	Jul-Sep 2022
8/19/2022	Jul-Sep 2022
9/26/2022	Jul-Sep 2022
10/14/2022	Oct-Dec 2022
10/29/2022	Oct-Dec 2022

The `transaction_quarter` field is created in the preceding load statement by using the `quartername()` function and passing the `date` field as the function's argument.

The `quartername()` function initially identifies the quarter into which the date value falls. It then returns a value showing the start-end months of this quarter, as well as the year.

Diagram of `quartername()` function, example with no additional arguments



Transaction 8203 took place on August 8, 2022. The `quartername()` function identifies that the transaction took place in the third quarter, and therefore returns Jul-Sep 2022. The months are displayed in the same format as the `MonthNames` system variable.

Example 2 – date with period_no argument

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, previous_quarter, that returns the previous quarter to when the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
  *,
  quartername(date,-1) as previous_quarter
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

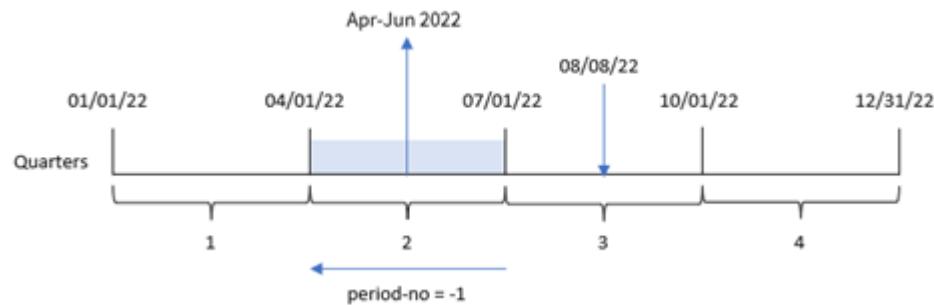
- date
- previous_quarter

Results table

date	previous_quarter
1/7/2022	Oct-Dec 2021
1/19/2022	Oct-Dec 2021
2/5/2022	Oct-Dec 2021
2/28/2022	Oct-Dec 2021
3/16/2022	Oct-Dec 2021
4/1/2022	Jan-Mar 2022
5/7/2022	Jan-Mar 2022
5/16/2022	Jan-Mar 2022
6/15/2022	Jan-Mar 2022
6/26/2022	Jan-Mar 2022
7/9/2022	Apr-Jun 2022
7/22/2022	Apr-Jun 2022
7/23/2022	Apr-Jun 2022
7/27/2022	Apr-Jun 2022
8/2/2022	Apr-Jun 2022
8/8/2022	Apr-Jun 2022
8/19/2022	Apr-Jun 2022
9/26/2022	Apr-Jun 2022
10/14/2022	Jul-Sep 2022
10/29/2022	Jul-Sep 2022

In this instance, because a `period_no` of -1 was used as the offset argument in the `quartername()` function, the function first identifies that the transactions took place in the third quarter. It then shifts one quarter prior and returns a value showing the start-end months of this quarter, as well as the year.

Diagram of quartername() function, period_no example



Transaction 8203 took place on August 8. The quartername() function identifies that the quarter before the transaction took place was between April 1 and June 30. Therefore, it returns Apr-Jun 2022.

Example 3 – date with first_week_day argument

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, we need to set March 1 as the beginning of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
  *,
  quartername(date,0,3) as transaction_quarter
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- transaction_quarter

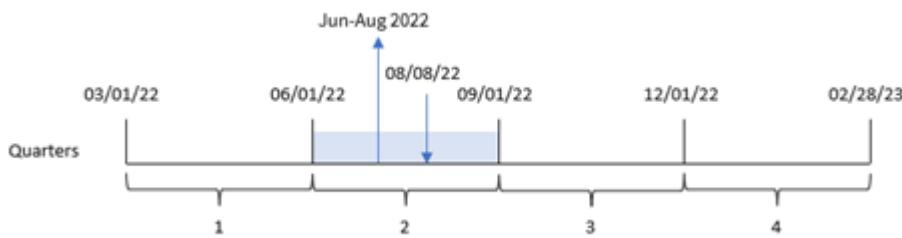
Results table

date	transaction_quarter
1/7/2022	Dec-Feb 2021
1/19/2022	Dec-Feb 2021
2/5/2022	Dec-Feb 2021
2/28/2022	Dec-Feb 2021
3/16/2022	Mar-May 2022
4/1/2022	Mar-May 2022
5/7/2022	Mar-May 2022
5/16/2022	Mar-May 2022
6/15/2022	Jun-Aug 2022
6/26/2022	Jun-Aug 2022
7/9/2022	Jun-Aug 2022
7/22/2022	Jun-Aug 2022
7/23/2022	Jun-Aug 2022
7/27/2022	Jun-Aug 2022
8/2/2022	Jun-Aug 2022
8/8/2022	Jun-Aug 2022
8/19/2022	Jun-Aug 2022
9/26/2022	Sep-Nov 2022

date	transaction_quarter
10/14/2022	Sep-Nov 2022
10/29/2022	Sep-Nov 2022

In this instance, because the `first_month_of_year` argument of 3 is used in the `quartername()` function, the start of the year moves from January 1 to March 1. Therefore, the quarters in the year are separated into March-May, June-August, September-November and December-February.

Diagram of quartername() function, first_week_day example



Transaction 8203 took place on August 8. The `quartername()` function identifies that the transaction took place in the second quarter, between the start of June and the end of August. Therefore, it returns Jun-Aug 2022.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the end of the quarter when the transactions took place is created as a measure in a chart object of the application.

Load script

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
```

```
8193,4/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Create the following measure:

```
=quartername(date)
```

Results table

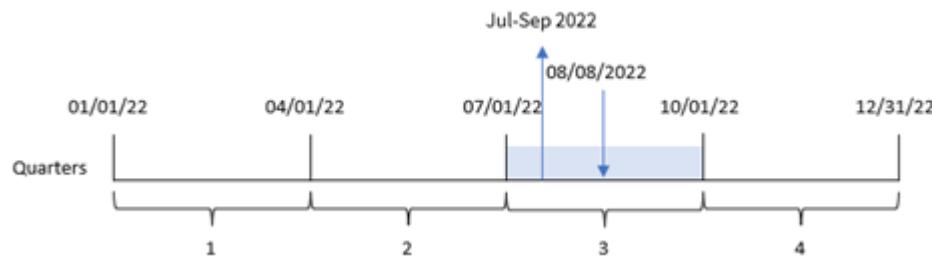
date	=quartername(date)
1/7/2022	Jan-Mar 2022
1/19/2022	Jan-Mar 2022
2/5/2022	Jan-Mar 2022
2/28/2022	Jan-Mar 2022
3/16/2022	Jan-Mar 2022
4/1/2022	Apr-Jun 2022
5/7/2022	Apr-Jun 2022
5/16/2022	Apr-Jun 2022
6/15/2022	Apr-Jun 2022
6/26/2022	Apr-Jun 2022
7/9/2022	Jul-Sep 2022
7/22/2022	Jul-Sep 2022
7/23/2022	Jul-Sep 2022
7/27/2022	Jul-Sep 2022
8/2/2022	Jul-Sep 2022

date	=quartername(date)
8/8/2022	Jul-Sep 2022
8/19/2022	Jul-Sep 2022
9/26/2022	Jul-Sep 2022
10/14/2022	Oct-Dec 2022
10/29/2022	Oct-Dec 2022

The `transaction_quarter` measure is created in the chart object by using the `quartername()` function and passing the date field as the function's argument.

The `quartername()` function initially identifies the quarter into which the date value falls. It then returns a value showing the start-end months of this quarter, as well as the year.

Diagram of quartername() function, chart object example



Transaction 8203 took place on August 8, 2022. The `quartername()` function identifies that the transaction took place in the third quarter, and therefore returns Jul-Sep 2022. The months are displayed in the same format as the `MonthNames` system variable.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

The end user would like a chart object that presents the total sales by quarter for the transactions. This could be achieved even when this dimension is not available in the data model, using the `quartername()` function as a calculated dimension in the chart.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create a calculated dimension using the following expression:
`=quartername(date)`
3. Next, calculate total sales using the following aggregation measure:
`=sum(amount)`
4. Set the measure's **Number formatting** to **Money**.

Results table

<code>=quartername(date)</code>	<code>=sum(amount)</code>
Jul-Sep 2022	\$446.31
Apr-Jun 2022	\$351.48
Jan-Mar 2022	\$253.89
Oct-Dec 2022	\$163.91

quarterstart

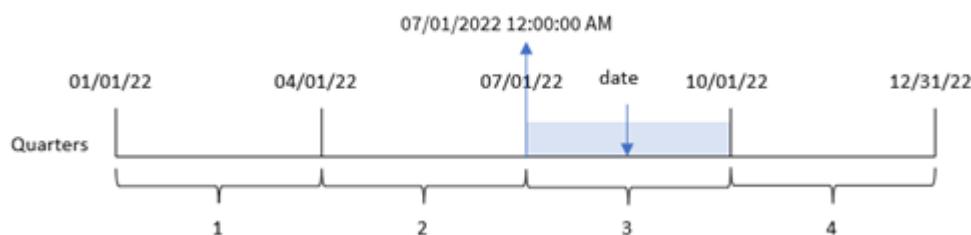
This function returns a value corresponding to a timestamp of the first millisecond of the quarter containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
QuarterStart(date[, period_no[, first_month_of_year]])
```

Return data type: dual

Diagram of quarterstart() function



The `quarterstart()` function determines which quarter the date falls into. It then returns a timestamp, in date format, for the first millisecond of the first month of that quarter.

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the quarter which contains date . Negative values in period_no indicate preceding quarters and positive values indicate succeeding quarters.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

When to use it

The `quarterstart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the quarter that has elapsed thus far. For example, it could be used if a user would like to calculate the interest that has been accumulated in a quarter to date.

Function examples

Example	Result
<code>quarterstart('10/29/2005')</code>	Returns 10/01/2005.
<code>quarterstart('10/29/2005', -1)</code>	Returns 07/01/2005.
<code>quarterstart('10/29/2005', 0, 3)</code>	Returns 09/01/2005.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `start_of_quarter`, which returns a timestamp for the start of the quarter when the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    quarterstart(date) as start_of_quarter,
    timestamp(quarterstart(date)) as start_of_quarter_timestamp
  ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
```

```
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- start_of_quarter
- start_of_quarter_timestamp

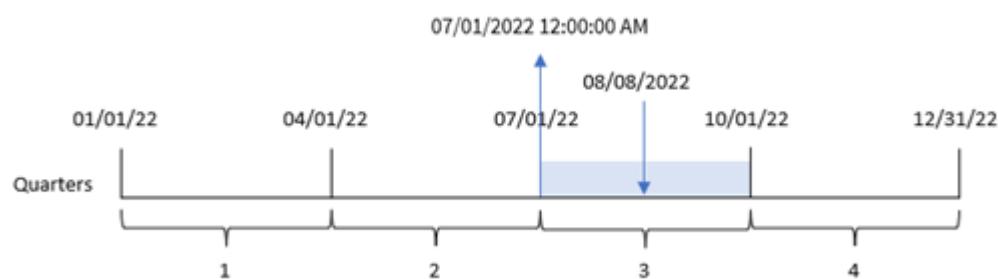
Results table

date	start_of_quarter	start_of_quarter_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2021 12:00:00 AM
5/7/2022	04/01/2022	4/1/2021 12:00:00 AM
5/16/2022	04/01/2022	4/1/2021 12:00:00 AM
6/15/2022	04/01/2022	4/1/2021 12:00:00 AM
6/26/2022	04/01/2022	4/1/2021 12:00:00 AM
7/9/2022	07/01/2022	7/1/2021 12:00:00 AM
7/22/2022	07/01/2022	7/1/2021 12:00:00 AM
7/23/2022	07/01/2022	7/1/2021 12:00:00 AM
7/27/2022	07/01/2022	7/1/2021 12:00:00 AM
8/2/2022	07/01/2022	7/1/2021 12:00:00 AM

date	start_of_quarter	start_of_quarter_timestamp
8/8/2022	07/01/2022	7/1/2021 12:00:00 AM
8/19/2022	07/01/2022	7/1/2021 12:00:00 AM
9/26/2022	07/01/2022	7/1/2021 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

The `start_of_quarter` field is created in the preceding load statement by using the `quarterstart()` function and passing the date field as the function's argument. The `quarterstart()` function initially identifies which quarter the date value falls into. It then returns a timestamp for the first millisecond of that quarter.

Diagram of quarterstart() function, example with no additional arguments



Transaction 8203 took place on August 8. The `quarterstart()` function identifies that the transaction took place in the third quarter, and returns the first millisecond of that quarter, which is July 1 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_quarter_start`, that returns the timestamp for the start of the quarter before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
  *,
```

```

quarterstart(date,-1) as previous_quarter_start,
timestamp(quarterstart(date,-1)) as previous_quarter_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_quarter_start
- previous_quarter_start_timestamp

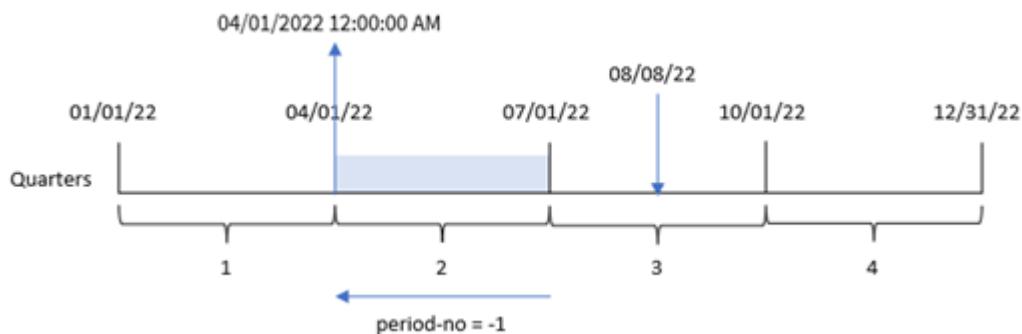
Results table

date	previous_quarter_start	previous_quarter_start_timestamp
1/7/2022	10/01/2021	10/1/2021 12:00:00 AM
1/19/2022	10/01/2021	10/1/2021 12:00:00 AM
2/5/2022	10/01/2021	10/1/2021 12:00:00 AM
2/28/2022	10/01/2021	10/1/2021 12:00:00 AM
3/16/2022	10/01/2021	10/1/2021 12:00:00 AM
4/1/2022	01/01/2022	1/1/2022 12:00:00 AM

date	previous_quarter_start	previous_quarter_start_timestamp
5/7/2022	01/01/2022	1/1/2022 12:00:00 AM
5/16/2022	01/01/2022	1/1/2022 12:00:00 AM
6/15/2022	01/01/2022	1/1/2022 12:00:00 AM
6/26/2022	01/01/2022	1/1/2022 12:00:00 AM
7/9/2022	04/01/2022	4/1/2021 12:00:00 AM
7/22/2022	04/01/2022	4/1/2021 12:00:00 AM
7/23/2022	04/01/2022	4/1/2021 12:00:00 AM
7/27/2022	04/01/2022	4/1/2021 12:00:00 AM
8/2/2022	04/01/2022	4/1/2021 12:00:00 AM
8/8/2022	04/01/2022	4/1/2021 12:00:00 AM
8/19/2022	04/01/2022	4/1/2021 12:00:00 AM
9/26/2022	04/01/2022	4/1/2021 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

In this instance, because a period_no of -1 was used as the offset argument in the quarterstart() function, the function first identifies the quarter that the transactions take place in. It then shifts one quarter prior and identifies the first millisecond of that quarter.

Diagram of quarterstart() function, period_no example



Transaction 8203 took place on August 8. The quarterstart() function identifies that the quarter before the transaction took place was between April 1 and June 30. It then returns the first millisecond of that quarter, April 1 at 12:00:00 AM.

Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, we need to set March 1 as the beginning of the financial year.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  quarterstart(date,0,3) as start_of_quarter,
  timestamp(quarterstart(date,0,3)) as start_of_quarter_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

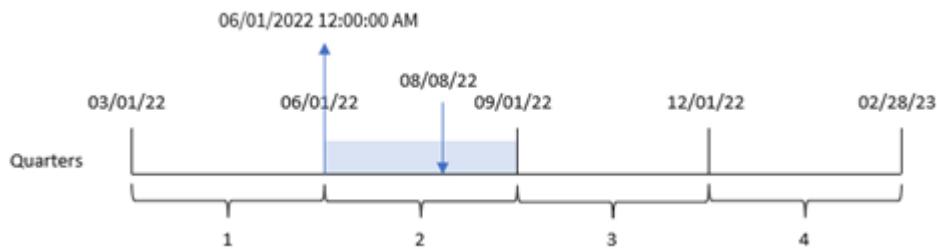
- `date`
- `start_of_quarter`
- `start_of_quarter_timestamp`

Results table

date	start_of_quarter	start_of_quarter_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	12/01/2021	12/1/2021 12:00:00 AM
2/28/2022	12/01/2021	12/1/2021 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/16/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	06/01/2022	6/1/2022 12:00:00 AM
8/8/2022	06/01/2022	6/1/2022 12:00:00 AM
8/19/2022	06/01/2022	6/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

In this instance, because the `first_month_of_year` argument of 3 is used in the `quarterstart()` function, the start of the year moves from January 1 to March 1.

Diagram of quarterstart() function, first_month_of_year example



Transaction 8203 took place on August 8. Because the beginning of the year is March 1, the quarters in the year occur between March-May, June-August, September-November and December-February. The quarterstart() function identifies that the transaction took place in the quarter between the start of June and of August and returns the first millisecond of that quarter, which is June 1 at 12:00:00 AM.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the end of the quarter when the transactions took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
```

```
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Add the following measures:

- `=quarterstart(date)`
- `=timestamp(quarterstart(date))`

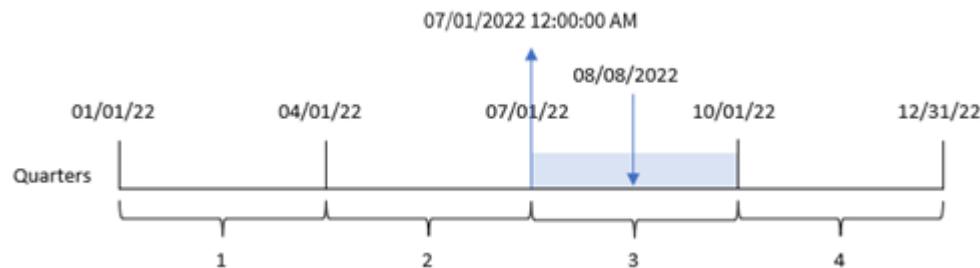
Results table

date	=quarterstart(date)	=timestamp(quarterstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	04/01/2022	4/1/2022 12:00:00 AM
6/26/2022	04/01/2022	4/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM

The `start_of_quarter` measure is created in the chart object by using the `quarterstart()` function and passing the date field as the function's argument.

The `quarterstart()` function identifies the quarter into which the date value falls, returning a timestamp for the first millisecond of that quarter.

Diagram of quarterstart() function, chart object example



Transaction 8203 took place on August 8. The `quarterstart()` function identifies that the transaction took place in the third quarter, and returns the first millisecond of that quarter. This returned value is July 1 at 12:00:00 AM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of loan balances, which is loaded into a table called `Loans`.
- Data consisting of loan IDs, the balance at the beginning of the quarter, and the simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan ID, the current interest that has been accrued on each loan in the quarter to date.

Load script

```
Loans:  
Load  
*  
Inline  
[  
loan_id,start_balance,rate  
8188,$10000.00,0.024  
8189,$15000.00,0.057  
8190,$17500.00,0.024  
8191,$21000.00,0.034
```

```
8192,$90000.00,0.084  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add these fields as dimensions:
 - loan_id
 - start_balance
2. Next, create this measure to calculate the accumulated interest:
`=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)`
3. Set the measure's **Number formatting** to **Money**.

Results table

loan_id	start_balance	=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)
8188	\$10000.00	\$15.07
8189	\$15000.00	\$128.84
8190	\$17500.00	\$63.29
8191	\$21000.00	\$107.59
8192	\$90000.00	\$1139.18

The `quarterstart()` function, using today's date as its only argument, returns the start date of the current year. By subtracting that result from the current date, the expression returns the number of days that have elapsed so far this quarter.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate incurred for this period. The result is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this quarter.

second

This function returns an integer representing the second when the fraction of the **expression** is interpreted as a time according to the standard number interpretation.

Syntax:

```
second (expression)
```

Return data type: integer

When to use it

The `second()` function is useful when you would like to compare aggregations by second. For example, the function can be used if you would like to see activity count distribution by second.

These dimensions can be created either in the load script by using the function to create a field in a Master Calendar table, or used directly in a chart as a calculated dimension.

Function examples

Example	Result
second('09:14:36')	returns 36
second('0.5555')	returns 55 (Because 0.5555 = 13:19:55)

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Variable

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing transactions by timestamp, which is loaded into a table called `Transactions`.
- The default `Timestamp` system variable (`M/D/YYYY h:mm:ss[.ffff] TT`) is used.
- The creation of a field, `second`, to calculate when purchases took place.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';
```

`Transactions:`

```
  Load
    *,
    second(date) as second
  ;
```

```
Load
*
```

```
Inline
[
```

```
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- second

Results table

date	second
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

The values in the second field are created by using the second() function and passing the date as the expression in the preceding load statement.

Example 2 – Chart object

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The second values are calculated via a measure in a chart object.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

Transactions:
Load
*
Inline
[
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:date.

Create the following measure:

```
=second(date)
```

Results table

date	=second(date)
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

The values for second are created by using the `second()` function and passing the date as the expression in a measure for the chart object.

Example 3 – Scenario

Load script and chart expressions

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of timestamps, which is generated to represent the traffic to a particular festival's ticket sales website. These timestamps and a corresponding id are loaded into a table called `web_Traffic`.
- The `TimeStamp` system variable `M/D/YYYY h:mm:ss[.ffff] TT` is used.

In this scenario, there were 10000 tickets, which went on sale at 9:00 AM on May 20, 2021. One minute later, the tickets were sold out.

The user would like a chart object that shows, by second, the count of visits to the website.

Load script

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.ffff] TT';

tmpTimeStampCreator:
Load
    makedate(2022,05,20) as date
AutoGenerate 1;

join load
    maketime(9+floor(rand()*2),0,floor(rand()*59)) as time
autogenerate 10000;

web_Traffic:
Load
    recno() as id,
    timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Next, create a calculated dimensions using the following expression:
`=second(timestamp)`

3. Create an aggregation measure to calculate the total count of entries:
`=count(id)`

The results table will look similar to the table below, but with different values for the aggregation measure:

Results table

second(timestamp)	=count(id)
0	150
1	184
2	163
3	178
4	179
5	158
6	177
7	169
8	149
9	186
10	169
11	179
12	186
13	182
14	180
15	153
16	191
17	203
18	158
19	159
20	163
+ 39 more rows	

setdateyear

This function takes as input a **timestamp** and a **year** and updates the **timestamp** with the **year** specified in input.

Syntax:

```
setdateyear (timestamp, year)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
timestamp	A standard Qlik Sense timestamp (often just a date).
year	A four-digit year.

Examples and results:

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Scripting examples

Example	Result
<code>setdateyear ('29/10/2005', 2013)</code>	Returns '29/10/2013'
<code>setdateyear ('29/10/2005 04:26:14', 2013)</code>	Returns '29/10/2013 04:26:14' To see the time part of the timestamp in a visualization, you must set the number formatting to Date and choose a value for Formatting that displays time values.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
SetYear:  
Load *,  
SetDateYear(testdates, 2013) as NewYear  
Inline [  
testdates  
1/11/2012  
10/12/2012  
1/5/2013  
2/1/2013  
19/5/2013  
15/9/2013  
11/12/2013  
2/3/2014  
14/5/2014  
13/6/2014  
7/7/2014  
4/8/2014  
];
```

The resulting table contains the original dates and a column in which the year has been set to 2013.

Results table

testdates	NewYear
1/11/2012	1/11/2013
10/12/2012	10/12/2013
2/1/2012	2/1/2013
1/5/2013	1/5/2013
19/5/2013	19/5/2013
15/9/2013	15/9/2013
11/12/2013	11/12/2013
2/3/2014	2/3/2013
14/5/2014	14/5/2013
13/6/2014	13/6/2013
7/7/2014	7/7/2013
4/8/2014	4/8/2013

setdateyearmonth

This function takes as input a **timestamp**, a **month** and a **year** and updates the **timestamp** with the **year** and the **month** specified in input..

Syntax:

```
SetDateYearMonth (timestamp, year, month)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
timestamp	A standard Qlik Sense timestamp (often just a date).
year	A four-digit year.
month	A one or two-digit month.

Examples and results:

These examples use the date format **DD/MM/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of your data load script. Change the format in the examples to suit your requirements.

Scripting examples

Example	Result
<code>setdateyearmonth ('29/10/2005', 2013, 3)</code>	Returns '29/03/2013'
<code>setdateyearmonth ('29/10/2005 04:26:14', 2013, 3)</code>	Returns '29/03/2013 04:26:14' To see the time part of the timestamp in a visualization, you must set the number formatting to Date and choose a value for Formatting that displays time values.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
SetYearMonth:  
Load *,  
SetDateYearMonth(testdates, 2013,3) as NewYearMonth  
Inline [  
testdates  
1/11/2012  
10/12/2012  
2/1/2013  
19/5/2013  
15/9/2013  
11/12/2013  
14/5/2014  
13/6/2014  
7/7/2014  
4/8/2014  
];
```

The resulting table contains the original dates and a column in which the year has been set to 2013.

Results table

testdates	NewYearMonth
1/11/2012	1/3/2013
10/12/2012	10/3/2013
2/1/2012	2/3/2013
19/5/2013	19/3/2013
15/9/2013	15/3/2013
11/12/2013	11/3/2013
14/5/2014	14/3/2013
13/6/2014	13/3/2013

testdates	NewYearMonth
7/7/2014	7/3/2013
4/8/2014	4/3/2013

timezone

This function returns the time zone, as defined on the computer where the Qlik engine is running.

Syntax:

```
TimeZone( )
```

Return data type: dual

Example:

```
timezone()
```

If you want to see a different timezone in a measure in your app, you can use the `localtime()` function in a measure.

today

This function returns the current date. The function returns values in the `dateFormat` system variable format.

Syntax:

```
today([ timer_mode ])
```

Return data type: dual

The `today()` function can be used either in the load script or in chart objects.

The default `timer_mode` value is 1.

Arguments

Argument	Description
timer_mode	<p>Can have the following values:</p> <ul style="list-style-type: none"> 0 (day of last finished data load) 1 (day of function call) 2 (day when the app was opened) <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>If you use the function in a load script, <code>timer_mode=0</code> will result in the day of the last finished data load, while <code>timer_mode=1</code> will give the day of the current data load.</i> </div>

Function examples

timer_mode value	Result if used in load script	Result if used in chart object
0	Returns a date, in the <code>DateFormat</code> system variable format, of the last successful data reload prior to the latest data reload.	Returns a date, in the <code>DateFormat</code> system variable format, for the latest data reload.
1	Returns a date, in the <code>DateFormat</code> system variable format, for the latest data reload.	Returns a date, in the <code>DateFormat</code> system variable format, of the function call.
2	Returns a date, in the <code>DateFormat</code> system variable format, for when the user's session in the application began. This will not be updated unless the user reloads the script.	Returns the date, in the <code>DateFormat</code> system variable format, for when the user's session in the application began. This will be refreshed once a new session begins or the data in the application is reloaded.

When to use it

The `today()` function is commonly used as a component within an expression. For example, it can be used to calculate the interest that has accumulated in a month up to the current date.

The following table provides an explanation of the result returned by the `today()` function, given different values for the `timer_mode` argument:

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Generation of objects using load script

Load script and results

Overview

The following example creates three variables using the `today()` function. Each variable uses one of the `timer_mode` options to demonstrate their effect.

For the variables to demonstrate their purpose, reload the script and then, after 24 hours, reload the script a second time. This will result in the `today(0)` and `today(1)` variables showing different values, thereby correctly demonstrating their purpose.

Load script

```
LET vPreviousDataLoad = today(0);  
LET vCurrentDataLoad = today(1);  
LET vApplicationOpened = today(2);
```

Results

Once the data has been loaded for a second time, create three textboxes using the directions below.

First, create a textbox for the data which has previously been loaded.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
`=vPreviousDataLoad`
3. Under **Appearance**, select **Show titles** and add the title 'Previous Reload Time' to the object.

Next, create a textbox for the data which is currently being loaded.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
`=vCurrentDataLoad`
3. Under **Appearance**, select **Show titles** and add the title 'Current Reload Time' to the object.

Create a final textbox to show when the user's session in the application was started.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure to the object:
`=vApplicationOpened`
3. Under **Appearance**, select **Show titles** and add the title 'User Session Started' to the object.

Diagram of variables created using `today()` function in load script

Previous Reload Time 06/22/2022	Current Reload Time 06/23/2022	User Session Began 06/23/2022
------------------------------------	-----------------------------------	----------------------------------

The above image shows example values for each of the created variables. For example, the values could be as follows:

- Previous Reload Time: 06/22/2022
- Current Reload Time: 06/23/2022
- User Session Began: 06/23/2022

Example 2 – Generation of objects without load script

Load script and chart expression

Overview

The following example creates three chart objects using the `today()` function. Each chart object uses one of the `timer_mode` options to demonstrate their effect.

There is no load script for this example.

Results

Once the data has been loaded for a second time, create three textboxes.

First, create a textbox for the latest data reload.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=today(0)`
3. Under **Appearance**, select **Show titles** and add the title 'Latest Data Reload' to the object.

Next, create a textbox to show the current time.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=today(1)`
3. Under **Appearance**, select **Show titles** and add the title 'Current Time' to the object.

Create a final textbox to show when the user's session in the application was started.

Do the following:

1. Using the **Text & Image** chart object, create a textbox.
2. Add the following measure:
`=today(2)`
3. Under **Appearance**, select **Show titles** and add the title 'User Session Began' to the object.

Diagram of objects created using today() function without load script

Latest Data Reload 06/23/2022	Current Time 06/23/2022	User Session Began 06/23/2022
----------------------------------	----------------------------	----------------------------------

The above image shows example values for each of the created objects. For example, the values could be as follows:

- Latest Data Reload: 06/23/2022
- Current Time: 06/23/2022
- User Session Began: 06/23/2022

The 'Latest Data Reload' chart object uses a `timer_mode` value of 0. This returns the timestamp for the last time the data was successfully reloaded.

The 'Current Time' chart object uses a `timer_mode` value of 1. This returns the current time according to the system clock. If the sheet or object is refreshed, this value will be updated.

The 'User Session Began' chart object uses a `timer_mode` value of 2. This returns the timestamp for when the application was opened, and the user's session began.

Example 3 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of loan balances, which is loaded into a table called `Loans`.
- Table data with fields for loan ID, balance at the start of the month, and the simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan ID, the current interest that has been accrued on each loan in the month to date. Although the application is only reloaded once per week, the user would like the results to be refreshed whenever the object or application is refreshed.

Load script

```
Loans:  
Load  
*  
Inline  
[
```

```
loan_id,start_balance,rate  
8188,$10000.00,0.024  
8189,$15000.00,0.057  
8190,$17500.00,0.024  
8191,$21000.00,0.034  
8192,$90000.00,0.084  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Add the following fields as dimensions:
 - loan_id
 - start_balance
3. Next, create a measure to calculate accumulated interest:
 $=start_balance*(rate*(today(1)-monthstart(today(1)))/365)$
4. Set the measure's **Number formatting** to **Money**.

Results table

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	\$10000.00	\$16.44
8189	\$15000.00	\$58.56
8190	\$17500.00	\$28.77
8191	\$21000.00	\$48.90
8192	\$90000.00	\$517.81

The monthstart() function, using the today() function to return today's date as its only argument, returns the start date of the current month. By subtracting that result from the current date, again using the today() function, the expression returns the number of days that have elapsed so far this month.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate incurred for this period. The result is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this month.

Because the value of 1 is used as the timer_mode argument in the today() functions inside the expression, each time the chart object is refreshed (by opening the application, refreshing the page, moving between sheets, etc.), the date returned will be for the current date, and the results will be refreshed accordingly.

UTC

Returns the current Coordinated Universal Time.

Syntax:

```
UTC( )
```

Return data type: dual

Example:

```
utc()
```

week

This function returns an integer representing the week number corresponding to the date entered.

Syntax:

```
week(timestamp [, first_week_day [, broken_weeks [, reference_day]])
```

Return data type: integer

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
first_week_day	Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used. The possible values first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday. For more information about the system variable, see <i>FirstWeekDay (page 215)</i> .
broken_weeks	If you don't specify broken_weeks , the value of variable BrokenWeeks will be used to define if weeks are broken or not.
reference_day	If you don't specify reference_day , the value of variable ReferenceDay will be used to define which day in January to set as reference day to define week 1. By default, Qlik Sense functions use 4 as the reference day. This means that week 1 must contain January 4, or put differently, that week 1 must always have at least 4 days in January.

The `week()` function determines which week the date falls into and returns the week number.

In Qlik Sense, the regional settings are fetched when the app is created, and the corresponding settings are stored in the script as environment variables. These are used to determine the week number.

This means that most European app developers get the following environment variables, corresponding to the ISO 8601 definition:

```
Set FirstWeekDay =0;      // Monday as first week day
Set BrokenWeeks =0;       // Use unbroken weeks
Set ReferenceDay =4;      // Jan 4th is always in week 1
```

A North American app developer often gets the following environment variables:

```
Set FirstWeekDay =6;      // Sunday as first week day
Set BrokenWeeks =1;       // Use broken weeks
Set ReferenceDay =1;      // Jan 1st is always in week 1
```

The first day of the week is determined by the `FirstWeekDay` system variable. You can also change the first day of the week by using the `first_week_day` argument in the `week()` function.

If your application uses broken weeks, the week number count begins on January 1 and ends on the day prior to the `FirstWeekDay` system variable regardless of how many days have occurred.

If your application is using unbroken weeks, week 1 can begin in the previous year or in the first few days of January. This depends on how you use the `FirstWeekDay` and the `ReferenceDay` environment variables.

When to use it

The `week()` function is useful when you would like to compare aggregations by weeks. For example, it could be used if you would like to see the total sales of products by week. The `week()` function is chosen over `weekname()` when the user would like the calculation to not necessarily use the application's `Brokenweeks`, `FirstWeekDay`, or `ReferenceDay` system variables.

For example, if you want to see the total sales of products by week.

If the application is using unbroken weeks, week 1 may contain dates from December of the previous year or exclude dates in January of the current year. If the application is using broken weeks, week 1 may contain less than seven days..

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

The examples below assume

```
Set DateFormat= 'MM/DD/YYYY';
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

Function examples

Example	Result
<code>week('12/28/2021')</code>	Returns 52.
<code>week(44614)</code>	Returns 8, since this is the serial number for 02/22/2022.

Example	Result
week('01/03/2021')	Returns 53.
week('01/03/2021',6)	Returns 1.

Example 1 – Default system variables

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2021 and the first two weeks of 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `week_number`, that returns the year and week number when the transactions took place.
- The creation of a field called `week_day`, showing the weekday value of each transaction date.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
Load
  *,
  weekDay(date) as week_day,
  week(date) as week_number
;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
```

```
8194,01/07/2022,40.56  
8195,01/08/2022,53.67  
8196,01/09/2022,26.63  
8197,01/10/2022,72.48  
8198,01/11/2022,18.37  
8199,01/12/2022,45.26  
8200,01/13/2022,58.23  
8201,01/14/2022,18.52  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **week_day**
- **week_number**

Results table

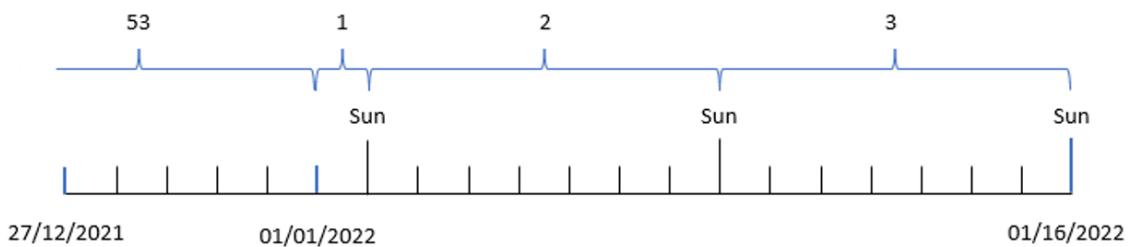
id	date	week_day	week_number
8183	12/27/2021	Mon	53
8184	12/28/2021	Tue	53
8185	12/29/2021	Wed	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	2
8190	01/03/2022	Mon	2
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	3
8197	01/10/2022	Mon	3
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

The week_number field is created in the preceding load statement by using the week() function and passing the date field as the function's argument.

No other parameters are passed into the function, and therefore the following default variables that affect the week() function are in effect:

- **Brokenweeks:** The week count begins on January 1
- **Firstweekday:** The first day of the week is Sunday

Diagram of week() function, using default system variables



Because the application is using the default `Brokenweeks` system variable, week 1 begins on January 1, a Saturday.

Because of the default `Firstweekday` system variable, weeks begin on a Sunday. The first Sunday after January 1 occurs on January 2, which is when week 2 begins.

Example 2 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The creation of a field, `week_number`, that returns the year and week number when the transactions took place.
- The creation of a field called `week_day`, showing the weekday value of each transaction date.

In this example, we would like to set the start of the work week to Tuesday.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;
```

Transactions:

```
Load
  *,
  WeekDay(date) as week_day,
  Week(date,1) as week_number
;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- week_day
- week_number

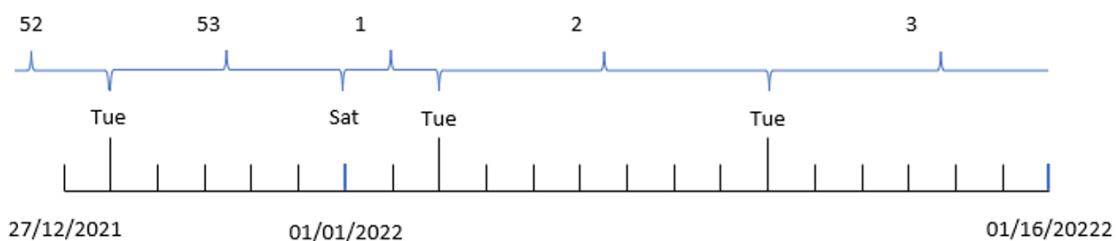
Results table

id	date	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	53
8185	12/29/2021	Wed	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53

id	date	week_day	week_number
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

The application is still using broken weeks. However, the `first_week_day` argument has been set to 1 in the `week()` function. This sets the first day of the week to a Tuesday.

Diagram of `week()` function, `first_week_day` example



The application is using the default `BrokenWeeks` system variable, so week 1 begins on January 1, a Saturday.

The `first_week_day` argument of the `week()` function sets the first week day to a Tuesday. Therefore, week 53 begins on December 28, 2021.

However, because the function is still using broken weeks, week 1 will only be two days long, due to the first Tuesday after January 1 occurring on January 3.

Example 3 – unbroken_weeks

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

In this example, we use unbroken weeks.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

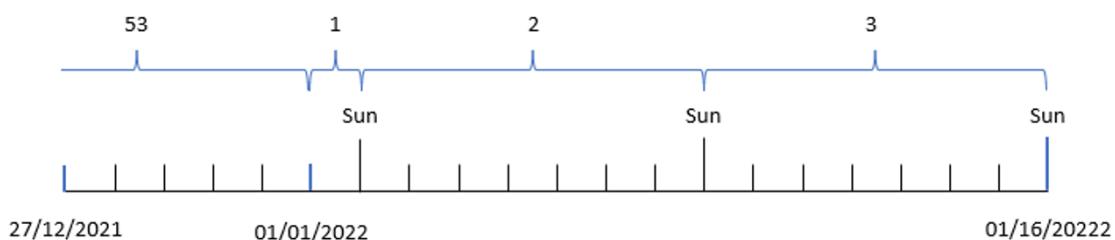
Transactions:
Load
  *,
  weekDay(date) as week_day,
  week(date,6,0) as week_number
;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`
- `week_day`
- `week_number`

Diagram of week() function, chart object example



Results table

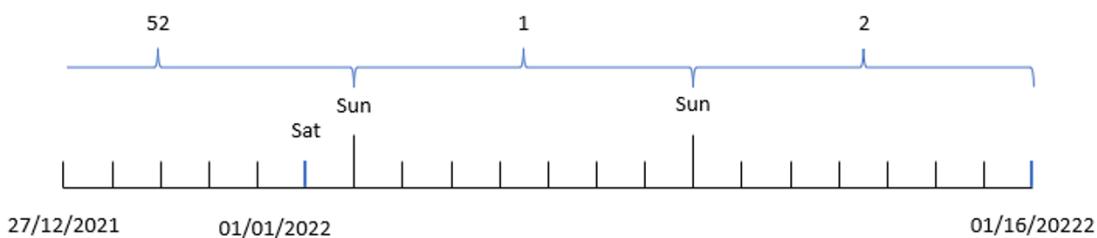
id	date	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	52
8185	12/29/2021	Wed	52
8186	12/30/2021	Thu	52
8187	12/31/2021	Fri	52
8188	01/01/2022	Sat	52
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	1
8192	01/05/2022	Wed	1
8193	01/06/2022	Thu	1
8194	01/07/2022	Fri	1
8195	01/08/2022	Sat	1
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	2

id	date	week_day	week_number
8199	01/12/2022	Wed	2
8200	01/13/2022	Thu	2
8201	01/14/2022	Fri	2

The `first_week_date` parameter is set to 1, making Tuesday the first day of the week. The `broken_weeks` parameter is set to 0, forcing the function to use unbroken weeks. Finally, the third parameter sets the `reference_day` to 2.

The `first_week_date` parameter is set to 6, making Sunday the first day of the week. The `broken_weeks` parameter is set to 0, forcing the function to use unbroken weeks.

Diagram of week() function, example using unbroken weeks



By using unbroken weeks, week 1 does not necessarily begin on January 1; instead, it is required to have a minimum of four days. Therefore, in the dataset, week 52 concludes on Saturday, January 1, 2022. Week 1 then begins on the `FirstWeekDay` system variable, which is Sunday, January 2. This week will conclude on the following Saturday, January 8.

Example 4 – `reference_day`

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the third example.
- The creation of a field, `week_number`, that returns the year and week number when the transactions took place.
- The creation of a field called `week_day`, showing the weekday value of each transaction date.

Additionally, the following conditions must be met:

- The work week begins on a Tuesday.
- The company uses unbroken weeks.
- The reference_day value is 2. In other words, the minimum number of days in January in week 1 will be 2.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;
```

Transactions:

```
Load
  *,
  weekDay(date) as week_day,
  week(date,1,0,2) as week_number
;

Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date

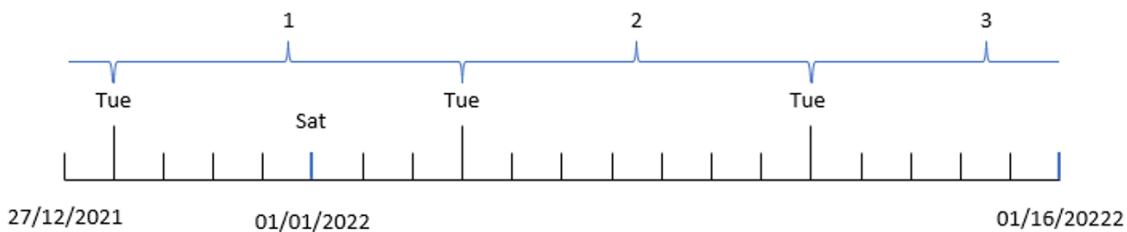
- week_day
- week_number

Results table

id	date	week_day	week_number
8183	12/27/2021	Mon	52
8184	12/28/2021	Tue	1
8185	12/29/2021	Wed	1
8186	12/30/2021	Thu	1
8187	12/31/2021	Fri	1
8188	01/01/2022	Sat	1
8189	01/02/2022	Sun	1
8190	01/03/2022	Mon	1
8191	01/04/2022	Tue	2
8192	01/05/2022	Wed	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	Sun	2
8197	01/10/2022	Mon	2
8198	01/11/2022	Tue	3
8199	01/12/2022	Wed	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

The `first_week_date` parameter is set to 1, making Tuesday the first day of the week. The `broken_weeks` parameter is set to 0, forcing the function to use unbroken weeks. Finally, the third parameter sets the `reference_day` parameter to 2.

Diagram of week() function, reference_day example



With the function using unbroken weeks and a `reference_day` value of 2 used as a parameter, week 1 only needs to include two days in January. Due to the first weekday being Tuesday, week 1 begins on December 28, 2021, and concludes on Monday, January 3, 2022.

Example 5 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns the week number is created as a measure in a chart object.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
```

```
8198,01/11/2022,18.37  
8199,01/12/2022,45.26  
8200,01/13/2022,58.23  
8201,01/14/2022,18.52  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Add the following fields as dimensions:
 - id
 - date
3. Next, create the following measure:
`=week (date)`
4. Create a measure, `week_day`, to show the weekday value of each transaction date:
`=weekday(date)`

Results table

id	date	=week(date)	=weekday(date)
8183	12/27/2021	53	Mon
8184	12/28/2021	53	Tue
8185	12/29/2021	53	Wed
8186	12/30/2021	53	Thu
8187	12/31/2021	53	Fri
8188	01/01/2022	1	Sat
8189	01/02/2022	2	Sun
8190	01/03/2022	2	Mon
8191	01/04/2022	2	Tue
8192	01/05/2022	2	Wed
8193	01/06/2022	2	Thu
8194	01/07/2022	2	Fri
8195	01/08/2022	2	Sat
8196	01/09/2022	3	Sun
8197	01/10/2022	3	Mon
8198	01/11/2022	3	Tue
8199	01/12/2022	3	Wed

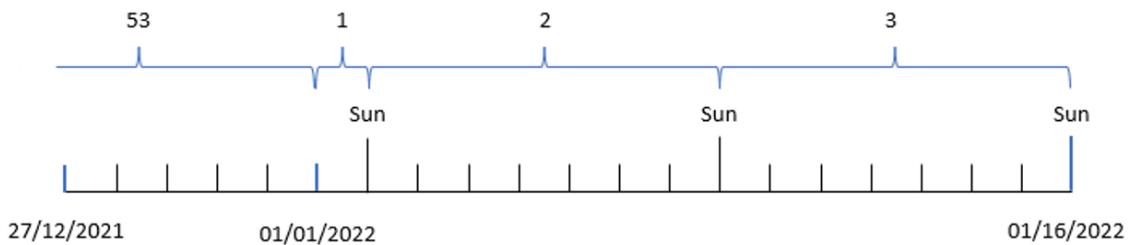
id	date	=week(date)	=weekday(date)
8200	01/13/2022	3	Thu
8201	01/14/2022	3	Fri

The week_number field is created in the preceding load statement by using the week() function and passing the date field as the function's argument.

No other parameters are passed into the function, and therefore the following default variables that affect the week() function are in effect:

- **Brokenweeks:** The week count begins on January 1
- **Firstweekday:** The first day of the week is Sunday

Diagram of week() function, chart object example



Because the application is using the default `Brokenweeks` system variable, week 1 begins on January 1, a Saturday.

Because of the default `Firstweekday` system variable, weeks begin on a Sunday. The first Sunday after January 1 occurs on January 2, which is when week 2 begins.

Example 6 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2019 and first two weeks of 2020, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

The application primarily uses broken weeks across its dashboard. However, the end user would like a chart object that presents the total sales by week using unbroken weeks. The reference day should be January 2, with weeks beginning on a Tuesday. This could be achieved even when this dimension is not available in the data model, using the `week()` function as a calculated dimension in the chart.

Load script

```
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8183,12/27/2019,58.27
8184,12/28/2019,67.42
8185,12/29/2019,23.80
8186,12/30/2019,82.06
8187,12/31/2019,40.56
8188,01/01/2020,37.23
8189,01/02/2020,17.17
8190,01/03/2020,88.27
8191,01/04/2020,57.42
8192,01/05/2020,53.80
8193,01/06/2020,82.06
8194,01/07/2020,40.56
8195,01/08/2020,53.67
8196,01/09/2020,26.63
8197,01/10/2020,72.48
8198,01/11/2020,18.37
8199,01/12/2020,45.26
8200,01/13/2020,58.23
8201,01/14/2020,18.52
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table.
2. Create the following calculated dimension:
`=week(date)`
3. Next, create the following aggregation measure:
`=sum(amount)`
4. Set the measure's **Number formatting** to **Money**.
5. Select the **Sorting** menu, and for the calculated dimension, remove custom sorting.
6. De-select the **Sort numerically** and **Sort alphabetically** options.

Results table

week(date)	sum(amount)
52	\$125.69
53	\$146.42
1	\$200.09
2	\$347.57
3	\$122.01

weekday

This function returns a dual value with:

- A day name as defined in the environment variable **DayNames**.
- An integer between 0-6 corresponding to the nominal day of the week (0-6).

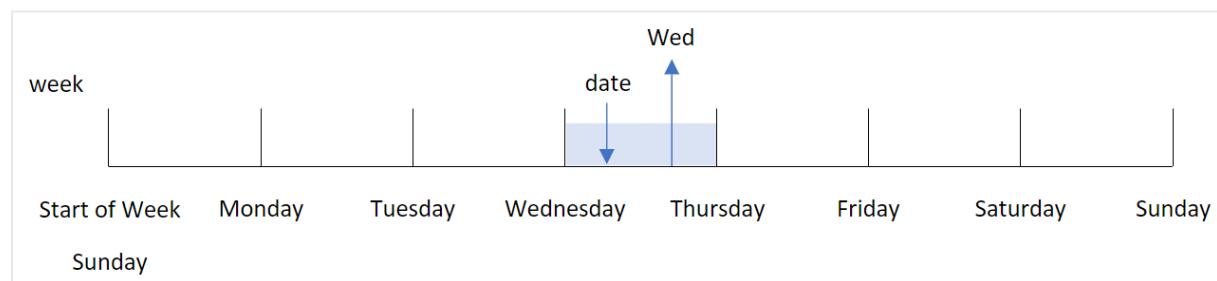
Syntax:

```
weekday(date [,first_week_day=0])
```

Return data type: dual

The weekday() function determines which day of the week a date occurs on. It then returns a string value representing that day.

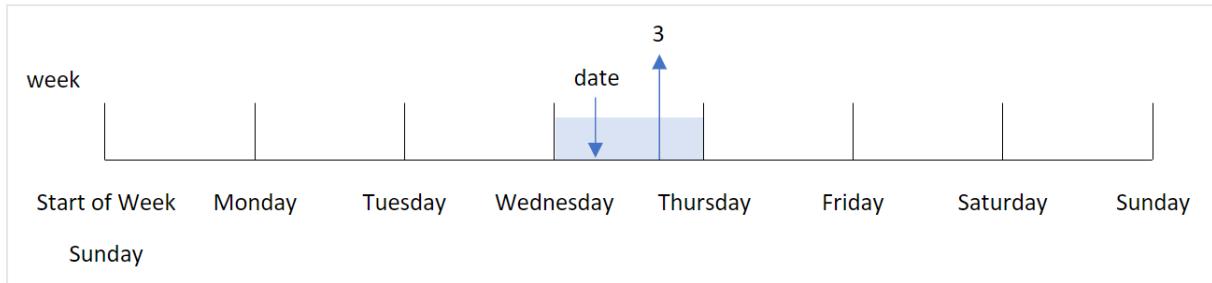
Diagram of weekday() function that returns the name of the day a date falls on



The result returns the number value corresponding to that day of the week (0-6), based on the week's start day. For example, if the first day of the week is set to Sunday, a Wednesday will return a number value of 3. This start day is determined either by the `FirstWeekDay` system variable, or the `first_week_day` function parameter.

You can use this number value as a part of an arithmetic expression. For example, multiply it by 1 to return the value itself.

Diagram of `weekday()` function with the number value of the day being shown instead of the name of the day



When to use it

The `weekday()` function is useful when you want to compare aggregations by day of the week. For example, if you want to compare the average sales of products by weekday.

These dimensions can be created in the load script by using the function to create a field in a **Master Calendar** table; or created directly in a chart as a calculated measure.

Related topics

Topics	Interaction
FirstWeekDay (page 215)	Defines the start day of each week.

Arguments

Argument	Description
date	The date or timestamp to evaluate.
first_week_day	Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used. <i>FirstWeekDay (page 215)</i>

You can use the following values to set the day on which the week starts in the `first_week_day` argument:

first_week_day values

Day	Value
Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.



Unless stated otherwise, `Firstweekday` is set to 0 in these examples.

Function examples

Example	Result
<code>weekday('10/12/1971')</code>	Returns 'Tue' and 1.
<code>weekday('10/12/1971' , 6)</code>	Returns 'Tue' and 2. In this example, Sunday (6) is the first day of the week.
<code>SET FirstWeekDay=6; ... weekday('10/12/1971')</code>	Returns 'Tue' and 2.

Example 1 - Weekday string

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 which is loaded into a table called ‘Transactions’.
- The `Firstweekday` system variable which is set to 6 (Sunday).
- The `DayNames` variable which is set to use the default day names.
- A preceding load which contains the `weekday()` function, which is set as the ‘week_day’ field and returns the weekday the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
Load
  *,
  weekDay(date) as week_day
;
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- week_day

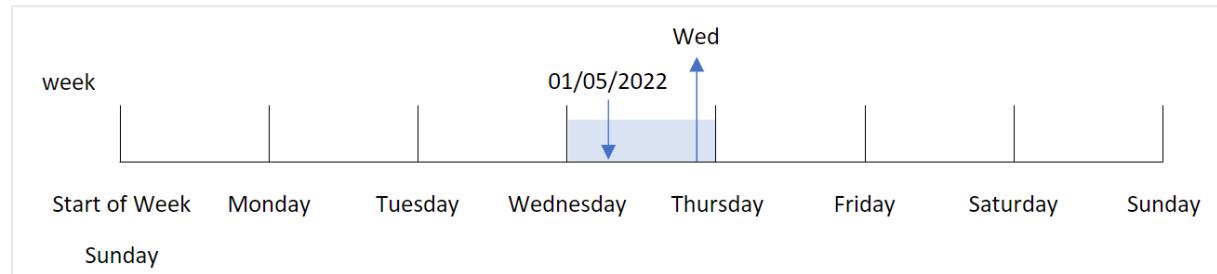
Results table

id	date	week_day
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

The ‘week_day’ field is created in the preceding load statement by using the `weekday()` function and passing the date field as the function’s argument.

The `weekday()` function returns the weekday string value; that is, it returns the name of the weekday which is set by the `DayNames` system variable.

Diagram of `weekday()` function that returns Wednesday as the weekday for transaction 8192



Transaction 8192 took place on January 5. The `FirstweekDay` system variable sets the first day of the week as Sunday. The `weekday()` function transaction took place on a Wednesday and returns this value, in the abbreviated form of the `DayNames` system variable, in the `week_day` field.

The values in the ‘`week_day`’ field are right aligned in the column because there is a dual number and text result for the field (Wednesday, 3). To convert the field value into its number equivalent, the field can be wrapped inside the `num()` function. For example, in Transaction 8192, the Wednesday value would be converted into the number 3.

Example 2 - `first_week_day`

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 which is loaded into a table called ‘Transactions’.
- The `FirstweekDay` system variable which is set to 6 (Sunday).
- The `DayNames` variable which is set to use the default day names.
- A preceding load which contains the `weekday()` function, which is set as the ‘`week_day`’ field and returns the weekday the transactions took place.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;
```

Transactions:

```
Load
  *,
  WeekDay(date,1) as week_day
;
Load
```

```

*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];

```

Results

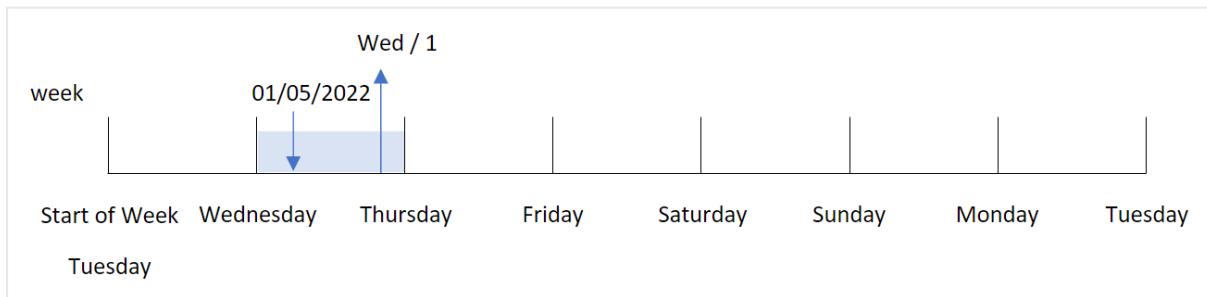
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- week_day

Results table

id	date	week_day
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

Diagram of weekday() function that shows Wednesday has the dual number value of 1



Because the `first_week_day` argument is set to 1 in the `weekday()` function, the first day of the week is Tuesday. Therefore, all transactions that take place on a Tuesday will have a dual number value of 0.

Transaction 8192 took place on January 5. The `weekday()` function identifies that this is a Wednesday, and so the expression would return the dual number value of 1.

Example 3 - Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 which is loaded into a table called ‘Transactions’.
- The `FirstWeekDay` system variable which is set to 6 (Sunday).
- The `DayNames` variable which is set to use the default day names.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that identifies the weekday value is created as a measure in a chart in the app.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`

To calculate the weekday value, create the following measure:

- `=weekday(date)`

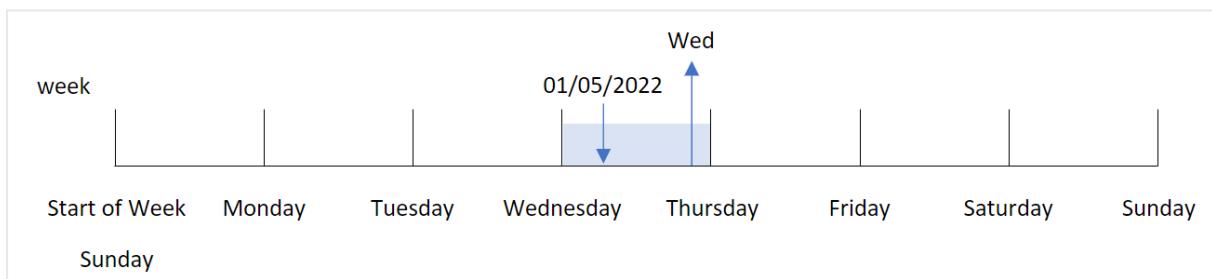
Results table

id	date	=weekday(date)
8188	01/01/2022	Sat
8189	01/02/2022	Sun
8190	01/03/2022	Mon
8191	01/04/2022	Tue
8192	01/05/2022	Wed
8193	01/06/2022	Thu
8194	01/07/2022	Fri

The ‘=weekday(date)’ field is created in the chart by using the `weekday()` function and passing the date field as the function’s argument.

The `weekday()` function returns the weekday string value; that is, it returns the name of the weekday which is set by the `DayNames` system variable.

Diagram of `weekday()` function that returns Wednesday as the weekday for transaction 8192



Transaction 8192 took place on January 5. The `Firstweekday` system variable sets the first day of the week as Sunday. The `weekday()` function transaction took place on a Wednesday and returns this value, in the abbreviated form of the `DayNames` system variable, in the `=weekday(date)` field.

Example 4 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022 which is loaded into a table called ‘Transactions’.
- The `Firstweekday` system variable which is set to 6 (Sunday).
- The `DayNames` variable which is set to use the default day names.

The end user would like a chart that presents the average sales by weekday for the transactions.

Load script

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
LOAD
    RecNo() AS id,
    MakeDate(2022, 1, Ceil(Rand() * 31)) as date,
    Rand() * 1000 AS amount

Autogenerate(1000);
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- =weekday(date)
- =avg(amount)

Set the measure's **Number Formatting** to **Money**.

Results table

weekday(date)	Avg(amount)
Sun	\$536.96
Mon	\$500.80
Tue	\$515.63
Wed	\$509.21
Thu	\$482.70
Fri	\$441.33
Sat	\$505.22

weekend

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the calendar week containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
WeekEnd(timestamp [, period_no [, first_week_day ]])
```

Return data type: dual

The `weekend()` function determines which week the date falls into. It then returns a timestamp, in date format, for the last millisecond of that week. The first day of the week is determined by the `FirstWeekDay` environment variable. However, this can be superseded by the `first_week_day` argument in the `weekend()` function.

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
period_no	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
first_week_day	Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used. The possible values for first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday. For more information about the system variable, see <i>FirstWeekDay (page 215)</i>

When to use it

The `weekend()` function is commonly used as part of an expression when the user would like the calculation to use remaining days of the week for the specified date. For example, it could be used if a user would like to calculate the total interest not yet incurred during the week.

The following examples assume:

```
SET FirstWeekDay=0;
```

Example	Result
<code>weekend('01/10/2013')</code>	Returns 01/12/2013 23:59:59.
<code>weekend('01/10/2013', -1)</code>	Returns 01/05/2013 23:59:59..
<code>weekend('01/10/2013', 0, 1)</code>	Returns 01/14/2013 23:59:59.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Examples:

If you want ISO settings for weeks and week numbers, make sure to have the following in the script:

```
Set DateFormat = 'YYYY-MM-DD';
Set FirstWeekDay =0; // Monday as first week day
Set BrokenWeeks =0; //(use unbroken weeks)
Set ReferenceDay =4; // Jan 4th is always in week 1
```

If you want US settings, make sure to have the following in the script:

```
Set DateFormat = 'M/D/YYYY';
Set FirstWeekDay =6; // Sunday as first week day
Set BrokenWeeks =1; //(use broken weeks)
Set ReferenceDay =1; // Jan 1st is always in week 1
```

The examples above results in the following from the weekend() function:

Example of Weekend function

Date	ISO week end	US week end
Sat 2020 Dec 26	2020-12-27	12/26/2020
Sun 2020 Dec 27	2020-12-27	1/2/2021
Mon 2020 Dec 28	2021-01-03	1/2/2021
Tue 2020 Dec 29	2021-01-03	1/2/2021
Wed 2020 Dec 30	2021-01-03	1/2/2021
Thu 2020 Dec 31	2021-01-03	1/2/2021
Fri 2021 Jan 1	2021-01-03	1/2/2021
Sat 2021 Jan 2	2021-01-03	1/2/2021
Sun 2021 Jan 3	2021-01-03	1/9/2021
Mon 2021 Jan 4	2021-01-10	1/9/2021
Tue 2021 Jan 5	2021-01-10	1/9/2021



The week ends are on Sundays in the ISO column, and on Saturdays in the US column.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `end_of_week`, that returns a timestamp for the end of the week when the transactions took place.

Load script

```
SET FirstWeekDay=6;

Transactions:
  Load
    *,
    weekend(date) as end_of_week,
    timestamp(weekend(date)) as end_of_week_timestamp
  ;
  Load
  *
  Inline
  [
  id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- end_of_week
- end_of_week_timestamp

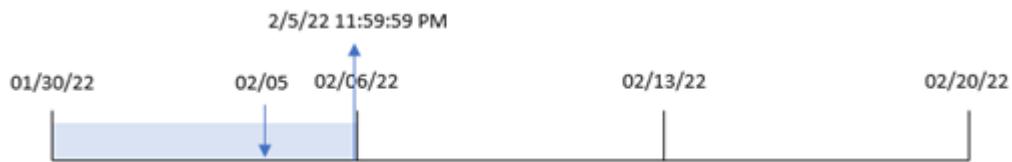
Results table

date	end_of_week	end_of_week_timestamp
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

The end_of_week field is created in the preceding load statement by using the weekend() function and passing the date field as the function's argument.

The weekend() function identifies which week the date value falls into and returns a timestamp for the last millisecond of that week.

Diagram of weekend() function, basic example



Transaction 8191 took place on February 5. The `Firstweekday` system variable sets the first day of the week to a Sunday. The `weekend()` function identifies that the first Saturday after February 5 – and therefore the end of the week – was on February 5. Therefore, the `end_of_week` value for that transaction returns the last millisecond of that day, which is February 5 at 11:59:59 PM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `previous_week_end`, that returns the timestamp for the start of the week before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    weekend(date,-1) as previous_week_end,
    timestamp(weekend(date,-1)) as previous_week_end_timestamp
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,1/7/2022,17.17
  8189,1/19/2022,37.23
  8190,2/28/2022,88.27
  8191,2/5/2022,57.42
  8192,3/16/2022,53.80
  8193,4/1/2022,82.06
  8194,5/7/2022,40.39
  8195,5/16/2022,87.21
  8196,6/15/2022,95.93
  8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_week_end
- previous_week_end_timestamp

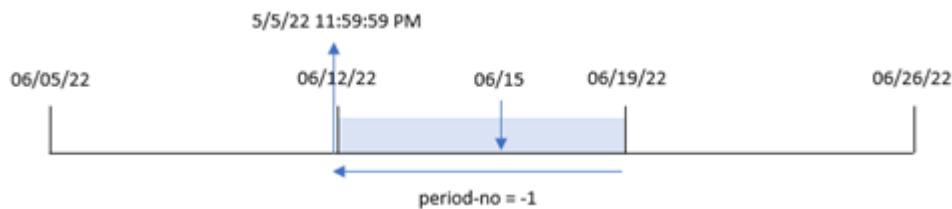
Results table

date	end_of_week	end_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 11:59:59 PM
1/19/2022	01/15/2022	1/15/2022 11:59:59 PM
2/5/2022	01/29/2022	1/29/2022 11:59:59 PM
2/28/2022	02/26/2022	2/26/2022 11:59:59 PM
3/16/2022	03/12/2022	3/12/2022 11:59:59 PM
4/1/2022	03/26/2022	3/26/2022 11:59:59 PM
5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
5/16/2022	05/14/2022	5/14/2022 11:59:59 PM
6/15/2022	06/11/2022	6/11/2022 11:59:59 PM
6/26/2022	06/25/2022	6/25/2022 11:59:59 PM
7/9/2022	07/02/2022	7/2/2022 11:59:59 PM
7/22/2022	07/16/2022	7/16/2022 11:59:59 PM
7/23/2022	07/16/2022	7/16/2022 11:59:59 PM
7/27/2022	07/23/2022	7/23/2022 11:59:59 PM
8/2/2022	07/30/2022	7/30/2022 11:59:59 PM
8/8/2022	08/06/2022	8/6/2022 11:59:59 PM
8/19/2022	08/13/2022	8/13/2022 11:59:59 PM

date	end_of_week	end_of_week_timestamp
9/26/2022	09/24/2022	9/24/2022 11:59:59 PM
10/14/2022	10/08/2022	10/8/2022 11:59:59 PM
10/29/2022	10/22/2022	10/22/2022 11:59:59 PM

In this instance, because a period_no of -1 was used as the offset argument in the weekend() function, the function first identifies the week in which the transactions take place. It then looks one week prior and identifies the last millisecond of that week.

Diagram of weekend() function, period_no example



Transaction 8196 took place on June 15. The weekend() function identifies that the week begins on June 12. Therefore, the previous week ends on June 11 at 11:59:59 PM; this is the value returned for the previous_week_end field.

Example 3 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, we need to set Tuesday as the first day of the work week.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    weekend(date,0,1) as end_of_week,
    timestamp(weekend(date,0,1)) as end_of_week_timestamp,
    ;
  Load
  *
  Inline
  [
  id,date,amount
```

```
8188,1/7/2022,17.17  
8189,1/19/2022,37.23  
8190,2/28/2022,88.27  
8191,2/5/2022,57.42  
8192,3/16/2022,53.80  
8193,4/1/2022,82.06  
8194,5/7/2022,40.39  
8195,5/16/2022,87.21  
8196,6/15/2022,95.93  
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- end_of_week
- end_of_week_timestamp

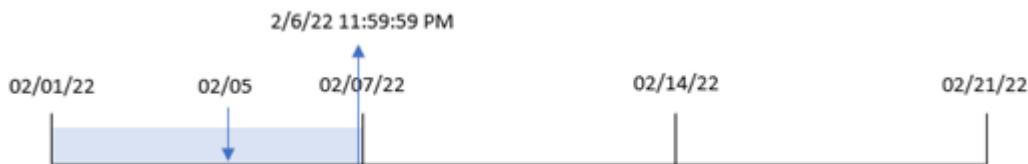
Results table

date	end_of_week	end_of_week_timestamp
1/7/2022	01/10/2022	1/10/2022 11:59:59 PM
1/19/2022	01/24/2022	1/24/2022 11:59:59 PM
2/5/2022	02/07/2022	2/7/2022 11:59:59 PM
2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
3/16/2022	03/21/2022	3/21/2022 11:59:59 PM
4/1/2022	04/04/2022	4/4/2022 11:59:59 PM
5/7/2022	05/09/2022	5/9/2022 11:59:59 PM
5/16/2022	05/16/2022	5/16/2022 11:59:59 PM
6/15/2022	06/20/2022	6/20/2022 11:59:59 PM
6/26/2022	06/27/2022	6/27/2022 11:59:59 PM
7/9/2022	07/11/2022	7/11/2022 11:59:59 PM

date	end_of_week	end_of_week_timestamp
7/22/2022	07/25/2022	7/25/2022 11:59:59 PM
7/23/2022	07/25/2022	7/25/2022 11:59:59 PM
7/27/2022	08/01/2022	8/1/2022 11:59:59 PM
8/2/2022	08/08/2022	8/8/2022 11:59:59 PM
8/8/2022	08/08/2022	8/8/2022 11:59:59 PM
8/19/2022	08/22/2022	8/22/2022 11:59:59 PM
9/26/2022	09/26/2022	9/26/2022 11:59:59 PM
10/14/2022	10/17/2022	10/17/2022 11:59:59 PM
10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

In this instance, because the `first_week_date` argument of 1 is used in the `weekend()` function, it sets the first day of the week to Tuesday.

Diagram of weekend() function, first_week_day example



Transaction 8191 took place on February 5. The `weekend()` function identifies that the first Monday after the this date – and therefore the end of the week and value returned – was on February 6 at 11:59:59 PM.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the end of the week when the transactions took place is created as a measure in a chart object of the application.

Load script

```
Transactions:
Load
*
Inline
[
```

```

id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate the start of the week that a transaction takes place in, add the following measures:

- `=weekend(date)`
- `=timestamp(weekend(date))`

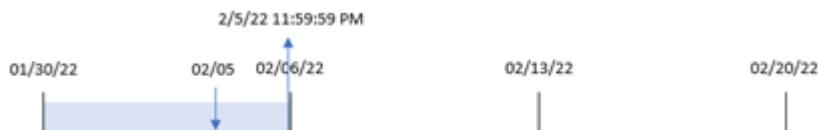
Results table

date	=weekend(date)	=timestamp(weekend(date))
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM

date	=weekend(date)	=timestamp(weekend(date))
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

The `end_of_week` measure is created in the chart object by using the `weekend()` function and passing the date field as the function's argument. The `weekend()` function identifies which week the date value falls into, returning a timestamp for the last millisecond of that week.

Diagram of weekend() function, chart object example



Transaction 8191 took place on February 5. The `Firstweekday` system variable sets the first day of the week to a Sunday. The `weekend()` function identifies that the first Saturday after February 5 – and therefore the end of the week – was on February 5. Therefore, the `end_of_week` value for that transaction returns the last millisecond of that day, which is February 5 at 11:59:59 PM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Employee_Expenses`.
- Data consisting of employee IDs, employee names, and the average daily expense claims of each employee.

The end user would like a chart object that displays, by employee ID and employee name, the estimated expense claims still to be incurred for the remainder of the week.

Load script

```
Employee_Expenses:  
Load  
*  
Inline  
[  
employee_id,employee_name,avg_daily_claim  
182,Mark, $15  
183,Deryck, $12.5  
184,Dexter, $12.5  
185,Sydney,$27  
186,Agatha,$18  
];
```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add these fields as dimensions:
 - employee_id
 - employee_name
2. Next, create a measure to calculate the accumulated interest:
$$=(\text{weekend}(\text{today}(1))-\text{today}(1))*\text{avg_daily_claim}$$
3. Set the measure's **Number formatting** to **Money**.

Results table

employee_id	employee_name	$=\text{weekend}(\text{today}(1))-\text{today}(1)*\text{avg_daily_claim}$
182	Mark	\$90.00
183	Deryck	\$75.00
184	Dexter	\$75.00
185	Sydney	\$162.00
186	Agatha	\$108.00

The `weekend()` function, by using today's date as its only argument, returns the end date of the current week. Then, by subtracting today's date from the week end date, the expression returns the number of days that remain this week.

This value is then multiplied by the average daily expense claim by each employee to calculate the estimated value of claims that each employee is expected to make in the remaining week.

weekname

This function returns a value showing the year and week number with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the week containing **date**.

Syntax:

```
WeekName(date[, period_no [, first_week_day [, broken_weeks [, reference_day]]]])
```

The `weekname()` function determines which week the date falls into and returns the week number and year of that week. The first day of the week is determined by the `Firstweekday` system variable. However, you can also change the first day of the week by using the `first_week_day` argument in the `weekname()` function.

In Qlik Sense, the regional settings are fetched when the app is created, and the corresponding settings are stored in the script as environment variables.

A North American app developer often gets `set Brokenweeks=1;` in the script, corresponding to broken weeks. A European app developer often gets `set Brokenweeks=0;` in the script, corresponding to unbroken weeks.

If your application uses broken weeks, the week number count begins on the January 1 and ends on the day prior to the `Firstweekday` system variable regardless of how many days have occurred.

However, if your application is using unbroken weeks, week 1 can begin in the previous year or in the first few days in January. This depends on how you use the `ReferenceDay` and `Firstweekday` system variables.

Example of Weekname function

Date	ISO week name	US week name
Sat 2020 Dec 26	2020/52	2020/52
Sun 2020 Dec 27	2020/52	2020/53
Mon 2020 Dec 28	2020/53	2020/53
Tue 2020 Dec 29	2020/53	2020/53
Wed 2020 Dec 30	2020/53	2020/53
Thu 2020 Dec 31	2020/53	2020/53
Fri 2021 Jan 1	2020/53	2021/01
Sat 2021 Jan 2	2020/53	2021/01
Sun 2021 Jan 3	2020/53	2021/02
Mon 2021 Jan 4	2021/01	2021/02
Tue 2021 Jan 5	2021/01	2021/02

When to use it

The `weekname()` function is useful for when you would like to compare aggregations by weeks.

For example, if you want to see the total sales of products by week. To maintain consistency with the `BrokenWeeks` environment variable in the application, use `weekname()` instead of `lunarweekname()`. If the application is using unbroken weeks, week 1 may contain dates from December of the previous year or exclude dates in January of the current year. If the application is using broken weeks, week 1 may contain less than seven days.

Return data type: dual

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
period_no	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.
first_week_day	Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used. The possible values first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday. For more information about the system variable, see <i>FirstWeekDay</i> (page 215).
broken_weeks	If you don't specify broken_weeks , the value of variable BrokenWeeks will be used to define if weeks are broken or not.
reference_day	If you don't specify reference_day , the value of variable ReferenceDay will be used to define which day in January to set as reference day to define week 1. By default, Qlik Sense functions use 4 as the reference day. This means that week 1 must contain January 4, or put differently, that week 1 must always have at least 4 days in January.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

The examples below assume:

```
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

Function examples

Example	Result
weekname('01/12/2013')	Returns 2013/02.
weekname('01/12/2013', -1)	Returns 2013/01.
weekname('01/12/2013', 0, 1)	Returns 2013/02.

Example 1 – Date with no additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2021 and first two weeks of 2022 is loaded into a table called ‘Transactions’.
- The `DateFormat` system variable which is set to the `MM/DD/YYYY` format.
- The `BrokenWeeks` system variable which is set to 1.
- The `Firstweekday` system variable which is set to 6.
- A preceding load which contains the following:
 - The `weekday()` function which is set as the field, ‘`week_number`’, that returns the year and week number when the transactions took place.
 - The `weekname()` function which is set as the field called ‘`week_day`’, to show the weekday value of each transaction date.

Load script

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;

Transactions:
Load
  *,
  WeekDay(date) as week_day,
  Weekname(date) as week_number
;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
```

```
8186,12/30/2021,82.06  
8187,12/31/2021,40.56  
8188,01/01/2022,37.23  
8189,01/02/2022,17.17  
8190,01/03/2022,88.27  
8191,01/04/2022,57.42  
8192,01/05/2022,53.80  
8193,01/06/2022,82.06  
8194,01/07/2022,40.56  
8195,01/08/2022,53.67  
8196,01/09/2022,26.63  
8197,01/10/2022,72.48  
8198,01/11/2022,18.37  
8199,01/12/2022,45.26  
8200,01/13/2022,58.23  
8201,01/14/2022,18.52  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **week_day**
- **week_number**

Results table

id	date	week_day	week_number
8183	12/27/2021	Mon	2021/53
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/02
8190	01/03/2022	Mon	2022/02
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02

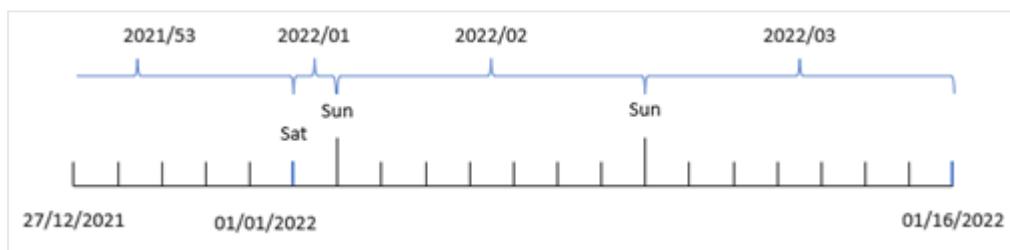
id	date	week_day	week_number
8196	01/09/2022	Sun	2022/03
8197	01/10/2022	Mon	2022/03
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

The ‘week_number’ field is created in the preceding load statement by using the `weekname()` function and passing the date field as the function’s argument.

The `weekname()` function initially identifies which week the date value falls into and returns the week number count and the year the transaction takes place.

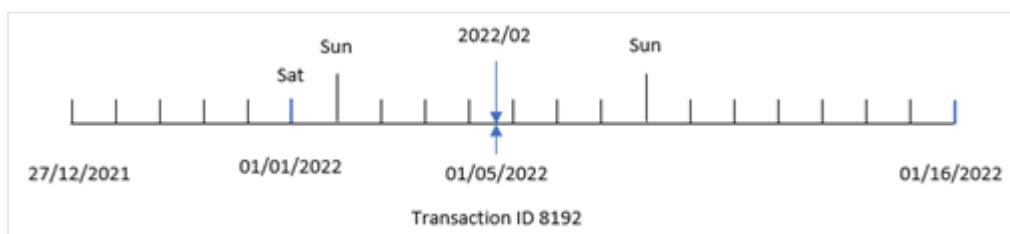
The `Firstweekday` system variable sets Sunday as the first day of the week. The `Brokenweeks` system variable sets the application to use broken weeks, meaning that week 1 will begin on January 1.

Diagram of `weekname()` function with the default variables.



Week 1 begins on January 1, which is a Saturday, and therefore transactions occurring on this date return the value 2022/01 (the year and week number).

Diagram of `weekname()` function identifying the week number of transaction 8192.



Because the application is using broken weeks and the first weekday is Sunday, transactions occurring from January 2 to January 8 return the value 2022/02 (week number 2 in 2022.) An example of this would be transaction 8192 which took place on January 5 and returns the value 2022/02 for the ‘week_number’ field.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the task is to create a field, ‘previous_week_number’, that returns the year, and week number, prior to when the transactions took place.

Open the Data load editor and add the following load script to a new tab.

Load script

```
SET BrokenWeeks=1;
SET FirstWeekDay=6;

Transactions:
  Load
    *,
    weekname(date,-1) as previous_week_number
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8183,12/27/2021,58.27
  8184,12/28/2021,67.42
  8185,12/29/2021,23.80
  8186,12/30/2021,82.06
  8187,12/31/2021,40.56
  8188,01/01/2022,37.23
  8189,01/02/2022,17.17
  8190,01/03/2022,88.27
  8191,01/04/2022,57.42
  8192,01/05/2022,53.80
  8193,01/06/2022,82.06
  8194,01/07/2022,40.56
  8195,01/08/2022,53.67
  8196,01/09/2022,26.63
  8197,01/10/2022,72.48
  8198,01/11/2022,18.37
  8199,01/12/2022,45.26
  8200,01/13/2022,58.23
  8201,01/14/2022,18.52
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

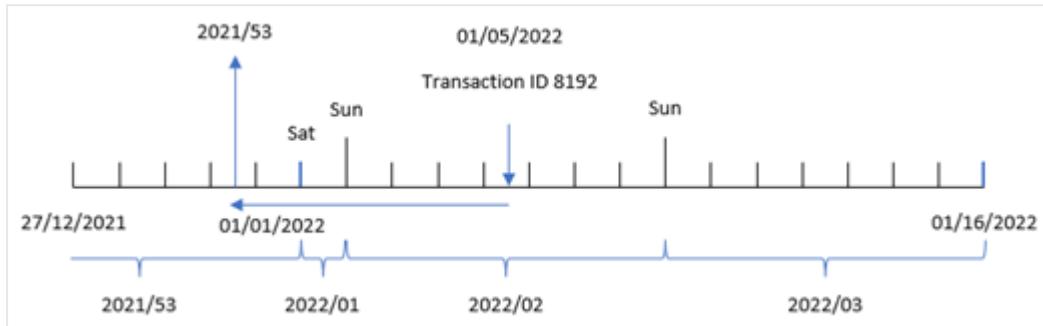
- `id`
- `date`
- `week_day`
- `week_number`

Results table

id	date	week_day	week_number
8183	12/27/2021	Mon	2021/52
8184	12/28/2021	Tue	2021/52
8185	12/29/2021	Wed	2021/52
8186	12/30/2021	Thu	2021/52
8187	12/31/2021	Fri	2021/52
8188	01/01/2022	Sat	2021/52
8189	01/02/2022	Sun	2021/53
8190	01/03/2022	Mon	2021/53
8191	01/04/2022	Tue	2021/53
8192	01/05/2022	Wed	2021/53
8193	01/06/2022	Thu	2021/53
8194	01/07/2022	Fri	2021/53
8195	01/08/2022	Sat	2022/01
8196	01/09/2022	Sun	2022/02
8197	01/10/2022	Mon	2022/02
8198	01/11/2022	Tue	2022/02
8199	01/12/2022	Wed	2022/02
8200	01/13/2022	Thu	2022/02
8201	01/14/2022	Fri	2022/02

Because a `period_no` of -1 is used as the offset argument in the `weekname()` function, the function first identifies the week that the transactions take place in. It then looks one week prior and identifies the first millisecond of that week.

Diagram of weekname() function with a period_no offset of -1.



Transaction 8192 took place on January 5, 2022. The weekname() function looks one week prior, December 30, 2021, and returns the week number and year for that date – 2021/53.

Example 3 – first_week_day

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the company policy is for the work week to begin on Tuesday.

Open the Data load editor and add the following load script to a new tab.

Load script

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  weekday(date) as week_day,
  weekname(date,0,1) as week_number
;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
```

```
8193,01/06/2022,82.06  
8194,01/07/2022,40.56  
8195,01/08/2022,53.67  
8196,01/09/2022,26.63  
8197,01/10/2022,72.48  
8198,01/11/2022,18.37  
8199,01/12/2022,45.26  
8200,01/13/2022,58.23  
8201,01/14/2022,18.52  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

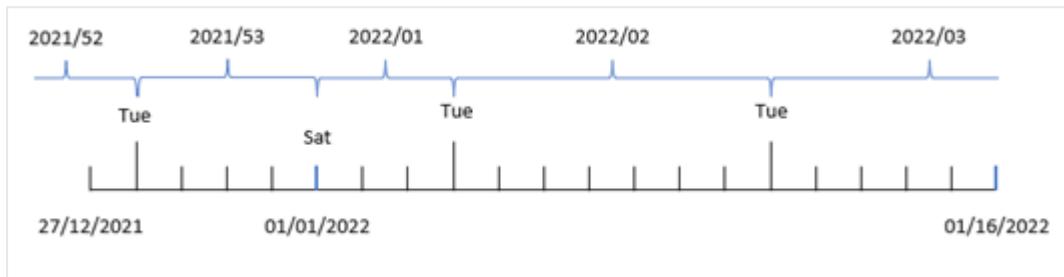
- **id**
- **date**
- **week_day**
- **week_number**

Results table

id	date	week_day	week_number
8183	12/27/2021	Mon	2021/52
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/01
8190	01/03/2022	Mon	2022/01
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02
8196	01/09/2022	Sun	2022/02
8197	01/10/2022	Mon	2022/02
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03

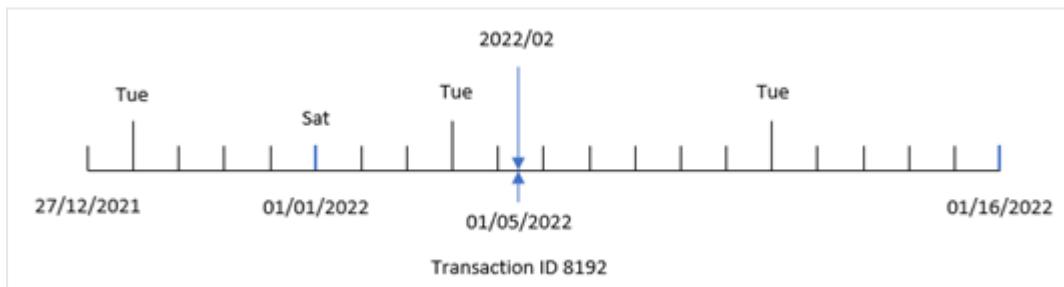
id	date	week_day	week_number
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

Diagram of weekname() function with Tuesday as the first day of the week.



Because the `first_week_date` argument of 1 is used in the `weekname()` function, it uses Tuesday as the first day of the week. The function therefore determines that week 53 of 2021 begins on Tuesday December 28; and, due to the application using broken weeks, week 1 begins on January 1, 2022, and ends on the last millisecond of Monday January 3, 2022.

Diagram showing week number of transaction 8192 with Tuesday as the first day of week.



Transaction 8192 took place on January 5, 2022. Therefore, using a `first_week_day` parameter of Tuesday, the `weekname()` function returns the value 2022/02 for the 'week_number' field.

Example 4 – Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that returns the year number of the week for when the transactions took place is created as a measure in a chart object of the application.

Load script

```
SET BrokenWeeks=1;
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- =week_day (date)

To calculate the start of the week that a transaction takes place in, create the following measure:

=weekname(date)

Results table

id	date	=weekday(date)	=weekname(date)
8183	12/27/2021	Mon	2021/53
8184	12/28/2021	Tue	2021/53
8185	12/29/2021	Wed	2021/53
8186	12/30/2021	Thu	2021/53

id	date	=weekday(date)	=weekname(date)
8187	12/31/2021	Fri	2021/53
8188	01/01/2022	Sat	2022/01
8189	01/02/2022	Sun	2022/02
8190	01/03/2022	Mon	2022/02
8191	01/04/2022	Tue	2022/02
8192	01/05/2022	Wed	2022/02
8193	01/06/2022	Thu	2022/02
8194	01/07/2022	Fri	2022/02
8195	01/08/2022	Sat	2022/02
8196	01/09/2022	Sun	2022/03
8197	01/10/2022	Mon	2022/03
8198	01/11/2022	Tue	2022/03
8199	01/12/2022	Wed	2022/03
8200	01/13/2022	Thu	2022/03
8201	01/14/2022	Fri	2022/03

The ‘week_number’ field is created as a measure in the chart object by using the `weekname()` function and passing the date field as the function’s argument.

The `weekname()` function initially identifies which week the date value falls into and returns the week number count and the year that the transaction takes place.

The `FirstWeekDay` system variable sets Sunday as the first day of the week. The `BrokenWeeks` system variable sets the application to use broken weeks, meaning that week 1 begins on January 1.

Diagram showing week number with Sunday as the first day of the week.

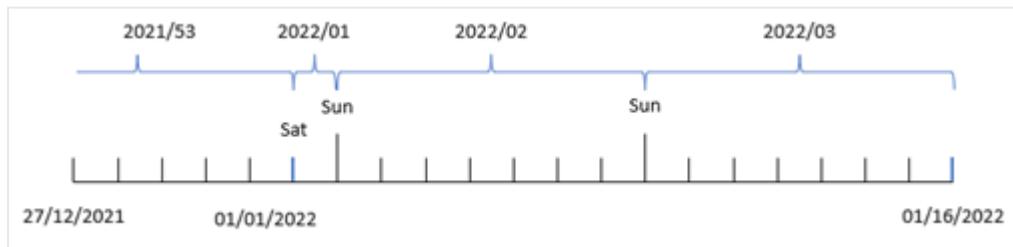
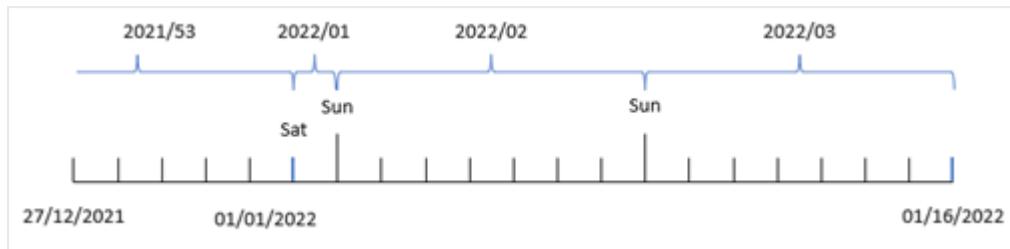


Diagram showing that transaction 8192 took place in week number two.



Because the application is using broken weeks and the first weekday is Sunday, transactions occurring from January 2 to January 8 return the value 2022/02, week number 2 in 2022. Note that transaction 8192 took place on January 5 and returns the value 2022/02 for the 'week_number' field.

Example 5 – Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2019 and first two weeks of 2020 is loaded into a table called 'Transactions'.
- The `BrokenWeeks` system variable which is set to 0.
- The `ReferenceDay` system variable which is set to 2.
- The `DateFormat` system variable which is set to the `MM/DD/YYYY` format.

Load script

```
SET BrokenWeeks=0;  
SET ReferenceDay=2;  
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load  
*  
Inline  
[  
id,date,amount  
8183,12/27/2019,58.27  
8184,12/28/2019,67.42  
8185,12/29/2019,23.80  
8186,12/30/2019,82.06  
8187,12/31/2019,40.56  
8188,01/01/2020,37.23  
8189,01/02/2020,17.17  
8190,01/03/2020,88.27  
8191,01/04/2020,57.42  
8192,01/05/2020,53.80
```

```
8193,01/06/2020,82.06  
8194,01/07/2020,40.56  
8195,01/08/2020,53.67  
8196,01/09/2020,26.63  
8197,01/10/2020,72.48  
8198,01/11/2020,18.37  
8199,01/12/2020,45.26  
8200,01/13/2020,58.23  
8201,01/14/2020,18.52  
];
```

Results

Load the data and open a sheet. Create a new table.

Create a calculated dimension using the following expression:

```
=weekname(date)
```

To calculate total sales create the following aggregation measure:

```
=sum(amount)
```

Set the measure's **Number Formatting** to **Money**.

Results table

weekname(date)	=sum(amount)
2019/52	\$125.69
2020/01	\$346.51
2020/02	\$347.57
2020/03	\$122.01

To demonstrate the results of using the weekname() function in this scenario, add the following field as a dimension:

```
date
```

Results table with date field

weekname(date)	date	=sum(amount)
2019/52	12/27/2019	\$58.27
2019/52	12/28/2019	\$67.42
2020/01	12/29/2019	\$23.80
2020/01	12/30/2019	\$82.06
2020/01	12/31/2019	\$40.56
2020/01	01/01/2020	\$37.23

weekname(date)	date	=sum(amount)
2020/01	01/02/2020	\$17.17
2020/01	01/03/2020	\$88.27
2020/01	01/04/2020	\$57.42
2020/02	01/05/2020	\$53.80
2020/02	01/06/2020	\$82.06
2020/02	01/07/2020	\$40.56
2020/02	01/08/2020	\$53.67
2020/02	01/09/2020	\$26.63
2020/02	01/10/2020	\$72.48
2020/02	01/11/2020	\$18.37
2020/03	01/12/2020	\$45.26
2020/03	01/13/2020	\$58.23
2020/03	01/14/2020	\$18.52

Because the application uses unbroken weeks, and week 1 requires a minimum of two days in January because of the `ReferenceDay` system variable, week 1 of 2020 includes transactions from December 29, 2019.

weekstart

This function returns a value corresponding to a timestamp of the first millisecond of the first day of the calendar week containing **date**. The default output format is the **DateFormat** set in the script.

Syntax:

```
WeekStart(timestamp [, period_no [, first_week_day ]])
```

Return data type:

The `weekstart()` function determines which week the date falls into. It then returns a timestamp, in date format, for the first millisecond of that week. The first day of the week is determined by the `Firstweekday` environment variable. However, this can be superseded by the `first_week_day` argument in the `weekstart()` function.

Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
period_no	shift is an integer, where the value 0 indicates the week which contains date . Negative values in shift indicate preceding weeks and positive values indicate succeeding weeks.

Argument	Description
first_week_day	<p>Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used.</p> <p>The possible values first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday.</p> <p>For more information about the system variable, see <i>FirstWeekDay</i> (page 215).</p>

When to use it

The `weekstart()` function is commonly used as part of an expression when the user would like the calculation to use the fraction of the week that has elapsed thus far. For example, it could be used if a user would like to calculate the total wages earned by employees in the week so far.

The following examples assume:

```
SET FirstWeekDay=0;
```

Function examples

Example	Result
<code>weekstart('01/12/2013')</code>	Returns 01/07/2013.
<code>weekstart('01/12/2013', -1)</code>	Returns 11/31/2012.
<code>weekstart('01/12/2013', 0, 1)</code>	Returns 01/08/2013.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Examples:

If you want ISO settings for weeks and week numbers, make sure to have the following in the script:

```
Set DateFormat = 'YYYY-MM-DD';
Set FirstWeekDay =0; // Monday as first week day
Set BrokenWeeks =0; //(use unbroken weeks)
Set ReferenceDay =4; // Jan 4th is always in week 1
```

If you want US settings, make sure to have the following in the script:

```
Set DateFormat = 'M/D/YYYY';
Set FirstWeekDay =6; // Sunday as first week day
Set BrokenWeeks =1; //(use broken weeks)
Set ReferenceDay =1; // Jan 1st is always in week 1
```

The examples above results in the following from the weekstart() function:

Example of Weekstart function

Date	ISO week start	US week start
Sat 2020 Dec 26	2020-12-21	12/20/2020
Sun 2020 Dec 27	2020-12-21	12/27/2020
Mon 2020 Dec 28	2020-12-28	12/27/2020
Tue 2020 Dec 29	2020-12-28	12/27/2020
Wed 2020 Dec 30	2020-12-28	12/27/2020
Thu 2020 Dec 31	2020-12-28	12/27/2020
Fri 2021 Jan 1	2020-12-28	12/27/2020
Sat 2021 Jan 2	2020-12-28	12/27/2020
Sun 2021 Jan 3	2020-12-28	1/3/2021
Mon 2021 Jan 4	2021-01-04	1/3/2021
Tue 2021 Jan 5	2021-01-04	1/3/2021



The week starts are on Mondays in the ISO column, and on Sundays in the US column.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `start_of_week`, that returns a timestamp for the start of the week when the transactions took place.

Load script

```
SET FirstWeekDay=6;
```

Transactions:

```

Load
  *,
  weekstart(date) as start_of_week,
  timestamp(weekstart(date)) as start_of_week_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];

```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- start_of_week
- start_of_week_timestamp

Results table

date	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM

date	start_of_week	start_of_week_timestamp
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

The `start_of_week` field is created in the preceding load statement by using the `weekstart()` function and passing the date field as the function's argument.

The `weekstart()` function initially identifies which week the date value falls into, returning a timestamp for the first millisecond of that week.

Diagram of weekstart() function, example with no additional arguments



Transaction 8191 took place on February 5. The `Firstweekday` system variable sets the first day of the week to a Sunday. The `weekstart()` function identifies that the first Sunday before February 5 – and therefore the start of the week – was on January 30. Therefore, the `start_of_week` value for that transaction returns the first millisecond of that day, which is January 30 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, previous_week_start, that returns the timestamp for the start of the quarter before the transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  weekstart(date,-1) as previous_week_start,
  timestamp(weekstart(date,-1)) as previous_week_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- previous_week_start
- previous_week_start_timestamp

Results table

date	previous_week_start	previous_week_start_timestamp
1/7/2022	12/26/2021	12/26/2021 12:00:00 AM
1/19/2022	01/09/2022	1/9/2022 12:00:00 AM
2/5/2022	01/23/2022	1/23/2022 12:00:00 AM
2/28/2022	02/20/2022	2/20/2022 12:00:00 AM
3/16/2022	03/06/2022	3/6/2022 12:00:00 AM
4/1/2022	03/20/2022	3/20/2022 12:00:00 AM
5/7/2022	04/24/2022	4/24/2022 12:00:00 AM
5/16/2022	05/08/2022	5/8/2022 12:00:00 AM
6/15/2022	06/05/2022	6/5/2022 12:00:00 AM
6/26/2022	06/19/2022	6/19/2022 12:00:00 AM
7/9/2022	06/26/2022	6/26/2022 12:00:00 AM
7/22/2022	07/10/2022	7/10/2022 12:00:00 AM
7/23/2022	07/10/2022	7/10/2022 12:00:00 AM
7/27/2022	07/17/2022	7/17/2022 12:00:00 AM
8/2/2022	07/24/2022	7/24/2022 12:00:00 AM
8/8/2022	07/31/2022	7/31/2022 12:00:00 AM
8/19/2022	08/07/2022	8/7/2022 12:00:00 AM
9/26/2022	09/18/2022	9/18/2022 12:00:00 AM
10/14/2022	10/02/2022	10/2/2022 12:00:00 AM
10/29/2022	10/16/2022	10/16/2022 12:00:00 AM

In this instance, because a period_no of -1 was used as the offset argument in the weekstart() function, the function first identifies the week that the transactions take place in. It then looks one week prior and identifies the first millisecond of that week.

Diagram of `weekstart()` function, period_no example



Transaction 8196 took place on June 15. The `weekstart()` function identifies that the week begins on June 12. Therefore, the previous week began on June 5 at 12:00:00 AM; this is the value that is returned for the `previous_week_start` field.

Example 3 – first_week_day

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example. However, in this example, we need to set Tuesday as the first day of the work week.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  weekstart(date,0,1) as start_of_week,
  timestamp(weekstart(date,0,1)) as start_of_week_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- start_of_week
- start_of_week_timestamp

Results table

date	start_of_week	start_of_week_timestamp
1/7/2022	01/04/2022	1/4/2022 12:00:00 AM
1/19/2022	01/18/2022	1/18/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/22/2022	2/22/2022 12:00:00 AM
3/16/2022	03/15/2022	3/15/2022 12:00:00 AM
4/1/2022	03/29/2022	3/29/2022 12:00:00 AM
5/7/2022	05/03/2022	5/3/2022 12:00:00 AM
5/16/2022	05/10/2022	5/10/2022 12:00:00 AM
6/15/2022	06/14/2022	6/14/2022 12:00:00 AM
6/26/2022	06/21/2022	6/21/2022 12:00:00 AM
7/9/2022	07/05/2022	7/5/2022 12:00:00 AM
7/22/2022	07/19/2022	7/19/2022 12:00:00 AM
7/23/2022	07/19/2022	7/19/2022 12:00:00 AM
7/27/2022	07/26/2022	7/26/2022 12:00:00 AM
8/2/2022	08/02/2022	8/2/2022 12:00:00 AM
8/8/2022	08/02/2022	8/2/2022 12:00:00 AM
8/19/2022	08/16/2022	8/16/2022 12:00:00 AM
9/26/2022	09/20/2022	9/20/2022 12:00:00 AM

date	start_of_week	start_of_week_timestamp
10/14/2022	10/11/2022	10/11/2022 12:00:00 AM
10/29/2022	10/25/2022	10/25/2022 12:00:00 AM

In this instance, because the `first_week_date` argument of 1 is used in the `weekstart()` function, it sets the first day of the week to Tuesday.

Diagram of weekstart() function, first_week_day example



Transaction 8191 took place on February 5. The `weekstart()` function identifies that the first Tuesday before this date – and therefore the start of the week and value returned – was February 1 at 12:00:00 AM.

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that returns a timestamp for the start of the week when the transactions took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89  
8198,7/9/2022,36.23  
8199,7/22/2022,25.66  
8200,7/23/2022,82.77  
8201,7/27/2022,69.98  
8202,8/2/2022,76.11  
8203,8/8/2022,25.12  
8204,8/19/2022,46.23  
8205,9/26/2022,84.21  
8206,10/14/2022,96.24  
8207,10/29/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

To calculate the start of the week in which a transaction takes place, add the following measures:

- `=weekstart(date)`
- `=timestamp(weekstart(date))`

Results table

date	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM

date	start_of_week	start_of_week_timestamp
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

The `start_of_week` measure is created in the chart object by using the `weekstart()` function and passing the date field as the function's argument.

The `weekstart()` function initially identifies which week the date value falls into, returning a timestamp for the first millisecond of that week.

Diagram of weekstart() function, chart object example



Transaction 8191 took place on February 5. The `Firstweekday` system variable sets the first day of the week to a Sunday. The `weekstart()` function identifies that the first Sunday before February 5 – and therefore the start of the week – was January 30. Therefore, the `start_of_week` value for that transaction returns the first millisecond of that day, which is January 30 at 12:00:00 AM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset which is loaded into a table called `Payroll1`.
- Data consisting of employee IDs, employee names, and the daily wage earned by each employee.

Employees begin work on Monday and work six days per week. The `Firstweekday` system variable must not be modified.

The end user would like a chart object that displays, by employee ID and employee name, the wages earned in the week to date.

Load script

```
Payroll1:  
Load
```

```

*
Inline
[
employee_id,employee_name,day_rate
182,Mark, $150
183,Deryck, $125
184,Dexter, $125
185,Sydney,$270
186,Agatha,$128
];

```

Results

Do the following:

1. Load the data and open a sheet. Create a new table and add these fields as dimensions:
 - employee_id
 - employee_name
2. Next, create a measure to calculate the wages earned in the week to date:
 $=\text{if}(\text{today}(1)-\text{weekstart}(\text{today}(1),0,0)<7, (\text{today}(1)-\text{weekstart}(\text{today}(1),0,0)) * \text{day_rate}, \text{day_rate} * 6)$
3. Set the measure's **Number formatting** to **Money**.

Results table

employee_id	employee_name	$=\text{if}(\text{today}(1)-\text{weekstart}(\text{today}(1),0,0)<7, (\text{today}(1)-\text{weekstart}(\text{today}(1),0,0)) * \text{day_rate}, \text{day_rate} * 6)$
182	Mark	\$600.00
183	Deryck	\$500.00
184	Dexter	\$500.00
185	Sydney	\$1080.00
186	Agatha	\$512.00

The `weekstart()` function, by using today's date as its first argument and 0 as its third argument, sets Monday as the first day of the week and returns the start date of the current week. By subtracting that result from the current date, the expression then returns the number of days that have elapsed so far this week.

The condition then evaluates whether there have been more than six days this week. If so, the employee's `day_rate` is multiplied by 6 days. Otherwise, the `day_rate` is multiplied by the number of days that have occurred so far this week.

weekyear

This function returns the year to which the week number belongs according to the environment variables. The week number ranges between 1 and approximately 52.

Syntax:

```
weekyear(timestamp [, first_week_day [, broken_weeks [, reference_day]]])
```

Return data type: integer

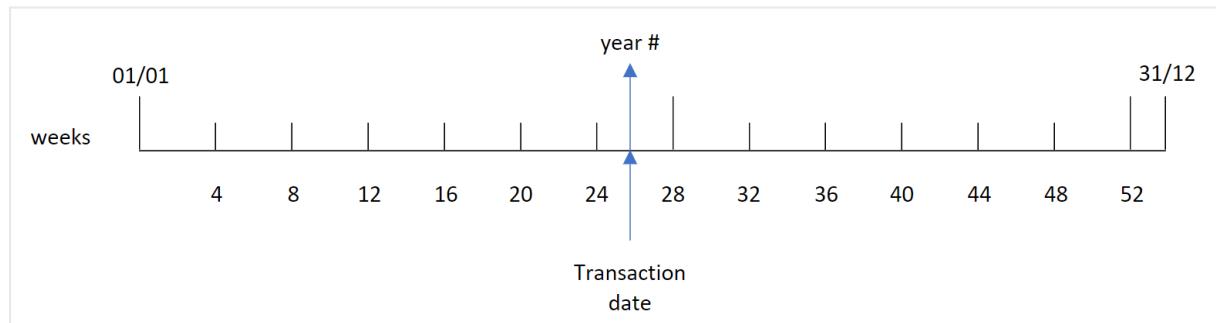
Arguments

Argument	Description
timestamp	The date or timestamp to evaluate.
first_week_day	<p>Specifies the day on which the week starts. If omitted, the value of variable FirstWeekDay is used.</p> <p>The possible values first_week_day are 0 for Monday, 1 for Tuesday, 2 for Wednesday, 3 for Thursday, 4 for Friday, 5 for Saturday, and 6 for Sunday.</p> <p>For more information about the system variable, see <i>FirstWeekDay</i> (page 215).</p>
broken_weeks	If you don't specify broken_weeks , the value of variable BrokenWeeks will be used to define if weeks are broken or not.
reference_day	If you don't specify reference_day , the value of variable ReferenceDay will be used to define which day in January to set as reference day to define week 1. By default, Qlik Sense functions use 4 as the reference day. This means that week 1 must contain January 4, or put differently, that week 1 must always have at least 4 days in January.

The `weekyear()` function determines which week of a year a date falls into. It then returns the year corresponding to that week number.

If `brokenweeks` is set to 0 (false), `weekyear()` will return the same as `year()`.

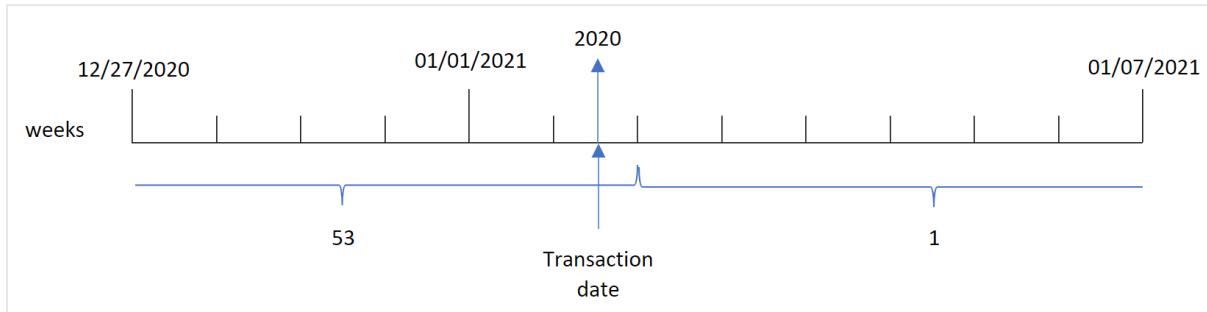
Diagram of weekyear() function's range



However, if the `BrokenWeeks` system variable is set to use unbroken weeks, week 1 must only contain a certain number of days in January based on the value specified in the `ReferenceDay` system variable.

For example, if a `ReferenceDay` value of 4 is used, week 1 must include at least four days in January. It is possible for week 1 to include dates in December of the previous year or for the final week number of a year to include dates in January of the following year. In situations like this, the `weekyear()` function will return a different value to the `year()` function.

Diagram of weekyear() function's range when using unbroken weeks



When to use it

The weekyear() function is useful when you would like to compare aggregations by years. For example, if you would like to see the total sales of products by year. The weekyear() function is chosen over year() when the user would like to retain consistency with the `BrokenWeeks` system variable in the app.

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>weekyear('12/30/1996',0,0,4)</code>	Returns 1997, because week 1 of 1997 starts on 12/30/1996
<code>weekyear('01/02/1997',0,0,4)</code>	Returns 1997
<code>weekyear('12/28/1997',0,0,4)</code>	Returns 1997
<code>weekyear('12/30/1997',0,0,4)</code>	Returns 1998, because week 1 of 1998 starts on 12/29/1997
<code>weekyear('01/02/1999',0,0,4)</code>	Returns 1998, because week 53 of 1998 ends on 01/03/1999

Related topics

Topic	Interaction
<i>week</i> (page 1021)	Returns an integer representing the week number according to ISO 8601
<i>year</i> (page 1094)	Returns an integer representing the year when the expression is interpreted as a date according to the standard number interpretation.

Example 1 - Broken weeks

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2020 and first week of 2021 which is loaded into a table called ‘Transactions’.
- The `Brokenweeks` variable which is set to 1.
- A preceding load which contains the following:
 - The `weekyear()` function, set as the field ‘`week_year`’ that returns the year in which the transactions took place.
 - The `week()` function, set as the field ‘`week`’ that shows the week number of each transaction date.

Load script

```
SET BrokenWeeks=1;

Transactions:
  Load
  *,
  week(date) as week,
  weekyear(date) as week_year
  ;
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
```

```
8182,01/03/2021,88.27  
8183,01/04/2021,57.42  
8184,01/05/2021,67.42  
8185,01/06/2021,23.80  
8186,01/07/2021,82.06  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **week**
- **week_year**

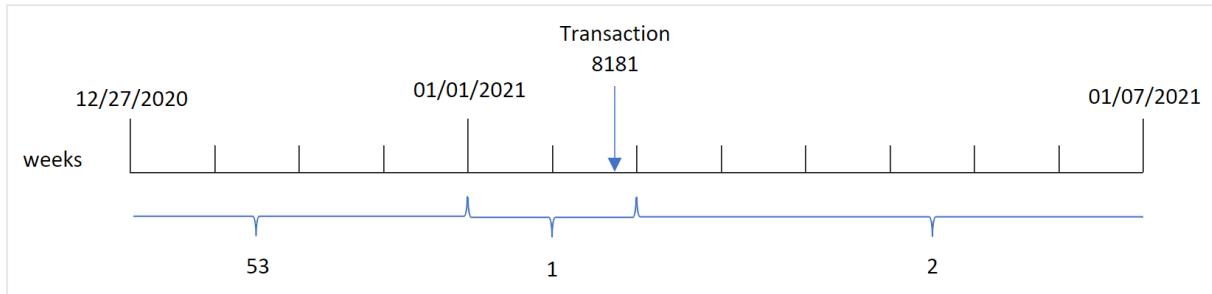
Results table

id	date	week	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

The ‘week_year’ field is created in the preceding load statement by using the `weekyear()` function and passing the date field as the function’s argument.

The `BrokenWeeks` system variable is set to 1 meaning that the app uses broken weeks. Week 1 begins on January 1.

Diagram of `weekyear()` function's range with the use of broken weeks



Transaction 8181 takes place on January 2, which is part of week 1. Therefore, it returns a value of 2021 for the 'week_year' field.

Example 2 - Unbroken weeks

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2020 and first week of 2021 which is loaded into a table called 'Transactions'.
- The `BrokenWeeks` variable which is set to 0.
- A preceding load which contains the following:
 - The `weekyear()` function, set as the field 'week_year' that returns the year in which the transactions took place.
 - The `week()` function, set as the field 'week' that shows the week number of each transaction date.

However, in this example, the company policy is to use unbroken weeks.

Load script

```
SET BrokenWeeks=0;

Transactions:
  Load
  *,
  week(date) as week,
  weekyear(date) as week_year
;
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
```

```
8177,12/29/2020,23.80  
8178,12/30/2020,82.06  
8179,12/31/2020,40.56  
8180,01/01/2021,37.23  
8181,01/02/2021,17.17  
8182,01/03/2021,88.27  
8183,01/04/2021,57.42  
8184,01/05/2021,67.42  
8185,01/06/2021,23.80  
8186,01/07/2021,82.06  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **week**
- **week_year**

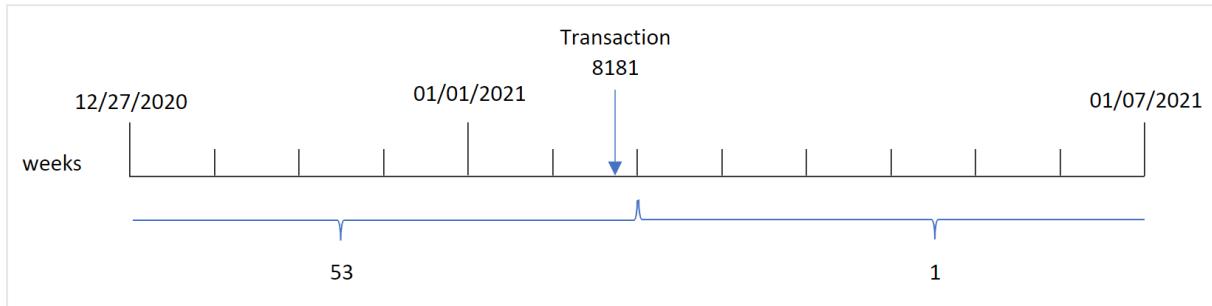
Results table

id	date	week	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	53	2020
8181	01/02/2021	53	2020
8182	01/03/2021	1	2021
8183	01/04/2021	1	2021
8184	01/05/2021	1	2021
8185	01/06/2021	1	2021
8186	01/07/2021	1	2021

The `BrokenWeeks` system variable is set to 0 meaning that the application uses unbroken weeks. Therefore, week 1 is not required to begin on January 1.

Week 53 of 2020 continues until the end of January 2, 2021, with week 1 of 2020 beginning on Sunday, January 3, 2021.

Diagram of weekyear() function's range with the use of unbroken weeks



Transaction 8181 takes place on January 2, which is part of week 1. Therefore, it returns a value of 2021 for the ‘week_year’ field.

Example 3 - Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example the dataset is unchanged and loaded into the application. The calculation that returns the week number of the year when the transactions took place is created as a measure in a chart in the app.

Load script

```
SET BrokenWeeks=1;
```

```
Transactions:
```

```
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date

To calculate the week that a transaction takes place in, create the following measure:

- =week(date)

To calculate the year that a transaction takes place in based on the week number, create the following measure:

- =weekyear(date)

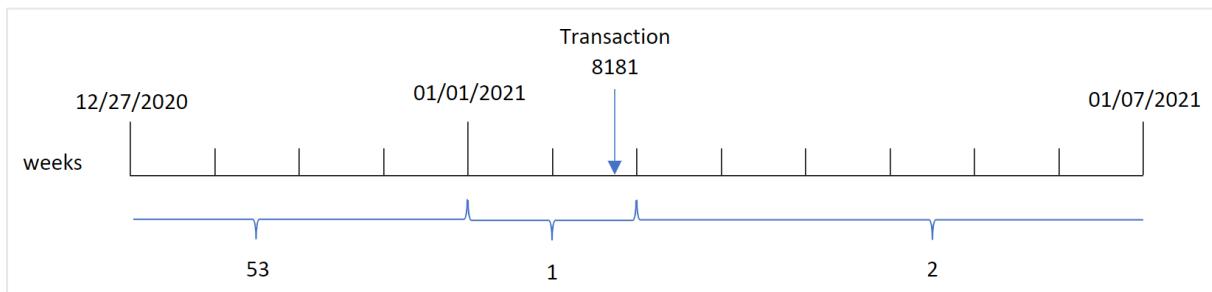
Results table

id	date	week	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

The ‘week_year’ field is created in the preceding load statement by using the `weekyear()` function and passing the date field as the function’s argument.

The `BrokenWeeks` system variable is set to 1 meaning that the app uses broken weeks. week 1 begins on January 1.

Diagram of `weekyear()` function's range with the use of broken weeks



Transaction 8181 takes place on January 2, which is part of week 1. Therefore, it returns a value of 2021 for the ‘week_year’ field.

Example 4 - Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions for the last week of 2020 and first week of 2021 which is loaded into a table called ‘Transactions’.
- The `BrokenWeeks` variable which is set to 0. This means the app will use unbroken weeks.
- The `ReferenceDay` variable which is set to 2. This means the year will begin on January 2 and will contain a minimum of two days in January.
- The `FirstWeekDay` variable which is set to 1. This means the first day of the week will be Tuesday.

The company policy is to use broken weeks. The end user would like a chart that presents the total sales by year. The app uses unbroken weeks with week 1 containing a minimum of two days in January.

Load script

```
SET BrokenWeeks=0;  
SET ReferenceDay=2;  
SET FirstWeekDay=1;
```

Transactions:

```
Load  
*  
Inline  
[  
id,date,amount  
8176,12/28/2020,19.42  
8177,12/29/2020,23.80  
8178,12/30/2020,82.06  
8179,12/31/2020,40.56  
8180,01/01/2021,37.23  
8181,01/02/2021,17.17  
8182,01/03/2021,88.27  
8183,01/04/2021,57.42  
8184,01/05/2021,67.42  
8185,01/06/2021,23.80  
8186,01/07/2021,82.06  
];
```

Results

Load the data and open a sheet. Create a new table.

To calculate the year that a transaction takes place in based on the week number, create the following measure:

- `=weekyear(date)`

To calculate total sales, create the following measure:

- `sum(amount)`

Set the measure's **Number Formatting** to **Money**.

Results table

<code>weekyear(date)</code>	<code>=sum(amount)</code>
2020	19.42
2021	373.37

year

This function returns an integer representing the year when the **expression** is interpreted as a date according to the standard number interpretation.

Syntax:

`year(expression)`

Return data type: integer

The `year()` function is available as both a script and chart function. The function returns the year for a particular date. It is commonly used to create a year field as a dimension in a Master Calendar.

When to use it

The `year()` function is useful when you would like to compare aggregations by year. For example, the function could be used if you would like to see the total sales of products by year.

These dimensions can be created either in the load script by using the function to create a field in a Master Calendar table. Alternatively, it could be used directly in a chart as a calculated dimension.

Function examples

Example	Result
<code>year('2012-10-12')</code>	returns 2012
<code>year('35648')</code>	returns 1997, because 35648 = 1997-08-06

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may

be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – DateFormat dataset (script)

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates, which is loaded into a table named `Master_Calendar`.
- The default `DateFormat` system variable (MM/DD/YYYY) is used.
- A preceding load, which is used to create an additional field, `year`, using the `year()` function.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Master_Calendar:
```

```
  Load
    date,
    year(date) as year
  ;
Load
date
Inline
[
date
12/28/2020
12/29/2020
12/30/2020
12/31/2020
01/01/2021
01/02/2021
01/03/2021
01/04/2021
01/05/2021
01/06/2021
01/07/2021
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- year

Results table

date	year
12/28/2020	2020
12/29/2020	2020
12/30/2020	2020
12/31/2020	2020
01/01/2021	2021
01/02/2021	2021
01/03/2021	2021
01/04/2021	2021
01/05/2021	2021
01/06/2021	2021
01/07/2021	2021

Example 2 – ANSI Dates

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates, which is loaded into a table named `Master_Calendar`.
- The default `DateFormat` system variable (MM/DD/YYYY) is used. However, the dates included in the dataset are in ANSI standard date format.
- A preceding load, which is used to create an additional field, named `year`, using the `year()` function.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Master_Calendar:
```

```
    Load
```

```
date,  
year(date) as year  
;  
Load  
date  
Inline  
[  
date  
2020-12-28  
2020-12-29  
2020-12-30  
2020-12-31  
2021-01-01  
2021-01-02  
2021-01-03  
2021-01-04  
2021-01-05  
2021-01-06  
2021-01-07  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- year

Results table

date	year
2020-12-28	2020
2020-12-29	2020
2020-12-30	2020
2020-12-31	2020
2021-01-01	2021
2021-01-02	2021
2021-01-03	2021
2021-01-04	2021
2021-01-05	2021
2021-01-06	2021
2021-01-07	2021

Example 3 – Unformatted dates

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset of dates in numerical format, which is loaded into a table named `Master_Calendar`.
- The default `DateFormat` system variable (MM/DD/YYYY) is used.
- A preceding load, which is used to create an additional field, `year`, using the `year()` function.

The original unformatted date is loaded, named `unformatted_date`, and to provide clarity, a further additional field, named `long_date`, is used to convert the numerical date into a formatted date field using the `date()` function.

Load script

```
SET DateFormat='MM/DD/YYYY';

Master_Calendar:
  Load
    unformatted_date,
    date(unformatted_date) as long_date,
    year(unformatted_date) as year
  ;
Load
unformatted_date
Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `unformatted_date`
- `long_date`
- `year`

Results table

unformatted_date	long_date	year
44868	11/03/2022	2022
44898	12/03/2022	2022
44928	01/02/2023	2023
44958	02/01/2023	2023
44988	03/03/2023	2023
45008	03/23/2023	2023
45018	04/02/2023	2023
45038	04/22/2023	2023
45048	05/02/2023	2023
45068	05/22/2023	2023
45078	06/01/2023	2023

Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

In this example, a dataset of orders placed is loaded into a table named Sales. The table contains three fields:

- `id`
- `sales_date`
- `amount`

Warranties on product sales last two years from the date of sale. The task is to create a measure in a chart to determine the year in which each warranty will expire.

Load script

```
Sales:  
Load  
id,  
sales_date,  
amount  
Inline
```

```
[  
id,sales_date,amount  
1,12/28/2020,231.24,  
2,12/29/2020,567.28,  
3,12/30/2020,364.28,  
4,12/31/2020,575.76,  
5,01/01/2021,638.68,  
6,01/02/2021,785.38,  
7,01/03/2021,967.46,  
8,01/04/2021,287.67  
9,01/05/2021,764.45,  
10,01/06/2021,875.43,  
11,01/07/2021,957.35  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: `sales_date`.

Create the following measure:

```
=year(sales_date+365*2)
```

Results table

<code>sales_date</code>	<code>=year(sales_date+365*2)</code>
12/28/2020	2022
12/29/2020	2022
12/30/2020	2022
12/31/2020	2022
01/01/2021	2023
01/02/2021	2023
01/03/2021	2023
01/04/2021	2023
01/05/2021	2023
01/06/2021	2023
01/07/2021	2023

The results of this measure can be seen in the table above. To add two years to a date, multiply 365 by 2 and add the result to the sales date. Therefore, sales that took place in 2020 have an expiry year of 2022.

`yearend`

This function returns a value corresponding to a timestamp of the last millisecond of the last day of the year containing `date`. The default output format will be the **DateFormat** set in the script.

Syntax:

```
YearEnd( date[, period_no[, first_month_of_year = 1]] )
```

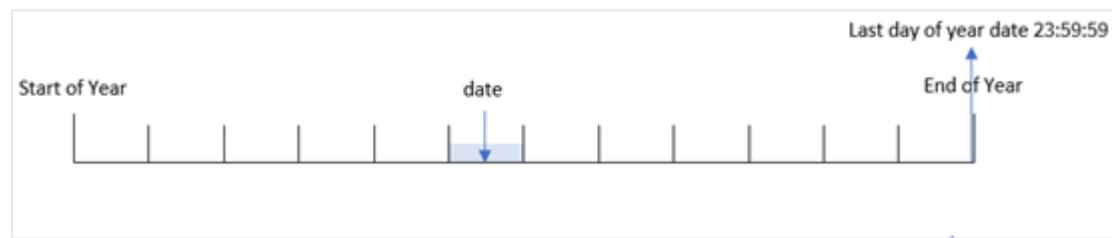
In other words, the `yearend()` function determines which year the date falls into. It then returns a timestamp, in date format, for the last millisecond of that year. The first month of the year is, by default, January.

However, you can change which month is set as first by using the `first_month_of_year` argument in the `yearend()` function.



The `yearend()` function does not consider the `FirstMonthofYear` system variable. The year begins on January 1 unless the `first_month_of_year` argument is used to change it.

Diagram of `yearend()` function.



When to use it

The `yearend()` function is used as part of an expression when you want the calculation to use the fraction of the year that has not yet occurred. For example, if you want to calculate the total interest not yet incurred during the year.

Return data type: dual

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the year which contains date . Negative values in period_no indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

You can use the following values to set the first month of year in the `first_month_of_year` argument:

`first_month_of_year` values

Month	Value
February	2
March	3

Month	Value
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>yearend('10/19/2001')</code>	Returns 12/31/2001 23:59:59.
<code>yearend('10/19/2001', -1)</code>	Returns 12/31/2000 23:59:59.
<code>yearend('10/19/2001', 0, 4)</code>	Returns 03/31/2002 23:59:59.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 is loaded into a table called ‘Transactions’.
- The date field has been provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- A preceding load statement which contains the following:
 - `yearend()` function which is set as the `year_end` field.
 - `Timestamp()` function which is set as the `year_end_timestamp` field.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yearend(date) as year_end,
  timestamp(yearend(date)) as year_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- `id`
- `date`

- year_end
- year_end_timestamp

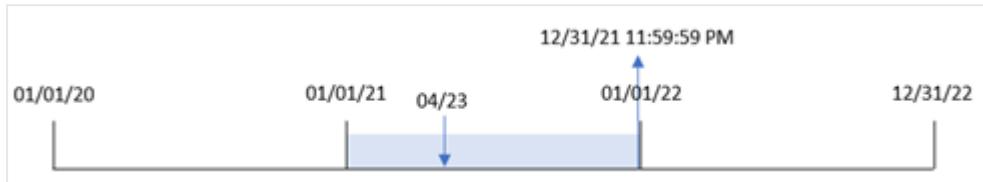
Results table

id	date	year_end	year_end_timestamp
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

The ‘year_end’ field is created in the preceding load statement by using the `yearend()` function and passing the date field as the function’s argument.

The `yearend()` function initially identifies which year the date value falls into and returns a timestamp for the last millisecond of that year.

Diagram of yearend() function with transaction 8199 selected.



Transaction 8199 took place on April 23, 2021. The yearend() function returns the last millisecond of that year, which is December 31 at 11:59:59 PM.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the task is to create a field, ‘previous_year_end’ , that returns the end date timestamp of the year prior to the year in which a transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yearend(date,-1) as previous_year_end,
  timestamp(yearend(date,-1)) as previous_year_end_timestamp
;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
```

```
8204,06/06/2022,46.23  
8205,07/18/2022,84.21  
8206,11/14/2022,96.24  
8207,12/12/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

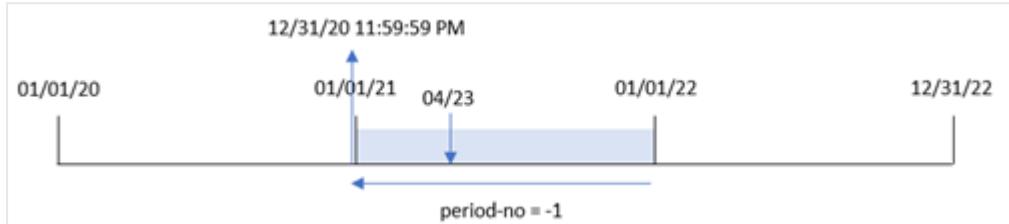
- **id**
- **date**
- **previous_year_end**
- **previous_year_end_timestamp**

Results table

id	date	previous_year_end	previous_year_end_timestamp
8188	01/13/2020	12/31/2019	12/31/2019 11:59:59 PM
8189	02/26/2020	12/31/2019	12/31/2019 11:59:59 PM
8190	03/27/2020	12/31/2019	12/31/2019 11:59:59 PM
8191	04/16/2020	12/31/2019	12/31/2019 11:59:59 PM
8192	05/21/2020	12/31/2019	12/31/2019 11:59:59 PM
8193	08/14/2020	12/31/2019	12/31/2019 11:59:59 PM
8194	10/07/2020	12/31/2019	12/31/2019 11:59:59 PM
8195	12/05/2020	12/31/2019	12/31/2019 11:59:59 PM
8196	01/22/2021	12/31/2020	12/31/2020 11:59:59 PM
8197	02/03/2021	12/31/2020	12/31/2020 11:59:59 PM
8198	03/17/2021	12/31/2020	12/31/2020 11:59:59 PM
8199	04/23/2021	12/31/2020	12/31/2020 11:59:59 PM
8200	05/04/2021	12/31/2020	12/31/2020 11:59:59 PM
8201	06/30/2021	12/31/2020	12/31/2020 11:59:59 PM
8202	07/26/2021	12/31/2020	12/31/2020 11:59:59 PM
8203	12/27/2021	12/31/2020	12/31/2020 11:59:59 PM
8204	06/06/2022	12/31/2021	12/31/2021 11:59:59 PM
8205	07/18/2022	12/31/2021	12/31/2021 11:59:59 PM
8206	11/14/2022	12/31/2021	12/31/2021 11:59:59 PM
8207	12/12/2022	12/31/2021	12/31/2021 11:59:59 PM

Because a period_no of -1 was used as the offset argument in the yearend() function, the function first identifies the year that the transactions take place in. It then looks one year prior and identifies the last millisecond of that year.

Diagram of yearend() function with a period_no of -1.



Transaction 8199 takes place on April 23, 2021. The yearend() function returns the last millisecond of the prior year, December 31, 2020 at 11:59:59 PM, for the 'previous_year_end' field.

Example 3 – first_month_of_year

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the company policy is for the year to begin from April 1.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    yearend(date,0,4) as year_end,
    timestamp(yearend(date,0,4)) as year_end_timestamp
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,01/13/2020,37.23
  8189,02/26/2020,17.17
  8190,03/27/2020,88.27
  8191,04/16/2020,57.42
  8192,05/21/2020,53.80
  8193,08/14/2020,82.06
  8194,10/07/2020,40.39
  8195,12/05/2020,87.21
  8196,01/22/2021,95.93
  8197,02/03/2021,45.89
  8198,03/17/2021,36.23
```

```
8199,04/23/2021,25.66  
8200,05/04/2021,82.77  
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,06/06/2022,46.23  
8205,07/18/2022,84.21  
8206,11/14/2022,96.24  
8207,12/12/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **year_end**
- **year_end_timestamp**

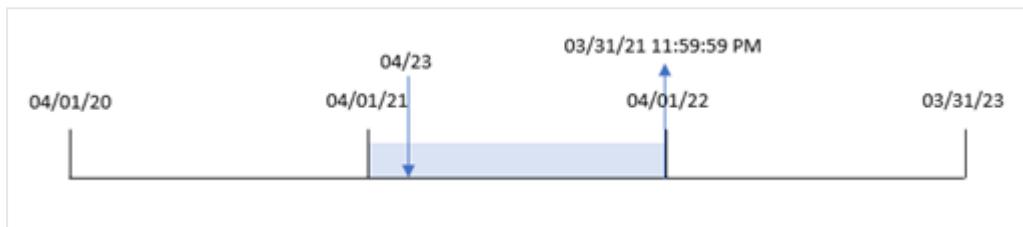
Results table

id	date	year_end	year_end_timestamp
8188	01/13/2020	03/31/2020	3/31/2020 11:59:59 PM
8189	02/26/2020	03/31/2020	3/31/2020 11:59:59 PM
8190	03/27/2020	03/31/2020	3/31/2020 11:59:59 PM
8191	04/16/2020	03/31/2021	3/31/2021 11:59:59 PM
8192	05/21/2020	03/31/2021	3/31/2021 11:59:59 PM
8193	08/14/2020	03/31/2021	3/31/2021 11:59:59 PM
8194	10/07/2020	03/31/2021	3/31/2021 11:59:59 PM
8195	12/05/2020	03/31/2021	3/31/2021 11:59:59 PM
8196	01/22/2021	03/31/2021	3/31/2021 11:59:59 PM
8197	02/03/2021	03/31/2021	3/31/2021 11:59:59 PM
8198	03/17/2021	03/31/2021	3/31/2021 11:59:59 PM
8199	04/23/2021	03/31/2022	3/31/2022 11:59:59 PM
8200	05/04/2021	03/31/2022	3/31/2022 11:59:59 PM
8201	06/30/2021	03/31/2022	3/31/2022 11:59:59 PM
8202	07/26/2021	03/31/2022	3/31/2022 11:59:59 PM
8203	12/27/2021	03/31/2022	3/31/2022 11:59:59 PM
8204	06/06/2022	03/31/2023	3/31/2023 11:59:59 PM

id	date	year_end	year_end_timestamp
8205	07/18/2022	03/31/2023	3/31/2023 11:59:59 PM
8206	11/14/2022	03/31/2023	3/31/2023 11:59:59 PM
8207	12/12/2022	03/31/2023	3/31/2023 11:59:59 PM

Because the `first_month_of_year` argument of 4 is used in the `yearend()` function, it sets the first day of the year to April 1, and the last day of the year to March 31.

Diagram of yearend() function with April as the first month of the year.



Transaction 8199 takes place on April 23, 2021. Because the `yearend()` function sets the start of the year to April 1, it returns March 31, 2022 as the 'year_end' value for the transaction.

Example 4 – Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that returns the end date timestamp of the year in which a transaction took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
```

```
8197,02/03/2021,45.89  
8198,03/17/2021,36.23  
8199,04/23/2021,25.66  
8200,05/04/2021,82.77  
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,06/06/2022,46.23  
8205,07/18/2022,84.21  
8206,11/14/2022,96.24  
8207,12/12/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date

To calculate in which year a transaction took place, create the following measures:

- =yearend(date)
- =timestamp(yearend(date))

Results table

id	date	=yearend(date)	=timestamp(yearend(date))
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM

id	date	=yearend(date)	=timestamp(yearend(date))
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

The ‘end_of_year’ measure is created in the chart object by using the yearend() function and passing the date field as the function’s argument.

The yearend() function initially identifies which year the date value falls into returning a timestamp for the last millisecond of that year.

Diagram of yearend() function that shows Transaction 8199 took place in April.



Transaction 8199 takes place on April 23, 2021. The yearend() function returns the last millisecond of that year, which is December 31 at 11:59:59 PM.

Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset is loaded into a table called ‘Employee_Expenses’. The table contains the following fields:
 - employee IDs
 - employee name
 - average daily expense claims of each employee

The end user would like a chart object that displays, by employee id and employee name, the estimated expense claims still to be incurred for the remainder of the year. The financial year begins in January.

Load script

```
Employee_Expenses:
Load
```

```
*  
Inline  
[  
employee_id,employee_name,avg_daily_claim  
182,Mark, $15  
183,Deryck, $12.5  
184,Dexter, $12.5  
185,Sydney,$27  
186,Agatha,$18  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- employee_id
- employee_name

To calculate the projected expense claims, create the following measure:

```
=yearend(today(1))-today(1)*avg_daily_claim
```

Set the measure's **Number Formatting** to **Money**.

Results table

employee_id	employee_name	=yearend(today(1))-today(1)*avg_daily_claim
182	Mark	\$3240.00
183	Deryck	\$2700.00
184	Dexter	\$2700.00
185	Sydney	\$5832.00
186	Agatha	\$3888.00

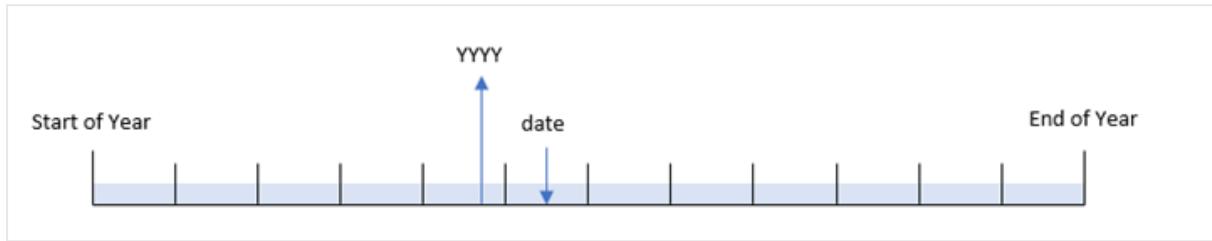
By using today's date as its only argument, the yearend() function returns the end date of the current year. Then, by subtracting today's date from the year end date, the expression returns the number of days remaining in this year.

This value is then multiplied by the average daily expense claim by each employee to calculate the estimated value of claims each employee is expected to make in the remaining year.

yearname

This function returns a four-digit year as display value with an underlying numeric value corresponding to a timestamp of the first millisecond of the first day of the year containing **date**.

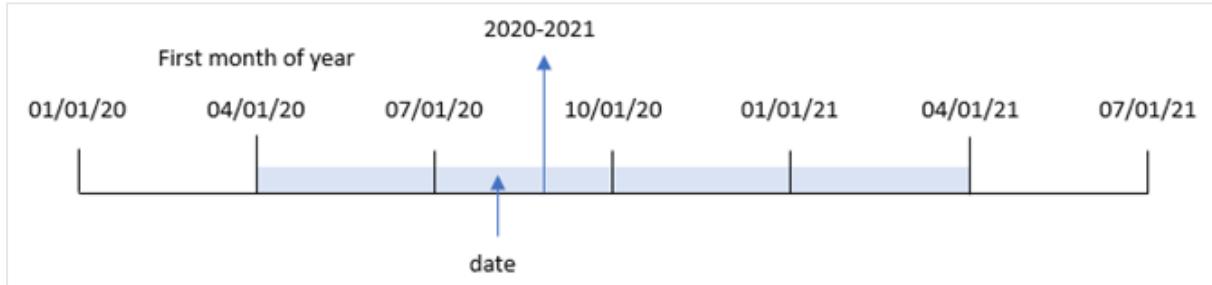
Diagram of range of time of the yearname() function.



The yearname() function is different to the year() function as it lets you offset the date you want evaluated and lets you set the first month of the year.

If the first month of the year is not January, the function will return the two four-digit years across the twelve month period that contain the date. For example, if the start of the year is April and the date being evaluated is 06/30/2020, the result returned would be 2020–2021.

Diagram of yearname() function with April set as the first month of the year.



Syntax:

```
YearName(date[, period_no[, first_month_of_year]] )
```

Return data type: dual

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the year which contains date . Negative values in period_no indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year . The display value will then be a string showing two years.

You can use the following values to set the first month of year in the **first_month_of_year** argument:

first_month_of_year values

Month	Value
February	2

Month	Value
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

When to use it

The `yearname()` function is useful for comparing aggregations by year. For example, if you want to see the total sales of products by year.

These dimensions can be created in the load script by using the function to create a field in a Master Calendar table. They can also be created in a chart as calculated dimensions

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>yearname('10/19/2001')</code>	Returns '2001.'
<code>yearname('10/19/2001', -1)</code>	Returns '2000.'
<code>yearname('10/19/2001', 0, 4)</code>	Returns '2001-2002.'

Related topics

Topic	Description
year (page 1094)	This function returns an integer representing the year when the expression is interpreted as a date according to the standard number interpretation.

Example 1 – No additional arguments

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 is loaded into a table called ‘Transactions’.
- The `DateFormat` system variable which is set to ‘MM/DD/YYYY’.
- A preceding load that uses the `yearname()` and which is set as the `year_name` field.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
  *,
  yearname(date) as year_name
;
```

Load

*

Inline

[

id,date,amount

```
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
```

```
8203, '12/27/2021', 25.12  
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- year_name

Results table

date	year_name
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

The ‘year_name’ field is created in the preceding load statement by using the `yearname()` function and passing the date field as the function’s argument.

The `yearname()` function identifies which year the date value falls into and returns this as a four-digit year value.

Diagram of `yearname()` function that shows 2021 as the year value.



Example 2 – period_no

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 is loaded into a table called ‘Transactions’.
- The `DateFormat` system variable which is set to ‘MM/DD/YYYY’.
- A preceding load that uses the `yearname()` and which is set as the `year_name` field.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    yearname(date,-1) as prior_year_name
  ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
```

```
8196, '01/22/2021', 95.93  
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '12/27/2021', 25.12  
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- prior_year_name

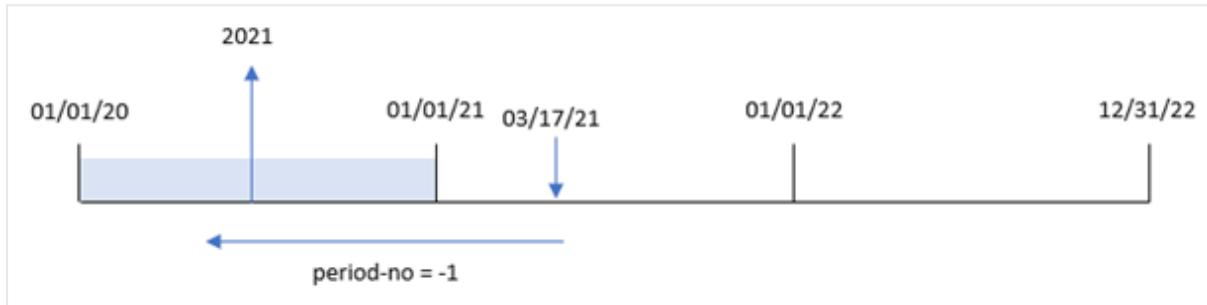
Results table

date	prior_year_name
01/13/2020	2019
02/26/2020	2019
03/27/2020	2019
04/16/2020	2019
05/21/2020	2019
08/14/2020	2019
10/07/2020	2019
12/05/2020	2019
01/22/2021	2020
02/03/2021	2020
03/17/2021	2020
04/23/2021	2020
05/04/2021	2020
06/30/2021	2020
07/26/2021	2020
12/27/2021	2020
06/06/2022	2021

date	prior_year_name
07/18/2022	2021
11/14/2022	2021
12/12/2022	2021

Because a period_no of -1 is used as the offset argument in the yearname() function, the function first identifies the year that the transactions take place in. The function then shifts one year prior and returns the resulting year.

Diagram of yearname() function with the period_no set -1.



Example 3 – first_month_of_year

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The dateFormat system variable which is set to 'MM/DD/YYYY'.
- A preceding load that uses the yearname() and which is set as the year_name field.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
  *,
  yearname(date,0,4) as year_name
;
Load
*
Inline
[
id,date,amount
```

```
8188, '01/13/2020', 37.23  
8189, '02/26/2020', 17.17  
8190, '03/27/2020', 88.27  
8191, '04/16/2020', 57.42  
8192, '05/21/2020', 53.80  
8193, '08/14/2020', 82.06  
8194, '10/07/2020', 40.39  
8195, '12/05/2020', 87.21  
8196, '01/22/2021', 95.93  
8197, '02/03/2021', 45.89  
8198, '03/17/2021', 36.23  
8199, '04/23/2021', 25.66  
8200, '05/04/2021', 82.77  
8201, '06/30/2021', 69.98  
8202, '07/26/2021', 76.11  
8203, '12/27/2021', 25.12  
8204, '06/06/2022', 46.23  
8205, '07/18/2022', 84.21  
8206, '11/14/2022', 96.24  
8207, '12/12/2022', 67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date
- year_name

Results table

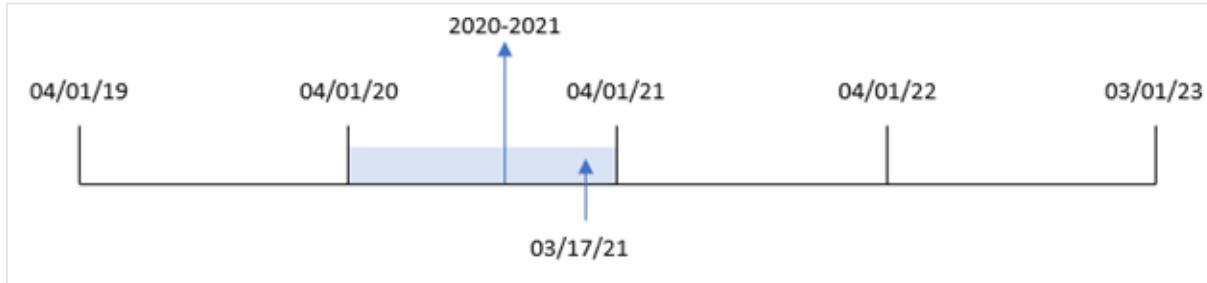
date	year_name
01/13/2020	2019-2020
02/26/2020	2019-2020
03/27/2020	2019-2020
04/16/2020	2020-2021
05/21/2020	2020-2021
08/14/2020	2020-2021
10/07/2020	2020-2021
12/05/2020	2020-2021
01/22/2021	2020-2021
02/03/2021	2020-2021
03/17/2021	2020-2021

date	year_name
04/23/2021	2021-2022
05/04/2021	2021-2022
06/30/2021	2021-2022
07/26/2021	2021-2022
12/27/2021	2021-2022
06/06/2022	2022-2023
07/18/2022	2022-2023
11/14/2022	2022-2023
12/12/2022	2022-2023

Because the `first_month_of_year` argument of 4 is used in the `yearname()` function, the start of the year moves from January 1 to April 1. Therefore, each twelve month period crosses two calendar years and the `yearname()` function returns the two four-digit years for dates evaluated.

Transaction 8198 takes place on March 17, 2021. The `yearname()` function sets the beginning of the year on April 1 and the ending on March 30. Therefore, transaction 8198 occurred in the year period from April 1, 2020 and March 30, 2021. As a result, the `yearname()` function returns the value 2020-2021.

Diagram of `yearname()` function with March set as the first month of the year.



Example 4 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The `DateFormat` system variable which is set to 'MM/DD/YYYY'.

However, the field that returns the year that the transaction took place in is created as a measure in a chart object.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension:

```
date
```

To calculate the 'year_name' field, create this measure:

```
=yearname(date)
```

Results table

date	=yearname(date)
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020

date	=yearname(date)
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

The ‘year_name’ measure is created in the chart object using the yearname() function and passing the date field as the function’s argument.

The yearname() function identifies which year the date value falls into and returns this as a four-digit year value.

Diagram of yearname() function with 2021 as the year value.



Example 5 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset from the first example.
- The `DateFormat` system variable which is set to 'MM/DD/YYYY'.

The end user would like a chart that presents the total sales by quarter for the transactions. Use the `yearname()` function as a calculated dimension to create this chart when the `yearname()` dimension is not available in the data model.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

Results

Load the data and open a sheet. Create a new table.

To compare aggregations by year, create this calculated dimension:

```
=yearname(date)
```

Create this measure:

```
=sum(amount)
```

Set the measure's **Number Formatting** to **Money**.

Results table

yearname(date)	=sum(amount)
2020	\$463.55
2021	\$457.69
2022	\$294.35

yearstart

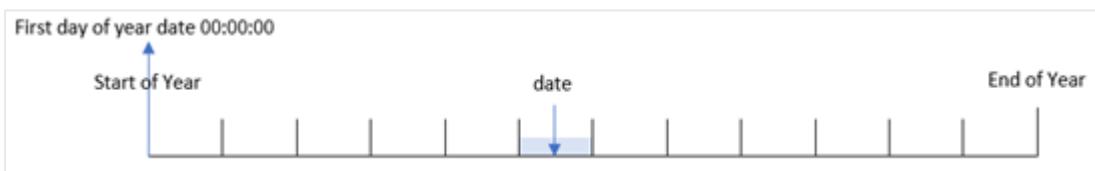
This function returns a timestamp corresponding to the start of the first day of the year containing **date**. The default output format will be the **DateFormat** set in the script.

Syntax:

```
YearStart(date[, period_no[, first_month_of_year]])
```

In other words, the `yearstart()` function determines which year the date falls into. It then returns a timestamp, in date format, for the first millisecond of that year. The first month of the year is, by default, January; however, you can change which month is set as first by using the `first_month_of_year` argument in the `yearstart()` function.

Diagram of `yearstart()` function that shows the range of time that the function can cover.



When to use it

The `yearstart()` function is used as part of an expression when you want the calculation to use the fraction of the year that has elapsed thus far. For example, if you want to calculate the interest that has accumulated in a year to date.

Return data type: dual

Arguments

Argument	Description
date	The date or timestamp to evaluate.
period_no	period_no is an integer, where the value 0 indicates the year which contains date . Negative values in period_no indicate preceding years and positive values indicate succeeding years.
first_month_of_year	If you want to work with (fiscal) years not starting in January, indicate a value between 2 and 12 in first_month_of_year .

The following months can be used in the `first_month_of_year` argument:

first_month_of_year values

Month	Value
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Function examples

Example	Result
<code>yearstart('10/19/2001')</code>	Returns 01/01/2001 00:00:00.
<code>yearstart('10/19/2001', -1)</code>	Returns 01/01/2000 00:00:00.
<code>yearstart('10/19/2001', 0, 4)</code>	Returns 04/01/2001 00:00:00.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022 is loaded into a table called ‘Transactions’.
- The date field has been provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- A preceding load statement which contains the following:
 - `yearstart()` function which is set as the `year_start` field.
 - `Timestamp()` function which is set as the `year_start_timestamp` field

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yearstart(date) as year_start,
  timestamp(yearstart(date)) as year_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
```

```
8207,12/12/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **year_start**
- **year_start_timestamp**

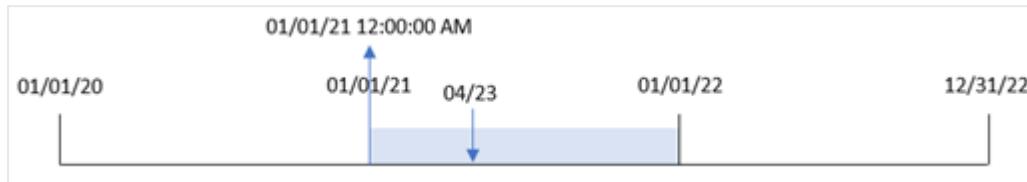
Results table

id	date	year_start	year_start_timestamp
8188	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8189	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8190	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8191	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8192	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8193	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8194	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM
8195	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM
8196	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM
8199	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8201	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8202	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM
8203	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8204	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM
8205	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8206	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8207	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM

The ‘year_start’ field is created in the preceding load statement by using the `yearstart()` function and passing the date field as the function’s argument.

The `yearstart()` function initially identifies which year the date value falls into and returns a timestamp for the first millisecond of that year.

Diagram of the `yearstart()` function and transaction 8199.



Transaction 8199 took place on April 23, 2021. The `yearstart()` function returns the first millisecond of that year, which is January 1 at 12:00:00 AM.

Example 2 – period_no

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the task is to create a field, ‘`previous_year_start`’, that returns the start date timestamp of the year prior to the year in which a transaction took place.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yearstart(date,-1) as previous_year_start,
  timestamp(yearstart(date,-1)) as previous_year_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
```

```
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **previous_year_start**
- **previous_year_start_timestamp**

Results table

id	date	previous_year_start	previous_year_start_timestamp
8188	01/13/2020	01/01/2019	1/1/2019 12:00:00 AM
8189	02/26/2020	01/01/2019	1/1/2019 12:00:00 AM
8190	03/27/2020	01/01/2019	1/1/2019 12:00:00 AM
8191	04/16/2020	01/01/2019	1/1/2019 12:00:00 AM
8192	05/21/2020	01/01/2019	1/1/2019 12:00:00 AM
8193	08/14/2020	01/01/2019	1/1/2019 12:00:00 AM
8194	10/07/2020	01/01/2019	1/1/2019 12:00:00 AM
8195	12/05/2020	01/01/2019	1/1/2019 12:00:00 AM
8196	01/22/2021	01/01/2020	1/1/2020 12:00:00 AM
8197	02/03/2021	01/01/2020	1/1/2020 12:00:00 AM
8198	03/17/2021	01/01/2020	1/1/2020 12:00:00 AM
8199	04/23/2021	01/01/2020	1/1/2020 12:00:00 AM
8200	05/04/2021	01/01/2020	1/1/2020 12:00:00 AM
8201	06/30/2021	01/01/2020	1/1/2020 12:00:00 AM
8202	07/26/2021	01/01/2020	1/1/2020 12:00:00 AM
8203	12/27/2021	01/01/2020	1/1/2020 12:00:00 AM
8204	06/06/2022	01/01/2021	1/1/2021 12:00:00 AM
8205	07/18/2022	01/01/2021	1/1/2021 12:00:00 AM

id	date	previous_year_start	previous_year_start_timestamp
8206	11/14/2022	01/01/2021	1/1/2021 12:00:00 AM
8207	12/12/2022	01/01/2021	1/1/2021 12:00:00 AM

In this instance, because a period_no of -1 is used as the offset argument in the yearstart() function, the function first identifies the year that the transactions take place in. It then looks one year prior and identifies the first millisecond of that year.

Diagram of the yearstart() function with a period_no of -1.



Transaction 8199 took place on April 23, 2021. The yearstart() function returns the first millisecond of the prior year, January 1, 2020 at 12:00:00 AM, for the 'previous_year_start' field.

Example 3 – first_month_of_year

Load script and results

Overview

The same dataset and scenario as the first example are used.

However, in this example, the company policy is for the year to begin from April 1.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

Transactions:

```

Load
  *,
  yearstart(date,0,4) as year_start,
  timestamp(yearstart(date,0,4)) as year_start_timestamp
;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
```

```
8193,08/14/2020,82.06  
8194,10/07/2020,40.39  
8195,12/05/2020,87.21  
8196,01/22/2021,95.93  
8197,02/03/2021,45.89  
8198,03/17/2021,36.23  
8199,04/23/2021,25.66  
8200,05/04/2021,82.77  
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,06/06/2022,46.23  
8205,07/18/2022,84.21  
8206,11/14/2022,96.24  
8207,12/12/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- **id**
- **date**
- **year_start**
- **year_start_timestamp**

Results table

id	date	year_start	year_start_timestamp
8188	01/13/2020	04/01/2019	4/1/2019 12:00:00 AM
8189	02/26/2020	04/01/2019	4/1/2019 12:00:00 AM
8190	03/27/2020	04/01/2019	4/1/2019 12:00:00 AM
8191	04/16/2020	04/01/2020	4/1/2020 12:00:00 AM
8192	05/21/2020	04/01/2020	4/1/2020 12:00:00 AM
8193	08/14/2020	04/01/2020	4/1/2020 12:00:00 AM
8194	10/07/2020	04/01/2020	4/1/2020 12:00:00 AM
8195	12/05/2020	04/01/2020	4/1/2020 12:00:00 AM
8196	01/22/2021	04/01/2020	4/1/2020 12:00:00 AM
8197	02/03/2021	04/01/2020	4/1/2020 12:00:00 AM
8198	03/17/2021	04/01/2020	4/1/2020 12:00:00 AM
8199	04/23/2021	04/01/2021	4/1/2021 12:00:00 AM
8200	05/04/2021	04/01/2021	4/1/2021 12:00:00 AM

id	date	year_start	year_start_timestamp
8201	06/30/2021	04/01/2021	4/1/2021 12:00:00 AM
8202	07/26/2021	04/01/2021	4/1/2021 12:00:00 AM
8203	12/27/2021	04/01/2021	4/1/2021 12:00:00 AM
8204	06/06/2022	04/01/2022	4/1/2022 12:00:00 AM
8205	07/18/2022	04/01/2022	4/1/2022 12:00:00 AM
8206	11/14/2022	04/01/2022	4/1/2022 12:00:00 AM
8207	12/12/2022	04/01/2022	4/1/2022 12:00:00 AM

In this instance, because the `first_month_of_year` argument of 4 is used in the `yearstart()` function, it sets the first day of the year to April 1, and the last day of the year to March 31.

Diagram of the `yearstart()` function with the first month set as April.



Transaction 8199 took place on April 23, 2021. Because the `yearstart()` function sets the start of the year to April 1 and returns it as the ‘`year_start`’ value for the transaction.

Example 4 – Chart object example

Load script and chart expression

Overview

The same dataset and scenario as the first example are used.

However, in this example, the dataset is unchanged and loaded into the application. The calculation that returns the start date timestamp of the year in which a transaction took place is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
```

```
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- id
- date

To calculate in which year a transaction took place, create the following measures:

- =yearstart(date)
- =timestamp(yearstart(date))

Results table

id	date	=yearstart(date)	=timestamp(yearstart(date))
8188	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM
8189	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8190	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8191	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM
8192	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8193	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8194	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM
8195	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8196	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM

id	date	=yearstart(date)	=timestamp(yearstart(date))
8199	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8201	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8202	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8203	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8204	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8205	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8206	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM
8207	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM

The ‘start_of_year’ measure is created in the chart object by using the `yearstart()` function and passing the date field as the function’s argument.

The `yearstart()` function initially identifies which year the date value falls into and returns a timestamp for the first millisecond of that year.

Diagram of the `yearstart()` function and transaction 8199.



Transaction 8199 took place on April 23, 2021. The `yearstart()` function returns the first millisecond of that year, which is January 1 at 12:00:00 AM.

Example 5 – Scenario

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset is loaded into a table called ‘Loans’. The table contains the following fields:
 - Loan IDs.
 - The balance at the beginning of the year.
 - The simple interest rate charged on each loan per annum.

The end user would like a chart object that displays, by loan id, the current interest that has been accrued on each loan in the year to date.

Load script

```
Loans:  
Load  
*  
Inline  
[  
loan_id,start_balance,rate  
8188,$10000.00,0.024  
8189,$15000.00,0.057  
8190,$17500.00,0.024  
8191,$21000.00,0.034  
8192,$90000.00,0.084  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- loan_id
- start_balance

To calculate the accumulated interest, create the following measure:

```
=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
```

Set the measure's **Number Formatting** to **Money**.

Results table

loan_id	start_balance	=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
8188	\$10000.00	\$39.73
8189	\$15000.00	\$339.66
8190	\$17500.00	\$166.85
8191	\$21000.00	\$283.64
8192	\$90000.00	\$3003.29

The `yearstart()` function, using today's date as its only argument, returns the start date of the current year. By subtracting that result from the current date, the expression returns the number of days that have elapsed so far this year.

This value is then multiplied by the interest rate and divided by 365 to return the effective interest rate for the period. The effective interest rate for the period is then multiplied by the starting balance of the loan to return the interest that has been accrued so far this year.

yeartodate

This function finds if the input timestamp falls within the year of the date the script was last loaded, and returns True if it does, False if it does not.

Syntax:

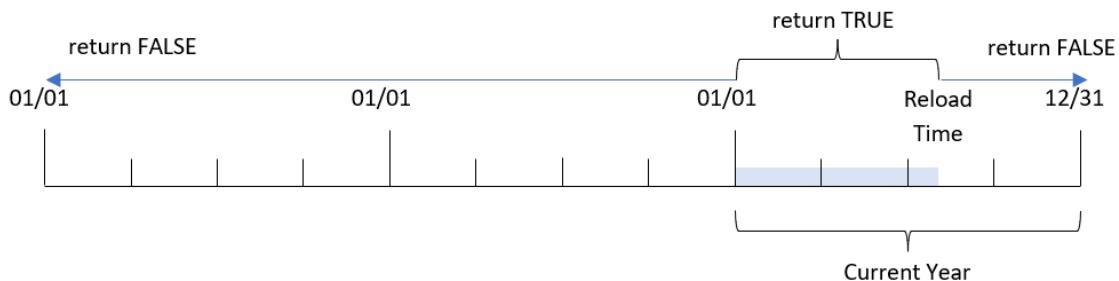
```
YearToDate(timestamp[ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

Return data type: Boolean



In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Example diagram of yeardate() function



If none of the optional parameters are used, the year to date means any date within one calendar year from January 1 up to and including the date of the last script execution.

In other words, the `yeardate()` function, when triggered with no additional parameters, is used to evaluate a timestamp and return a Boolean result based on whether the date occurred within the calendar year up to and including the date that the reload took place.

However, it is also possible to supersede the start date of the year using the `firstmonth` argument, as well as to make comparisons with preceding or following years using the `yearoffset` argument.

Finally, in instances of historical datasets, the `yeardate()` function provides a parameter to set `todaydate`, which will instead compare the timestamp to the calendar year up to and including the date provided in the `todaydate` argument.

Arguments

Argument	Description
<code>timestamp</code>	The timestamp to evaluate, for example '10/12/2012'.
<code>yearoffset</code>	By specifying a <code>yearoffset</code> , <code>yeardate</code> returns True for the same period in another year. A negative <code>yearoffset</code> indicates a previous year, a positive offset a future year. The most recent year-to-date is achieved by specifying <code>yearoffset = -1</code> . If omitted, 0 is assumed.

Argument	Description
firstmonth	By specifying a firstmonth between 1 and 12 (1 if omitted), the beginning of the year may be moved forward to the first day of any month. For example, if you want to work with a fiscal year beginning on May 1, specify firstmonth = 5. A value of 1 would indicate a fiscal year starting on January 1, and a value of 12 would indicate a fiscal year starting on December 1.
todaydate	By specifying a todaydate (timestamp of the last script execution if omitted) it is possible to move the day used as the upper boundary of the period.

When to use it

The `yeartodate()` function returns a Boolean result. Typically, this type of function will be used as a condition in an if expression. This would return an aggregation or calculation dependent on whether the evaluated date occurred in the year up to and including the last reload date of the application.

For example, the `YearToDate()` function can be used to identify all equipment manufactured so far in the current year.

The following examples assume last reload time = 11/18/2011.

Function examples

Example	Result
<code>yeartodate('11/18/2010')</code>	returns False
<code>yeartodate('02/01/2011')</code>	returns True
<code>yeartodate('11/18/2011')</code>	returns True
<code>yeartodate('11/19/2011')</code>	returns False
<code>yeartodate('11/19/2011', 0, 1, '12/31/2011')</code>	returns True
<code>yeartodate('11/18/2010', -1)</code>	returns True
<code>yeartodate('11/18/2011', -1)</code>	returns False
<code>yeartodate('04/30/2011', 0, 5)</code>	returns False
<code>yeartodate('05/01/2011', 0, 5)</code>	returns True

Regional settings

Unless otherwise specified, the examples in this topic use the following date format: MM/DD/YYYY. The date format is specified in the `SET DateFormat` statement in your data load script. The default date formatting may be different in your system, due to your regional settings and other factors. You can change the formats in the examples below to suit your requirements. Or you can change the formats in your load script to match these examples.

Default regional settings in apps are based on the regional system settings of the computer or server where Qlik Sense is installed. If the Qlik Sense server you are accessing is set to Sweden, the Data load editor will use Swedish regional settings for dates, time, and currency. These regional format settings are not related to the language displayed in the Qlik Sense user interface. Qlik Sense will be displayed in the same language as the browser you are using.

Example 1 – Basic example

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.
- The creation of a field, `year_to_date`, that determines which transactions took place in the calendar year up to the date of the last reload.

At the time of writing, the date is April 26, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    yeartodate(date) as year_to_date
  ;
  Load
  *
  Inline
  [
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
```

```
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,02/02/2022,46.23  
8205,02/26/2022,84.21  
8206,03/07/2022,96.24  
8207,03/11/2022,67.67  
];
```

Results

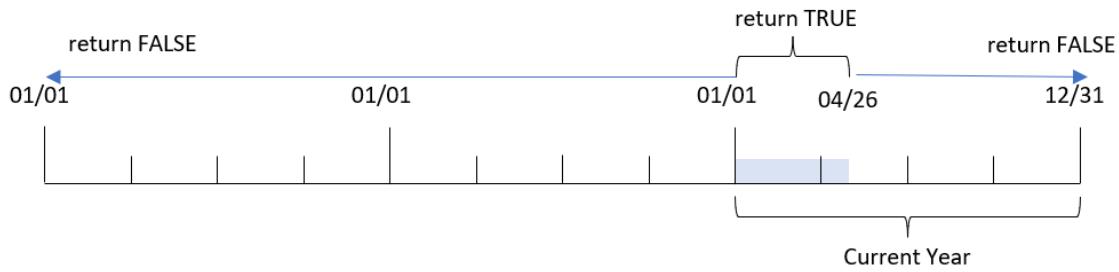
Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- year_to_date

Results table

date	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

Diagram of yeartodate() function, basic example



The `year_to_date` field is created in the preceding load statement by using the `yeartodate()` function and passing the date field as the function's argument.

Because no further parameters are passed into the function, the `yeartodate()` function initially identifies the reload date and therefore the boundaries for the current calendar year (starting January 1) that will return a Boolean result of `TRUE`.

Therefore, any transaction that occurs between January 1 and April 26, the reload date, will return a Boolean result of `TRUE`. Any transaction that occurs before the start of 2022 will return a Boolean result of `FALSE`.

Example 2 – yearoffset

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, `two_years_prior`, that determines which transactions took place a full two years before the calendar year to date.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yeartodate(date,-2) as two_years_prior
;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
```

```
8189,02/28/2020,17.17  
8190,04/09/2020,88.27  
8191,04/16/2020,57.42  
8192,05/21/2020,53.80  
8193,08/14/2020,82.06  
8194,10/07/2020,40.39  
8195,12/05/2020,87.21  
8196,01/22/2021,95.93  
8197,02/03/2021,45.89  
8198,03/17/2021,36.23  
8199,04/23/2021,25.66  
8200,05/04/2021,82.77  
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,02/02/2022,46.23  
8205,02/26/2022,84.21  
8206,03/07/2022,96.24  
8207,03/11/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- two_years_prior

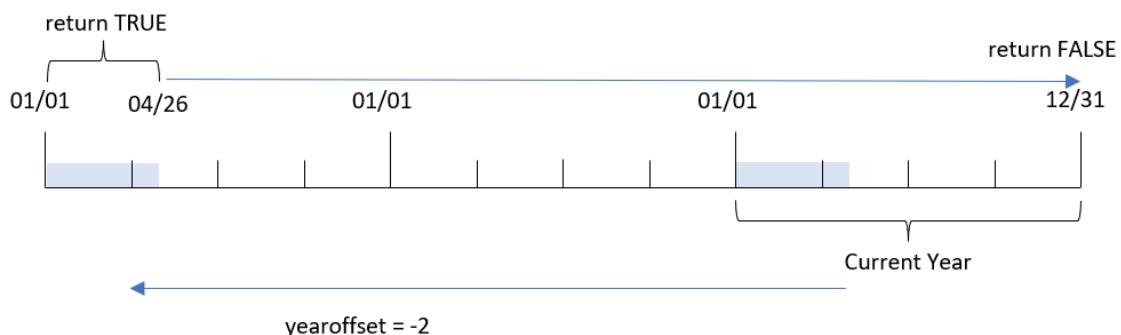
Results table

date	two_years_prior
01/10/2020	-1
02/28/2020	-1
04/09/2020	-1
04/16/2020	-1
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0

date	two_years_prior
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	0
02/26/2022	0
03/07/2022	0
03/11/2022	0

By using -2 as the yearoffset argument in the yeartodate() function, the function shifts the boundaries of the comparator calendar year segment by a full two years. Initially, the year segment equates to between January 1 and April 26, 2022. The yearoffset argument then offsets this segment to two years prior. The date boundaries will then fall between the January 1 and April 26, 2020.

Diagram of yeartodate() function, yearoffset example



Therefore, any transaction that occurs between January 1 and April 26, 2020 will return a Boolean result of TRUE. Any transactions that appear before or after this segment will return FALSE.

Example 3 – firstmonth

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, year_to_date, that determines which transactions took place in the calendar year up to the date of the last reload.

In this example, we set the start of the fiscal year to July 1.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
  *,
  yeartodate(date,0,7) as year_to_date
;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- date
- year_to_date

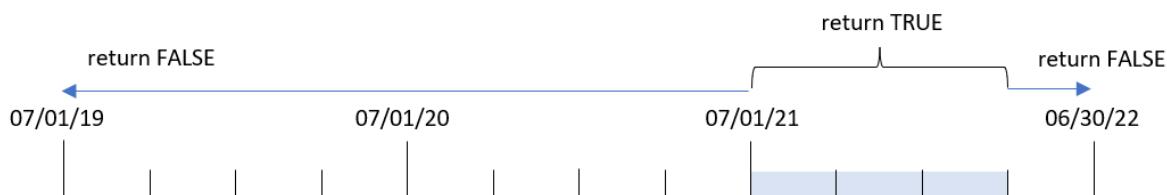
Results table

date	year_to_date
01/10/2020	0
02/28/2020	0

date	year_to_date
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	-1
12/27/2021	-1
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

In this instance, because the `firstmonth` argument of 7 is used in the `yeartodate()` function, it sets the first day of the year to July 1, and the last day of the year to June 30.

Diagram of yeardate() function, firstmonth example



Therefore, any transaction that occurs between July 1, 2021 and April 26, 2022, the reload date, will return a Boolean result of `TRUE`. Any transaction that occurs before July 1, 2021 will return a Boolean result of `FALSE`.

Example 4 – todaydate

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- The same dataset and scenario as the first example.
- The creation of a field, year_to_date, that determines which transactions took place in the calendar year up to the date of the last reload.

However, in this example, we need to identify all transactions that took place in the calendar year up to and including March 1, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';

Transactions:
  Load
    *,
    yeartodate(date, 0, 1, '03/01/2022') as year_to_date
  ;
  Load
  *
  Inline
  [
  id,date,amount
  8188,01/10/2020,37.23
  8189,02/28/2020,17.17
  8190,04/09/2020,88.27
  8191,04/16/2020,57.42
  8192,05/21/2020,53.80
  8193,08/14/2020,82.06
  8194,10/07/2020,40.39
  8195,12/05/2020,87.21
  8196,01/22/2021,95.93
  8197,02/03/2021,45.89
  8198,03/17/2021,36.23
  8199,04/23/2021,25.66
  8200,05/04/2021,82.77
  8201,06/30/2021,69.98
  8202,07/26/2021,76.11
  8203,12/27/2021,25.12
  8204,02/02/2022,46.23
  8205,02/26/2022,84.21
  8206,03/07/2022,96.24
  8207,03/11/2022,67.67
  ];
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

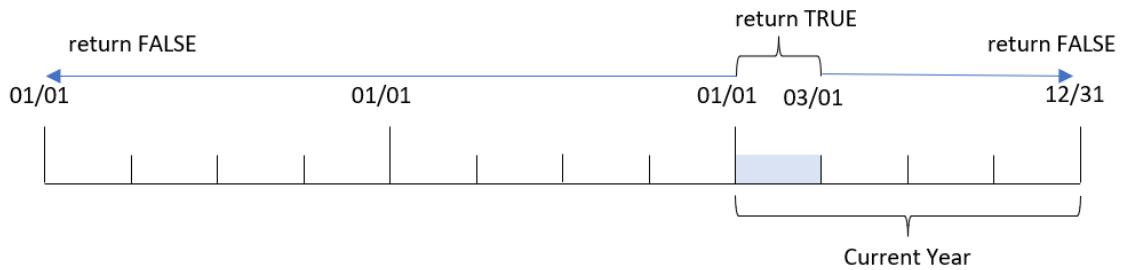
- date
- year_to_date

Results table

date	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	0
03/11/2022	0

In this instance, because the `todaydate` argument of 03/01/2022 is used in the `yeartodate()` function, it sets the end boundary of the comparator calendar year segment to March 1, 2022. It is critical to provide the `firstmonth` parameter (between 1 and 12); otherwise the function will return null results.

Diagram of yeartodate() function, example using todaydate argument



Therefore, any transaction that occurs between January 1, 2022 and March 1, 2022, the `todaydate` parameter, will return a Boolean result of `TRUE`. Any transaction that occurs before January 1, 2022 or after March 1, 2022 will return a Boolean result of `FALSE`.

Example 5 – Chart object example

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains the same dataset and scenario as the first example.

However, in this example, the unchanged dataset is loaded into the application. The calculation that determines which transactions took place in the calendar year up to the date of the last reload is created as a measure in a chart object of the application.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
```

```
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,02/02/2022,46.23  
8205,02/26/2022,84.21  
8206,03/07/2022,96.24  
8207,03/11/2022,67.67  
];
```

Results

Load the data and open a sheet. Create a new table and add this field as a dimension: date.

Add the following measure:

```
=yeartodate(date)
```

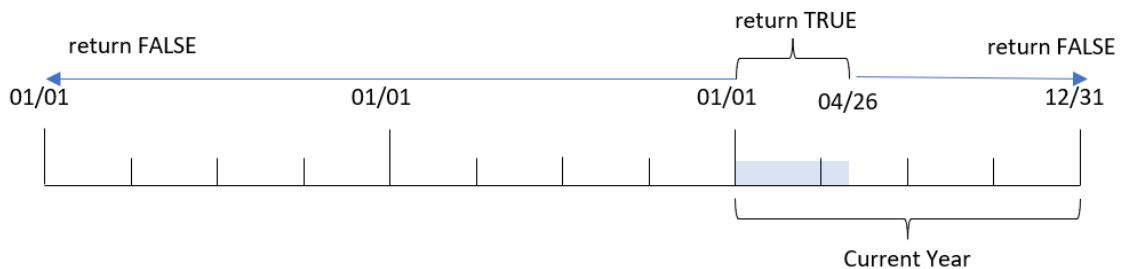
Results table

date	=yeartodate(date)
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

The year_to_date measure is created in the chart object by using the `yeartodate()` function and passing the date field as the function's argument.

Because no further parameters are passed into the function, the `yeartodate()` function initially identifies the reload date, and therefore the boundaries for the current calendar year (starting January 1) that will return a Boolean result of `TRUE`.

Diagram of `yeartodate()` function, example using chart object



Any transaction that occurs between January 1 and April 26, the reload date, will return a Boolean result of `TRUE`. Any transaction that occurs before the start of 2022 will return a Boolean result of `FALSE`.

Example 6 – Scenario

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- A dataset containing a set of transactions between 2020 and 2022, which is loaded into a table called `Transactions`.
- The date field provided in the `DateFormat` system variable (MM/DD/YYYY) format.

The end user would like a KPI object that presents the total sales for the equivalent period in 2021 as the current year to date as at the last reload time.

At the time of writing, the date is June 16, 2022.

Load script

```
SET DateFormat='MM/DD/YYYY';
```

```
Transactions:
```

```
Load
```

```
*
```

```
Inline
```

```
[
```

```
id,date,amount
```

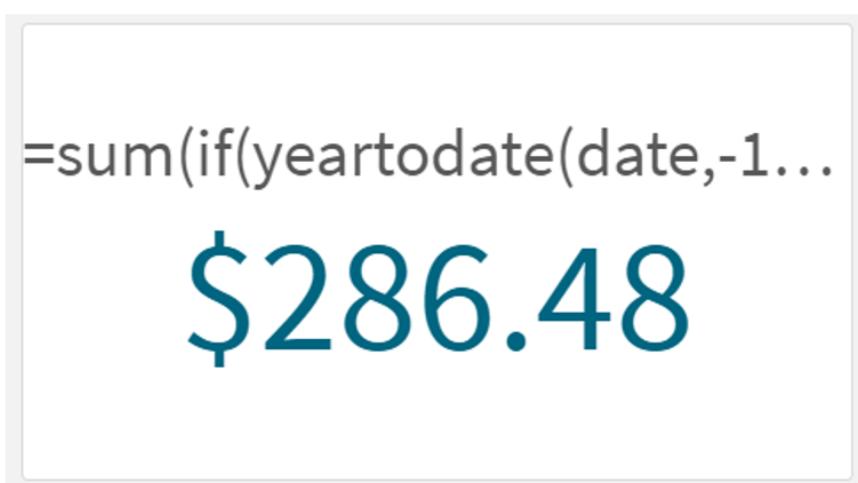
```
8188,01/10/2020,37.23  
8189,02/28/2020,17.17  
8190,04/09/2020,88.27  
8191,04/16/2020,57.42  
8192,05/21/2020,53.80  
8193,08/14/2020,82.06  
8194,10/07/2020,40.39  
8195,12/05/2020,87.21  
8196,01/22/2021,95.93  
8197,02/03/2021,45.89  
8198,03/17/2021,36.23  
8199,04/23/2021,25.66  
8200,05/04/2021,82.77  
8201,06/30/2021,69.98  
8202,07/26/2021,76.11  
8203,12/27/2021,25.12  
8204,02/02/2022,46.23  
8205,02/26/2022,84.21  
8206,03/07/2022,96.24  
8207,03/11/2022,67.67  
];
```

Results

Do the following:

1. Create a KPI object.
2. Create the following aggregation measure to calculate total sales:
`=sum(if(yeartodate(date,-1),amount,0))`
3. Set the measure's **Number formatting** to **Money**.

KPI yeartodate() chart for 2021



=sum(if(yeartodate(date,-1...

\$286.48

The yeartodate() function returns a Boolean value when evaluating the dates of each transaction ID. Because the reload took place on June 16, 2022, the yeartodate function segments the year period to between 01/01/2022 and 06/16/2022. However, since a period_no value of -1 was used in the function, these boundaries are then shifted to the previous year. Therefore, for any transaction that occurs between 01/01/2021 and 06/16/2021, the yeartodate() function returns a Boolean value of TRUE and sums the amount.

5.8 Exponential and logarithmic functions

This section describes functions related to exponential and logarithmic calculations. All functions can be used in both the data load script and in chart expressions.

In the functions below, the parameters are expressions where **x** and **y** should be interpreted as real valued numbers.

exp

The natural exponential function, e^x , using the natural logarithm **e** as base. The result is a positive number.

```
exp(x )
```

Examples and results:

exp(3) returns 20.085.

log

The natural logarithm of **x**. The function is only defined if **x**> 0. The result is a number.

```
log(x )
```

Examples and results:

log(3) returns 1.0986

log10

The common logarithm (base 10) of **x**. The function is only defined if **x**> 0. The result is a number.

```
log10(x )
```

Examples and results:

log10(3) returns 0.4771

pow

Returns **x** to the power of **y**. The result is a number.

```
pow(x,y )
```

Examples and results:

pow(3, 3) returns 27

sqr

x squared (x to the power of 2). The result is a number.

```
sqr (x )
```

Examples and results:

sqr(3) returns 9

sqrt

Square root of x. The function is only defined if x >= 0. The result is a positive number.

```
sqrt(x )
```

Examples and results:

sqrt(3) returns 1.732

5.9 Field functions

These functions can only be used in chart expressions.

Field functions either return integers or strings identifying different aspects of field selections.

Count functions

GetAlternativeCount

GetAlternativeCount() is used to find the number of alternative (light gray) values in the identified field.

```
GetAlternativeCount - chart function (field_name)
```

GetExcludedCount

GetExcludedCount() finds the number of excluded distinct values in the identified field. Excluded values include alternative (light gray), excluded (dark gray), and selected excluded (dark gray with check mark) fields.

```
GetExcludedCount - chart function (page 1157)(field_name)
```

GetNotSelectedCount

This chart function returns the number of not-selected values in the field named **fieldname**. The field must be in and-mode for this function to be relevant.

```
GetNotSelectedCount - chart function(fieldname [, includeexcluded=false])
```

GetPossibleCount

GetPossibleCount() is used to find the number of possible values in the identified field. If the identified field includes selections, then the selected (green) fields are counted. Otherwise associated (white) values are counted.

```
GetPossibleCount - chart function(field_name)
```

GetSelectedCount

GetSelectedCount() finds the number of selected (green) values in a field.

```
GetSelectedCount - chart function (field_name [, include_excluded])
```

Field and selection functions

GetCurrentSelections

GetCurrentSelections() returns a list of the current selections in the app. If the selections are instead made using a search string in a search box, **GetCurrentSelections()** returns the search string.

```
GetCurrentSelections - chart function([record_sep [,tag_sep [,value_sep [,max_values]]]])
```

GetFieldSelections

GetFieldSelections() returns a **string** with the current selections in a field.

```
GetFieldSelections - chart function ( field_name [, value_sep [, max_values]])
```

GetObjectDimension

GetObjectDimension() returns the name of the dimension. **Index** is an optional integer denoting the dimension that should be returned.

```
GetObjectDimension - chart function ([index])
```

GetObjectField

GetObjectField() returns the name of the dimension. **Index** is an optional integer denoting the dimension that should be returned.

```
GetObjectField - chart function ([index])
```

GetObjectMeasure

GetObjectMeasure() returns the name of the measure. **Index** is an optional integer denoting the measure that should be returned.

```
GetObjectMeasure - chart function ([index])
```

GetAlternativeCount - chart function

GetAlternativeCount() is used to find the number of alternative (light gray) values in the identified field.

Syntax:

```
GetAlternativeCount (field_name)
```

Return data type: integer

Arguments:

Arguments	
Argument	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses the **First name** field loaded to a filter pane.

Examples and results	
Examples	Results
Given that John is selected in First name . GetAlternativeCount ([First name])	4 as there are 4 unique and excluded (gray) values in First name .
Given that John and Peter are selected. GetAlternativeCount ([First name])	3 as there are 3 unique and excluded (gray) values in First name .
Given that no values are selected in First name . GetAlternativeCount ([First name])	0 as there are no selections.

Data used in example:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetCurrentSelections - chart function

GetCurrentSelections() returns a list of the current selections in the app. If the selections are instead made using a search string in a search box, **GetCurrentSelections()** returns the search string.

If options are used, you will need to specify record_sep. To specify a new line, set **record_sep** to **chr(13)&chr(10)**.

If all but two, or all but one, values, are selected, the format 'NOT x,y' or 'NOT y' will be used respectively. If you select all values and the count of all values is greater than max_values, the text ALL will be returned.

Syntax:

```
GetCurrentSelections ([record_sep [, tag_sep [, value_sep [, max_values [, state_name]]]]])
```

Return data type: string

Arguments:

Arguments

Arguments	Description
record_sep	Separator to be put between field records. The default is <CR><LF> meaning a new line.
tag_sep	Separator to be put between the field name tag and the field values. The default is ': '.
value_sep	The separator to be put between field values. The default is ', '.
max_values	The maximum number of field values to be individually listed. When a larger number of values is selected, the format 'x of y values' will be used instead. The default is 6.
state_name	The name of an alternate state that has been chosen for the specific visualization. If the state_name argument is used, only the selections associated with the specified state name are taken into account.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples and results

Examples	Results
Given that John is selected in First name . GetCurrentselections ()	'First name: John'
Given that John and Peter are selected in First name . GetCurrentselections ()	'First name: John, Peter'
Given that John and Peter are selected in First name and JA is selected in Initials . GetCurrentselections ()	'First name: John, Peter Initials: JA'
Given that John is selected in First name and JA is selected in Initials . GetCurrentselections (chr(13)&chr(10) , ' = ')	'First name = John Initials = JA'
Given that you have selected all names except Sue in First name and no selections in Initials . GetCurrentselections (chr(13)&chr(10), '=' , ',' ,3)	'First name=NOT Sue'

Data used in example:

```
Names:  
LOAD * inline [  
First name|Last name|Initials|Has cellphone  
John|Anderson|JA|Yes  
Sue|Brown|SB|Yes  
Mark|Carr|MC|No  
Peter|Devonshire|PD|No  
Jane|Elliot|JE|Yes  
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetExcludedCount - chart function

GetExcludedCount() finds the number of excluded distinct values in the identified field. Excluded values include alternative (light gray), excluded (dark gray), and selected excluded (dark gray with check mark) fields.

Syntax:

```
GetExcludedCount (field_name)
```

Return data type: string

Arguments:

Arguments

Arguments	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses three fields loaded to different filter panes, one for **First name**, one for **Last name**, and one for **Initials**.

Examples and results

Examples	Results
If no values are selected in First name .	GetExcludedCount (Initials) = 0 There are no selections.
If John is selected in First name .	GetExcludedCount (Initials) = 5 There are 5 excluded values in Initials with dark gray color. The sixth cell (JA) will be white as it is associated with the selection John in First name .
If John and Peter are selected.	GetExcludedCount (Initials) = 3 John is associated with 1 value and Peter is associated with 2 values, in Initials .

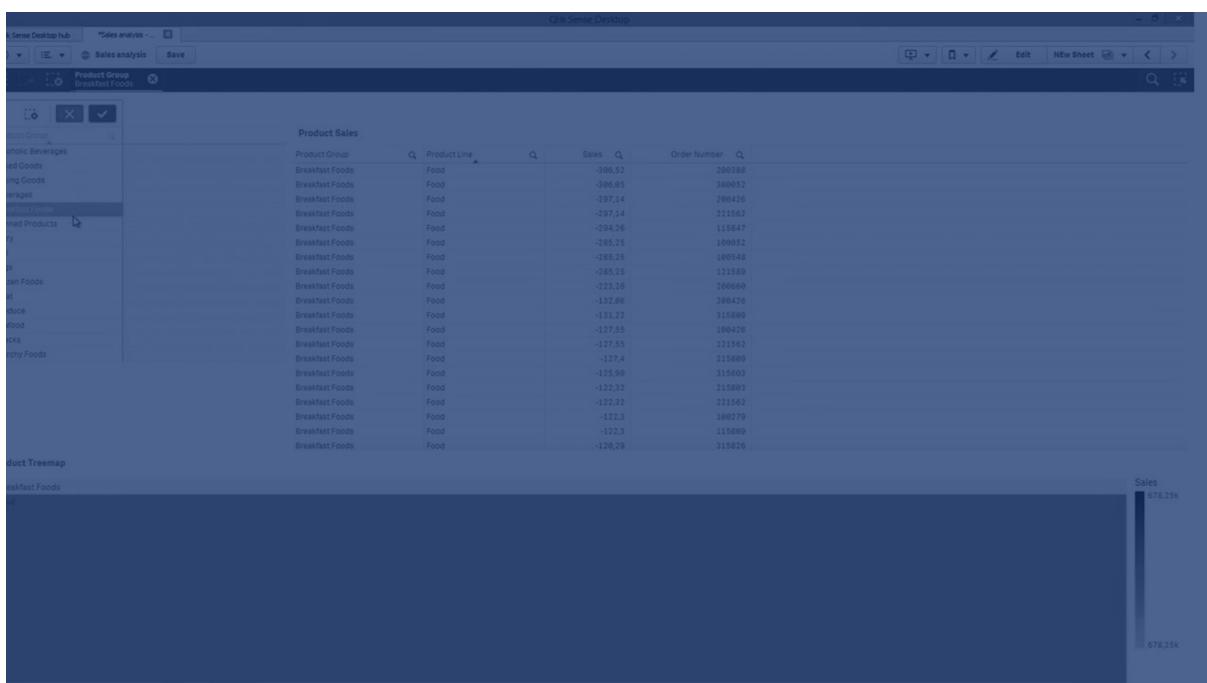
Examples	Results
If John and Peter are selected in First name , and then Franc is selected in Last name .	GetExcludedCount ([First name]) = 4 There are 4 excluded values in First name with dark gray color. GetExcludedCount() evaluates for fields with excluded values, including alternative and selected excluded fields.
If John and Peter are selected in First name , and then Franc and Anderson are selected in Last name .	GetExcludedCount (Initials) = 4 There are 4 excluded values in Initials with dark gray color. The other two cells (JA and PF) will be white as they associated with the selections John and Peter in First name .
If John and Peter are selected in First name , and then Franc and Anderson are selected in Last name .	GetExcludedCount ([Last name]) = 4 There are 4 excluded values in Initials . Devonshire has light gray color while Brown, Carr, and Elliot have dark gray color.

Data used in example:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetFieldSelections - chart function

GetFieldSelections() returns a **string** with the current selections in a field.



5 Script and chart functions

If all but two, or all but one of the values are selected, the format 'NOT x,y' or 'NOT y' will be used respectively. If you select all values and the count of all values is greater than max_values, the text ALL will be returned.

Syntax:

```
GetFieldSelections ( field_name [, value_sep [, max_values [, state_name]]])
```

Return data type: string

Return string formats

Format	Description
'a, b, c'	If the number of selected values is max_values or less, the string returned is a list of the selected values. The values are separated with value_sep as delimiter.
'NOT a, b, c'	If the number of non-selected values is max_values or less, the string returned is a list of the non-selected values with NOT as a prefix. The values are separated with value_sep as delimiter.
'x of y'	x = the number of selected values y = the total number of values This is returned when max_values < x < (y - max_values).
'ALL'	Returned if all values are selected.
'-'	Returned if no value is selected.
<search string>	If you have selected using search, the search string is returned.

Arguments:

Arguments

Arguments	Description
field_name	The field containing the range of data to be measured.
value_sep	The separator to be put between field values. The default is ','.
max_values	The maximum number of field values to be individually listed. When a larger number of values is selected, the format 'x of y values' will be used instead. The default is 6.
state_name	The name of an alternate state that has been chosen for the specific visualization. If the state_name argument is used, only the selections associated with the specified state name are taken into account.

Examples and results:

The following example uses the **First name** field loaded to a filter pane.

Examples and results

Examples	Results
Given that John is selected in First name . GetFieldSelections ([First name])	'John'
Given that John and Peter are selected. GetFieldSelections ([First name])	'John,Peter'
Given that John and Peter are selected. GetFieldSelections ([First name],'; ')	'John; Peter'
Given that John, Sue, Mark are selected in First name . GetFieldSelections ([First name],'; ',2)	'NOT Jane;Peter', because the value 2 is stated as the value of the max_values argument. Otherwise, the result would have been John; Sue; Mark.

Data used in example:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetNotSelectedCount - chart function

This chart function returns the number of not-selected values in the field named **fieldname**. The field must be in and-mode for this function to be relevant.

Syntax:

```
GetNotSelectedCount (fieldname [, includeexcluded=false])
```

Arguments:

Arguments

Argument	Description
fieldname	The name of the field to be evaluated.

Argument	Description
includeexcluded	If includeexcluded is stated as True, the count will include selected values which are excluded by selections in another field.

Examples:

```
GetNotSelectedCount( Country )
GetNotSelectedCount( Country, true )
```

GetObjectDimension - chart function

GetObjectDimension() returns the name of the dimension. **Index** is an optional integer denoting the dimension that should be returned.



You cannot use this function in a chart in the following locations: title, subtitle, footer, reference line expression.



You cannot reference the name of a dimension or measure in another object using the Object ID.

Syntax:

```
GetObjectDimension ([index])
```

Example:

```
GetObjectDimension(1)
```

Example: Chart expression

Qlik Sense table showing examples of the GetObjectDimension function in a chart expression

transaction_date	customer_id	transaction_quantity	=GetObjectDimension ()	=GetObjectDimension (0)	=GetObjectDimension (1)
2018/08/30	049681	13	transaction_date	transaction_date	customer_id
2018/08/30	203521	6	transaction_date	transaction_date	customer_id
2018/08/30	203521	21	transaction_date	transaction_date	customer_id

If you want to return the name of a measure, use the **GetObjectMeasure** function instead.

GetObjectField - chart function

GetObjectField() returns the name of the dimension. **Index** is an optional integer denoting the dimension that should be returned.



You cannot use this function in a chart in the following locations: title, subtitle, footer, reference line expression.



You cannot reference the name of a dimension or measure in another object using the Object ID.

Syntax:

```
GetObjectField ([index])
```

Example:

```
GetObjectField(1)
```

Example: Chart expression

Qlik Sense table showing examples of the GetObjectField function in a chart expression.

transaction_date	customer_id	transaction_quantity	=GetObjectField() (0)	=GetObjectField (0)	=GetObjectField (1)
2018/08/30	049681	13	transaction_date	transaction_date	customer_id
2018/08/30	203521	6	transaction_date	transaction_date	customer_id
2018/08/30	203521	21	transaction_date	transaction_date	customer_id

If you want to return the name of a measure, use the **GetObjectMeasure** function instead.

GetObjectMeasure - chart function

GetObjectMeasure() returns the name of the measure. **Index** is an optional integer denoting the measure that should be returned.



You cannot use this function in a chart in the following locations: title, subtitle, footer, reference line expression.



You cannot reference the name of a dimension or measure in another object using the Object ID.

Syntax:

```
GetObjectMeasure ([index])
```

Example:

```
GetObjectMeasure(1)
```

Example: Chart expression

Qlik Sense table showing examples of the GetObjectMeasure function in a chart expression

customer_id	sum(transaction_quantity)	Avg(transaction_quantity)	=GetObjectMeasure ()	=GetObjectMeasure(0)	=GetObjectMeasure(1)
49681	13	13	sum(transaction_quantity)	sum(transaction_quantity)	Avg(transaction_quantity)
203521	27	13.5	sum(transaction_quantity)	sum(transaction_quantity)	Avg(transaction_quantity)

If you want to return the name of a dimension, use the **GetObjectField** function instead.

GetPossibleCount - chart function

GetPossibleCount() is used to find the number of possible values in the identified field. If the identified field includes selections, then the selected (green) fields are counted. Otherwise associated (white) values are counted..

For fields with selections, **GetPossibleCount()** returns the number of selected (green) fields.

Return data type: integer

Syntax:

```
GetPossibleCount (field_name)
```

Arguments:

Arguments

Arguments	Description
field_name	The field containing the range of data to be measured.

Examples and results:

The following example uses two fields loaded to different filter panes, one for **First name** name and one for **Initials**.

Examples and results

Examples	Results
Given that John is selected in First name . GetPossibleCount ([Initials])	1 as there is 1 value in Initials associated with the selection, John , in First name .
Given that John is selected in First name . GetPossibleCount ([First name])	1 as there is 1 selection, John , in First name .

Examples	Results
Given that Peter is selected in First name . GetPossibleCount ([Initials])	2 as Peter is associated with 2 values in Initials .
Given that no values are selected in First name . GetPossibleCount ([First name])	5 as there are no selections and there are 5 unique values in First name .
Given that no values are selected in First name . GetPossibleCount ([Initials])	6 as there are no selections and there are 6 unique values in Initials .

Data used in example:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

GetSelectedCount - chart function

GetSelectedCount() finds the number of selected (green) values in a field.

Syntax:

```
GetSelectedCount (field_name [, include_excluded [, state_name]])
```

Return data type: integer

Arguments:

Arguments

Arguments	Description
field_name	The field containing the range of data to be measured.
include_excluded	If set to True() , the count will include selected values, which are currently excluded by selections in other fields. If False or omitted, these values will not be included.
state_name	The name of an alternate state that has been chosen for the specific visualization. If the state_name argument is used, only the selections associated with the specified state name are taken into account.

Examples and results:

The following example uses three fields loaded to different filter panes, one for **First name** name, one for **Initials** and one for **Has cellphone**.

Examples and results

Examples	Results
Given that John is selected in First name . GetSelectedCount ([First name])	1 as one value is selected in First name .
Given that John is selected in First name . GetSelectedCount ([Initials])	0 as no values are selected in Initials .
With no selections in First name , select all values in Initials and after that select the value Yes in Has cellphone . GetSelectedCount ([Initials], True())	6. Although selections with Initials MC and PD have Has cellphone set to No , the result is still 6, because the argument include_excluded is set to True() .

Data used in example:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

5.10 File functions

The file functions (only available in script expressions) return information about the table file which is currently being read. These functions will return NULL for all data sources except table files (exception: **ConnectionString()**).

File functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Attribute

This script function returns the value of the meta tags of different media files as text. The following file formats are supported: MP3, WMA, WMV, PNG and JPG. If the file **filename** does not exist, is not a supported file format or does not contain a meta tag named **attributename**, NULL will be returned.

Attribute (filename, attributename)

ConnectionString

The **ConnectionString()** function returns the name of the active data connection for ODBC or OLE DB connections. The function returns an empty string if no **connect** statement has been executed, or after a **disconnect** statement.

ConnectionString ()

FileBaseName

The **FileBaseName** function returns a string containing the name of the table file currently being read, without path or extension.

FileBaseName ()

FileDir

The **FileDir** function returns a string containing the path to the directory of the table file currently being read.

FileDir ()

FileExtension

The **FileExtension** function returns a string containing the extension of the table file currently being read.

FileExtension ()

FileName

The **FileName** function returns a string containing the name of the table file currently being read, without path but including the extension.

FileName ()

FilePath

The **FilePath** function returns a string containing the full path to the table file currently being read.

FilePath ()

FileSize

The **FileSize** function returns an integer containing the size in bytes of the file filename or, if no filename is specified, of the table file currently being read.

FileSize ()

FileTime

The **FileTime** function returns a timestamp in UTC format of the last modification of a specified file. If a file is not specified, the function returns a timestamp in UTC of the last modification of the currently read table file.

FileTime ([filename])

GetFolderPath

The **GetFolderPath** function returns the value of the Microsoft Windows *SHGetFolderPath* function. This function takes as input the name of a Microsoft Windows folder and returns the full path of the folder.

GetFolderPath ()

QvdCreateTime

This script function returns the XML-header timestamp from a QVD file, if any is present, otherwise it returns NULL. In the timestamp, time is provided in UTC.

QvdCreateTime (filename)

QvdFieldName

This script function returns the name of field number **fieldno** in a QVD file. If the field does not exist NULL is returned.

QvdFieldName (filename , fieldno)

QvdNoOfFields

This script function returns the number of fields in a QVD file.

QvdNoOfFields (filename)

QvdNoOfRecords

This script function returns the number of records currently in a QVD file.

QvdNoOfRecords (filename)

QvdTableName

This script function returns the name of the table stored in a QVD file.

QvdTableName (filename)

Attribute

This script function returns the value of the meta tags of different media files as text. The following file formats are supported: MP3, WMA, WMV, PNG and JPG. If the file **filename** does not exist, is not a supported file format or does not contain a meta tag named **attributename**, NULL will be returned.

Syntax:

Attribute(filename, attributename)

A large number of meta tags can be read. The examples in this topic show which tags can be read for the respective supported file types.



*You can only read meta tags saved in the file according to the relevant specification, for example ID2v3 for MP3 files or EXIF for JPG files, not meta information saved in the **Windows File Explorer**.*

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a media file including path, if needed, as a folder data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: <i>c:\data\</i></p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: <i>data\</i></p>
attributename	The name of a meta tag.

The examples use the **GetFolderPath** function to find the paths to media files. As **GetFolderPath** is only supported in legacy mode, you need to replace the references to **GetFolderPath** with a lib:// data connection path when you use this function in standard mode or in Qlik Sense SaaS.

File system access restriction (page 1454)

Example 1: MP3 files

This script reads all possible MP3 meta tags in folder *MyMusic*.

```
// Script to read MP3 meta tags
for each vExt in 'mp3'
for each vFoundFile in filelist( GetFolderPath('MyMusic') & '\*.'& vExt )
FileList:
LOAD FileLongName,
    subfield(FileLongName,'\',-1) as FileshortName,
    num(FileSize(FileLongName),'# ### ### ##','.') as Filesize,
    FileTime(FileLongName) as FileTime,
    // ID3v1.0 and ID3v1.1 tags
    Attribute(FileLongName, 'Title') as Title,
    Attribute(FileLongName, 'Artist') as Artist,
    Attribute(FileLongName, 'Album') as Album,
    Attribute(FileLongName, 'Year') as Year,
    Attribute(FileLongName, 'Comment') as Comment,
    Attribute(FileLongName, 'Track') as Track,
    Attribute(FileLongName, 'Genre') as Genre,
    // ID3v2.3 tags
    Attribute(FileLongName, 'AENC') as AENC, // Audio encryption
    Attribute(FileLongName, 'APIC') as APIC, // Attached picture
    Attribute(FileLongName, 'COMM') as COMM, // Comments
    Attribute(FileLongName, 'COMR') as COMR, // Commercial frame
    Attribute(FileLongName, 'ENCR') as ENCR, // Encryption method registration
```

```

Attribute(FileLongName, 'EQUA') as EQUA, // Equalization
Attribute(FileLongName, 'ETCO') as ETCO, // Event timing codes
Attribute(FileLongName, 'GEOB') as GEOB, // General encapsulated object
Attribute(FileLongName, 'GRID') as GRID, // Group identification registration
Attribute(FileLongName, 'IPLS') as IPLS, // Involved people list
Attribute(FileLongName, 'LINK') as LINK, // Linked information
Attribute(FileLongName, 'MCDI') as MCDI, // Music CD identifier
Attribute(FileLongName, 'MLLT') as MLLT, // MPEG location lookup table
Attribute(FileLongName, 'OWNE') as OWNE, // Ownership frame
Attribute(FileLongName, 'PRIV') as PRIV, // Private frame
Attribute(FileLongName, 'PCNT') as PCNT, // Play counter
Attribute(FileLongName, 'POPM') as POPM, // Popularimeter
Attribute(FileLongName, 'POSS') as POSS, // Position synchronisation frame
Attribute(FileLongName, 'RBUF') as RBUF, // Recommended buffer size
Attribute(FileLongName, 'RVAD') as RVAD, // Relative volume adjustment
Attribute(FileLongName, 'RVRB') as RVRB, // Reverb
Attribute(FileLongName, 'SYLT') as SYLT, // Synchronized lyric/text
Attribute(FileLongName, 'SYTC') as SYTC, // Synchronized tempo codes
Attribute(FileLongName, 'TALB') as TALB, // Album/Movie>Show title
Attribute(FileLongName, 'TBPM') as TBPM, // BPM (beats per minute)
Attribute(FileLongName, 'TCOM') as TCOM, // Composer
Attribute(FileLongName, 'TCON') as TCON, // Content type
Attribute(FileLongName, 'TCOP') as TCOP, // Copyright message
Attribute(FileLongName, 'TDAT') as TDAT, // Date
Attribute(FileLongName, 'TDLY') as TDLY, // Playlist delay
Attribute(FileLongName, 'TENC') as TENC, // Encoded by
Attribute(FileLongName, 'TEXT') as TEXT, // Lyricist/Text writer
Attribute(FileLongName, 'TFLT') as TFLT, // File type
Attribute(FileLongName, 'TIME') as TIME, // Time
Attribute(FileLongName, 'TIT1') as TIT1, // Content group description
Attribute(FileLongName, 'TIT2') as TIT2, // Title/songname/content description
Attribute(FileLongName, 'TIT3') as TIT3, // Subtitle/Description refinement
Attribute(FileLongName, 'TKEY') as TKEY, // Initial key
Attribute(FileLongName, 'TLAN') as TLAN, // Language(s)
Attribute(FileLongName, 'TLEN') as TLEN, // Length
Attribute(FileLongName, 'TMED') as TMED, // Media type
Attribute(FileLongName, 'TOAL') as TOAL, // Original album/movie/show title
Attribute(FileLongName, 'TOFN') as TOFN, // Original filename
Attribute(FileLongName, 'TOLY') as TOLY, // Original lyricist(s)/text writer(s)
Attribute(FileLongName, 'TOPE') as TOPE, // Original artist(s)/performer(s)
Attribute(FileLongName, 'TORY') as TORY, // Original release year
Attribute(FileLongName, 'TOWN') as TOWN, // File owner/licensee
Attribute(FileLongName, 'TPE1') as TPE1, // Lead performer(s)/Soloist(s)
Attribute(FileLongName, 'TPE2') as TPE2, // Band/orchestra/accompaniment
Attribute(FileLongName, 'TPE3') as TPE3, // Conductor/performer refinement
Attribute(FileLongName, 'TPE4') as TPE4, // Interpreted, remixed, or otherwise modified by
Attribute(FileLongName, 'TPOS') as TPOS, // Part of a set
Attribute(FileLongName, 'TPUB') as TPUB, // Publisher
Attribute(FileLongName, 'TRCK') as TRCK, // Track number/Position in set
Attribute(FileLongName, 'TRDA') as TRDA, // Recording dates
Attribute(FileLongName, 'TRSN') as TRSN, // Internet radio station name
Attribute(FileLongName, 'TRSO') as TRSO, // Internet radio station owner
Attribute(FileLongName, 'TSIZ') as TSIZ, // Size
Attribute(FileLongName, 'TSRC') as TSRC, // ISRC (international standard recording code)
Attribute(FileLongName, 'TSSE') as TSSE, // Software/Hardware and settings used for
encoding

```

```

Attribute(FileLongName, 'TYER') as TYER, // Year
Attribute(FileLongName, 'TXXX') as TXXX, // User defined text information frame
Attribute(FileLongName, 'UFID') as UFID, // Unique file identifier
Attribute(FileLongName, 'USER') as USER, // Terms of use
Attribute(FileLongName, 'USLT') as USLT, // Unsynchronized lyric/text transcription
Attribute(FileLongName, 'WCOM') as WCOM, // Commercial information
Attribute(FileLongName, 'WCOP') as WCOP, // Copyright/Legal information
Attribute(FileLongName, 'WOAF') as WOAF, // Official audio file webpage
Attribute(FileLongName, 'WOAR') as WOAR, // Official artist/performer webpage
Attribute(FileLongName, 'WOAS') as WOAS, // Official audio source webpage
Attribute(FileLongName, 'WORS') as WORS, // Official internet radio station homepage
Attribute(FileLongName, 'WPAY') as WPAY, // Payment
Attribute(FileLongName, 'WPUB') as WPUB, // Publishers official webpage
Attribute(FileLongName, 'WXXX') as WXXX; // User defined URL link frame
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt

```

Example 2: JPEG

This script reads all possible EXIF meta tags from JPG files in folder *MyPictures*.

```

// Script to read Jpeg Exif meta tags
for each vExt in 'jpg', 'jpeg', 'jpe', 'jfif', 'jif', 'jfi'
for each vFoundFile in filelist( GetFolderPath('MyPictures') & '\*.' & vExt )
FileList:
LOAD FileLongName,
    subfield(FileLongName, '\',-1) as FileshortName,
    num(FileSize(FileLongName), '# ### ### ##', ',', ' ') as Filesize,
    FileTime(FileLongName) as FileTime,
    // ***** Exif Main (IFD0) Attributes *****
    Attribute(FileLongName, 'ImageWidth') as ImageWidth,
    Attribute(FileLongName, 'ImageLength') as ImageLength,
    Attribute(FileLongName, 'BitsPerSample') as BitsPerSample,
    Attribute(FileLongName, 'Compression') as Compression,
    // examples: 1=uncompressed, 2=CCITT, 3=CCITT 3, 4=CCITT 4,
    // 5=LZW, 6=JPEG (old style), 7=JPEG, 8=Deflate, 32773=PackBits RLE,
    Attribute(FileLongName, 'PhotometricInterpretation') as PhotometricInterpretation,
    // examples: 0=WhiteIsZero, 1=BlackIsZero, 2=RGB, 3=Palette, 5=CMYK, 6=YCbCr,
    Attribute(FileLongName, 'ImageDescription') as ImageDescription,
    Attribute(FileLongName, 'Make') as Make,
    Attribute(FileLongName, 'Model') as Model,
    Attribute(FileLongName, 'StripOffsets') as StripOffsets,
    Attribute(FileLongName, 'Orientation') as Orientation,
    // examples: 1=TopLeft, 2=TopRight, 3=BottomRight, 4=BottomLeft,
    // 5=LeftTop, 6=RightTop, 7=RightBottom, 8=LeftBottom,
    Attribute(FileLongName, 'SamplesPerPixel') as SamplesPerPixel,
    Attribute(FileLongName, 'RowsPerStrip') as RowsPerStrip,
    Attribute(FileLongName, 'StripByteCounts') as StripByteCounts,
    Attribute(FileLongName, 'XResolution') as XResolution,
    Attribute(FileLongName, 'YResolution') as YResolution,
    Attribute(FileLongName, 'PlanarConfiguration') as PlanarConfiguration,
    // examples: 1=chunky format, 2=planar format,
    Attribute(FileLongName, 'ResolutionUnit') as ResolutionUnit,
    // examples: 1=none, 2=inches, 3=centimeters,
    Attribute(FileLongName, 'TransferFunction') as TransferFunction,

```

```

Attribute(FileLongName, 'Software') as Software,
Attribute(FileLongName, 'DateTime') as DateTime,
Attribute(FileLongName, 'Artist') as Artist,
Attribute(FileLongName, 'HostComputer') as HostComputer,
Attribute(FileLongName, 'WhitePoint') as WhitePoint,
Attribute(FileLongName, 'PrimaryChromaticities') as PrimaryChromaticities,
Attribute(FileLongName, 'YcbCrCoefficients') as YcbCrCoefficients,
Attribute(FileLongName, 'YcbCrSubSampling') as YcbCrSubSampling,
Attribute(FileLongName, 'YcbCrPositioning') as YcbCrPositioning,
// examples: 1=centered, 2=co-sited,
Attribute(FileLongName, 'ReferenceBlackwhite') as ReferenceBlackwhite,
Attribute(FileLongName, 'Rating') as Rating,
Attribute(FileLongName, 'RatingPercent') as RatingPercent,
Attribute(FileLongName, 'ThumbnailFormat') as ThumbnailFormat,
// examples: 0=Raw Rgb, 1=Jpeg,
Attribute(FileLongName, 'Copyright') as Copyright,
Attribute(FileLongName, 'ExposureTime') as ExposureTime,
Attribute(FileLongName, 'FNumber') as FNumber,
Attribute(FileLongName, 'ExposureProgram') as ExposureProgram,
// examples: 0=Not defined, 1=Manual, 2=Normal program, 3=Aperture priority, 4=Shutter
priority,
// 5=Creative program, 6=Action program, 7=Portrait mode, 8=Landscape mode, 9=Bulb,
Attribute(FileLongName, 'ISOSpeedRatings') as ISOSpeedRatings,
Attribute(FileLongName, 'TimeZoneOffset') as TimeZoneOffset,
Attribute(FileLongName, 'SensitivityType') as SensitivityType,
// examples: 0=Unknown, 1=Standard output sensitivity (SOS), 2=Recommended exposure index
(REI),
// 3=ISO speed, 4=Standard output sensitivity (SOS) and Recommended exposure index (REI),
// 5=Standard output sensitivity (SOS) and ISO Speed, 6=Recommended exposure index (REI)
and ISO Speed,
// 7=Standard output sensitivity (SOS) and Recommended exposure index (REI) and ISO speed,
Attribute(FileLongName, 'ExifVersion') as ExifVersion,
Attribute(FileLongName, 'DateTimeOriginal') as DateTimeOriginal,
Attribute(FileLongName, 'DateTimeDigitized') as DateTimeDigitized,
Attribute(FileLongName, 'ComponentsConfiguration') as ComponentsConfiguration,
// examples: 1=Y, 2=Cb, 3=Cr, 4=R, 5=G, 6=B,
Attribute(FileLongName, 'CompressedBitsPerPixel') as CompressedBitsPerPixel,
Attribute(FileLongName, 'ShutterSpeedValue') as ShutterSpeedValue,
Attribute(FileLongName, 'ApertureValue') as ApertureValue,
Attribute(FileLongName, 'BrightnessValue') as BrightnessValue, // examples: -1=Unknown,
Attribute(FileLongName, 'ExposureBiasValue') as ExposureBiasValue,
Attribute(FileLongName, 'MaxApertureValue') as MaxApertureValue,
Attribute(FileLongName, 'SubjectDistance') as SubjectDistance,
// examples: 0=Unknown, -1=Infinity,
Attribute(FileLongName, 'MeteringMode') as MeteringMode,
// examples: 0=Unknown, 1=Average, 2=CenterweightedAverage, 3=Spot,
// 4=MultiSpot, 5=Pattern, 6=Partial, 255=Other,
Attribute(FileLongName, 'LightSource') as LightSource,
// examples: 0=Unknown, 1=Daylight, 2=Fluorescent, 3=Tungsten, 4=Flash, 9=Fine weather,
// 10=Cloudy weather, 11=Shade, 12=Daylight fluorescent,
// 13=Day white fluorescent, 14=Cool white fluorescent,
// 15=White fluorescent, 17=Standard light A, 18=Standard light B, 19=Standard light C,
// 20=D55, 21=D65, 22=D75, 23=D50, 24=ISO studio tungsten, 255=other light source,
Attribute(FileLongName, 'Flash') as Flash,
Attribute(FileLongName, 'FocalLength') as FocalLength,
Attribute(FileLongName, 'SubjectArea') as SubjectArea,
Attribute(FileLongName, 'MakerNote') as MakerNote,

```

```

Attribute(FileLongName, 'UserComment') as UserComment,
Attribute(FileLongName, 'SubSecTime') as SubSecTime,
Attribute(FileLongName, 'SubsecTimeOriginal') as SubsecTimeOriginal,
Attribute(FileLongName, 'SubsecTimeDigitized') as SubsecTimeDigitized,
Attribute(FileLongName, 'XPTitle') as XPTitle,
Attribute(FileLongName, 'XPComment') as XPComment,
Attribute(FileLongName, 'XPAuthor') as XPAuthor,
Attribute(FileLongName, 'XPKeywords') as XPKeywords,
Attribute(FileLongName, 'XPSubject') as XPSubject,
Attribute(FileLongName, 'FlashpixVersion') as FlashpixVersion,
Attribute(FileLongName, 'ColorSpace') as ColorSpace, // examples: 1=sRGB,
65535=Uncalibrated,
Attribute(FileLongName, 'PixelXDimension') as PixelXDimension,
Attribute(FileLongName, 'PixelYDimension') as PixelYDimension,
Attribute(FileLongName, 'RelatedSoundFile') as RelatedSoundFile,
Attribute(FileLongName, 'FocalPlaneXResolution') as FocalPlaneXResolution,
Attribute(FileLongName, 'FocalPlaneYResolution') as FocalPlaneYResolution,
Attribute(FileLongName, 'FocalPlaneResolutionUnit') as FocalPlaneResolutionUnit,
// examples: 1=None, 2=Inch, 3=Centimeter,
Attribute(FileLongName, 'ExposureIndex') as ExposureIndex,
Attribute(FileLongName, 'SensingMethod') as SensingMethod,
// examples: 1=Not defined, 2=One-chip color area sensor, 3=Two-chip color area sensor,
// 4=Three-chip color area sensor, 5=Color sequential area sensor,
// 7=Trilinear sensor, 8=Color sequential linear sensor,
Attribute(FileLongName, 'FileSource') as FileSource,
// examples: 0=Other, 1=Scanner of transparent type,
// 2=Scanner of reflex type, 3=Digital still camera,
Attribute(FileLongName, 'SceneType') as SceneType,
// examples: 1=A directly photographed image,
Attribute(FileLongName, 'CFAPattern') as CFAPattern,
Attribute(FileLongName, 'CustomRendered') as CustomRendered,
// examples: 0=Normal process, 1=Custom process,
Attribute(FileLongName, 'ExposureMode') as ExposureMode,
// examples: 0=Auto exposure, 1=Manual exposure, 2=Auto bracket,
Attribute(FileLongName, 'whiteBalance') as whiteBalance,
// examples: 0=Auto white balance, 1=Manual white balance,
Attribute(FileLongName, 'DigitalZoomRatio') as DigitalZoomRatio,
Attribute(FileLongName, 'FocalLengthIn35mmFilm') as FocalLengthIn35mmFilm,
Attribute(FileLongName, 'SceneCaptureType') as SceneCaptureType,
// examples: 0=Standard, 1=Landscape, 2=Portrait, 3=Night scene,
Attribute(FileLongName, 'GainControl') as GainControl,
// examples: 0=None, 1=Low gain up, 2=High gain up, 3=Low gain down, 4=High gain down,
Attribute(FileLongName, 'Contrast') as Contrast,
// examples: 0=Normal, 1=Soft, 2=Hard,
Attribute(FileLongName, 'Saturation') as Saturation,
// examples: 0=Normal, 1=Low saturation, 2=High saturation,
Attribute(FileLongName, 'Sharpness') as Sharpness,
// examples: 0=Normal, 1=Soft, 2=Hard,
Attribute(FileLongName, 'SubjectDistanceRange') as SubjectDistanceRange,
// examples: 0=Unknown, 1=Macro, 2=Close view, 3=Distant view,
Attribute(FileLongName, 'ImageUniqueID') as ImageUniqueID,
Attribute(FileLongName, 'BodySerialNumber') as BodySerialNumber,
Attribute(FileLongName, 'CMNT_GAMMA') as CMNT_GAMMA,
Attribute(FileLongName, 'PrintImageMatching') as PrintImageMatching,
Attribute(FileLongName, 'OffsetSchema') as OffsetSchema,
// ***** Interoperability Attributes *****
Attribute(FileLongName, 'InteroperabilityIndex') as InteroperabilityIndex,

```

```

Attribute(FileLongName, 'InteroperabilityVersion') as InteroperabilityVersion,
Attribute(FileLongName, 'InteroperabilityRelatedImageFileFormat') as
InteroperabilityRelatedImageFileFormat,
Attribute(FileLongName, 'InteroperabilityRelatedImageWidth') as
InteroperabilityRelatedImageWidth,
Attribute(FileLongName, 'InteroperabilityRelatedImageLength') as
InteroperabilityRelatedImageLength,
Attribute(FileLongName, 'InteroperabilityColorSpace') as InteroperabilityColorSpace,
// examples: 1=sRGB, 65535=Uncalibrated,
Attribute(FileLongName, 'InteroperabilityPrintImageMatching') as
InteroperabilityPrintImageMatching,
// ***** GPS Attributes *****
Attribute(FileLongName, 'GPSVersionID') as GPSVersionID,
Attribute(FileLongName, 'GPSLatitudeRef') as GPSLatitudeRef,
Attribute(FileLongName, 'GPSLatitude') as GPSLatitude,
Attribute(FileLongName, 'GPSLongitudeRef') as GPSLongitudeRef,
Attribute(FileLongName, 'GPSLongitude') as GPSLongitude,
Attribute(FileLongName, 'GPSAltitudeRef') as GPSAltitudeRef,
// examples: 0=Above sea level, 1=Below sea level,
Attribute(FileLongName, 'GPSAltitude') as GPSAltitude,
Attribute(FileLongName, 'GPSTimeStamp') as GPSTimeStamp,
Attribute(FileLongName, 'GPSSatellites') as GPSSatellites,
Attribute(FileLongName, 'GPSStatus') as GPSStatus,
Attribute(FileLongName, 'GPSMeasureMode') as GPSMeasureMode,
Attribute(FileLongName, 'GPSDOP') as GPSDOP,
Attribute(FileLongName, 'GPSSpeedRef') as GPSSpeedRef,
Attribute(FileLongName, 'GPSSpeed') as GPSSpeed,
Attribute(FileLongName, 'GPSTrackRef') as GPSTrackRef,
Attribute(FileLongName, 'GPSTrack') as GPSTrack,
Attribute(FileLongName, 'GPSImgDirectionRef') as GPSImgDirectionRef,
Attribute(FileLongName, 'GPSImgDirection') as GPSImgDirection,
Attribute(FileLongName, 'GPSMapDatum') as GPSMapDatum,
Attribute(FileLongName, 'GPSDestLatitudeRef') as GPSDestLatitudeRef,
Attribute(FileLongName, 'GPSDestLatitude') as GPSDestLatitude,
Attribute(FileLongName, 'GPSDestLongitudeRef') as GPSDestLongitudeRef,
Attribute(FileLongName, 'GPSDestLongitude') as GPSDestLongitude,
Attribute(FileLongName, 'GPSDestBearingRef') as GPSDestBearingRef,
Attribute(FileLongName, 'GPSDestBearing') as GPSDestBearing,
Attribute(FileLongName, 'GPSDestDistanceRef') as GPSDestDistanceRef,
Attribute(FileLongName, 'GPSDestDistance') as GPSDestDistance,
Attribute(FileLongName, 'GPSProcessingMethod') as GPSProcessingMethod,
Attribute(FileLongName, 'GPSAreaInformation') as GPSAreaInformation,
Attribute(FileLongName, 'GPSDateStamp') as GPSDateStamp,
Attribute(FileLongName, 'GPSDifferential') as GPSDifferential;
// examples: 0=No correction, 1=Differential correction,
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt

```

Example 3: Windows media files

This script reads all possible WMA/WMV ASF meta tags in folder *MyMusic*.

```

/ Script to read WMA/WMV ASF meta tags
for each vExt in 'ASF', 'WMA', 'WMV'
for each vFoundFile in filelist( GetFolderPath('MyMusic') & '\*.' & vExt )

```

```

FileList:
LOAD FileLongName,
    subfield(FileLongName, '\',-1) as FileshortName,
    num(FileSize(FileLongName), '# ### ### ###', ',', ' ') as Filesize,
    FileTime(FileLongName) as FileTime,
    Attribute(FileLongName, 'Title') as Title,
    Attribute(FileLongName, 'Author') as Author,
    Attribute(FileLongName, 'Copyright') as Copyright,
    Attribute(FileLongName, 'Description') as Description,
    Attribute(FileLongName, 'Rating') as Rating,
    Attribute(FileLongName, 'PlayDuration') as PlayDuration,
    Attribute(FileLongName, 'MaximumBitrate') as MaximumBitrate,
    Attribute(FileLongName, 'WMFSDKVersion') as WMFSDKVersion,
    Attribute(FileLongName, 'WMFSDKNeeded') as WMFSDKNeeded,
    Attribute(FileLongName, 'ISVBR') as ISVBR,
    Attribute(FileLongName, 'ASFLeakyBucketPairs') as ASFLeakyBucketPairs,
    Attribute(FileLongName, 'Peakvalue') as Peakvalue,
    Attribute(FileLongName, 'AverageLevel') as AverageLevel;
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt

```

Example 4: PNG

This script reads all possible PNG meta tags in folder *MyPictures*.

```

// Script to read PNG meta tags
for each vExt in 'png'
for each vFoundFile in filelist( GetFolderPath('MyPictures') & '\*.'& vExt )
FileList:
LOAD FileLongName,
    subfield(FileLongName, '\',-1) as FileshortName,
    num(FileSize(FileLongName), '# ### ### ###', ',', ' ') as Filesize,
    FileTime(FileLongName) as FileTime,
    Attribute(FileLongName, 'Comment') as Comment,
    Attribute(FileLongName, 'Creation Time') as Creation_Time,
    Attribute(FileLongName, 'Source') as Source,
    Attribute(FileLongName, 'Title') as Title,
    Attribute(FileLongName, 'Software') as Software,
    Attribute(FileLongName, 'Author') as Author,
    Attribute(FileLongName, 'Description') as Description,
    Attribute(FileLongName, 'Copyright') as Copyright;
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt

```

ConnectionString

The **ConnectionString()** function returns the name of the active data connection for ODBC or OLE DB connections. The function returns an empty string if no **connect** statement has been executed, or after a **disconnect** statement.

Syntax:

ConnectionString()

Examples and results:

Scripting examples

Example	Result
<pre>LIB CONNECT TO 'Tutorial ODBC'; ConnectionString: Load ConnectString() as ConnectionString AutoGenerate 1;</pre>	Returns 'Tutorial ODBC' in field ConnectString. This examples assumes that you have an available data connection called Tutorial ODBC.

FileBaseName

The **FileBaseName** function returns a string containing the name of the table file currently being read, without path or extension.

Syntax:

FileBaseName()

Examples and results:

Scripting examples

Example	Result
<pre>LOAD *, filebasename() as x from C:\UserFiles\abc.txt</pre>	Will return 'abc' in field X in each record read.

FileDir

The **FileDir** function returns a string containing the path to the directory of the table file currently being read.

Syntax:

FileDir()



This function supports only folder data connections in standard mode.

Examples and results:

Scripting examples

Example	Result
<pre>Load *, filedir() as x from C:\UserFiles\abc.txt</pre>	Will return 'C:\UserFiles' in field X in each record read.

FileExtension

The **FileExtension** function returns a string containing the extension of the table file currently being read.

Syntax:

```
FileExtension()
```

Examples and results:

Scripting examples

Example	Result
LOAD *, FileExtension() as X from C:\UserFiles\abc.txt	Will return 'txt' in field X in each record read.

FileName

The **FileName** function returns a string containing the name of the table file currently being read, without path but including the extension.

Syntax:

```
FileName()
```

Examples and results:

Scripting examples

Example	Result
LOAD *, FileName() as X from C:\UserFiles\abc.txt	Will return 'abc.txt' in field X in each record read.

FilePath

The **FilePath** function returns a string containing the full path to the table file currently being read.

Syntax:

```
FilePath()
```



This function supports only folder data connections in standard mode.

Examples and results:

Scripting examples

Example	Result
Load *, FilePath() as X from C:\UserFiles\abc.txt	Will return 'C:\UserFiles\abc.txt' in field X in each record read.

FileSize

The **FileSize** function returns an integer containing the size in bytes of the file filename or, if no filename is specified, of the table file currently being read.

Syntax:

```
FileSize([filename])
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a file, if necessary including path, as a folder or web file data connection. If you don't specify a file name, the table file currently being read is used.</p> <p>Example: <code>'lib://Table Files/'</code></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> • absolute <p>Example: <code>c:\data\</code></p> <ul style="list-style-type: none"> • relative to the Qlik Sense app working directory. <p>Example: <code>data\</code></p> <ul style="list-style-type: none"> • URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <code>http://wwwqlik.com</code></p>

Examples and results:

Scripting examples

Example	Result
<code>LOAD *, Filesize() as X from abc.txt;</code>	Will return the size of the specified file (abc.txt) as an integer in field X in each record read.
<code>Filesize('lib://DataFiles/xyz.xls')</code>	Will return the size of the file xyz.xls.

FileTime

The **FileTime** function returns a timestamp in UTC format of the last modification of a specified file. If a file is not specified, the function returns a timestamp in UTC of the last modification of the currently read table file.

Syntax:

```
FileTime([ filename ])
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a file, if necessary including path, as a folder or web file data connection.</p> <p>Example: 'lib://Table Files/'</p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: c:\data\</p> <ul style="list-style-type: none"> relative to the Qlik Sense app working directory. <p>Example: data\</p> <ul style="list-style-type: none"> URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: http://wwwqlik.com</p>

Examples and results:

Script examples

Example	Result
LOAD *, FileTime() as X from abc.txt;	Will return the timestamp of the last modification of the file (abc.txt) in field X in each record read.
FileTime('xyz.xls')	Will return the timestamp of the last modification of the file xyz.xls.

GetFolderPath

The **GetFolderPath** function returns the value of the Microsoft Windows *SHGetFolderPath* function. This function takes as input the name of a Microsoft Windows folder and returns the full path of the folder.



This function is not supported in standard mode. .

Syntax:

```
GetFolderPath(foldername)
```

Arguments:

Arguments

Argument	Description
foldername	<p>Name of the Microsoft Windows folder.</p> <p>The folder name should not contain any space. Any space in the folder name seen in Windows Explorer should be removed from the folder name.</p> <p>Examples:</p> <p><i>MyMusic</i></p> <p><i>MyDocuments</i></p>

Examples and results:

The goal of this example is to get the paths of the following Microsoft Windows folders: *MyMusic*, *MyPictures* and *Windows*. Add the example script to your app and reload it.

```
LOAD
GetFolderPath('MyMusic') as MyMusic,
GetFolderPath('MyPictures') as MyPictures,
GetFolderPath('Windows') as Windows
AutoGenerate 1;
```

Once the app is reloaded, the fields *MyMusic*, *MyPictures* and *Windows* are added to the data model. Each field contains the path to the folder defined in input. For example:

- C:\Users\smu\Music for the folder *MyMusic*
- C:\Users\smu\Pictures for the folder *MyPictures*
- C:\Windows for the folder *Windows*

QvdCreateTime

This script function returns the XML-header timestamp from a QVD file, if any is present, otherwise it returns NULL. In the timestamp, time is provided in UTC.

Syntax:

```
QvdCreateTime(filename)
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> • absolute <p>Example: <i>c: data </i></p> <ul style="list-style-type: none"> • relative to the Qlik Sense app working directory. <p>Example: <i>data </i></p> <ul style="list-style-type: none"> • URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <i>http://wwwqlik.com</i></p>

Example:

```
QvdCreateTime('MyFile.qvd')
QvdCreateTime('C:\MyDir\MyFile.qvd')
QvdCreateTime('lib://DataFiles/MyFile.qvd')
```

QvdFieldName

This script function returns the name of field number **fieldno** in a QVD file. If the field does not exist NULL is returned.

Syntax:

```
QvdFieldName(filename , fieldno)
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> • absolute <p>Example: <i>c:\ data </i></p> <ul style="list-style-type: none"> • relative to the Qlik Sense app working directory. <p>Example: <i>data </i></p> <ul style="list-style-type: none"> • URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <i>http://wwwqlik.com</i></p>
fieldno	The number of the field within the table contained in the QVD file.

Examples:

```
QvdFieldName ('MyFile.qvd', 5)
QvdFieldName ('C:\MyDir\MyFile.qvd', 5)
QvdFieldName ('lib://DataFiles/MyFile.qvd', 5)
```

All three examples return the name of the fifth field of the table contained in the QVD file.

QvdNoOfFields

This script function returns the number of fields in a QVD file.

Syntax:

```
QvdNoOfFields(filename)
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">• absolute <p>Example: <i>c:\ data </i></p> <ul style="list-style-type: none">• relative to the Qlik Sense app working directory. <p>Example: <i>data </i></p> <ul style="list-style-type: none">• URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <i>http://wwwqlik.com</i></p>

Examples:

```
QvdNoOfFields ('MyFile.qvd')
QvdNoOfFields ('C:\MyDir\MyFile.qvd')
QvdNoOfFields ('lib://DataFiles/MyFile.qvd')
```

QvdNoOfRecords

Example: This script function returns the number of records currently in a QVD file.

Syntax:

```
QvdNoOfRecords (filename)
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <i>'lib://Table Files/'</i></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none">• absolute <p>Example: <i>c: data </i></p> <ul style="list-style-type: none">• relative to the Qlik Sense app working directory. <p>Example: <i>data </i></p> <ul style="list-style-type: none">• URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <i>http://wwwqlik.com</i></p>

Examples:

```
QvdNoOfRecords ('MyFile.qvd')
QvdNoOfRecords ('C:\MyDir\MyFile.qvd')
QvdNoOfRecords ('lib://DataFiles/MyFile.qvd')
```

QvdTableName

This script function returns the name of the table stored in a QVD file.

Syntax:

```
QvdTableName(filename)
```

Arguments:

Arguments	
Argument	Description
filename	<p>The name of a QVD file, if necessary including path, as a folder or web data connection.</p> <p>Example: <code>'lib://Table Files/'</code></p> <p>In legacy scripting mode, the following path formats are also supported:</p> <ul style="list-style-type: none"> absolute <p>Example: <code>c:\data\</code></p> relative to the Qlik Sense app working directory. <p>Example: <code>data\</code></p> URL address (HTTP or FTP), pointing to a location on the Internet or an intranet. <p>Example: <code>http://wwwqlik.com</code></p>

Examples:

```
QvdTableName ('MyFile.qvd')
QvdTableName ('C:\MyDir\MyFile.qvd')
QvdTableName ('lib://data\MyFile.qvd')
```

5.11 Financial functions

Financial functions can be used in the data load script and in chart expressions to calculate payments and interest rates.

For all the arguments, cash that is paid out is represented by negative numbers. Cash received is represented by positive numbers.

Listed here are the arguments that are used in the financial functions (excepting the ones beginning with `range-`).



*For all financial functions it is vital that you are consistent when specifying units for `rate` and `nper`. If monthly payments are made on a five-year loan at 6% annual interest, use 0.005 (6%/12) for `rate` and 60 (5*12) for `nper`. If annual payments are made on the same loan, use 6% for `rate` and 5 for `nper`.*

Financial functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

FV

This function returns the future value of an investment based on periodic, constant payments and a simple annual interest.

```
FV (rate, nper, pmt [ ,pv [ , type ] ])
```

nPer

This function returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

```
nPer (rate, pmt, pv [ ,fv [ , type ] ])
```

Pmt

This function returns the payment for a loan based on periodic, constant payments and a constant interest rate. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.

```
Pmt (rate, nper, pv [ ,fv [ , type ] ] )
```

PV

This function returns the present value of an investment.

```
PV (rate, nper, pmt [ ,fv [ , type ] ])
```

Rate

This function returns the interest rate per period on annuity. The result has a default number format of **Fix** two decimals and %.

```
Rate (nper, pmt , pv [ ,fv [ , type ] ] )
```

BlackAndSchole

The Black and Scholes model is a mathematical model for financial market derivative instruments. The formula calculates the theoretical value of an option. In Qlik Sense, the **BlackAndSchole** function returns the value according to the Black and Scholes unmodified formula (European style options).

```
BlackAndSchole(strike , time_left , underlying_price , vol , risk_free_rate , type)
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
strike	The future purchase price of the stock.
time_left	The number of time periods remaining.
underlying_price	The current value of the stock.

Argument	Description
vol	Volatility (of the stock price) expressed as a percentage in decimal form, per time period.
risk_free_rate	The risk-free rate expressed as a percentage in decimal form, per time period.
call_or_put	The type of option: 'c', 'call' or any non-zero numeric value for call options 'p', 'put' or 0 for put options.

Limitations:

The value of strike, time_left, and underlying_price must be >0.

The value of vol and risk_free_rate must be: <0 or >0.

Examples and results:

Scripting examples

Example	Result
<code>BlackAndSchole(130, 4, 68.5, 0.4, 0.04, 'call')</code> This calculates the theoretical price of an option to buy a share that is worth 68.5 today, at a value of 130 in 4 years. The formula uses a volatility of 0.4 (40%) per year and a risk-free interest rate of 0.04 (4%).	Returns 11.245

FV

This function returns the future value of an investment based on periodic, constant payments and a simple annual interest.

Syntax:

```
FV(rate, nper, pmt [ ,pv [ , type ] ])
```

Return data type: numeric. By default, the result will be formatted as currency..

Arguments:

Arguments

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.

Argument	Description
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero).
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Scripting example

Example	Result
You are paying a new household appliance by 36 monthly installments of \$20. The interest rate is 6% per annum. The bill comes at the end of every month. What is the total invested, when the last bill has been paid? <code>FV(0.005, 36, -20)</code>	Returns \$786.72

nPer

This function returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Syntax:

```
nPer(rate, pmt, pv [ ,fv [ , type ] ])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero).
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Scripting example

Example	Result
You want to sell a household appliance by monthly installments of \$20. The interest rate is 6% per annum. The bill comes at the end of every month. How many periods are required if the value of the money received after the last bill has been paid should equal \$800? nPer(0.005,-20,0,800)	Returns 36.56

Pmt

This function returns the payment for a loan based on periodic, constant payments and a constant interest rate. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.

```
Pmt(rate, nper, pv [ ,fv [ , type ] ] )
```

Return data type: numeric. By default, the result will be formatted as currency..

To find the total amount paid over the duration of the loan, multiply the returned **pmt** value by **nper**.

Arguments:

Arguments

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero).
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Scripting examples

Example	Result
The following formula returns the monthly payment on a \$20,000 loan at an annual rate of 10 percent, that must be paid off in 8 months: Pmt(0.1/12,8,20000)	Returns - \$2,594.66

Example	Result
For the same loan, if payment is due at the beginning of the period, the payment is: <code>Pmt(0.1/12,8,20000,0,1)</code>	Returns - \$2,573.21

PV

This function returns the present value of an investment.

```
PV(rate, nper, pmt [ ,fv [ , type ] ])
```

Return data type: numeric. By default, the result will be formatted as currency..

The present value is the total amount that a series of future payments is worth right now. For example, when borrowing money, the loan amount is the present value to the lender.

Arguments:

Arguments

Argument	Description
rate	The interest rate per period.
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Scripting example

Example	Result
What is the present value of a debt, when you have to pay \$100 at the end of each month during a five-year period, given an interest rate of 7%? <code>PV(0.07/12,12*5,-100,0,0)</code>	Returns \$5,050.20

Rate

This function returns the interest rate per period on annuity. The result has a default number format of **Fix** two decimals and %.

Syntax:

```
Rate(nper, pmt , pv [ ,fv [ , type ] ])
```

Return data type: numeric.

The **rate** is calculated by iteration and can have zero or more solutions. If the successive results of **rate** do not converge, a NULL value will be returned.

Arguments:

Arguments

Argument	Description
nper	The total number of payment periods in an annuity.
pmt	The payment made each period. It cannot change over the life of the annuity. A payment is stated as a negative number, for example, -20.
pv	The present value, or lump-sum amount, that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero).
fv	The future value, or cash balance, you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0.
type	Should be 0 if payments are due at the end of the period and 1 if payments are due at the beginning of the period. If type is omitted, it is assumed to be 0.

Examples and results:

Scripting example

Example	Result
What is the interest rate of a five-year \$10,000 annuity loan with monthly payments of \$300? Rate(60, -300, 10000)	Returns 2.00%

5.12 Formatting functions

The formatting functions impose the display format on the input numeric fields or expressions. Depending on data type, you can specify the characters for the decimal separator, thousands separator, and so on.

The functions all return a dual value with both the string and the number value, but can be thought of as performing a number-to-string conversion. **Dual()** is a special case, but the other formatting functions take the numeric value of the input expression and generate a string representing the number.

In contrast, the interpretation functions do the opposite: they take string expressions and evaluate them as numbers, specifying the format of the resulting number.

The functions can be used both in data load scripts and chart expressions.



All number representations are given with a decimal point as the decimal separator.

Formatting functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

ApplyCodepage

ApplyCodepage() applies a different code page character set to the field or text stated in the expression. The **codepage** argument must be in number format.

```
ApplyCodepage (text, codepage)
```

Date

Date() formats an expression as a date using the format set in the system variables in the data load script, or the operating system, or a format string, if supplied.

```
Date (number[, format])
```

Dual

Dual() combines a number and a string into a single record, such that the number representation of the record can be used for sorting and calculation purposes, while the string value can be used for display purposes.

```
Dual (text, number)
```

Interval

Interval() formats a number as a time interval using the format in the system variables in the data load script, or the operating system, or a format string, if supplied.

```
Interval (number[, format])
```

Money

Money() formats an expression numerically as a money value, in the format set in the system variables set in the data load script, or in the operating system, unless a format string is supplied, and optional decimal and thousands separators.

```
Money (number[, format[, dec_sep [, thou_sep]]])
```

Num

Num() formats a number, that is it converts the numeric value of the input to display text using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.

```
Num (number[, format[, dec_sep [, thou_sep]]])
```

Time

Time() formats an expression as a time value, in the time format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.

```
Time (number[, format])
```

Timestamp

TimeStamp() formats an expression as a date and time value, in the timestamp format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.

```
Timestamp (number[, format])
```

See also:

□ [Interpretation functions \(page 1224\)](#)

ApplyCodepage

ApplyCodepage() applies a different code page character set to the field or text stated in the expression. The **codepage** argument must be in number format.



Although ApplyCodepage can be used in chart expressions, it is more commonly used as a script function in the data load editor. For example, as you load files that might have been saved in different character sets out of your control, you can apply the code page that represents the character set you require.

Syntax:

```
ApplyCodepage (text, codepage)
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	Field or text to which you want to apply a different code page, given by the argument codepage .
codepage	Number representing the code page to be applied to the field or expression given by text .

Examples and results:

Scripting examples

Example	Result
<pre>LOAD ApplyCodepage(ROWX,1253) as GreekProduct, ApplyCodepage (ROWY, 1255) as HebrewProduct, ApplyCodepage (ROWZ, 65001) as EnglishProduct; SQL SELECT ROWX, ROWY, ROWZ From Products;</pre>	<p>When loading from SQL the source might have a mixture of different character sets: Cyrillic, Hebrew, and so on, from the UTF-8 format. These would be required to be loaded row by row, applying a different code page for each row.</p> <p>The codepage value 1253 represents Windows Greek character set, the value 1255 represents Hebrew, and the value 65001 represents standard Latin UTF-8 characters.</p>

See also: *Character set (page 161)*

Date

Date() formats an expression as a date using the format set in the system variables in the data load script, or the operating system, or a format string, if supplied.

Syntax:

```
Date(number[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String describing the format of the resulting string. If no format string is supplied, the date format set in the system variables in the data load script, or the operating system is used.

Examples and results:

The examples below assume the following default settings:

- Date setting 1: YY-MM-DD
- Date setting 2: M/D/YY

Example:

```
Date( A )
where A=35648
```

Results table

Results	Setting 1	Setting 2
String:	97-08-06	8/6/97
Number:	35648	35648

Example:

```
Date( A, 'YY.MM.DD' )
where A=35648
```

Results table

Results	Setting 1	Setting 2
String:	97.08.06	97.08.06
Number:	35648	35648

Example:

```
Date( A, 'DD.MM.YYYY' )
where A=35648.375
```

Results table

Results	Setting 1	Setting 2
String:	06.08.1997	06.08.1997
Number:	35648.375	35648.375

Example:

```
Date( A, 'YY.MM.DD' )
where A=8/6/97
```

Results table

Results	Setting 1	Setting 2
String:	NULL (nothing)	97.08.06
Number:	NULL	35648

Dual

Dual() combines a number and a string into a single record, such that the number representation of the record can be used for sorting and calculation purposes, while the string value can be used for display purposes.

Syntax:

```
Dual(text, number)
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
text	The string value to be used in combination with the number argument.
number	The number to be used in combination with the string in the string argument.

In Qlik Sense, all field values are potentially dual values. This means that the field values can have both a numeric value and a textual value. An example is a date that could have a numeric value of 40908 and the textual representation '2011-12-31'.



When several data items read into one field have different string representations but the same valid number representation, they will all share the first string representation encountered.



*The **dual** function is typically used early in the script, before other data is read into the field concerned, in order to create that first string representation, which will be shown in filter panes.*

Examples and results:

Scripting examples

Example	Description
<pre>Add the following examples to your script and run it. Load dual (NameDay,NumDay) as DayOfWeek inline [NameDay,NumDay Monday,0 Tuesday,1 Wednesday,2 Thursday,3 Friday,4 Saturday,5 Sunday,6];</pre>	<p>The field DayOfWeek can be used in a visualization, as a dimension, for example. In a table with the week days are automatically sorted into their correct number sequence, instead of alphabetical order.</p>
<pre>Load Dual('Q' & Ceil(Month(Now())/3), Ceil(Month(Now())/3)) as Quarter AutoGenerate 1;</pre>	<p>This example finds the current quarter. It is displayed as Q1 when the Now() function is run in the first three months of the year, Q2 for the second three months, and so on. However, when used in sorting, the field Quarter will behave as its numerical value: 1 to 4.</p>

Example	Description
Dual('Q' & Ceil(Month(Date)/3), Ceil(Month(Date)/3)) as Quarter	As in the previous example, the field Quarter is created with the text values 'Q1' to 'Q4', and assigned the numeric values 1 to 4. In order to use this in the script the values for Date must be loaded.
Dual(weekYear(Date) & '-w' & Week(Date), WeekStart(Date)) as Yearweek	This example creates a field YearWeek with text values of the form '2012-W22' and at the same time, assigns a numeric value corresponding to the date number of the first day of the week, for example: 41057. In order to use this in the script the values for Date must be loaded.

Interval

Interval() formats a number as a time interval using the format in the system variables in the data load script, or the operating system, or a format string, if supplied.

Intervals may be formatted as a time, as days or as a combination of days, hours, minutes, seconds and fractions of seconds.

Syntax:

```
Interval(number[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String describing how the resulting interval string is to be formatted. If omitted, the short date format, time format, and decimal separator set in the operating system are used.

Examples and results:

The examples below assume the following default settings:

- Date format setting 1: YY-MM-DD
- Date format setting 2: hh:mm:ss
- Number decimal separator: .

Results table

Example	String	Number
Interval(A) where A=0.375	09:00:00	0.375

Example	String	Number
<code>Interval(A)</code> where A=1.375	33:00:00	1.375
<code>Interval(A, 'D hh:mm')</code> where A=1.375	1 09:00	1.375
<code>Interval(A-B, 'D hh:mm')</code> where A=97-08-06 09:00:00 and B=96-08-06 00:00:00	365 09:00	365.375

Money

Money() formats an expression numerically as a money value, in the format set in the system variables set in the data load script, or in the operating system, unless a format string is supplied, and optional decimal and thousands separators.

Syntax:

```
Money(number[, format[, dec_sep[, thou_sep]]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String describing how the resulting money string is to be formatted.
dec_sep	String specifying the decimal number separator.
thou_sep	String specifying the thousands number separator.

If arguments 2-4 are omitted, the currency format set in the operating system is used.

Examples and results:

The examples below assume the following default settings:

- MoneyFormat setting 1: kr ##0,00, MoneyThousandSep','
- MoneyFormat setting 2: \$ #,##0.00, MoneyThousandSep','

Example:

```
Money( A )
where A=35648
```

Results table

Results	Setting 1	Setting 2
String:	kr 35 648,00	\$ 35,648.00
Number:	35648.00	35648.00

Example:

```
Money( A, '#,##0 ¥', '.', ',')  
where A=3564800
```

Results table

Results	Setting 1	Setting 2
String:	3,564,800 ¥	3,564,800 ¥
Number:	3564800	3564800

Num

Num() formats a number, that is it converts the numeric value of the input to display text using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.

Syntax:

```
Num(number[, format[, dec_sep [, thou_sep]]])
```

Return data type: dual

The Num function returns a dual value with both the string and the numeric value. The function takes the numeric value of the input expression and generates a string representing the number.

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String specifying how the resulting string is to be formatted. If omitted, the decimal and thousand separators that are set in the data load script are used.
dec_sep	String specifying the decimal number separator. If omitted, the value of the variable DecimalSep that is set in the data load script is used.
thou_sep	String specifying the thousands number separator. If omitted, the value of the variable ThousandSep that is set in the data load script is used.

Example: Chart expression

Example:

The following table shows the results when field A equals 35648.312.

A	Results
Result	
Num(A)	35648.312 (depends on environment variables in script)
Num(A, '0.0', ',')	35648.3
Num(A, '0,00', ',')	35648,31
Num(A, '#,##0.0', ',', ',')	35,648.3
Num(A, '# ##0', ',', ',')	35 648

Example: Load script

Load script

Num can be used in load script to format a number, even if the thousand and decimal separators are already set in the script. The load script below includes specific thousand and decimal separators but then uses *Num* to format data in different ways.

In the **Data load editor**, create a new section, and then add the example script and run it. Then add, at least, the fields listed in the results column to a sheet in your app to see the result.

```
SET ThousandSep=',';
SET DecimalSep='.';
Transactions:
Load
*,
Num(transaction_amount) as [No formatting],
Num(transaction_amount,'0') as [0],
Num(transaction_amount,'#,##0') as [#,##0],
Num(transaction_amount,'# ###,00') as [# ###,00],
Num(transaction_amount,'# ###,00','.',',') as [# ###,00 , , , ],
Num(transaction_amount,'# ###.00','.',',') as [#,###.00 , . , , ],
Num(transaction_amount,'$#,###.00') as [$#,###.00],
;
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, s, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

Qlik Sense table showing the results from different uses of the *Num* function in the load script. The fourth column of the table contains incorrect formatting use, for example purposes.

No formatting	0	#,##0	# ###,00	# ###,00 ,',' , ''	#,###.00 ,',' , ''	\$#,###.00
-59.18	-59	-59	-59###,00	-59,18	-59.18	\$-59,18
15.75	16	16	16###,00	15,75	15.75	\$15,75
1251	1251	1,251	1251###,00	1 251,00	1,251.00	\$1,251.00
3177.4	3177	3,177	3177###,00	3 177,40	3,177.40	\$3,177.40
5356.31	5356	5,356	5356###,00	5 356,31	5,356.31	\$5,356.31
12423.56	12424	12,424	12424###,00	12 423,56	12,423.56	\$12,423.56
21484.21	21484	21,484	21484###,00	21 484,21	21,484.21	\$21,484.21

Example: Load script

Load script

Num can be used in a load script to format a number as a percentage.

In the **Data load editor**, create a new section, and then add the example script and run it. Then add, at least, the fields listed in the results column to a sheet in your app to see the result.

```
SET ThousandSep=',';
SET DecimalSep='.';
Transactions:
Load
*,
Num(discount,'#,##0%') as [Discount #,##0%]
;
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

Qlik Sense table showing the results of the *Num* function being used in the load script to format percentages.

Discount	Discount #,##0%
0.333333333333333	33%

Discount	Discount #,##0%
0.22	22%
0	0%
.14	14%
0.1	10%
0	0%
75	7,500%

Time

Time() formats an expression as a time value, in the time format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.

Syntax:

```
Time(number[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String describing how the resulting time string is to be formatted. If omitted, the short date format, time format, and decimal separator set in the operating system is used.

Examples and results:

The examples below assume the following default settings:

- Time format setting 1: hh:mm:ss
- Time format setting 2: hh.mm.ss

Example:

```
Time( A )  
where A=0.375
```

Results table

Results	Setting 1	Setting 2
String:	09:00:00	09.00.00
Number:	0.375	0.375

Example:

```
Time( A )
where A=35648.375
```

Results table

Results	Setting 1	Setting 2
String:	09:00:00	09.00.00
Number:	35648.375	35648.375

Example:

```
Time( A, 'hh-mm' )
where A=0.99999
```

Results table

Results	Setting 1	Setting 2
String:	23-59	23-59
Number:	0.99999	0.99999

Timestamp

TimeStamp() formats an expression as a date and time value, in the timestamp format set in the system variables in the data load script, or in the operating system, unless a format string is supplied.

Syntax:

```
Timestamp(number[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
number	The number to be formatted.
format	String describing how the resulting timestamp string is to be formatted. If omitted, the short date format, time format, and decimal separator set in the operating system is used.

Examples and results:

The examples below assume the following default settings:

- TimeStampFormat setting 1: YY-MM-DD hh:mm:ss
- TimeStampFormat setting 2: M/D/YY hh:mm:ss

Example:

```
Timestamp( A )
where A=35648.375
```

Results table

Results	Setting 1	Setting 2
String:	97-08-06 09:00:00	8/6/97 09:00:00
Number:	35648.375	35648.375

Example:

```
Timestamp( A , 'YYYY-MM-DD hh.mm' )
where A=35648
```

Results table

Results	Setting 1	Setting 2
String:	1997-08-06 00.00	1997-08-06 00.00
Number:	35648	35648

5.13 General numeric functions

In these general numeric functions, the arguments are expressions where **x** should be interpreted as a real valued number. All functions can be used in both data load scripts and chart expressions.

General numeric functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

bitcount

BitCount() returns how many bits in the binary equivalent of a decimal number are set to 1. That is, the function returns the number of set bits in **integer_number**, where **integer_number** is interpreted as a signed 32-bit integer.

```
BitCount(integer_number)
```

div

Div() returns the integer part of the arithmetic division of the first argument by the second argument. Both parameters are interpreted as real numbers, that is, they do not have to be integers.

```
Div(integer_number1, integer_number2)
```

fabs

Fabs() returns the absolute value of **x**. The result is a positive number.

Fabs (x)

fact

Fact() returns the factorial of a positive integer **x**.

Fact (x)

frac

Frac() returns the fraction part of **x**.

Frac (x)

sign

Sign() returns 1, 0 or -1 depending on whether **x** is a positive number, 0, or a negative number.

Sign (x)

Combination and permutation functions

combin

Combin() returns the number of combinations of **q** elements that can be picked from a set of **p** items. As represented by the formula: $\text{combin}(p, q) = p! / q!(p-q)!$ The order in which the items are selected is insignificant.

Combin (p, q)

permut

Permut() returns the number of permutations of **q** elements that can be selected from a set of **p** items. As represented by the formula: $\text{Permut}(p, q) = (p)! / (p - q)!$ The order in which the items are selected is significant.

Permut (p, q)

Modulo functions

fmod

fmod() is a generalized modulo function that returns the remainder part of the integer division of the first argument (the dividend) by the second argument (the divisor). The result is a real number. Both arguments are interpreted as real numbers, that is, they do not have to be integers.

Fmod (a, b)

mod

Mod() is a mathematical modulo function that returns the non-negative remainder of an integer division. The first argument is the dividend, the second argument is the divisor, Both arguments must be integer values.

Mod(integer_number1, integer_number2)

Parity functions

even

Even() returns True (-1), if **integer_number** is an even integer or zero. It returns False (0), if **integer_number** is an odd integer, and NULL if **integer_number** is not an integer.

```
Even(integer_number)
```

odd

Odd() returns True (-1), if **integer_number** is an odd integer or zero. It returns False (0), if **integer_number** is an even integer, and NULL if **integer_number** is not an integer.

```
Odd(integer_number)
```

Rounding functions

ceil

Ceil() rounds up a number to the nearest multiple of the **step** shifted by the **offset** number.

```
Ceil(x[, step[, offset]])
```

floor

Floor() rounds down a number to the nearest multiple of the **step** shifted by the **offset** number.

```
Floor(x[, step[, offset]])
```

round

Round() returns the result of rounding a number up or down to the nearest multiple of **step** shifted by the **offset** number.

```
Round( x [ , step [ , offset ]])
```

BitCount

BitCount() returns how many bits in the binary equivalent of a decimal number are set to 1. That is, the function returns the number of set bits in **integer_number**, where **integer_number** is interpreted as a signed 32-bit integer.

Syntax:

```
BitCount(integer_number)
```

Return data type: integer

Examples and results:

Examples and results

Examples	Results
BitCount (3)	3 is binary 11, therefore this returns 2
BitCount (-1)	-1 is 64 ones in binary, therefore this returns 64

Ceil

Ceil() rounds up a number to the nearest multiple of the **step** shifted by the **offset** number.

Compare with the **floor** function, which rounds input numbers down.

Syntax:

```
Ceil(x[, step[, offset]])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
x	Input number.
step	Interval increment. The default value is 1.
offset	Defines the base of the step interval. The default value is 0.

Examples and results:

Examples and results

Examples	Results
Ceil(2.4)	Returns 3 In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 < x <=1, 1 < x <= 2, 2< x <=3 , 3 < x <=4...
Ceil(4.2)	Returns 5
Ceil(3.88 ,0.1)	Returns 3.9 In this example, the size of the interval is 0.1 and the base of the interval is 0. The intervals are ... 3.7 < x <= 3.8, 3.8 < x <= 3.9 , 3.9 < x <= 4.0...
ceil(3.88 ,5)	Returns 5
ceil(1.1 ,1)	Returns 2

Examples	Results
<code>ceil(1.1 ,1,0.5)</code>	Returns 1.5 In this example, the size of the step is 1 and the offset is 0.5. It means that the base of the step interval is 0.5 and not 0. The intervals are ... 0.5 < x <=1.5 , 1.5 < x <= 2.5, 2.5 < x <=3.5, 3.5 < x <=4.5...
<code>ceil(1.1 ,1,-0.01)</code>	Returns 1.99 The intervals are ...-0.01 < x <= 0.99, 0.99 < x <= 1.99 , 1.99 < x <=2.99...

Combin

Combin() returns the number of combinations of **q** elements that can be picked from a set of **p** items. As represented by the formula: $\text{Combin}(p,q) = p! / q!(p-q)!$ The order in which the items are selected is insignificant.

Syntax:

```
Combin(p, q)
```

Return data type: integer

Limitations:

Non-integer items will be truncated.

Examples and results:

Examples and results

Examples	Results
How many combinations of 7 numbers can be picked from a total of 35 lottery numbers? <code>combin(35,7)</code>	Returns 6,724,520

Div

Div() returns the integer part of the arithmetic division of the first argument by the second argument. Both parameters are interpreted as real numbers, that is, they do not have to be integers.

Syntax:

```
Div(integer_number1, integer_number2)
```

Return data type: integer

Examples and results:

Examples and results	
Examples	Results
Div(7,2)	Returns 3
Div(7.1,2.3)	Returns 3
Div(9,3)	Returns 3
Div(-4,3)	Returns -1
Div(4,-3)	Returns -1
Div(-4,-3)	Returns 1

Even

Even() returns True (-1), if **integer_number** is an even integer or zero. It returns False (0), if **integer_number** is an odd integer, and NULL if **integer_number** is not an integer.

Syntax:

```
Even(integer_number)
```

Return data type: Boolean

Examples and results:

Examples and results	
Examples	Results
Even(3)	Returns 0, False
Even(2 * 10)	Returns -1, True
Even(3.14)	Returns NULL

Fabs

Fabs() returns the absolute value of **x**. The result is a positive number.

Syntax:

```
fabs(x)
```

Return data type: numeric

Examples and results:

Examples and results

Examples	Results
<code>fabs(2.4)</code>	Returns 2.4
<code>fabs(-3.8)</code>	Returns 3.8

Fact

Fact() returns the factorial of a positive integer **x**.

Syntax:

`Fact(x)`

Return data type: integer

Limitations:

If the number **x** is not an integer, it will be truncated. Non-positive numbers will return NULL.

Examples and results:

Examples and results

Examples	Results
<code>Fact(1)</code>	Returns 1
<code>Fact(5)</code>	Returns 120 ($1 * 2 * 3 * 4 * 5 = 120$)
<code>Fact(-5)</code>	Returns NULL

Floor

Floor() rounds down a number to the nearest multiple of the **step** shifted by the **offset** number.

Compare with the **ceil** function, which rounds input numbers up.

Syntax:

`Floor(x[, step[, offset]])`

Return data type: numeric

Arguments:

Arguments	
Argument	Description
x	Input number.
step	Interval increment. The default value is 1.
offset	Defines the base of the step interval. The default value is 0.

Examples and results:

Examples and results	
Examples	Results
Floor(2.4)	Returns 2 In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 <= x <1, 1 <= x < 2, 2<= x <3 , 3<= x <4....
Floor(4.2)	Returns 4
Floor(3.88 ,0.1)	Returns 3.8 In this example, the size of the interval is 0.1 and the base of the interval is 0. The intervals are ... 3.7 <= x < 3.8, 3.8 <= x < 3.9 , 3.9 <= x < 4.0...
Floor(3.88 ,5)	Returns 0
Floor(1.1 ,1)	Returns 1
Floor(1.1 ,1,0.5)	Returns 0.5 In this example, the size of the step is 1 and the offset is 0.5. It means that the base of the step interval is 0.5 and not 0. The intervals are ... 0.5 <= x <1.5 , 1.5 <= x < 2.5, 2.5<= x <3.5,...

Fmod

fmod() is a generalized modulo function that returns the remainder part of the integer division of the first argument (the dividend) by the second argument (the divisor). The result is a real number. Both arguments are interpreted as real numbers, that is, they do not have to be integers.

Syntax:

```
fmod(a, b)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
a	Dividend
b	Divisor

Examples and results:

Examples and results	
Examples	Results
fmod(7,2)	Returns 1
fmod(7.5,2)	Returns 1.5
fmod(9,3)	Returns 0
fmod(-4,3)	Returns -1
fmod(4,-3)	Returns 1
fmod(-4,-3)	Returns -1

Frac

Frac() returns the fraction part of x.

The fraction is defined in such a way that `Frac(x) + Floor(x) = x`. In simple terms, this means that the fractional part of a positive number is the difference between the number (x) and the integer that precedes the fractional part.

For example: The fractional part of 11.43 = 11.43 - 11 = 0.43

For a negative number, say -1.4, `Floor(-1.4) = -2`, which produces the following result:

The fractional part of -1.4 = -1.4 - (-2) = -1.4 + 2 = 0.6

Syntax:

```
Frac(x)
```

Return data type: numeric

Arguments:

Arguments	
Argument	Description
x	Number to return fraction for.

Examples and results:

Examples and results	
Examples	Results
Frac(11.43)	Returns 0.43
Frac(-1.4)	Returns 0.6
Extract the time component from the numeric representation of a timestamp, thus omitting the date.	Returns 3:56:00 PM
Time(Frac(44518.663888889))	

Mod

Mod() is a mathematical modulo function that returns the non-negative remainder of an integer division. The first argument is the dividend, the second argument is the divisor, Both arguments must be integer values.

Syntax:

```
Mod(integer_number1, integer_number2)
```

Return data type: integer

Limitations:

integer_number2 must be greater than 0.

Examples and results:

Examples and results	
Examples	Results
Mod(7,2)	Returns 1
Mod(7.5,2)	Returns NULL
Mod(9,3)	Returns 0

Examples	Results
Mod(-4,3)	Returns 2
Mod(4,-3)	Returns NULL
Mod(-4,-3)	Returns NULL

Odd

Odd() returns True (-1), if **integer_number** is an odd integer or zero. It returns False (0), if **integer_number** is an even integer, and NULL if **integer_number** is not an integer.

Syntax:

```
Odd(integer_number)
```

Return data type: Boolean

Examples and results:

Examples and results

Examples	Results
odd(3)	Returns -1, True
odd(2 * 10)	Returns 0, False
odd(3.14)	Returns NULL

Permut

Permut() returns the number of permutations of **q** elements that can be selected from a set of **p** items. As represented by the formula: $\text{Permut}(p,q) = (p)! / (p - q)!$ The order in which the items are selected is significant.

Syntax:

```
Permut(p, q)
```

Return data type: integer

Limitations:

Non-integer arguments will be truncated.

Examples and results:

Examples and results	
Examples	Results
In how many ways could the gold, silver and bronze medals be distributed after a 100 m final with 8 participants? Permut(8,3)	Returns 336

Round

Round() returns the result of rounding a number up or down to the nearest multiple of **step** shifted by the **offset** number.

If the number to round is exactly in the middle of an interval, it is rounded upwards.

Syntax:

```
Round(x[, step[, offset]])
```

Return data type: numeric



If you are rounding a floating point number you may observe erroneous results. These rounding errors occur because floating point numbers are represented by a finite number of binary digits. Therefore, results are calculated using a number that is already rounded. If these rounding errors will affect your work, multiply the numbers to convert them to integers before rounding.

Arguments:

Arguments

Argument	Description
x	Input number.
step	Interval increment. The default value is 1.
offset	Defines the base of the step interval. The default value is 0.

Examples and results:

Examples and results	
Examples	Results
Round(3.8)	Returns 4 In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 <= x <1, 1 <= x < 2, 2<= x <3, 3<= x <4...
Round(3.8 ,4)	Returns 4
Round(2.5)	Returns 3. In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 <= x <1, 1 <= x <2, 2<= x <3...
Round(2 ,4)	Returns 4. Rounded up because 2 is exactly half of the step interval of 4. In this example, the size of the step is 4 and the base of the step interval is 0. The intervals are ... 0 <= x <4 , 4 <= x <8, 8<= x <12...
Round(2 ,6)	Returns 0. Rounded down because 2 is less than half of the step interval of 6. In this example, the size of the step is 6 and the base of the step interval is 0. The intervals are ... 0 <= x <6 , 6 <= x <12, 12<= x <18...
Round(3.88 ,0.1)	Returns 3.9 In this example, the size of the step is 0.1 and the base of the step interval is 0. The intervals are ... 3.7 <= x <3.8, 3.8 <= x <3.9 , 3.9 <= x < 4.0...
Round(3.8875 ,1/1000)	Returns 3.889 In this example, the size of the step is 0.001, which rounds the number up and limits it to three decimal places.
Round(3.88 ,5)	Returns 5
Round(1.1 ,1,0.5)	Returns 1.5 In this example, the size of the step is 1 and the base of the step interval is 0.5. The intervals are ... 0.5 <= x <1.5 , 1.5 <= x <2.5, 2.5<= x <3.5...

Sign

Sign() returns 1, 0 or -1 depending on whether **x** is a positive number, 0, or a negative number.

Syntax:

Sign(x)

Return data type: numeric

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Examples and results	
Examples	Results
Sign(66)	Returns 1
Sign(0)	Returns 0
Sign(- 234)	Returns -1

5.14 Geospatial functions

These functions are used to handle geospatial data in map visualizations. Qlik Sense follows GeoJSON specifications for geospatial data and supports the following:

- Point
- Linestring
- Polygon
- Multipolygon

For more information on GeoJSON specifications, see:

 [GeoJSON.org](#)

Geospatial functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

There are two categories of geospatial functions: aggregation and non-aggregation.

Aggregation functions take a geometry set (points or areas) as input, and return a single geometry. For example, multiple areas can be merged together, and a single boundary for the aggregation can be drawn on the map.

Non-aggregation functions take a single geometry and return one geometry. For example, for the function `GeoGetPolygonCenter()`, if the boundary geometry of one area is set as input, the point geometry (longitude and latitude) for the center of that area is returned.

The following are aggregation functions:

GeoAggrGeometry

GeoAggrGeometry() is used to aggregate a number of areas into a larger area, for example aggregating a number of sub-regions to a region.

```
GeoAggrGeometry (field_name)
```

GeoBoundingBox

GeoBoundingBox() is used to aggregate a geometry into an area and calculate the smallest bounding box that contains all coordinates.

```
GeoBoundingBox (field_name)
```

GeoCountVertex

GeoCountVertex() is used to find the number of vertices a polygon geometry contains.

```
GeoCountVertex(field_name)
```

GeoInvProjectGeometry

GeoInvProjectGeometry() is used to aggregate a geometry into an area and apply the inverse of a projection.

```
GeoInvProjectGeometry(type, field_name)
```

GeoProjectGeometry

GeoProjectGeometry() is used to aggregate a geometry into an area and apply a projection.

```
GeoProjectGeometry(type, field_name)
```

GeoReduceGeometry

GeoReduceGeometry() is used to reduce the number of vertices of a geometry, and to aggregate a number of areas into one area, but still displaying the boundary lines from the individual areas.

```
GeoReduceGeometry (geometry)
```

The following are non-aggregation functions:

GeoGetBoundingBox

GeoGetBoundingBox() is used in scripts and chart expressions to calculate the smallest geospatial bounding box that contains all coordinates of a geometry.

```
GeoGetBoundingBox (geometry)
```

GeoGetPolygonCenter

GeoGetPolygonCenter() is used in scripts and chart expressions to calculate and return the center point of a geometry.

```
GeoGetPolygonCenter (geometry)
```

GeoMakePoint

GeoMakePoint() is used in scripts and chart expressions to create and tag a point with latitude and longitude.

```
GeoMakePoint (lat_field_name, lon_field_name)
```

GeoProject

GeoProject() is used in scripts and chart expressions to apply a projection to a geometry.

```
GeoProject (type, field_name)
```

GeoAggrGeometry

GeoAggrGeometry() is used to aggregate a number of areas into a larger area, for example aggregating a number of sub-regions to a region.

Syntax:

```
GeoAggrGeometry (field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

Typically, **GeoAggrGeometry()** can be used to combine geospatial boundary data. For example, you might have postcode areas for suburbs in a city and sales revenues for each area. If a sales person's territory covers several postcode areas, it might be useful to present total sales by sales territory, rather than individual areas, and show the results on a color-filled map.

GeoAggrGeometry() can calculate the aggregation of the individual suburb geometries and generate the merged territory geometry in the data model. If then, the sales territory boundaries are adjusted, when the data is reloaded the new merged boundaries and revenues are reflected in the map.

As **GeoAggrGeometry()** is an aggregating function, if you use it in the script a **LOAD** statement with a **Group by** clause is required.



*The boundary lines of maps created using **GeoAggrGeometry()** are those of the merged areas. If you want to display the individual boundary lines of the pre-aggregated areas, use **GeoReduceGeometry()**.*

Examples:

This example loads a KML file with area data, and then loads a table with the aggregated area data.

```
[MapSource]:  
LOAD [world.Name],  
      [world.Point],  
      [world.Area]  
FROM [lib://Downloads/world.kml]  
(kml, Table is [world.shp/Features]);
```

```
Map:  
LOAD world.Name,  
    GeoAggrGeometry(world.Area) as [AggrArea]  
resident MapSource Group By world.Name;  
  
Drop Table MapSource;
```

GeoBoundingBox

GeoBoundingBox() is used to aggregate a geometry into an area and calculate the smallest bounding box that contains all coordinates.

A GeoBoundingBox is represented as a list of four values: left, right, top, bottom.

Syntax:

```
GeoBoundingBox(field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

GeoBoundingBox() aggregates a set of geometries and returns four coordinates for the smallest rectangle that contains all the coordinates of that aggregated geometry.

To visualize the result on a map, transfer the resulting string of four coordinates into a polygon format, tag the transferred field with a geopolygon format, and drag and drop that field into the map object. The rectangular boxes will then be displayed in the map visualization.

GeoCountVertex

GeoCountVertex() is used to find the number of vertices a polygon geometry contains.

Syntax:

```
GeoCountVertex(field_name)
```

Return data type: integer

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

GeoGetBoundingBox

GeoGetBoundingBox() is used in scripts and chart expressions to calculate the smallest geospatial bounding box that contains all coordinates of a geometry.

A geospatial bounding box, created by the function GeoBoundingBox() is represented as a list of four values: left, right, top, bottom.

Syntax:

```
GeoGetBoundingBox(field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.



*Do not use the **Group by** clause in the data load editor with this and other non-aggregating geospatial functions, because this will cause an error on load.*

GeoGetPolygonCenter

GeoGetPolygonCenter() is used in scripts and chart expressions to calculate and return the center point of a geometry.

In some cases, the requirement is to plot a dot instead of color fill on a map. If the existing geospatial data is only available in the form of area geometry (for example, a boundary), use **GeoGetPolygonCenter()** to retrieve a pair of longitude and latitude for the center of area.

Syntax:

```
GeoGetPolygonCenter(field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.



*Do not use the **Group by** clause in the data load editor with this and other non-aggregating geospatial functions, because this will cause an error on load.*

GeoInvProjectGeometry

GeoInvProjectGeometry() is used to aggregate a geometry into an area and apply the inverse of a projection.

Syntax:

```
GeoInvProjectGeometry(type, field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
type	Projection type used in transforming the geometry of the map. This can take one of two values: 'unit', (default), which results in a 1:1 projection, or 'mercator', which uses the standard Mercator projection.
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

Example:

Scripting example

Example	Result
In a Load statement: GeoInvProjectGeometry ('mercator',AreaPolygon) as InvProjectGeometry	The geometry loaded as AreaPolygon is transformed using the inverse transformation of the Mercator projection and stored as InvProjectGeometry for use in visualizations.

GeoMakePoint

GeoMakePoint() is used in scripts and chart expressions to create and tag a point with latitude and longitude. GeoMakePoint returns points in the order of longitude and latitude.

Syntax:

```
GeoMakePoint(lat_field_name, lon_field_name)
```

Return data type: string, formatted [longitude, latitude]

Arguments:

Arguments

Argument	Description
lat_field_name	A field or expression referring to a field representing the latitude of the point.
lon_field_name	A field or expression referring to a field representing the longitude of the point.



*Do not use the **Group by** clause in the data load editor with this and other non-aggregating geospatial functions, because this will cause an error on load.*

GeoProject

GeoProject() is used in scripts and chart expressions to apply a projection to a geometry.

Syntax:

```
GeoProject(type, field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
type	Projection type used in transforming the geometry of the map. This can take one of two values: 'unit', (default), which results in a 1:1 projection, or 'mercator', which uses the web Mercator projection.
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.



*Do not use the **Group by** clause in the data load editor with this and other non-aggregating geospatial functions, because this will cause an error on load.*

Example:

Script examples

Example	Result
In a Load statement: <code>GEOProject('mercator',Area) as GetProject</code>	The Mercator projection is applied to the geometry loaded as Area , and the result is stored as GetProject .

GeoProjectGeometry

GeoProjectGeometry() is used to aggregate a geometry into an area and apply a projection.

Syntax:

```
GeoProjectGeometry(type, field_name)
```

Return data type: string

Arguments:

Arguments

Argument	Description
type	Projection type used in transforming the geometry of the map. This can take one of two values: 'unit', (default), which results in a 1:1 projection, or 'mercator', which uses the web Mercator projection.
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

Example:

Example	Result
In a Load statement: GeoProjectGeometry ('mercator',AreaPolygon) as ProjectGeometry	The geometry loaded as AreaPolygon is transformed using the Mercator projection and stored as ProjectGeometry for use in visualizations.

GeoReduceGeometry

GeoReduceGeometry() is used to reduce the number of vertices of a geometry, and to aggregate a number of areas into one area, but still displaying the boundary lines from the individual areas.

Syntax:

```
GeoReduceGeometry(field_name[, value])
```

Return data type: string

Arguments:

Arguments

Argument	Description
field_name	A field or expression referring to a field containing the geometry to be represented. This could be either a point (or set of points) giving longitude and latitude, or an area.

Argument	Description
value	<p>The amount of reduction to apply to the geometry. The range is from 0 to 1, with 0 representing no reduction and 1 representing maximal reduction of vertices.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <i>Using a value of 0.9 or higher with a complex data set can reduce the number of vertices to a level where the visual representation is inaccurate.</i> </div>

GeoReduceGeometry() also performs a similar function to, **GeoAggrGeometry()** in that it aggregates a number of areas into one area. The difference being that individual boundary lines from the pre-aggregation data are displayed on the map if you use **GeoReduceGeometry()**.

As **GeoReduceGeometry()** is an aggregating function, if you use it in the script a **LOAD** statement with a **Group by** clause is required.

Examples:

This example loads a KML file with area data, and then loads a table with the reduced and aggregated area data.

```
[MapSource]:
LOAD [world.Name],
    [world.Point],
    [world.Area]
FROM [lib://Downloads/world.kml]
(kml, Table is [world.shp/Features]);
```

Map:

```
LOAD world.Name,
    GEOReduceGeometry(world.Area,0.5) as [ReducedArea]
resident MapSource Group By world.Name;
```

```
Drop Table MapSource;
```

5.15 Interpretation functions

The interpretation functions evaluate the contents of input text fields or expressions, and impose a specified data format on the resulting numeric value. With these functions, you can specify the format of the number, in accordance with its data type, including attributes such as: decimal separator, thousands separator, and date format.

The interpretation functions all return a dual value with both the string and the number value, but can be thought of as performing a string-to-number conversion. The functions take the text value of the input expression and generate a number representing the string.

In contrast, the formatting functions do the opposite: they take numeric expressions and evaluate them as strings, specifying the display format of the resulting text.

If no interpretation functions are used, Qlik Sense interprets the data as a mix of numbers, dates, times, time stamps and strings, using the default settings for number format, date format, and time format, defined by script variables and by the operating system.

All interpretation functions can be used in both data load scripts and chart expressions.



All number representations are given with a decimal point as the decimal separator.

Interpretation functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Date#

Date#() evaluates an expression as a date in the format specified in the second argument, if supplied. If the format code is omitted, the default date format set in the operating system is used.

```
Date# (page 1226) (text[, format])
```

Interval#

Interval#() evaluates a text expression as a time interval in the format set in the operating system, by default, or in the format specified in the second argument, if supplied.

```
Interval# (page 1227) (text[, format])
```

Money#

Money#() converts a text string to a money value, in the format set in the load script or the operating system, unless a format string is supplied. Custom decimal and thousand separator symbols are optional parameters.

```
Money# (page 1227) (text[, format[, dec_sep[, thou_sep ] ] ])
```

Num#

Num#() interprets a text string as a numerical value, that is it converts the input string to a number using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.

```
Num# (page 1229) (text[ , format[, dec_sep[ , thou_sep]]])
```

Text

Text() forces the expression to be treated as text, even if a numeric interpretation is possible.

```
Text(expr)
```

Time#

Time#() evaluates an expression as a time value, in the time format set in the data load script or the operating system, unless a format string is supplied..

```
Time# (page 1230) (text[, format])
```

Timestamp#

Timestamp#() evaluates an expression as a date and time value, in the timestamp format set in the data load script or the operating system, unless a format string is supplied.

```
Timestamp# (page 1231) (text[, format])
```

See also:

□ [Formatting functions \(page 1190\)](#)

Date#

Date# evaluates an expression as a date in the format specified in the second argument, if supplied.

Syntax:

```
Date#(text[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
text	The text string to be evaluated.
format	String describing the format of the text string to be evaluated. If omitted, the date format set in the system variables in the data load script, or the operating system is used.

Examples and results:

The following example uses the date format **M/D/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of the data load script.

Add this example script to your app and run it.

```
Load *,  
Num(Date#(StringDate)) as Date;  
LOAD * INLINE [  
StringDate  
8/7/97  
8/6/1997  
]
```

If you create a table with **StringDate** and **Date** as dimensions, the results are as follows:

Results

StringDate	Date
8/7/97	35649
8/6/1997	35648

Interval#

Interval#() evaluates a text expression as a time interval in the format set in the operating system, by default, or in the format specified in the second argument, if supplied.

Syntax:

```
Interval#(text[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
text	The text string to be evaluated.
format	String describing the expected input format to use when converting the string to a numeric interval. If omitted, the short date format, time format, and decimal separator set in the operating system are used.

The **interval#** function converts a text time interval to a numeric equivalent.

Examples and results:

The examples below assume the following operating system settings:

- Short date format: YY-MM-DD
- Time format: M/D/YY
- Number decimal separator: .

Results

Example	Result
Interval#(A, 'D hh:mm') where A='1 09:00'	1.375

Money#

Money#() converts a text string to a money value, in the format set in the load script or the operating system, unless a format string is supplied. Custom decimal and thousand separator symbols are optional parameters.

Syntax:

```
Money#(text[, format[, dec_sep [, thou_sep ] ] ])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
text	The text string to be evaluated.
format	String describing the expected input format to use when converting the string to a numeric interval. If omitted, the money format set in the operating system is used.
dec_sep	String specifying the decimal number separator. If omitted, the MoneyDecimalSep value set in the data load script is used.
thou_sep	String specifying the thousands number separator. If omitted, the MoneyThousandSep value set in the data load script is used.

The **money#** function generally behaves just like the **num#** function but takes its default values for decimal and thousand separator from the script variables for money format or the system settings for currency.

Examples and results:

The examples below assume the two following operating system settings:

- Money format default setting 1: kr # ##0,00
- Money format default setting 2: \$ #,##0.00

Money#(A , '# ##0,00 kr')

where A=35 648,37 kr

Results

Results	Setting 1	Setting 2
String	35 648.37 kr	35 648.37 kr
Number	35648.37	3564837

Money#(A, '\$#, ','.', ',')

where A= \$35,648.37

Results

Results	Setting 1	Setting 2
String	\$35,648.37	\$35,648.37
Number	35648.37	35648.37

Num#

Num#() interprets a text string as a numerical value, that is it converts the input string to a number using the format specified in the second parameter. If the second parameter is omitted, it uses the decimal and thousand separators set in the data load script. Custom decimal and thousand separator symbols are optional parameters.

Syntax:

```
Num#(text[, format[, dec_sep [, thou_sep ] ]])
```

Return data type: dual

The **Num#()** function returns a dual value with both the string and the numeric value. The function takes the textual representation of the input expression and generates a number. It does not change the format of the number: the output is formatted in the same way as the input.

Arguments:

Arguments

Argument	Description
text	The text string to be evaluated.
format	String specifying the number format used in the first parameter. If omitted, the decimal and thousand separators that are set in the data load script are used.
dec_sep	String specifying the decimal number separator. If omitted, the value of the variable DecimalSep that is set in the data load script is used.
thou_sep	String specifying the thousands number separator. If omitted, the value of the variable ThousandSep that is set in the data load script is used.

Examples and results:

The following table shows the result of *Num#(A, '#', '.', ',')* for different values of A.

A	String representation	Results	
		Numeric value (here displayed with decimal point)	
35,648.31	35,648.31	35648.31	
35 648.312	35 648.312	35648.312	
35.648,3123	35.648,3123	-	
35 648,31234	35 648,31234	-	

Text

Text() forces the expression to be treated as text, even if a numeric interpretation is possible.

Syntax:

```
Text (expr)
```

Return data type: dual

Example:

```
Text( A )  
where A=1234
```

Results

String	Number
1234	-

Example:

```
Text( pi() )
```

Results

String	Number
3.1415926535898	-

Time#

Time#() evaluates an expression as a time value, in the time format set in the data load script or the operating system, unless a format string is supplied..

Syntax:

```
time#(text[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
text	The text string to be evaluated.
format	String describing the format of the text string to be evaluated. If omitted, the short date format, time format, and decimal separator set in the operating system is used.

Example:

- Time format default setting 1: hh:mm:ss
- Time format default setting 2: hh.mm.ss

```
time#( A )
where A=09:00:00
```

Results

Results	Setting 1	Setting 2
String:	09:00:00	09:00:00
Number:	0.375	-

Example:

- Time format default setting 1: hh:mm:ss
- Time format default setting 2: hh.mm.ss

```
time#( A, 'hh.mm' )
where A=09.00
```

Results

Results	Setting 1	Setting 2
String:	09.00	09.00
Number:	0.375	0.375

Timestamp#

Timestamp#() evaluates an expression as a date and time value, in the timestamp format set in the data load script or the operating system, unless a format string is supplied.

Syntax:

```
timestamp#(text[, format])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
text	The text string to be evaluated.
format	String describing the format of the text string to be evaluated. If omitted, the short date format, time format, and decimal separator set in the operating system is used. ISO 8601 is supported for timestamps.

Example:

The following example uses the date format **M/D/YYYY**. The date format is specified in the **SET DateFormat** statement at the top of the data load script.

Add this example script to your app and run it.

```
Load *,  
Timestamp(Timestamp#(string)) as TS;  
LOAD * INLINE [  
String  
2015-09-15T12:13:14  
1952-10-16T13:14:00+0200  
1109-03-01T14:15  
];
```

If you create a table with **String** and **TS** as dimensions, the results are as follows:

Results	
String	TS
2015-09-15T12:13:14	9/15/2015 12:13:14 PM
1952-10-16T13:14:00+0200	10/16/1952 11:14:00 AM
1109-03-01T14:15	3/1/1109 2:15:00 PM

5.16 Inter-record functions

Inter-record functions are used:

- In the data load script, when a value from previously loaded records of data is needed for the evaluation of the current record.
- In a chart expression, when another value from the data set of a visualization is needed.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when an inter-record chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use an inter-record chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to the inter-record function. This limitation does not apply to the equivalent script function, if there is one.



Self-referencing expression definitions can only reliably be made in tables with fewer than 100 rows, but this may vary depending on the hardware that the Qlik engine is running on.

Row functions

These functions can only be used in chart expressions.

Above

Above() evaluates an expression at a row above the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly above. For charts other than tables, **Above()** evaluates for the row above the current row in the chart's straight table equivalent.

```
Above - chart function([TOTAL [<fld{,fld}>]] expr [ , offset [,count]])
```

Below

Below() evaluates an expression at a row below the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly below. For charts other than tables, **Below()** evaluates for the row below the current column in the chart's straight table equivalent.

```
Below - chart function([TOTAL[<fld{,fld}>]] expression [ , offset [,count ]])
```

Bottom

Bottom() evaluates an expression at the last (bottom) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the bottom row. For charts other than tables, the evaluation is made on the last row of the current column in the chart's straight table equivalent.

```
Bottom - chart function([TOTAL[<fld{,fld}>]] expr [ , offset [,count ]])
```

Top

Top() evaluates an expression at the first (top) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the top row. For charts other than tables, the **Top()** evaluation is made on the first row of the current column in the chart's straight table equivalent.

```
Top - chart function([TOTAL [<fld{,fld}>]] expr [ , offset [,count ]])
```

NoOfRows

NoOfRows() returns the number of rows in the current column segment in a table. For bitmap charts, **NoOfRows()** returns the number of rows in the chart's straight table equivalent.

```
NoOfRows - chart function([TOTAL])
```

Column functions

These functions can only be used in chart expressions.

Column

Column() returns the value found in the column corresponding to **ColumnNo** in a straight table, disregarding dimensions. For example **Column(2)** returns the value of the second measure column.

```
Column - chart function(ColumnNo)
```

Dimensionality

Dimensionality() returns the number of dimensions for the current row. In the case of pivot tables, the function returns the total number of dimension columns that have non-aggregation content, that is, do not contain partial sums or collapsed aggregates.

```
Dimensionality - chart function ( )
```

Secondarydimensionality

SecondaryDimensionality() returns the number of dimension pivot table rows that have non-aggregation content, that is, do not contain partial sums or collapsed aggregates. This function is the equivalent of the **dimensionality()** function for horizontal pivot table dimensions.

```
SecondaryDimensionality - chart function ( )
```

Field functions

FieldIndex

FieldIndex() returns the position of the field value **value** in the field **field_name** (by load order).

```
FieldIndex(field_name , value)
```

FieldValue

FieldValue() returns the value found in position **elem_no** of the field **field_name** (by load order).

```
FieldValue(field_name , elem_no)
```

FieldValueCount

FieldValueCount() is an **integer** function that returns the number of distinct values in a field.

```
FieldValueCount(field_name)
```

Pivot table functions

These functions can only be used in chart expressions.

After

After() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the column after the current column within a row segment in the pivot table.

```
After - chart function([TOTAL] expression [ , offset [,n]])
```

Before

Before() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the column before the current column within a row segment in the pivot table.

```
Before - chart function([TOTAL] expression [ , offset [,n]])
```

First

First() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the first column of the current row segment in the pivot table. This function returns NULL in all chart types except pivot tables.

```
First - chart function([TOTAL] expression [ , offset [,n]])
```

Last

Last() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the last column of the current row segment in the pivot table. This function returns NULL in all chart types except pivot tables.

```
Last - chart function([TOTAL] expression [ , offset [,n]])
```

ColumnNo

ColumnNo() returns the number of the current column within the current row segment in a pivot table. The first column is number 1.

```
ColumnNo - chart function([TOTAL])
```

NoOfColumns

NoOfColumns() returns the number of columns in the current row segment in a pivot table.

```
NoOfColumns - chart function([TOTAL])
```

Inter-record functions in the data load script

Exists

Exists() determines whether a specific field value has already been loaded into the field in the data load script. The function returns TRUE or FALSE, so can be used in the **where** clause of a **LOAD** statement or an **IF** statement.

```
Exists (field_name [, expr])
```

LookUp

Lookup() looks into a table that is already loaded and returns the value of **field_name** corresponding to the first occurrence of the value **match_field_value** in the field **match_field_name**. The table can be the current table or another table previously loaded.

```
LookUp (field_name, match_field_name, match_field_value [, table_name])
```

Peek

Peek() returns the value of a field in a table for a row that has already been loaded. The row number can be specified, as can the table. If no row number is specified, the last previously loaded record will be used.

```
Peek (field_name[, row_no[, table_name ] ])
```

Previous

Previous() finds the value of the **expr** expression using data from the previous input record that has not been discarded because of a **where** clause. In the first record of an internal table, the function will return NULL.

```
Previous (page 1271) (expr)
```

See also:

 [Range functions \(page 1290\)](#)

Above - chart function

Above() evaluates an expression at a row above the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly above. For charts other than tables, **Above()** evaluates for the row above the current row in the chart's straight table equivalent.

Syntax:

```
Above ([TOTAL] expr [ , offset [,count]])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offsetn, greater than 0, moves the evaluation of the expression n rows further up from the current row. Specifying an offset of 0 will evaluate the expression on the current row. Specifying a negative offset number makes the Above function work like the Below function with the corresponding positive offset number.
count	By specifying a third argument count greater than 1, the function will return a range of count values, one for each of count table rows counting upwards from the original cell. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 1290)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

On the first row of a column segment, a NULL value is returned, as there is no row above it.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the TOTAL qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

- Recursive calls will return NULL.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Examples and results:

Example 1:

Table visualization for Example 1

Customer	Sum([Sales])	Above(Sum(Sales))	Sum(Sales)+Above(Sum(Sales))	Above offset 3	Higher?
	2566	-	-	-	-
Astrida	587	-	-	-	-
Betacab	539	587	1126	-	-
Canutility	683	539	1222	-	Higher
Divadip	757	683	1440	1344	Higher

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: **Sum(Sales)** and **Above(Sum(Sales))**.

The column **Above(Sum(Sales))** returns NULL for the **Customer** row containing **Astrida**, because there is no row above it. The result for the row **Betacab** shows the value of **Sum(Sales)** for **Astrida**, the result for **Canutility** shows the value for **Sum(Sales)** for **Betacab**, and so on.

For the column labeled **Sum(Sales)+Above(Sum(Sales))**, the row for **Betacab** shows the result of the addition of the **Sum(Sales)** values for the rows **Betacab + Astrida** (539+587). The result for the row **Canutility** shows the result of the addition of **Sum(Sales)** values for **Canutility + Betacab** (683+539).

The measure labeled **Above offset 3** created using the expression **sum(Sales)+Above(Sum(Sales), 3)** has the argument **offset**, set to 3, and has the effect of taking the value in the row three rows above the current row. It adds the **Sum(Sales)** value for the current **Customer** to the value for the **Customer** three rows above. The values returned for the first three **Customer** rows are null.

The table also shows more complex measures: one created from **sum(Sales)+Above(Sum(Sales))** and one labeled **Higher?**, which is created from **IF(Sum(Sales)>Above(Sum(Sales)), 'Higher')**.



This function can also be used in charts other than tables, for example bar charts.



For other chart types, convert the chart to the straight table equivalent so you can easily interpret which row the function relates to.

Example 2:

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

In the following screenshot of table visualization for Example 2, the last-sorted dimension is **Month**, so the **Above** function evaluates based on months. There is a series of results for each **Product** value for each month (**Jan** to **Aug**) - a column segment. This is followed by a series for the next column segment: for each **Month** for the next **Product**. There will be a column segment for each **Customer** value for each **Product**.

Table visualization for Example 2

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	60
Astrida	AA	Apr	13	70
Astrida	AA	May	78	13
Astrida	AA	Jun	20	78
Astrida	AA	Jul	45	20
Astrida	AA	Aug	65	45

Example 3:

In the screenshot of table visualization for Example 3, the last sorted dimension is **Product**. This is done by moving the dimension **Product** to position 3 in the **Sorting** tab in the properties panel. The **Above** function is evaluated for each **Product**, and because there are only two products, **AA** and **BB**, there is only one non-null result in each series. In row **BB** for the month **Jan**, the value for **Above(Sum(Sales))**, is 46. For row **AA**, the value is null. The value in each row **AA** for any month will always be null, as there is no value of **Product** above **AA**. The second series is evaluated on **AA** and **BB** for the month **Feb**, for the **Customer** value, **Astrida**. When all the months have been evaluated for **Astrida**, the sequence is repeated for the second **Customer** **Betacab**, and so on.

Table visualization for Example 3

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	-
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	-
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	-
Astrida	BB	Apr	13	13

Example 4

Example 4:	Result								
The Above function can be used as input to the range functions. For example: RangeAvg(Above(Sum(Sales),1,3)).	<p>In the arguments for the Above() function, offset is set to 1 and count is set to 3. The function finds the results of the expression Sum(Sales) on the three rows immediately above the current row in the column segment (where there is a row). These three values are used as input to the RangeAvg() function, which finds the average of the values in the supplied range of numbers.</p> <p>A table with Customer as dimension gives the following results for the RangeAvg() expression.</p> <table> <tbody> <tr> <td>Astrida</td> <td>-</td> </tr> <tr> <td>Betacab</td> <td>587</td> </tr> <tr> <td>Canutility</td> <td>563</td> </tr> <tr> <td>Divadip:</td> <td>603</td> </tr> </tbody> </table>	Astrida	-	Betacab	587	Canutility	563	Divadip:	603
Astrida	-								
Betacab	587								
Canutility	563								
Divadip:	603								

Data used in examples:

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
```

```

Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');

```

See also:

- [Below - chart function \(page 1240\)](#)
- [Bottom - chart function \(page 1243\)](#)
- [Top - chart function \(page 1272\)](#)
- [RangeAvg \(page 1293\)](#)

Below - chart function

Below() evaluates an expression at a row below the current row within a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the row directly below. For charts other than tables, **Below()** evaluates for the row below the current column in the chart's straight table equivalent.

Syntax:

```
Below([TOTAL] expr [ , offset [,count ]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
offset	<p>Specifying an offsetn, greater than 1 moves the evaluation of the expression n rows further down from the current row.</p> <p>Specifying an offset of 0 will evaluate the expression on the current row.</p> <p>Specifying a negative offset number makes the Below function work like the Above function with the corresponding positive offset number.</p>
count	<p>By specifying a third parameter count greater than 1, the function will return a range of count values, one for each of count table rows counting downwards from the original cell.</p> <p>In this form, the function can be used as an argument to any of the special range functions.</p> <p><i>Range functions (page 1290)</i></p>

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

On the last row of a column segment, a NULL value is returned, as there is no row below it.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the TOTAL qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

- Recursive calls will return NULL.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Examples and results:

Example 1:

Table visualization for Example 1

Customer	Sum([Sales])	Below(Sum(Sales))	Sum(Sales)+Below(Sum(Sales))	Below + Offset 3	Higher
	2566	-	-	-	-
Astrida	587	539	1126	1344	Higher
Betacob	539	683	1222	-	-
Canutility	683	757	1440	-	-
Divadip	757	-	-	-	-

In the table shown in screenshot for Example 1, the table visualization is created from the dimension **Customer** and the measures: **sum(sales)** and **below(sum(sales))**.

The column **Below(Sum(Sales))** returns NULL for the **Customer** row containing **Divadip**, because there is no row below it. The result for the row **Canutility** shows the value of **Sum(Sales)** for **Divadip**, the result for **Betacob** shows the value for **Sum(Sales)** for **Canutility**, and so on.

The table also shows more complex measures, which you can see in the columns labeled: `sum(Sales)+Below(Sum(Sales))`, **Below +Offset 3**, and **Higher?**. These expressions work as described in the following paragraphs.

For the column labeled **Sum(Sales)+Below(Sum(Sales))**, the row for **Astrida** shows the result of the addition of the **Sum(Sales)** values for the rows **Betocab + Astrida** (539+587). The result for the row **Betocab** shows the result of the addition of **Sum(Sales)** values for **Canutility + Betocab** (539+683).

The measure labeled **Below +Offset 3** created using the expression `sum(Sales)+Below(Sum(Sales), 3)` has the argument **offset**, set to 3, and has the effect of taking the value in the row three rows below the current row. It adds the **Sum(Sales)** value for the current **Customer** to the value from the **Customer** three rows below. The values for the lowest three **Customer** rows are null.

The measure labeled **Higher?** is created from the expression: `IF(Sum(Sales)>Below(Sum(Sales)), 'Higher')`. This compares the values of the current row in the measure **Sum(Sales)** with the row below it. If the current row is a greater value, the text "Higher" is output.



This function can also be used in charts other than tables, for example bar charts.



For other chart types, convert the chart to the straight table equivalent so you can easily interpret which row the function relates to.

For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table. Please refer to Example: 2 in the **Above** function for further details.

Example 2

Example 2:	Result
The Below function can be used as input to the range functions. For example: <code>RangeAvg(Below(Sum(Sales), 1, 3))</code> .	In the arguments for the Below() function, offset is set to 1 and count is set to 3. The function finds the results of the expression Sum(Sales) on the three rows immediately below the current row in the column segment (where there is a row). These three values are used as input to the <code>RangeAvg()</code> function, which finds the average of the values in the supplied range of numbers. A table with Customer as dimension gives the following results for the <code>RangeAvg()</code> expression.

Example 2:	Result	
	Astrida	659.67
	Betacab	720
	Canutility	757
	Divadip:	-

Data used in examples:

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

-  [Above - chart function \(page 1235\)](#)
-  [Bottom - chart function \(page 1243\)](#)
-  [Top - chart function \(page 1272\)](#)
-  [RangeAvg \(page 1293\)](#)

Bottom - chart function

Bottom() evaluates an expression at the last (bottom) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the bottom row. For charts other than tables, the evaluation is made on the last row of the current column in the chart's straight table equivalent.

Syntax:

```
Bottom([TOTAL] expr [ , offset [,count ]])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offset greater than 1 moves the evaluation of the expression up n rows above the bottom row. Specifying a negative offset number makes the Bottom function work like the Top function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return not one but a range of count values, one for each of the last count rows of the current column segment. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 1290)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the **TOTAL** qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

- Recursive calls will return NULL.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Examples and results:

Table visualization for Example 1

Customer	Sum([Sales])	Bottom(Sum(Sales))	Sum(Sales)+Bottom(Sum(Sales))	Bottom offset 3
	2566	757	3323	3105
Astrida	587	757	1344	1126
Betacob	539	757	1296	1078
Canutility	683	757	1440	1222
Divadip	757	757	1514	1296

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: `sum(Sales)` and `Bottom(Sum(Sales))`.

The column **Bottom(Sum(Sales))** returns 757 for all rows because this is the value of the bottom row: **Divadip**.

The table also shows more complex measures: one created from `sum(Sales)+Bottom(Sum(Sales))` and one labeled **Bottom offset 3**, which is created using the expression `sum(Sales)+Bottom(Sum(Sales), 3)` and has the argument **offset** set to 3. It adds the **Sum(Sales)** value for the current row to the value from the third row from the bottom row, that is, the current row plus the value for **Betacob**.

Example: 2

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

In the first table, the expression is evaluated based on **Month**, and in the second table it is evaluated based on **Product**. The measure **End value** contains the expression `Bottom(sum(Sales))`. The bottom row for **Month** is Dec, and the value for Dec both the values of **Product** shown in the screenshot is 22. (Some rows have been edited out of the screenshot to save space.)

First table for Example 2. The value of Bottom for the End value measure based on Month (Dec).

5 Script and chart functions

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	22
Astrida	AA	Feb	60	22
Astrida	AA	Mar	70	22
Astrida	AA	Sep	78	22
Astrida	AA	Oct	12	22
Astrida	AA	Nov	78	22
Astrida	AA	Dec	22	22
Astrida	BB	Jan	46	22

Second table for Example 2. The value of Bottom for the End value measure based on Product (BB for Astrida).

Customer	Product	Month	Sum(Sales)	End value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

Please refer to Example: 2 in the **Above** function for further details.

Example 3

Example: 3	Result								
The Bottom function can be used as input to the range functions. For example: RangeAvg (Bottom (Sum(Sales),1,3)).	<p>In the arguments for the Bottom() function, offset is set to 1 and count is set to 3. The function finds the results of the expression Sum(Sales) on the three rows starting with the row above the bottom row in the column segment (because offset=1), and the two rows above that (where there is a row). These three values are used as input to the RangeAvg() function, which finds the average of the values in the supplied range of numbers.</p> <p>A table with Customer as dimension gives the following results for the RangeAvg() expression.</p> <table> <tbody> <tr> <td>Astrida</td> <td>659.67</td> </tr> <tr> <td>Betacab</td> <td>659.67</td> </tr> <tr> <td>Canutility</td> <td>659.67</td> </tr> <tr> <td>Divadip:</td> <td>659.67</td> </tr> </tbody> </table>	Astrida	659.67	Betacab	659.67	Canutility	659.67	Divadip:	659.67
Astrida	659.67								
Betacab	659.67								
Canutility	659.67								
Divadip:	659.67								
	<table> <tbody> <tr> <td>Astrida</td> <td>659.67</td> </tr> <tr> <td>Betacab</td> <td>659.67</td> </tr> <tr> <td>Canutility</td> <td>659.67</td> </tr> <tr> <td>Divadip:</td> <td>659.67</td> </tr> </tbody> </table>	Astrida	659.67	Betacab	659.67	Canutility	659.67	Divadip:	659.67
Astrida	659.67								
Betacab	659.67								
Canutility	659.67								
Divadip:	659.67								

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];
```

Sales2013:

```
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

 [Top - chart function \(page 1272\)](#)

Column - chart function

Column() returns the value found in the column corresponding to **ColumnNo** in a straight table, disregarding dimensions. For example **Column(2)** returns the value of the second measure column.

Syntax:

Column(ColumnNo)

Return data type: dual

Arguments:

Arguments	
Argument	Description
ColumnNo	<p>Column number of a column in the table containing a measure.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  <i>The Column() function disregards dimension columns.</i> </div>

Limitations:

- Recursive calls will return NULL.
- If **ColumnNo** references a column for which there is no measure, a NULL value is returned.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Examples and results:**Example: Percentage total sales**

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	505	29.70
A	AA	16	4	64	505	12.67
A	BB	9	9	81	505	16.04
B	BB	10	5	50	505	9.90

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
B	CC	20	2	40	505	7.92
B	DD	25	-	0	505	0.00
C	AA	15	8	120	505	23.76
C	CC	19	-	0	505	0.00

Example: Percentage of sales for selected customer

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	295	50.85
A	AA	16	4	64	295	21.69
A	BB	9	9	81	295	27.46

Examples and results

Examples	Results
Order Value is added to the table as a measure with the expression: sum(UnitPrice*Unitsales).	The result of Column(1) is taken from the column Order Value, because this is the first measure column.
Total Sales Value is added as a measure with the expression: sum(TOTAL UnitPrice*Unitsales)	The result of Column(2) is taken from Total Sales Value, because this is the second measure column.
% Sales is added as a measure with the expression 100*Column(1)/Column(2)	See the results in the column % Sales in the example <i>Percentage total sales (page 1248)</i> .
Make the selection Customer A.	The selection changes the Total Sales Value, and therefore the %Sales. See the example <i>Percentage of sales for selected customer (page 1249)</i> .

Data used in examples:

```
ProductData:
LOAD * inline [
Customer|Product|unitsales|unitprice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|1|25
Canutility|AA|8|15
Canutility|CC|19
] (delimiter is '|');
```

Dimensionality - chart function

Dimensionality() returns the number of dimensions for the current row. In the case of pivot tables, the function returns the total number of dimension columns that have non-aggregation content, that is, do not contain partial sums or collapsed aggregates.

Syntax:

```
Dimensionality ( )
```

Return data type: integer

Limitations:

This function is only available in charts. For all chart types, except pivot table, it will return the number of dimensions in all rows except the total, which will be 0.

Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Example: Chart expression using Dimensionality

Example: Chart expression

The **Dimensionality()** function can be used with a pivot table as a chart expression where you want to apply different cell formatting depending on the number of dimensions in a row that has non-aggregated data. This example uses the Dimensionality() function to apply a background color to table cells that match a given condition.

Load script

Load the following data as an inline load in the data load editor to create the chart expression example below.

Productsales:

```
Load * inline [
Country,Product,Sales,Budget
Sweden,AA,100000,50000
Germany,AA,125000,175000
Canada,AA,105000,98000
Norway,AA,74850,68500
Ireland,AA,49000,48000
Sweden,BB,98000,99000
Germany,BB,115000,175000
Norway,BB,71850,68500
Ireland,BB,31000,48000
] (delimiter is ',');
```

Chart expression

Create a pivot table visualization in a Qlik Sense sheet with **Country** and **Product** as dimensions. Add **Sum(Sales)**, **Sum(Budget)**, and **Dimensionality()** as measures.

In the **Properties** panel, enter the following expression as the **Background color expression** for the **Sum(Sales)** measure:

```
if(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156),  
if(Dimensionality()=2 and Sum(Sales)<Sum(Budget),RGB(178,29,29)  
))
```

Result:

	Country	Values		
		Sum(Sales)	Sum([Budget])	Dimensionality()
-	Canada	105000	98000	1
	AA	105000	98000	2
+	Germany	240000	350000	1
-	Ireland	80000	96000	1
	AA	49000	48000	2
	BB	31000	48000	2
-	Norway	146700	137000	1
	AA	74850	68500	2
	BB	71850	68500	2
+	Sweden	198000	149000	1

Explanation

The expression `if(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156), if(Dimensionality()=2 and Sum(Sales)<Sum(Budget),RGB(178,29,29)))` contains conditional statements that check the dimensionality value and the Sum(Sales) and Sum(Budget) for each product. If the conditions are met, a background color is applied to the Sum(Sales) value.

Exists

Exists() determines whether a specific field value has already been loaded into the field in the data load script. The function returns TRUE or FALSE, so can be used in the **where** clause of a **LOAD** statement or an **IF** statement.



You can also use **Not Exists()** to determine if a field value has not been loaded, but caution is recommended if you use **Not Exists()** in a where clause. The **Exists()** function tests both previously loaded tables and previously loaded values in the current table. So, only the first occurrence will be loaded. When the second occurrence is encountered, the value is already loaded. See the examples for more information.

Syntax:

```
Exists(field_name [, expr])
```

Return data type: Boolean

Arguments:

Arguments

Argument	Description
field_name	<p>The name of the field where you want to search for a value. You can use an explicit field name without quotes.</p> <p>The field must already be loaded by the script. That means, you cannot refer to a field that is loaded in a clause further down in the script.</p>
expr	<p>The value that you want to check if it exists. You can use an explicit value or an expression that refers to one or several fields in the current load statement.</p> <p> You cannot refer to fields that are not included in the current load statement.</p> <p>This argument is optional. If you omit it, the function will check if the value of field_name in the current record already exists.</p>

Examples and results:

Example 1

```
Exists (Employee)
```

Returns -1 (True) if the value of the field **Employee** in the current record already exists in any previously read record containing that field.

The statements `Exists (Employee, Employee)` and `Exists (Employee)` are equivalent.

Example 2

```
Exists(Employee, 'Bill')
```

Returns -1 (True) if the field value '**Bill**' is found in the current content of the field **Employee**.

Example 3

```
Employees:  
LOAD * inline [  
Employee|ID|Salary  
Bill|001|20000  
John|002|30000  
Steve|003|35000  
] (delimiter is '|');  
  
Citizens:  
Load * inline [  
Employee|Address  
Bill|New York  
Mary|London  
Steve|Chicago  
Lucy|Madrid  
Lucy|Paris  
John|Miami  
] (delimiter is '|') where Exists (Employee);  
  
Drop Tables Employees;
```

This results in a table that you can use in a table visualization using the dimensions Employee and Address.

The where clause, `where Exists (Employee)`, means only the names from the table Citizens that are also in Employees are loaded into the new table. The Drop statement removes the table Employees to avoid confusion.

Results

Employee	Address
Bill	New York
John	Miami
Steve	Chicago

Example 4

```
Employees:  
Load * inline [  
Employee|ID|Salary  
Bill|001|20000  
John|002|30000  
Steve|003|35000  
] (delimiter is '|');
```

Citizens:

```
Load * inline [  
Employee|Address
```

```
Bill|New York
Mary|London
Steve|Chicago
Lucy|Madrid
Lucy|Paris
John|Miami
] (delimiter is '|') where not Exists (Employee);

Drop Tables Employees;
```

The where clause includes not: where not Exists (Employee).

This means that only the names from the table Citizens that are not in Employees are loaded into the new table.

Note that there are two values for Lucy in the Citizens table, but only one is included in the result table. When you load the first row with the value Lucy, it is included in the Employee field. Hence, when the second line is checked, the value already exists.

Results

Employee	Address
Mary	London
Lucy	Madrid

Example 5

This example shows how to load all values.

```
Employees:
Load Employee As Name;
LOAD * inline [
Employee|ID|Salary
Bill|001|20000
John|002|30000
Steve|003|35000
] (delimiter is '|');

Citizens:
Load * inline [
Employee|Address
Bill|New York
Mary|London
Steve|Chicago
Lucy|Madrid
Lucy|Paris
John|Miami
] (delimiter is '|') where not Exists (Name, Employee);

Drop Tables Employees;
```

To be able to get all values for Lucy, two things were changed:

- A preceding load to the Employees table was inserted where Employee was renamed to Name.
Load Employee As Name;
- The Where condition in Citizens was changed to:
not Exists (Name, Employee).

This creates fields for Name and Employee. When the second row with Lucy is checked, it still does not exist in Name.

Results	
Employee	Address
Mary	London
Lucy	Madrid
Lucy	Paris

FieldIndex

FieldIndex() returns the position of the field value **value** in the field **field_name** (by load order).

Syntax:

```
FieldIndex(field_name , value)
```

Return data type: integer

Arguments:

Arguments	
Argument	Description
field_name	Name of the field for which the index is required. For example, the column in a table. Must be given as a string value. This means that the field name must be enclosed by single quotes.
value	The value of the field field_name .

Limitations:

- If **value** cannot be found among the field values of the field **field_name**, 0 is returned.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function. This limitation does not apply to the equivalent script function.

Examples and results:

The following examples use the field: **First name** from the tableNames.

Examples and results

Examples	Results
Add the example data to your app and run it.	The table Names is loaded, as in the sample data.
Chart function: In a table containing the dimension First name, add as a measure:	
<code>FieldIndex ('First name', 'John')</code>	1, because 'John' appears first in the load order of the First name field. Note that in a filter pane John would appear as number 2 from the top as it's sorted alphabetically and not as in the load order.
<code>FieldIndex ('First name', 'Peter')</code>	4, because FieldIndex() returns only one value, that is the first occurrence in the load order.
Script function: Given the table Names is loaded, as in the example data:	
<code>John1: Load FieldIndex('First name', 'John') as MyJohnPos Resident Names;</code>	MyJohnPos=1, because 'John' appears first in the load order of the First name field. Note that in a filter pane John would appear as number 2 from the top as it's sorted alphabetically and not as in the load order.
<code>Peter1: Load FieldIndex('First name', 'Peter') as MyPeterPos Resident Names;</code>	MyPeterPos=4, because FieldIndex() returns only one value, that is the first occurrence in the load order.

Data used in example:

```

Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');

John1:
Load FieldIndex('First name', 'John') as MyJohnPos
Resident Names;

Peter1:
Load FieldIndex('First name', 'Peter') as MyPeterPos
Resident Names;

```

FieldValue

FieldValue() returns the value found in position **elem_no** of the field **field_name** (by load order).

Syntax:

```
FieldValue(field_name , elem_no)
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
field_name	Name of the field for which the value is required. For example, the column in a table. Must be given as a string value. This means that the field name must be enclosed by single quotes.
elem_no	The position (element) number of the field, following the load order, that the value is returned for. This could correspond to the row in a table, but it depends on the order in which the elements (rows) are loaded.

Limitations:

- If **elem_no** is larger than the number of field values, NULL is returned.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function. This limitation does not apply to the equivalent script function.

Example

Load script

Load the following data as an inline load in the data load editor to create the example below.

Names :

```
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC |No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

John1:

```
Load FieldValue('First name',1) as MyPos1
Resident Names;
```

Peter1:

```
Load FieldValue('First name',5) as MyPos2  
Resident Names;
```

Create a visualization

Create a table visualization in a Qlik Sense sheet. Add fields **First name**, **MyPos1**, and **MyPos2** to the table.

Result

First name	MyPos1	MyPos2
Jane	John	Jane
John	John	Jane
Mark	John	Jane
Peter	John	Jane
Sue	John	Jane

Explanation

FieldValue('First name','1') results in John as the value for **MyPos1** for all first names because John appears first in the load order of the **First name** field. Note that in a filter pane John would appear as number 2 from the top, after Jane, as it's sorted alphabetically and not as in the load order.

FieldValue('First name','5') results in Jane as the value for **MyPos2** for all first names because Jane appears fifth in the load order of the **First name** field.

FieldValueCount

FieldValueCount() is an **integer** function that returns the number of distinct values in a field.

A partial reload can remove values from the data, which will not be reflected in the number returned. The returned number will correspond to all distinct values that were loaded in either the initial reload or any subsequent partial reload.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function. This limitation does not apply to the equivalent script function.

Syntax:

```
FieldValueCount(field_name)
```

Return data type: integer

Arguments:

Arguments	
Argument	Description
field_name	Name of the field for which the value is required. For example, the column in a table. Must be given as a string value. This means that the field name must be enclosed by single quotes.

Examples and results:

The following examples use the field **First name** from the table **Names**.

Examples and results

Examples	Results
Add the example data to your app and run it.	The table Names is loaded, as in the sample data.
Chart function: In a table containing the dimension First name, add as a measure:	
FieldValueCount('First name')	5 as Peter appears twice.
FieldValueCount('Initials')	6 as Initials only has distinct values.
Script function: Given the table Names is loaded, as in the example data:	
FieldCount1: Load FieldValueCount('First name') as MyFieldCount1 Resident Names;	MyFieldCount1=5, because 'Peter' appears twice.
FieldCount2: Load FieldValueCount('Initials') as MyInitialsCount1 Resident Names;	MyFieldCount1=6, because 'Initials' only has distinct values.

Data used in examples:

```
Names:
LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC|No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');

FieldCount1:
Load FieldValueCount('First name') as MyFieldCount1
Resident Names;
```

```
FieldCount2:  
Load FieldValueCount('Initials') as MyInitialsCount1  
Resident Names;
```

LookUp

Lookup() looks into a table that is already loaded and returns the value of **field_name** corresponding to the first occurrence of the value **match_field_value** in the field **match_field_name**. The table can be the current table or another table previously loaded.

Syntax:

```
lookup(field_name, match_field_name, match_field_value [, table_name])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
field_name	Name of the field for which the return value is required. Input value must be given as a string (for example, quoted literals).
match_field_name	Name of the field to look up match_field_value in. Input value must be given as a string (for example, quoted literals).
match_field_value	Value to look up in match_field_name field.
table_name	Name of the table in which to look up the value. Input value must be given as a string (for example quoted literals). If table_name is omitted the current table is assumed.



Arguments without quotes refer to the current table. To refer to other tables, enclose an argument in single quotes.

Limitations:

The order in which the search is made is the load order, unless the table is the result of complex operations such as joins, in which case, the order is not well defined. Both **field_name** and **match_field_name** must be fields in the same table, specified by **table_name**.

If no match is found, NULL is returned.

Example

Load script

Load the following data as an inline load in the data load editor to create the example below.

```

ProductList:
Load * Inline [
ProductID|Product|Category|Price
1|AA|1|1
2|BB|1|3
3|CC|2|8
4|DD|3|2
] (delimiter is '|');

OrderData:
Load *, Lookup('Category', 'ProductID', ProductID, 'ProductList') as CategoryID
Inline [
InvoiceID|CustomerID|ProductID|Units
1|Astrida|1|8
1|Astrida|2|6
2|Betacab|3|10
3|Divadip|3|5
4|Divadip|4|10
] (delimiter is '|');

Drop Table ProductList;

```

Create a visualization

Create a table visualization in a Qlik Sense sheet. Add fields **ProductID**, **InvoiceID**, **CustomerID**, **Units**, and **CategoryID** to the table.

Result

Resulting table

ProductID	InvoiceID	CustomerID	Units	CategoryID
1	1	Astrida	8	1
2	1	Astrida	6	1
3	2	Betacab	10	2
3	3	Divadip	5	2
4	4	Divadip	10	3

Explanation

The sample data uses the **Lookup()** function in the following form:

```
Lookup('Category', 'ProductID', ProductID, 'ProductList')
```

The **ProductList** table is loaded first.

The **Lookup()** function is used to build the **OrderData** table. It specifies the third argument as **ProductID**. This is the field for which the value is to be looked up in the second argument '**ProductID**' in the **ProductList**, as denoted by the enclosing single quotes.

The function returns the value for '**Category**' (in the **ProductList** table), loaded as **CategoryID**.

The **drop** statement deletes the **ProductList** table from the data model because it is not required, which leaves the resulting **OrderData** table.



*The **Lookup()** function is flexible and can access any previously loaded table. However, it is slow compared with the **Applymap()** function.*

See also:

[ApplyMap \(page 1283\)](#)

NoOfRows - chart function

NoOfRows() returns the number of rows in the current column segment in a table. For bitmap charts, **NoOfRows()** returns the number of rows in the chart's straight table equivalent.

If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Syntax:

NoOfRows ([TOTAL])

Return data type: integer

Arguments:

Arguments

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

Example: Chart expression using NoOfRows

Example - chart expression

Load script

Load the following data as an inline load in the data load editor to create the chart expression examples below.

```

Temp:
LOAD * inline [
Region|SubRegion|RowNo()|NoOfRows()
Africa|Eastern
Africa|Western
Americas|Central
Americas|Northern
Asia|Eastern
Europe|Eastern
Europe|Northern
Europe|Western
Oceania|Australia
] (delimiter is '|');

```

Chart expression

Create a table visualization in a Qlik Sense sheet with **Region** and **SubRegion** as dimensions. Add `RowNo()`, `NoOfRows()`, and `NoOfRows(Total)` as measures.

Result

Region	SubRegion	RowNo()	NoOfRows()	NoOfRows (Total)
Africa	Eastern	1	2	9
Africa	Western	2	2	9
Americas	Central	1	2	9
Americas	Northern	2	2	9
Asia	Eastern	1	1	9
Europe	Eastern	1	3	9
Europe	Northern	2	3	9
Europe	Western	3	3	9
Oceania	Australia	1	1	9

Explanation

In this example, the sort order is by the first dimension, Region. As a result, each column segment is made up of a group of regions that has the same value, for example, Africa.

The **RowNo()** column shows the row numbers for each column segment, for example, there are two rows for the Africa region. The row numbering then begins at 1 again for the next column segment, which is Americas.

The **NoOfRows()** column counts the number of rows in each column segment, for example, Europe has three rows in the column segment.

The **NoOfRows(Total)** column disregards the dimensions because of the **TOTAL** argument for **NoOfRows()** and counts the rows in the table.

If the table was sorted on the second dimension, **SubRegion**, the column segments would be based on that dimension so the row numbering would change for each **SubRegion**.

See also:

- [RowNo - chart function \(page 566\)](#)

Peek

Peek() returns the value of a field in a table for a row that has already been loaded. The row number can be specified, as can the table. If no row number is specified, the last previously loaded record will be used.

The **peek()** function is most often used to find the relevant boundaries in a previously loaded table, that is, the first value or last value of a specific field. In most cases, this value is stored in a variable for later use, for example, as a condition in a do-while loop.

Syntax:

```
Peek(  
    field_name  
    [, row_no[, table_name ] ])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
field_name	Name of the field for which the return value is required. Input value must be given as a string (for example, quoted literals).
row_no	The row in the table that specifies the field required. Can be an expression, but must resolve to an integer. 0 denotes the first record, 1 the second, and so on. Negative numbers indicate order from the end of the table. -1 denotes the last record read. If no row_no is stated, -1 is assumed.
table_name	A table label without the ending colon. If no table_name is stated, the current table is assumed. If used outside the LOAD statement or referring to another table, the table_name must be included.

Limitations:

The function can only return values from already loaded records. This means that in the first record of a table, a call using -1 as **row_no** will return NULL.

Examples and results:

Example 1

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
EmployeeDates:
Load * Inline [
EmployeeCode|StartDate|EndDate
101|02/11/2010|23/06/2012
102|01/11/2011|30/11/2013
103|02/01/2012|
104|02/01/2012|31/03/2012
105|01/04/2012|31/01/2013
106|02/11/2013|
] (delimiter is '|');

First_last_Employee:
Load
EmployeeCode,
Peek('EmployeeCode',0,'EmployeeDates') As FirstCode,
Peek('EmployeeCode',-1,'EmployeeDates') As LastCode
Resident EmployeeDates;
```

Resulting table

Employee code	StartDate	EndDate	FirstCode	LastCode
101	02/11/2010	23/06/2012	101	106
102	01/11/2011	30/11/2013	101	106
103	02/01/2012		101	106
104	02/01/2012	31/03/2012	101	106
105	01/04/2012	31/01/2013	101	106
106	02/11/2013		101	106

FirstCode = 101 because Peek('EmployeeCode',0, 'EmployeeDates') returns the first value of EmployeeCode in the table EmployeeDates.

LastCode = 106 because Peek('EmployeeCode',-1, 'EmployeeDates') returns the last value of EmployeeCode in the table EmployeeDates.

Substituting the value of the argument **row_no** returns the values of other rows in the table, as follows:

Peek('EmployeeCode',2, 'EmployeeDates') returns the third value, 103, in the table as the FirstCode.

However, note that without specifying the table as the third argument **table_name** in these examples, the function references the current (in this case, internal) table.

Example 2

If you want to access data further down in a table, you need to do it in two steps: first, load the entire table into a temporary table, and then re-sort it when using **Peek()**.

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
T1:
LOAD * inline [
ID|value
1|3
1|4
1|6
3|7
3|8
2|1
2|11
5|2
5|78
5|13
] (delimiter is '|');
```

```
T2:
```

```
LOAD *,
IF(ID=Peek('ID'), Peek('List')&','&value,value) AS List
RESIDENT T1
ORDER BY ID ASC;
DROP TABLE T1;
```

Create a table in a sheet in your app with **ID**, **List**, and **Value** as the dimensions.

Resulting table

ID	List	Value
1	3,4	4
1	3,4,6	6
1	3	3
2	1,11	11
2	1	1
3	7,8	8
3	7	7
5	2,78	78
5	2,78,13	13
5	2	2

The **IF()** statement is built from the temporary table T1.

`Peek('ID')` references the field ID in the previous row in the current table T2.

`Peek('List')` references the field List in the previous row in the table T2, currently being built as the expression is evaluated.

The statement is evaluated as follows:

If the current value of ID is the same as the previous value of ID, then write the value of `Peek('List')` concatenated with the current value of Value. Otherwise, write the current value of Value only.

If `Peek('List')` already contains a concatenated result, the new result of `Peek('List')` will be concatenated to it.



*Note the **Order by** clause. This specifies how the table is ordered (by ID in ascending order). Without this, the `Peek()` function will use whatever arbitrary ordering the internal table has, which can lead to unpredictable results.*

Example 3

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
Amounts:  
Load  
Date#(Month, 'YYYY-MM') as Month,  
Amount,  
Peek(Amount) as AmountMonthBefore  
Inline  
[Month,Amount  
2022-01,2  
2022-02,3  
2022-03,7  
2022-04,9  
2022-05,4  
2022-06,1];
```

Resulting table

Amount	AmountMonthBefore	Month
1	4	2022-06
2	-	2022-01
3	2	2022-02
4	9	2022-05
7	3	2022-03
9	7	2022-04

The field `AmountMonthBefore` will hold the amount from the previous month.

Here, the row_no and table_name parameters are omitted, so the default values are used. In this example, the following three function calls are equivalent:

- Peek(Amount)
- Peek(Amount,-1)
- Peek(Amount,-1,'Amounts')

Using -1 as row_no means that the value from previous row will be used. By substituting this value, values of other rows in the table can be fetched:

Peek(Amount,2) returns the third value in the table: 7.

Example 4

Data needs to be correctly sorted in order to get the correct results but, unfortunately, this is not always the case. Furthermore, the Peek() function cannot be used to reference data that has not yet been loaded. By using temporary tables and running multiple passes through the data, such problems can be avoided.

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
tmp1Amounts:  
Load * Inline  
[Month,Product,Amount  
2022-01,B,3  
2022-01,A,8  
2022-02,B,4  
2022-02,A,6  
2022-03,B,1  
2022-03,A,6  
2022-04,A,5  
2022-04,B,5  
2022-05,B,6  
2022-05,A,7  
2022-06,A,4  
2022-06,B,8];  
  
tmp2Amounts:  
Load *,  
If(Product=Peek(Product),Peek(Amount)) as AmountMonthBefore  
Resident tmp1Amounts  
Order By Product, Month Asc;  
Drop Table tmp1Amounts;  
  
Amounts:  
Load *,  
If(Product=Peek(Product),Peek(Amount)) as AmountMonthAfter  
Resident tmp2Amounts  
Order By Product, Month Desc;  
Drop Table tmp2Amounts;
```

Explanation

The initial table is sorted according to month, which means that the peek() function would in many cases return the amount for the wrong product. Hence, this table needs to be re-sorted. This is done by running a second pass through the data creating a new table tmp2Amounts. Note the Order By clause. It orders the records first by product, then by month in ascending order.

The If() function is needed since the AmountMonthBefore only should be calculated if the previous row contains the data for the same product but for the previous month. By comparing the product on the current row with the product on the previous row, this condition can be validated.

When the second table is created, the first table tmp1Amounts is dropped using a Drop Table statement.

Finally, a third pass is made through the data, but now with the months sorted in reverse order. This way, AmountMonthAfter can also be calculated.



Order by clauses specify how the table is ordered; without these, the Peek() function will use whatever arbitrary ordering the internal table has, which can lead to unpredictable results.

Result

Resulting table

Month	Product	Amount	AmountMonthBefore	AmountMonthAfter
2022-01	A	8	-	6
2022-02	B	3	-	4
2022-03	A	6	8	6
2022-04	B	4	3	1
2022-05	A	6	6	5
2022-06	B	1	4	5
2022-01	A	5	6	7
2022-02	B	5	1	6
2022-03	A	7	5	4
2022-04	B	6	5	8
2022-05	A	4	7	-
2022-06	B	8	6	-

Example 5

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
T1:
Load * inline [
Quarter, Value
2003q1, 10000
2003q1, 25000
2003q1, 30000
2003q2, 1250
2003q2, 55000
2003q2, 76200
2003q3, 9240
2003q3, 33150
2003q3, 89450
2003q4, 1000
2003q4, 3000
2003q4, 5000
2004q1, 1000
2004q1, 1250
2004q1, 3000
2004q2, 5000
2004q2, 9240
2004q2, 10000
2004q3, 25000
2004q3, 30000
2004q3, 33150
2004q4, 55000
2004q4, 76200
2004q4, 89450 ];
```

```
T2:
Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal;
Load Quarter, sum(value) as SumVal resident T1 group by Quarter;
```

Result

Resulting table

Quarter	SumVal	AccSumVal
2003q1	65000	65000
2003q2	132450	197450
2003q3	131840	329290
2003q4	9000	338290
2004q1	5250	343540
2004q2	24240	367780
2004q3	88150	455930
2004q4	220650	676580

Explanation

The load statement **Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal** includes a recursive call where the previous values are added to the current value. This operation is used to calculate an accumulation of values in the script.

See also:

Previous

Previous() finds the value of the **expr** expression using data from the previous input record that has not been discarded because of a **where** clause. In the first record of an internal table, the function will return NULL.

Syntax:

```
Previous(expr)
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured. The expression can contain nested previous() functions in order to access records further back. Data are fetched directly from the input source, making it possible to refer also to fields that have not been loaded into Qlik Sense, that is, even if they have not been stored in its associative database.

Limitations:

In the first record of an internal table, the function returns NULL.

Example:

Input the following into your load script

```
Sales2013:  
Load *, (Sales - Previous(Sales)) as Increase inline [  
Month|sales  
1|12  
2|13  
3|15  
4|17  
5|21  
6|21  
7|22  
8|23
```

```
9|32
10|35
11|40
12|41
] (delimiter is '|');
```

By using the **Previous()** function in the **Load** statement, we can compare the current value of Sales with the preceding value, and use it in a third field, Increase.

Resulting table

Month	Sales	Increase
1	12	-
2	13	1
3	15	2
4	17	2
5	21	4
6	21	0
7	22	1
8	23	1
9	32	9
10	35	3
11	40	5
12	41	1

Top - chart function

Top() evaluates an expression at the first (top) row of a column segment in a table. The row for which it is calculated depends on the value of **offset**, if present, the default being the top row. For charts other than tables, the **Top()** evaluation is made on the first row of the current column in the chart's straight table equivalent.

Syntax:

```
Top([TOTAL] expr [ , offset [,count ]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.

Argument	Description
offset	Specifying an offset of n, greater than 1, moves the evaluation of the expression down n rows below the top row. Specifying a negative offset number makes the Top function work like the Bottom function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return a range of count values, one for each of the last count rows of the current column segment. In this form, the function can be used as an argument to any of the special range functions. <i>Range functions (page 1290)</i>
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.



A column segment is defined as a consecutive subset of cells having the same values for the dimensions in the current sort order. Inter-record chart functions are computed in the column segment excluding the right-most dimension in the equivalent straight table chart. If there is only one dimension in the chart, or if the **TOTAL** qualifier is specified, the expression evaluates across full table.



If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns, except for the column showing the last dimension in the inter-field sort order.

Limitations:

- Recursive calls will return NULL.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Examples and results:

Example: 1

In the screenshot of the table shown in this example, the table visualization is created from the dimension **Customer** and the measures: `sum(Sales)` and `Top(sum(Sales))`.

The column **Top(Sum(Sales))** returns 587 for all rows because this is the value of the top row: **Astrida**.

The table also shows more complex measures: one created from `sum(Sales)+Top(Sum(Sales))` and one labeled **Top offset 3**, which is created using the expression `sum(Sales)+Top(Sum(Sales), 3)` and has the argument **offset** set to 3. It adds the **Sum(Sales)** value for the current row to the value from the third row from the top row, that is, the current row plus the value for **Canutility**.

Example 1

Top and Bottom					
Customer	Q	Sum(Sales)	Top(Sum(Sales))	Sum(Sales)+Top(Sum(Sales))	Top offset 3
Totals		2566	587	3153	3249
Astrida		587	587	1174	1270
Betacab		539	587	1126	1222
Canutility		683	587	1270	1366
Divadip		757	587	1344	1440

Example: 2

In the screenshots of tables shown in this example, more dimensions have been added to the visualizations: **Month** and **Product**. For charts with more than one dimension, the results of expressions containing the **Above**, **Below**, **Top**, and **Bottom** functions depend on the order in which the column dimensions are sorted by Qlik Sense. Qlik Sense evaluates the functions based on the column segments that result from the dimension that is sorted last. The column sort order is controlled in the properties panel under **Sorting** and is not necessarily the order in which the columns appear in a table.

First table for Example 2. The value of Top for the First value measure based on Month (Jan).

Customer	Product	Month	Sum(Sales)	Firstvalue
			2566	-
Astrida	AA	Jan	46	46
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	46
Astrida	AA	Apr	13	46
Astrida	AA	May	78	46
Astrida	AA	Jun	20	46
Astrida	AA	Jul	45	46
Astrida	AA	Aug	65	46
Astrida	AA	Sep	78	46
Astrida	AA	Oct	12	46
Astrida	AA	Nov	78	46
Astrida	AA	Dec	22	46

Second table for Example 2. The value of Top for the First value measure based on Product (AA for Astrida).

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

Please refer to Example: 2 in the **Above** function for further details.

Example 3

Example: 3	Result								
The Top function can be used as input to the range functions. For example: RangeAvg (Top(Sum(Sales),1,3)).	In the arguments for the Top() function, offset is set to 1 and count is set to 3. The function finds the results of the expression Sum(Sales) on the three rows starting with the row below the bottom row in the column segment (because the offset=1), and the two rows below that (where there is a row). These three values are used as input to the RangeAvg() function, which finds the average of the values in the supplied range of numbers. A table with Customer as dimension gives the following results for the RangeAvg() expression.								
	<table> <tbody> <tr> <td>Astrida</td> <td>603</td> </tr> <tr> <td>Betacab</td> <td>603</td> </tr> <tr> <td>Canutility</td> <td>603</td> </tr> <tr> <td>Divadip:</td> <td>603</td> </tr> </tbody> </table>	Astrida	603	Betacab	603	Canutility	603	Divadip:	603
Astrida	603								
Betacab	603								
Canutility	603								
Divadip:	603								

Monthnames:

```
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
```

```
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

-  [Bottom - chart function \(page 1243\)](#)
-  [Above - chart function \(page 1235\)](#)
-  [Sum - chart function \(page 336\)](#)
-  [RangeAvg \(page 1293\)](#)
-  [Range functions \(page 1290\)](#)

SecondaryDimensionality - chart function

SecondaryDimensionality() returns the number of dimension pivot table rows that have non-aggregation content, that is, do not contain partial sums or collapsed aggregates. This function is the equivalent of the **dimensionality()** function for horizontal pivot table dimensions.

Syntax:

```
SecondaryDimensionality( )
```

Return data type: integer

Limitations:

- Unless used in pivot tables, the **SecondaryDimensionality** function always returns 0.
- Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

After - chart function

After() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the column after the current column within a row segment in the pivot table.

Syntax:

```
after([TOTAL] expr [, offset [, count ]])
```



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.



This function returns NULL in all chart types except pivot tables.

Arguments:

Argument	Description
expr	The expression or field containing the data to be measured.
offset	<p>Specifying an offset n, greater than 1 moves the evaluation of the expression n rows further to the right from the current row.</p> <p>Specifying an offset of 0 will evaluate the expression on the current row.</p> <p>Specifying a negative offset number makes the After function work like the Before function with the corresponding positive offset number.</p>
count	By specifying a third parameter count greater than 1, the function will return a range of values, one for each of the table rows up to the value of count , counting to the right from the original cell.
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

On the last column of a row segment a NULL value will be returned, as there is no column after this one.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.

Example:

```
after( sum( sales ))
after( sum( sales ), 2 )
after( total sum( sales ))
rangeavg( after(sum(x),1,3)) returns an average of the three results of the sum(x) function evaluated in
the three columns immediately to the right of the current column.
```

Before - chart function

Before() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the column before the current column within a row segment in the pivot table.

Syntax:

```
before([TOTAL] expr [, offset [, count]])
```



This function returns NULL in all chart types except pivot tables.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offset n, greater than 1 moves the evaluation of the expression n rows further to the left from the current row. Specifying an offset of 0 will evaluate the expression on the current row. Specifying a negative offset number makes the Before function work like the After function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return a range of values, one for each of the table rows up to the value of count , counting to the left from the original cell.
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

On the first column of a row segment a NULL value will be returned, as there is no column before this one.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.

Examples:

```
before( sum( sales ))
before( sum( Sales ), 2 )
before( total sum( sales ))
```

rangeavg (before(sum(x),1,3)) returns an average of the three results of the **sum(x)** function evaluated in the three columns immediately to the left of the current column.

First - chart function

First() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the first column of the current row segment in the pivot table. This function returns NULL in all chart types except pivot tables.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Syntax:

```
first([TOTAL] expr [, offset [, count]])
```

Arguments:

Arguments	
Argument	Description
expression	The expression or field containing the data to be measured.
offset	<p>Specifying an offset n, greater than 1 moves the evaluation of the expression n rows further to the right from the current row.</p> <p>Specifying an offset of 0 will evaluate the expression on the current row.</p> <p>Specifying a negative offset number makes the First function work like the Last function with the corresponding positive offset number.</p>
count	By specifying a third parameter count greater than 1, the function will return a range of values, one for each of the table rows up to the value of count , counting to the right from the original cell.
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.

Examples:

```
first( sum( Sales ))
first( sum( Sales ), 2 )
first( total sum( Sales ) )
rangeavg (first(sum(x),1,5)) returns an average of the results of the sum(x) function evaluated on the five leftmost columns of the current row segment.
```

Last - chart function

Last() returns the value of an expression evaluated with a pivot table's dimension values as they appear in the last column of the current row segment in the pivot table. This function returns NULL in all chart types except pivot tables.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Syntax:

```
last([TOTAL] expr [, offset [, count]])
```

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
offset	Specifying an offset n, greater than 1 moves the evaluation of the expression n rows further to the left from the current row. Specifying an offset of 0 will evaluate the expression on the current row. Specifying a negative offset number makes the First function work like the Last function with the corresponding positive offset number.
count	By specifying a third parameter count greater than 1, the function will return a range of values, one for each of the table rows up to the value of count , counting to the left from the original cell.
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.

Example:

```
last( sum( sales ))
last( sum( sales ), 2 )
last( total sum( Sales ) )
rangeavg (last(sum(x),1,5)) returns an average of the results of the sum(x) function evaluated on the five rightmost columns of the current row segment.
```

ColumnNo - chart function

ColumnNo() returns the number of the current column within the current row segment in a pivot table. The first column is number 1.

Syntax:

```
ColumnNo([total])
```

Arguments:

Arguments	
Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Example:

```
if( columnNo( )=1, 0, sum( sales ) / before( sum( sales )))
```

NoOfColumns - chart function

NoOfColumns() returns the number of columns in the current row segment in a pivot table.



Sorting on y-values in charts or sorting by expression columns in tables is not allowed when this chart function is used in any of the chart's expressions. These sort alternatives are therefore automatically disabled. When you use this chart function in a visualization or table, the sorting of the visualization will revert back to the sorted input to this function.

Syntax:

```
NoOfColumns([total])
```

Arguments:

Arguments

Argument	Description
TOTAL	If the table is one-dimensional or if the qualifier TOTAL is used as argument, the current column segment is always equal to the entire column.

If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last dimension in the inter-field sort order. The inter-field sort order for horizontal dimensions in pivot tables is defined simply by the order of the dimensions from top to bottom.

Example:

```
if( columnNo( )=NoOfColumns( ), 0, after( sum( sales ) ) )
```

5.17 Logical functions

This section describes functions handling logical operations. All functions can be used in both the data load script and in chart expressions.

IsNum

Returns -1 (True) if the expression can be interpreted as a number, otherwise 0 (False).

```
IsNum( expr )
```

IsText

Returns -1 (True) if the expression has a text representation, otherwise 0 (False).

```
IsText( expr )
```



Both **IsNum** and **IsText** return 0 if the expression is NULL.

Example:

The following example loads an inline table with mixed text and numerical values, and adds two fields to check if the value is a numerical value, respectively a text value.

```
Load *, IsNum(value), IsText(value)
Inline [
Value
23
Green
Blue
12
33Red];
```

The resulting table looks like this:

Resulting table		
Value	IsNum(Value)	IsText(Value)
23	-1	0
Green	0	-1
Blue	0	-1
12	-1	0
33Red	0	-1

5.18 Mapping functions

This section describes functions for handling mapping tables. A mapping table can be used to replace field values or field names during script execution.

Mapping functions can only be used in the data load script.

Mapping functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

ApplyMap

The **ApplyMap** script function is used for mapping the output of an expression to a previously loaded mapping table.

```
ApplyMap ('mapname', expr [ , defaultexpr ] )
```

MapSubstring

The **MapSubstring** script function is used to map parts of any expression to a previously loaded mapping table. The mapping is case sensitive and non-iterative, and substrings are mapped from left to right.

```
MapSubstring ('mapname', expr)
```

ApplyMap

The **ApplyMap** script function is used for mapping the output of an expression to a previously loaded mapping table.

Syntax:

```
ApplyMap('map_name', expression [ , default_mapping ] )
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
map_name	The name of a mapping table that has previously been created through the mapping load or the mapping select statement. Its name must be enclosed by single, straight quotation marks.
	 <i>If you use this function in a macro expanded variable and refer to a mapping table that does not exist, the function call fails and a field is not created.</i>
expression	The expression, the result of which should be mapped.
default_mapping	If stated, this value will be used as a default value if the mapping table does not contain a matching value for expression. If not stated, the value of expression will be returned as is.



The output field of ApplyMap should not have the same name as one of its input fields. This may cause unexpected results. Example not to use: ApplyMap('Map', A) as A

Example:

In this example we load a list of salespersons with a country code representing their country of residence. We use a table mapping a country code to a country to replace the country code with the country name. Only three countries are defined in the mapping table, other country codes are mapped to 'Rest of the world'.

```
// Load mapping table of country codes:
map1:
mapping LOAD *
Inline [
CCode, Country
Sw, Sweden
Dk, Denmark
No, Norway
] ;

// Load list of salesmen, mapping country code to country
// If the country code is not in the mapping table, put Rest of the world
Salespersons:
LOAD *,
ApplyMap('map1', CCode, 'Rest of the world') As Country
Inline [
CCode, Salesperson
Sw, John
Sw, Mary
Sw, Per
```

```
Dk, Preben
Dk, olle
No, ole
Sf, Risttu
] ;
```

// we don't need the CCode anymore
Drop Field 'CCode';

The resulting table (Salespersons) looks like this:

Resulting table

Salesperson	Country
John	Sweden
Mary	Sweden
Per	Sweden
Preben	Denmark
Olle	Denmark
Ole	Norway
Risttu	Rest of the world

MapSubstring

The **MapSubstring** script function is used to map parts of any expression to a previously loaded mapping table. The mapping is case sensitive and non-iterative, and substrings are mapped from left to right.

Syntax:

```
MapSubstring('map_name', expression)
```

Return data type: string

Arguments:

Arguments

Argument	Description
map_name	The name of a mapping table previously read by a mapping load or a mapping select statement. The name must be enclosed by single straight quotation marks.  <i>If you use this function in a macro expanded variable and refer to a mapping table that does not exist, the function call fails and a field is not created.</i>
expression	The expression whose result is to be mapped by substrings.

Example:

In this example we load a list of product models. Each model has a set of attributes that are described by a composite code. Using the mapping table with MapSubstring, we can expand the attribute codes to a description.

```
map2:
mapping LOAD *
Inline [
AttCode, Attribute
R, Red
Y, Yellow
B, Blue
C, Cotton
P, Polyester
S, Small
M, Medium
L, Large
] ;

Productmodels:
LOAD *,
MapSubstring('map2', AttCode) as Description
Inline [
Model, AttCode
Twixie, R C S
Boomer, B P L
Raven, Y P M
Seedling, R C L
SeedlingPlus, R C L with hood
Younger, B C with patch
MultiStripe, R Y B C S/M/L
] ;
// we don't need the AttCode anymore
Drop Field 'AttCode';
```

The resulting table looks like this:

Resulting table

Model	Description
Twixie	Red Cotton Small
Boomer	Blue Polyester Large
Raven	Yellow Polyester Medium
Seedling	Red Cotton Large
SeedlingPlus	Red Cotton Large with hood
Younger	Blue Cotton with patch
MultiStripe	Red Yellow Blue Cotton Small/Medium/Large

5.19 Mathematical functions

This section describes functions for mathematical constants and Boolean values. These functions do not have any parameters, but the parentheses are still required.

All functions can be used in both the data load script and in chart expressions.

e

The function returns the base of the natural logarithms, e (2.71828...).

```
e( )
```

false

The function returns a dual value with text value 'False' and numeric value 0, which can be used as logical false in expressions.

```
false( )
```

pi

The function returns the value of π (3.14159...).

```
pi( )
```

rand

The function returns a random number between 0 and 1. This can be used to create sample data.

```
rand( )
```

Example:

This example script creates a table of 1000 records with randomly selected upper case characters, that is, characters in the range 65 to 91 (65+26).

Load

```
Chr( Floor(rand() * 26) + 65) as ucasechar,  
RecNo() as ID  
Autogenerate 1000;
```

true

The function returns a dual value with text value 'True' and numeric value -1, which can be used as logical true in expressions.

```
true( )
```

5.20 NULL functions

This section describes functions for returning or detecting NULL values.

All functions can be used in both the data load script and in chart expressions.

NULL functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

EmptyIsNull

The **EmptyIsNull** function converts empty strings to NULL. Hence, it returns NULL if the parameter is an empty string, otherwise it returns the parameter.

```
EmptyIsNull (expr )
```

IsNull

The **IsNull** function tests if the value of an expression is NULL and if so, returns -1 (True), otherwise 0 (False).

```
IsNull (expr )
```

Null

The **Null** function returns a NULL value.

```
NULL( )
```

EmptyIsNull

The **EmptyIsNull** function converts empty strings to NULL. Hence, it returns NULL if the parameter is an empty string, otherwise it returns the parameter.

Syntax:

```
EmptyIsNull(exp )
```

Examples and results:

Scripting examples

Example	Result
EmptyIsNull(AdditionalComments)	This expression will return as null any empty string values of the <i>AdditionalComments</i> field, instead of empty strings. Non-empty strings and numbers are returned.
EmptyIsNull(PurgeChar (PhoneNumber, ' -()'))	This expression will strip any dashes, spaces and parentheses from the <i>PhoneNumber</i> field. If there are no characters left, the EmptyIsNull function returns the empty string as null; an empty phone number is the same as no phone number.

IsNull

The **IsNull** function tests if the value of an expression is NULL and if so, returns -1 (True), otherwise 0 (False).

Syntax:

```
IsNull(expr )
```



A string with length zero is not considered as a NULL and will cause **IsNull** to return False.

Example: Data load script

In this example, an inline table with four rows is loaded, where the first three lines contain either nothing, - or 'NULL' in the Value column. We convert these values to true NULL value representations with the middle preceding **LOAD** using the **Null** function.

The first preceding **LOAD** adds a field checking if the value is NULL, using the **IsNull** function.

NullsDetectedAndConverted:

```
LOAD *,
If(IsNull(ValueNullConv), 'T', 'F') as IsItNull;

LOAD *,
If(len(trim(value))= 0 or value='NULL' or value='-', Null(), value ) as valueNullConv;

LOAD * Inline
[ID, value
0,
1,NULL
2,-
3,value];
```

This is the resulting table. In the ValueNullConv column, the NULL values are represented by -.

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

NULL

The **Null** function returns a NULL value.

Syntax:

```
Null( )
```

Example: Data load script

In this example, an inline table with four rows is loaded, where the first three lines contain either nothing, - or 'NULL' in the Value column. We want to convert these values to true NULL value representations.

The middle preceding **LOAD** performs the conversion using the **Null** function.

The first preceding **LOAD** adds a field checking if the value is NULL, just for illustration purposes in this example.

NullsDetectedAndConverted:

```
LOAD *,
If(IsNull(valueNullConv), 'T', 'F') as IsItNull;

LOAD *,
If(len(trim(value))= 0 or value='NULL' or value='-', Null(), value ) as valueNullConv;

LOAD * Inline
[ID, value
0,
1,NULL
2,-
3,value];
```

This is the resulting table. In the ValueNullConv column, the NULL values are represented by -.

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

5.21 Range functions

The range functions are functions that take an array of values and produce a single value as a result. All range functions can be used in both the data load script and in chart expressions.

For example, in a visualization, a range function can calculate a single value from an inter-record array. In the data load script, a range function can calculate a single value from an array of values in an internal table.



*Range functions replace the following general numeric functions: **numsum**, **numavg**, **numcount**, **nummin** and **nummax**, which should now be regarded as obsolete.*

Basic range functions

RangeMax

RangeMax() returns the highest numeric values found within the expression or field.

```
RangeMax(first_expr[, Expression])
```

RangeMaxString

RangeMaxString() returns the last value in the text sort order that it finds in the expression or field.

```
RangeMaxString(first_expr[, Expression])
```

RangeMin

RangeMin() returns the lowest numeric values found within the expression or field.

```
RangeMin(first_expr[, Expression])
```

RangeMinString

RangeMinString() returns the first value in the text sort order that it finds in the expression or field.

```
RangeMinString(first_expr[, Expression])
```

RangeMode

RangeMode() finds the most commonly occurring value (mode value) in the expression or field.

```
RangeMode(first_expr[, Expression])
```

RangeOnly

RangeOnly() is a dual function that returns a value if the expression evaluates to one unique value. If this is not the case then **NULL** is returned.

```
RangeOnly(first_expr[, Expression])
```

RangeSum

RangeSum() returns the sum of a range of values. All non-numeric values are treated as 0.

```
RangeSum(first_expr[, Expression])
```

Counter range functions

RangeCount

RangeCount() returns the number of values, both text and numeric, in the expression or field.

```
RangeCount(first_expr[, Expression])
```

RangeMissingCount

RangeMissingCount() returns the number of non-numeric values (including NULL) in the expression or field.

```
RangeMissingCount(first_expr[, Expression])
```

RangeNullCount

RangeNullCount() finds the number of NULL values in the expression or field.

```
RangeNullCount(first_expr[, Expression])
```

RangeNumericCount

RangeNumericCount() finds the number of numeric values in an expression or field.

```
RangeNumericCount(first_expr[, Expression])
```

RangeTextCount

RangeTextCount() returns the number of text values in an expression or field.

```
RangeTextCount(first_expr[, Expression])
```

Statistical range functions

RangeAvg

RangeAvg() returns the average of a range. Input to the function can be either a range of values or an expression.

```
RangeAvg(first_expr[, Expression])
```

RangeCorrel

RangeCorrel() returns the correlation coefficient for two sets of data. The correlation coefficient is a measure of the relationship between the data sets.

```
RangeCorrel(x_values , y_values[, Expression])
```

RangeFractile

RangeFractile() returns the value that corresponds to the n-th **fractile** (quantile) of a range of numbers.

```
RangeFractile(fractile, first_expr[ ,Expression])
```

RangeKurtosis

RangeKurtosis() returns the value that corresponds to the kurtosis of a range of numbers.

```
RangeKurtosis(first_expr[, Expression])
```

RangeSkew

RangeSkew() returns the value corresponding to the skewness of a range of numbers.

```
RangeSkew(first_expr[, Expression])
```

RangeStdev

RangeStdev() finds the standard deviation of a range of numbers.

```
RangeStdev(expr1[, Expression])
```

Financial range functions

RangeIRR

RangeIRR() returns the internal rate of return for a series of cash flows represented by the input values.

```
RangeIRR (value[, value][, Expression])
```

RangeNPV

RangeNPV() returns the net present value of an investment based on a discount rate and a series of future periodic payments (negative values) and incomes (positive values). The result has a default number format of **money**.

```
RangeNPV (discount_rate, value[, value][, Expression])
```

RangeXIRR

RangeXIRR() returns the internal rate of return (yearly) for a schedule of cash flows that is not necessarily periodic. To calculate the internal rate of return for a series of periodic cash flows, use the **RangeIRR** function.

```
RangeXIRR (values, dates[, Expression])
```

RangeXNPV

RangeXNPV() returns the net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date**. All payments are discounted based on a 365-day year.

```
RangeXNPV (discount_rate, values, dates[, Expression])
```

See also:

- [Inter-record functions \(page 1232\)](#)

RangeAvg

RangeAvg() returns the average of a range. Input to the function can be either a range of values or an expression.

Syntax:

```
RangeAvg(first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Scripting examples

Examples	Results
RangeAvg (1,2,4)	Returns 2.33333333

Examples	Results
RangeAvg (1, 'xyz')	Returns 1
RangeAvg (null(), 'abc')	Returns NULL

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeTab3:
LOAD recno() as RangeID, RangeAvg(Field1,Field2,Field3) as MyRangeAvg INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

The resulting table shows the returned values of MyRangeAvg for each of the records in the table.

Resulting table

RangeID	MyRangeAvg
1	7
2	4
3	6
4	12.666
5	6.333
6	5

Example with expression:

```
RangeAvg (Above(MyField),0,3))
```

Returns a sliding average of the result of the range of three values of **MyField** calculated on the current row and two rows above the current row. By specifying the third argument as 3, the **Above()** function returns three values, where there are sufficient rows above, which are taken as input to the **RangeAvg()** function.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeAvg (Above (MyField,0,3))	Comments
10	10	Because this is the top row, the range consists of one value only.
2	6	There is only one row above this row, so the range is: 10,2.
8	6.6666666667	The equivalent to RangeAvg(10,2,8)
18	9.333333333	-
5	10. 333333333	-
9	10.6666666667	-

RangeTab:

```
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

See also:

- [Avg - chart function \(page 390\)](#)
- [Count - chart function \(page 341\)](#)

RangeCorrel

RangeCorrel() returns the correlation coefficient for two sets of data. The correlation coefficient is a measure of the relationship between the data sets.

Syntax:

```
RangeCorrel(x_value , y_value[, Expression])
```

Return data type: numeric

Data series should be entered as (x,y) pairs. For example, to evaluate two series of data, array 1 and array 2, where the array 1 = 2,6,9 and array 2 = 3,8,4 you would write RangeCorrel (2,3,6,8,9,4) which returns 0.269.

Arguments:

Arguments

Argument	Description
x-value, y-value	Each value represents a single value or a range of values as returned by an inter-record functions with a third optional parameter. Each value or range of values must correspond to an x-value or a range of y-values .
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

The function needs at least two pairs of coordinates to be calculated.

Text values, NULL values and missing values return NULL.

Examples and results:

Function examples

Examples	Results
RangeCorrel (2,3,6,8,9,4,8,5)	Returns 0.2492. This function can be loaded in the script or added into a visualization in the expression editor.

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeList:  
Load * Inline [  
ID1|x1|y1|x2|y2|x3|y3|x4|y4|x5|y5|x6|y6  
01|46|60|70|13|78|20|45|65|78|12|78|22  
02|65|56|22|79|12|56|45|24|32|78|55|15  
03|77|68|34|91|24|68|57|36|44|90|67|27  
04|57|36|44|90|67|27|57|68|47|90|80|94  
] (delimiter is '|');
```

```
XY:  
LOAD recno() as RangeID, * Inline [  
X|Y  
2|3  
6|8  
9|4  
8|5  
](delimiter is '|');
```

In a table with ID1 as a dimension and the measure: RangeCorrel(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6)), the **RangeCorrel()** function finds the value of **Correl** over the range of six x,y pairs, for each of the ID1 values.

Resulting table

ID1	MyRangeCorrel
01	-0.9517
02	-0.5209
03	-0.5209
04	-0.1599

Example:

```
XY:
LOAD recno() as RangeID, * Inline [
X|Y
2|3
6|8
9|4
8|5
](delimiter is '|');
```

In a table with RangeID as a dimension and the measure: RangeCorrel(Below(X,0,4,BelowY,0,4)), the **RangeCorrel()** function uses the results of the **Below()** functions, which because of the third argument (count) set to 4, produce a range of four x-y values from the loaded table XY.

Resulting table

RangeID	MyRangeCorrel2
01	0.2492
02	-0.9959
03	-1.0000
04	-

The value for RangeID 01 is the same as manually entering RangeCorrel(2,3,6,8,9,4,8,5). For the other values of RangeID, the series produced by the Below() function are: (6,8,9,4,8,5), (9,4,8,5), and (8,5), the last of which produces a null result.

See also:

- [Correl - chart function \(page 393\)](#)

RangeCount

RangeCount() returns the number of values, both text and numeric, in the expression or field.

Syntax:

```
RangeCount(first_expr[, Expression])
```

Return data type: integer

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be counted.
Expression	Optional expressions or fields containing the range of data to be counted.

Limitations:

NULL values are not counted.

Examples and results:

Function examples

Examples	Results
RangeCount (1,2,4)	Returns 3
RangeCount (2, 'xyz')	Returns 2
RangeCount (null())	Returns 0
RangeCount (2, 'xyz', null())	Returns 2

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeTab3:  
LOAD recno() as RangeID, RangeCount(Field1,Field2,Field3) as MyRangeCount INLINE [  
Field1, Field2, Field3  
10,5,6  
2,3,7  
8,2,8  
18,11,9  
5,5,9  
9,4,2  
];
```

The resulting table shows the returned values of MyRangeCount for each of the records in the table.

Results table

RangID	MyRangeCount
1	3
2	3
3	3
4	3
5	3
6	3

Example with expression:

`RangeCount (Above(MyField,1,3))`

Returns the number of values contained in the three results of **MyField**. By specifying the first argument of the **Above()** function as 1 and second argument as 3, it returns the values from the first three fields above the current row, where there are sufficient rows, which are taken as input to the **RangeCount()** function.

Data used in examples:

Sample data

MyField	RangeCount(Above(MyField,1,3))
10	0
2	1
8	2
18	3
5	3
9	3

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
2  
8  
18  
5  
9  
] ;
```

See also:

- ❑ [Count - chart function \(page 341\)](#)

RangeFractile

RangeFractile() returns the value that corresponds to the n-th **fractile** (quantile) of a range of numbers.



RangeFractile() uses linear interpolation between closest ranks when calculating the fractile.

Syntax:

```
RangeFractile(fractile, first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
fractile	A number between 0 and 1 corresponding to the fractile (quantile expressed as a fraction) to be calculated.
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeFractile (0.24,1,2,4,6)	Returns 1.72
RangeFractile(0.5,1,2,3,4,6)	Returns 3
RangeFractile (0.5,1,2,5,6)	Returns 3.5

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeTab:  
LOAD recno() as RangeID, RangeFractile(0.5,Field1,Field2,Field3) as MyRangeFrac INLINE [  
Field1, Field2, Field3  
10,5,6  
2,3,7  
8,2,8  
18,11,9  
5,5,9  
9,4,2  
];
```

The resulting table shows the returned values of MyRangeFrac for each of the records in the table.

Resulting table

RangefID	MyRangeFrac
1	6
2	3
3	8
4	11
5	5
6	4

Example with expression:

```
RangeFractile (0.5, Above(Sum(MyField),0,3))
```

In this example, the inter-record function **Above()** contains the optional offset and count arguments. This produces a range of results that can be used as input to the any of the range functions. In this case, `Above(Sum(MyField),0,3)` returns the values of `MyField` for the current row and the two rows above. These values provide the input to the **RangeFractile()** function. So, for the bottom row in the table below, this is the equivalent of `RangeFractile(0.5, 3,4,6)`, that is, calculating the 0.5 fractile for the series 3, 4, and 6. The first two rows in the table below, the number of values in the range is reduced accordingly, where there no rows above the current row. Similar results are produced for other inter-record functions.

Sample data

MyField	RangeFractile(0.5, Above(Sum(MyField),0,3))
1	1
2	1.5
3	2
4	3
5	4
6	5

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
1  
2  
3  
4  
5  
6
```

] ;

See also:

- [Above - chart function \(page 1235\)](#)
- [Fractile - chart function \(page 396\)](#)

RangeIRR

RangeIRR() returns the internal rate of return for a series of cash flows represented by the input values.

The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods.

This function uses a simplified version of the Newton method for calculating the internal rate of return (IRR).

Syntax:

```
RangeIRR(value[, value][, Expression])
```

Return data type: numeric

Arguments

Argument	Description
value	A single value or a range of values as returned by an inter record function with a third optional parameter. The function needs at least one positive and one negative value to be calculated.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

Text values, NULL values and missing values are disregarded.

Example table

Examples	Results
RangeIRR(-70000,12000,15000,18000,21000,26000)	Returns 0.0866

Examples	Results	
Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.	The resulting table shows the returned values of RangeIRR for each of the records in the table.	
RangeTab3: LOAD *, recno() as RangeID, RangeIRR(Field1,Field2,Field3) as RangeIRR; LOAD * INLINE [Field1 Field2 Field3 -10000 5000 6000 -2000 NULL 7000 -8000 'abc' 8000 -1800 11000 9000 -5000 5000 9000 -9000 4000 2000] (delimiter is ' ');	RangeID	RangeIRR
	1	0.0639
	2	0.8708
	3	-
	4	5.8419
	5	0.9318
	6	-0.2566

See also:

- [Inter-record functions \(page 1232\)](#)

RangeKurtosis

RangeKurtosis() returns the value that corresponds to the kurtosis of a range of numbers.

Syntax:

```
RangeKurtosis(first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Function examples

Examples	Results
RangeKurtosis (1,2,4,7)	Returns -0.28571428571429

See also:

- [Kurtosis - chart function \(page 404\)](#)

RangeMax

RangeMax() returns the highest numeric values found within the expression or field.

Syntax:

```
RangeMax(first_expr[, Expression])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Function examples

Examples	Results
RangeMax (1,2,4)	Returns 4
RangeMax (1,'xyz')	Returns 1
RangeMax (null(), 'abc')	Returns NULL

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

RangeTab3:

```
LOAD recno() as RangeID, RangeMax(Field1,Field2,Field3) as MyRangeMax INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

The resulting table shows the returned values of MyRangeMax for each of the records in the table.

Resulting table

RangeID	MyRangeMax
1	10
2	7
3	8
4	18
5	9
6	9

Example with expression:

```
RangeMax (Above(MyField,0,3))
```

Returns the maximum value in the range of three values of **MyField** calculated on the current row and two rows above the current row. By specifying the third argument as 3, the **Above()** function returns three values, where there are sufficient rows above, which are taken as input to the **RangeMax()** function.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeMax (Above(Sum(MyField),1,3))
10	10
2	10
8	10
18	18
5	18
9	18

Data used in examples:

RangeTab:

```
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

RangeMaxString

RangeMaxString() returns the last value in the text sort order that it finds in the expression or field.

Syntax:

```
RangeMaxString(first_expr[, Expression])
```

Return data type: string

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeMaxString (1,2,4)	Returns 4
RangeMaxString ('xyz','abc')	Returns 'xyz'
RangeMaxString (5,'abc')	Returns 'abc'
RangeMaxString (null())	Returns NULL

Example with expression:

```
RangeMaxString (Above(MaxString(MyField),0,3))
```

Returns the last (in text sort order) of the three results of the **MaxString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
10	10
abc	abc
8	abc
def	def
xyz	xyz
9	xyz

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
'abc'  
8  
'def'  
'xyz'  
9  
] ;
```

See also:

□ [MaxString - chart function \(page 517\)](#)

RangeMin

RangeMin() returns the lowest numeric values found within the expression or field.

Syntax:

```
RangeMin(first_expr[, Expression])
```

Return data type: numeric

Arguments:

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Function examples

Examples	Results
RangeMin (1,2,4)	Returns 1
RangeMin (1,'xyz')	Returns 1
RangeMin (null(), 'abc')	Returns NULL

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeTab3:
LOAD recno() as RangeID, RangeMin(Field1,Field2,Field3) as MyRangeMin INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

The resulting table shows the returned values of MyRangeMin for each of the records in the table.

Resulting table

RangeID	MyRangeMin
1	5
2	2
3	2
4	9
5	5
6	2

Example with expression:

```
RangeMin (Above(MyField,0,3)
```

Returns the minimum value in the range of three values of **MyField** calculated on the current row and two rows above the current row. By specifying the third argument as 3, the **Above()** function returns three values, where there are sufficient rows above, which are taken as input to the **RangeMin()** function.

Data used in examples:

Sample data

MyField	RangeMin(Above(MyField,0,3))
10	10
2	2
8	2
18	2
5	5
9	5

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
2  
8  
18  
5  
9  
] ;
```

See also:

□ [Min - chart function \(page 328\)](#)

RangeMinString

RangeMinString() returns the first value in the text sort order that it finds in the expression or field.

Syntax:

```
RangeMinString(first_expr[, Expression])
```

Return data type: string

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeMinString (1,2,4)	Returns 1
RangeMinString ('xyz','abc')	Returns 'abc'
RangeMinString (5,'abc')	Returns 5
RangeMinString (null())	Returns NULL

Example with expression:

```
RangeMinString (Above(MinString(MyField),0,3))
```

Returns the first (in text sort order) of the three results of the **MinString(MyField)** function evaluated on the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeMinString(Above(MinString(MyField),0,3))
10	10
abc	10
8	8
def	8
xyz	8
9	9

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
'abc'  
8  
'def'  
'xyz'  
9  
];
```

See also:

- [MinString - chart function \(page 520\)](#)

RangeMissingCount

RangeMissingCount() returns the number of non-numeric values (including NULL) in the expression or field.

Syntax:

```
RangeMissingCount(first_expr[, Expression])
```

Return data type: integer

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be counted.
Expression	Optional expressions or fields containing the range of data to be counted.

Examples and results:

Function examples

Examples	Results
RangeMissingCount (1,2,4)	Returns 0
RangeMissingCount (5, 'abc')	Returns 1
RangeMissingCount (null())	Returns 1

Example with expression:

```
RangeMissingCount (Above(MinString(MyField),0,3))
```

Returns the number of non-numeric values in the three results of the **MinString(MyField)** function evaluated on the current row and two rows above the current row.



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeMissingCount (Above(MinString (MyField),0,3))	Explanation
10	2	Returns 2 because there are no rows above this row so 2 of the 3 values are missing.
abc	2	Returns 2 because there is only 1 row above the current row and the current row is non-numeric ('abc').
8	1	Returns 1 because 1 of the 3 rows includes a non-numeric ('abc').
def	2	Returns 2 because 2 of the 3 rows include non-numeric values ('def' and 'abc').
xyz	2	Returns 2 because 2 of the 3 rows include non-numeric values ('xyz' and 'def').
9	2	Returns 2 because 2 of the 3 rows include non-numeric values ('xyz' and 'def').

Data used in examples:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

See also:

 [MissingCount - chart function \(page 344\)](#)

RangeMode

RangeMode() finds the most commonly occurring value (mode value) in the expression or field.

Syntax:

```
RangeMode(first_expr {, Expression})
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If more than one value shares the highest frequency, NULL is returned.

Examples and results:

Function examples

Examples	Results
RangeMode (1,2,9,2,4)	Returns 2
RangeMode ('a',4,'a',4)	Returns NULL
RangeMode (null())	Returns NULL

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

```
RangeTab3:
LOAD recno() as RangeID, RangeMode(Field1,Field2,Field3) as MyRangeMode INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

The resulting table shows the returned values of **MyRangeMode** for each of the records in the table.

Results table

RangeID	MyRangMode
1	-
2	-
3	8
4	-
5	5
6	-

Example with expression:

```
RangeMode (Above(MyField,0,3))
```

Returns the most commonly occurring value in the three results of **MyField** evaluated on the current row and two rows above the current row. By specifying the third argument as 3, the **Above()** function returns three values, where there are sufficient rows above, which are taken as input to the **RangeMode()** function.

Data used in example:

RangeTab:

```
LOAD * INLINE [  
    MyField  
    10  
    2  
    8  
    18  
    5  
    9  
];
```



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeMode(Above(MyField,0,3))
10	Returns 10 because there are no rows above so the single value is the most commonly occurring.
2	-
8	-
18	-
5	-
9	-

See also:

- [Mode - chart function \(page 331\)](#)

RangeNPV

RangeNPV() returns the net present value of an investment based on a discount rate and a series of future periodic payments (negative values) and incomes (positive values). The result has a default number format of **money**.

For cash flows that are not necessarily periodic, see *RangeXNPV (page 1327)*.

Syntax:

```
RangeNPV(discount_rate, value[,value] [, Expression])
```

Return data type: numeric

Arguments

Argument	Description
discount_rate	The interest rate per period.
value	A payment or income occurring at the end of each period. Each value may be a single value or a range of values as returned by an inter-record function with a third optional parameter.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

Text values, NULL values and missing values are disregarded.

Examples	Results														
<pre>RangeNPV(0.1,-10000,3000,4200,6800)</pre> <p>Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.</p> <pre>RangeTab3: LOAD *, recno() as RangeID, RangeNPV(Field1,Field2,Field3) as RangeNPV; LOAD * INLINE [Field1 Field2 Field3 10 5 -6000 2 NULL 7000 8 'abc' 8000 18 11 9000 5 5 9000 9 4 2000] (delimiter is ' ');</pre>	<p>Returns 1188.44</p> <p>The resulting table shows the returned values of RangeNPV for each of the records in the table.</p> <table> <thead> <tr> <th>RangeID</th> <th>RangeNPV</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>\$-49.13</td> </tr> <tr> <td>2</td> <td>\$777.78</td> </tr> <tr> <td>3</td> <td>\$98.77</td> </tr> <tr> <td>4</td> <td>\$25.51</td> </tr> <tr> <td>5</td> <td>\$250.83</td> </tr> <tr> <td>6</td> <td>\$20.40</td> </tr> </tbody> </table>	RangeID	RangeNPV	1	\$-49.13	2	\$777.78	3	\$98.77	4	\$25.51	5	\$250.83	6	\$20.40
RangeID	RangeNPV														
1	\$-49.13														
2	\$777.78														
3	\$98.77														
4	\$25.51														
5	\$250.83														
6	\$20.40														

See also:

- [Inter-record functions \(page 1232\)](#)

RangeNullCount

RangeNullCount() finds the number of NULL values in the expression or field.

Syntax:

```
RangeNullCount(first_expr [, Expression])
```

Return data type: integer

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeNullCount (1,2,4)	Returns 0
RangeNullCount (5,'abc')	Returns 0
RangeNullCount (null(), null())	Returns 2

Example with expression:

`RangeNullCount (Above(Sum(MyField),0,3))`

Returns the number of NULL values in the three results of the **Sum(MyField)** function evaluated on the current row and two rows above the current row.



Copying **MyField** in example below will not result in NULL value.

Sample data

MyField	RangeNullCount(Above(Sum(MyField),0,3))
10	Returns 2 because there are no rows above this row so 2 of the 3 values are missing (=NULL).
'abc'	Returns 1 because there is only one row above the current row, so one of the three values is missing (=NULL).
8	Returns 0 because none of the three rows is a NULL value.

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
'abc'  
8  
] ;
```

See also:

- ❑ [NullCount - chart function \(page 347\)](#)

RangeNumericCount

RangeNumericCount() finds the number of numeric values in an expression or field.

Syntax:

```
RangeNumericCount(first_expr[, Expression])
```

Return data type: integer

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeNumericCount (1,2,4)	Returns 3
RangeNumericCount (5, 'abc')	Returns 1
RangeNumericCount (null())	Returns 0

Example with expression:

```
RangeNumericCount (Above(MaxString(MyField),0,3))
```

Returns the number of numeric values in the three results of the **MaxString(MyField)** function evaluated on the current row and two rows above the current row.



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
10	1

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
abc	1
8	2
def	1
xyz	1
9	1

Data used in examples:

RangeTab:
 LOAD * INLINE [
 MyField
 10
 'abc'
 8
 def
 xyz
 9
] ;

See also:

□ [NumericCount - chart function \(page 350\)](#)

RangeOnly

RangeOnly() is a dual function that returns a value if the expression evaluates to one unique value. If this is not the case then **NULL** is returned.

Syntax:

```
RangeOnly(first_expr[, Expression])
```

Return data type: dual

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Examples	Results
RangeOnly (1,2,4)	Returns NULL
RangeOnly (5,'abc')	Returns NULL
RangeOnly (null(), 'abc')	Returns 'abc'
RangeOnly(10,10,10)	Returns 10

See also:

□ [Only - chart function \(page 334\)](#)

RangeSkew

RangeSkew() returns the value corresponding to the skewness of a range of numbers.

Syntax:

```
RangeSkew(first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:**Function examples**

Examples	Results
rangeskew (1,2,4)	Returns 0.93521952958283
rangeskew (above (salesvalue,0,3))	Returns a sliding skewness of the range of three values returned from the above() function calculated on the current row and the two rows above the current row.

Data used in example:

Sample data

CustID	RangeSkew(Above(SalesValue,0,3))
1-20	-, -, 0.5676, 0.8455, 1.0127, -0.8741, 1.7243, -1.7186, 1.5518, 1.4332, 0, 1.1066, 1.3458, 1.5636, 1.5439, 0.6952, -0.3766

```
Salestable:
LOAD recno() as CustID, * inline [
Salesvalue
101
163
126
139
167
86
83
22
32
70
108
124
176
113
95
32
42
92
61
21
] ;
```

See also:

- [Skew - chart function \(page 433\)](#)

RangeStdev

RangeStdev() finds the standard deviation of a range of numbers.

Syntax:

```
RangeStdev(first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

If no numeric value is found, NULL is returned.

Examples and results:

Function examples

Examples	Results
RangeStdev (1,2,4)	Returns 1.5275252316519
RangeStdev (null())	Returns NULL
RangeStdev (above (Salesvalue),0,3))	Returns a sliding standard of the range of three values returned from the above() function calculated on the current row and the two rows above the current row.

Data used in example:

Sample data

CustID	RangeStdev(SalesValue, 0,3))
1-20	-,43.841, 34.192, 18.771, 20.953, 41.138, 47.655, 36.116, 32.716, 25.325, 38,000, 27.737, 35.553, 33.650, 42.532, 33.858, 32.146, 25.239, 35.595

SalesTable:

```
LOAD recno() as CustID, * inline [
Salesvalue
101
163
126
139
167
86
83
22
32
70
108
124
176
113
95
32
42
92
61
21
```

] ;

See also:

- [Stdev - chart function \(page 436\)](#)

RangeSum

RangeSum() returns the sum of a range of values. All non-numeric values are treated as 0.

Syntax:

```
RangeSum(first_expr[, Expression])
```

Return data type: numeric

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Arguments

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Limitations:

The **RangeSum** function treats all non-numeric values as 0.

Examples and results:

Examples

Examples	Results
RangeSum (1,2,4)	Returns 7
RangeSum (5,'abc')	Returns 5
RangeSum (null())	Returns 0

Example:

Add the example script to your app and run it. To see the result, add the fields listed in the results column to a sheet in your app.

RangeTab3:

```
LOAD recno() as RangeID, Rangesum(Field1,Field2,Field3) as MyRangeSum INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
```

```
5,5,9
9,4,2
];
```

The resulting table shows the returned values of MyRangeSum for each of the records in the table.

Resulting table

RangeID	MyRangeSum
1	21
2	12
3	18
4	38
5	19
6	15

Example with expression:

```
RangeSum (Above(MyField,0,3))
```

Returns the sum of the three values of **MyField**: from the current row and two rows above the current row. By specifying the third argument as 3, the **Above()** function returns three values, where there are sufficient rows above, which are taken as input to the **RangeSum()** function.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Sample data

MyField	RangeSum(Above(MyField,0,3))
10	10
2	12
8	20
18	28
5	31
9	32

Data used in examples:

```
RangeTab:  
LOAD * INLINE [  
MyField  
10  
2  
8  
18
```

```
5  
9  
] ;
```

See also:

- [Sum - chart function \(page 336\)](#)
- [Above - chart function \(page 1235\)](#)

RangeTextCount

RangeTextCount() returns the number of text values in an expression or field.

Syntax:

```
RangeTextCount(first_expr[, Expression])
```

Return data type: integer

Arguments:

The arguments of this function may contain inter-record functions which in themselves return a list of values.

Argument

Argument	Description
first_expr	The expression or field containing the data to be measured.
Expression	Optional expressions or fields containing the range of data to be measured.

Examples and results:

Function examples

Examples	Results
RangeTextCount (1,2,4)	Returns 0
RangeTextCount (5,'abc')	Returns 1
RangeTextCount (null())	Returns 0

Example with expression:

```
RangeTextCount (Above(MaxString(MyField),0,3))
```

Returns the number of text values within the three results of the **MaxString(MyField)** function evaluated over the current row and two rows above the current row.

Data used in examples:



*Disable sorting of **MyField** to ensure that the example works as expected.*

Example data

MyField	MaxString(MyField)	RangeTextCount(Above(Sum(MyField),0,3))
10	10	0
abc	abc	1
8	8	1
def	def	2
xyz	xyz	2
9	9	2

Data used in examples:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
null()
'xyz'
9
] ;
```

See also:

- [TextCount - chart function \(page 353\)](#)

RangeXIRR

RangeXIRR() returns the internal rate of return (yearly) for a schedule of cash flows that is not necessarily periodic. To calculate the internal rate of return for a series of periodic cash flows, use the **RangeIRR** function.

Qlik's XIRR functionality (**XIRR()** and **RangeXIRR()** functions) uses the following equation, solving for the Rate value, to determine the correct XIRR value:

```
XNPV(Rate, pmt, date) = 0
```

The equation is solved using a simplified version of the Newton method.

Syntax:

```
RangeXIRR(value, date{, value, date})
```

Return data type: numeric

Arguments

Argument	Description
value	A cash flow or a series of cash flows that correspond to a schedule of payments in dates. The series of values must contain at least one positive and one negative value.
date	A payment date or a schedule of payment dates that corresponds to the cash flow payments.

When working with this function, the following limitations apply:

- Text values, NULL values and missing values are disregarded.
- All payments are discounted based on a 365-day year.
- This function requires at least one valid negative and at least one valid positive payment (with corresponding valid dates). If these payments are not provided, a NULL value is returned.

The following topics might help you work with this function:

- *RangeXNPV* (page 1327): Use this function calculate the net present value for a schedule of cash flows that is not necessarily periodic.
- *XIRR* (page 367): The **XIRR()** function calculates the aggregated internal rate of return (yearly) for a schedule of cash flows (that is not necessarily periodic).



Across different versions of Qlik Sense Client-Managed, there are variations in the underlying algorithm used by this function. For more information about recent updates to the algorithm, see support article [XIRR function Fix and Update](#).

Examples and results:

Examples and results

Examples	Results
<code>RangeXIRR(-2500, '2008-01-01', 2750, '2008-09-01')</code>	Returns 0.1532

See also:

- [RangeIRR](#) (page 1302)
- [RangeXNPV](#) (page 1327)
- [XIRR](#) (page 367)
- ↗ [XIRR function Fix and Update](#)

RangeXNPV

RangeXNPV() returns the net present value for a schedule of cash flows (not necessarily periodic) represented by paired numbers in the expressions given by **pmt** and **date**. All payments are discounted based on a 365-day year.

Syntax:

```
RangeXNPV(discount_rate, values, dates[, Expression])
```

Return data type: numeric

Arguments	
Argument	Description
discount_rate	discount_rate is the yearly rate that the payments should be discounted by.
values	A cash flow or a series of cash flows that corresponds to a schedule of payments in dates. Each value may be a single value or a range of values as returned by an inter-record function with a third optional parameter. The series of values must contain at least one positive and one negative value.
dates	A payment date or a schedule of payment dates that corresponds to the cash flow payments.

When working with this function, the following limitations apply:

- Text values, NULL values and missing values are disregarded.
- All payments are discounted based on a 365-day year.

Example - script

Load script and results

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Financial data contained in a table called RangeTab3.
- The use of the **RangeXNPV()** function to compute net present value.

Load script

```
RangeTab3:  
LOAD *,  
recno() as RangeID,  
RangeXNPV(DiscountRate,Value1,Date1,Value2,Date2) as RangeXNPV;  
LOAD * INLINE [
```

```
DiscountRate|value1|Date1|value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

Results

Load the data and open a sheet. Create a new table and add these fields as dimensions:

- RangeID
- RangeXNPV

Results table

RangeID	RangeXNPV
1	-\$9.09
2	-\$0.00
3	\$13.64

Example - chart expression

Load script and chart expression

Overview

Open the Data load editor and add the load script below to a new tab.

The load script contains:

- Financial data contained in a table called RangeTab3.
- The use of the **RangeXNPV()** function to compute net present value.

Load script

```
RangeTab3:
LOAD *,
recno() as RangeID,
RangeXNPV(DiscountRate,value1,Date1,value2,Date2) as RangeXNPV;
LOAD * INLINE [
DiscountRate|value1|Date1|value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

Results

Do the following:

Load the data and open a sheet. Create a new table and add the following calculation as a measure:

```
=RangeXNPV(0.1, -2500,'2008-01-01',2750,'2008-09-01')
```

Results table

=XIRR(Payments, Date)
\$80.25

See also:

 [XNPV \(page 373\)](#)

5.22 Relational functions

This is a group of functions that calculate properties of individual dimensional values in a chart, using already aggregated numbers.

The functions are relational in the sense that the function output depends not only on the value of the data point itself, but also on the value's relation to other data points. For example, a rank cannot be calculated without a comparison with other dimensional values.

These functions can only be used in chart expressions. They cannot be used in the load script.

A dimension is needed in the chart, since this defines the other data points needed for the comparison. Consequently, a relational function is not meaningful in a dimensionless chart (for example, a KPI object).

Ranking functions



Suppression of zero values is automatically disabled when these functions are used. NULL values are disregarded.

Rank

Rank() evaluates the rows of the chart in the expression, and for each row, displays the relative position of the value of the dimension evaluated in the expression. When evaluating the expression, the function compares the result with the result of the other rows containing the current column segment and returns the ranking of the current row within the segment.

```
Rank - chart function([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

HRank

HRank() evaluates the expression, and compares the result with the result of the other columns containing the current row segment of a pivot table. The function then returns the ranking of the current column within the segment.

```
HRank - chart function([TOTAL] expr[, mode[, fmt]])
```

Clustering functions

KMeans2D

The property group **Site license** contains properties related to the license for the Qlik Sense system. All fields are mandatory and must not be empty.

Site licence properties

Property name	Description
Owner name	The user name of the Qlik Sense product owner.
Owner organization	The name of the organization that the Qlik Sense product owner is a member of.
Serial number	The serial number assigned to the Qlik Sense software.
Control number	The control number assigned to the Qlik Sense software.
LEF access	The License Enabler File (LEF) assigned to the Qlik Sense software.

KMeans2D() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the cluster id of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, respectively. These are both aggregations. The number of clusters that are created is determined by the num_clusters parameter. Data can be optionally normalized by the norm parameter.

```
KMeans2D - chart function(num_clusters, coordinate_1, coordinate_2 [, norm])
```

KMeansND

KMeansND() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the cluster id of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, etc., up to n columns. These are all aggregations. The number of clusters that are created is determined by the num_clusters parameter.

```
KMeansND - chart function(num_clusters, num_iter, coordinate_1, coordinate_2 [, coordinate_3 [, ...]])
```

KMeansCentroid2D

KMeansCentroid2D() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the desired coordinate of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, respectively. These are both aggregations. The number of clusters that are created is determined by the num_clusters parameter. Data can be optionally normalized by the norm parameter.

```
KMeansCentroid2D - chart function(num_clusters, coordinate_no, coordinate_1,  
coordinate_2 [, norm])
```

KMeansCentroidND

KMeansCentroidND() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the desired coordinate of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, coordinate_2, etc., up to n columns. These are all aggregations. The number of clusters that are created is determined by the num_clusters parameter.

```
KMeansCentroidND - chart function(num_clusters, num_iter, coordinate_no,  
coordinate_1, coordinate_2 [, coordinate_3 [, ...]])
```

Time series decomposition functions

STL_Trend

STL_Trend is a time series decomposition function. Along with **STL_Seasonal** and **STL_Residual**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. The **STL_Trend** function will identify a general trend, independent of seasonal patterns or cycles, from time series data.

```
STL_Trend - chart function(target_measure, period_int [,seasonal_smoker  
[,trend_smoker]])
```

STL_Seasonal

STL_Seasonal is a time series decomposition function. Along with **STL_Trend** and **STL_Residual**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. The **STL_Seasonal** function can identify a seasonal pattern within a time series, separating this from the general trend displayed by the data.

```
STL_Seasonal - chart function(target_measure, period_int [,seasonal_smoker  
[,trend_smoker]])
```

STL_Residual

STL_Residual is a time series decomposition function. Along with **STL_Seasonal** and **STL_Trend**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. In performing this operation, part of the variation in the input metric will neither fit within the seasonal nor the trend component, and will be defined as the residual component. The **STL_Residual** chart function captures this portion of the calculation.

```
STL_Residual - chart function(target_measure, period_int [,seasonal_smoker  
[,trend_smoker]])
```

Rank - chart function

Rank() evaluates the rows of the chart in the expression, and for each row, displays the relative position of the value of the dimension evaluated in the expression. When evaluating the expression, the function compares the result with the result of the other rows containing the current column segment and returns the ranking of the current row within the segment.

Column segments

	Region	Country	Population	Rank(Population)
Column segment #1	Americas	Mexico	128,932,771	2
	Americas	Canada	37,742,154	3
Column segment #2	Americas	United States of America	331,002,651	1
	Europe	Sweden	10,099,265	4
Europe	Europe	United Kingdom	67,886,011	2
	Europe	France	65,273,511	3
Europe	Europe	Germany	83,783,942	1

For charts other than tables, the current column segment is defined as it appears in the chart's straight table equivalent.

Syntax:

```
Rank( [TOTAL] expr[, mode[, fmt]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
mode	Specifies the number representation of the function result.
fmt	Specifies the text representation of the function result.
TOTAL	If the chart is one-dimensional, or if the expression is preceded by the TOTAL qualifier, the function is evaluated along the entire column. If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns except for the column showing the last dimension in the inter-field sort order.

The ranking is returned as a dual value, which in the case when each row has a unique ranking, is an integer between 1 and the number of rows in the current column segment.

In the case where several rows share the same ranking, the text and number representation can be controlled with the **mode** and **fmt** parameters.

mode

The second argument, **mode**, can take the following values:

mode examples

Value	Description
0 (default)	If all ranks within the sharing group fall on the low side of the middle value of the entire ranking, all rows get the lowest rank within the sharing group. If all ranks within the sharing group fall on the high side of the middle value of the entire ranking, all rows get the highest rank within the sharing group. If ranks within the sharing group span over the middle value of the entire ranking, all rows get the value corresponding to the average of the top and bottom ranking in the entire column segment.
1	Lowest rank on all rows.
2	Average rank on all rows.
3	Highest rank on all rows.
4	Lowest rank on first row, then incremented by one for each row.

fmt

The third argument, **fmt**, can take the following values:

fmt examples

Value	Description
0 (default)	Low value - high value on all rows (for example 3 - 4).
1	Low value on all rows.
2	Low value on first row, blank on the following rows.

The order of rows for **mode** 4 and **fmt** 2 is determined by the sort order of the chart dimensions.

Examples and results:

Create two visualizations from the dimensions Product and Sales and another from Product and UnitSales. Add measures as shown in the following table.

Rank examples

Examples	Results
Example 1. Create a table with the dimensions Customer and Sales and the measure Rank(Sales)	<p>The result depends on the sort order of the dimensions. If the table is sorted on Customer, the table lists all the values of Sales for Astrida, then Betocab, and so on. The results for Rank(Sales) will show 10 for the Sales value 12, 9 for the Sales value 13, and so on, with the rank value of 1 returned for the Sales value 78. The next column segment begins with Betocab, for which the first value of Sales in the segment is 12. The rank value of Rank(Sales) for this is given as 11.</p> <p>If the table is sorted on Sales, the column segments consist of the values of Sales and the corresponding Customer. Because there are two Sales values of 12 (for Astrida and Betocab), the value of Rank(Sales) for that column segment is 1-2, for each value of Customer. This is because there are two values of Customer for the Sales value 12. If there had been 4 values, the result would be 1-4, for all rows. This shows what the result looks like for the default value (0) of the argument fmt.</p>
Example 2. Replace the dimension Customer with Product and add the measure Rank(Sales,1,2)	This returns 1 on the first row on each column segment and leaves all other rows blank, because arguments mode and fmt are set to 1 and 2 respectively.

Results for example 1, with table sorted on Customer:

Results table

Customer	Sales	Rank(Sales)
Astrida	12	10
Astrida	13	9
Astrida	20	8
Astrida	22	7
Astrida	45	6
Astrida	46	5
Astrida	60	4
Astrida	65	3
Astrida	70	2
Astrida	78	1
Betocab	12	11

Results for example 1, with table sorted on Sales:

Results table

Customer	Sales	Rank(Sales)
Astrida	12	1-2
Betacab	12	1-2
Astrida	13	1
Betacab	15	1
Astrida	20	1
Astrida	22	1-2
Betacab	22	1-2
Betacab	24	1-2
Canutility	24	1-2

Data used in examples:

```
ProductData:
Load * inline [
Customer|Product|unitsales|unitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|0|25
Canutility|AA|8|15
Canutility|CC|0|19
] (delimiter is '|');

Sales2013:
crosstable (Month, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

See also:

-  [Sum - chart function \(page 336\)](#)

HRank - chart function

HRank() evaluates the expression, and compares the result with the result of the other columns containing the current row segment of a pivot table. The function then returns the ranking of the current column within the segment.

Syntax:

```
HRank([ TOTAL ] expr [ , mode [, fmt ] ])
```

Return data type: dual



This function only works in pivot tables. In all other chart types it returns NULL.

Arguments:

Arguments

Argument	Description
expr	The expression or field containing the data to be measured.
mode	Specifies the number representation of the function result.
fmt	Specifies the text representation of the function result.
TOTAL	If the chart is one-dimensional, or if the expression is preceded by the TOTAL qualifier, the function is evaluated along the entire column. If the table or table equivalent has multiple vertical dimensions, the current column segment will include only rows with the same values as the current row in all dimension columns except for the column showing the last dimension in the inter-field sort order.

If the pivot table is one-dimensional or if the expression is preceded by the **total** qualifier, the current row segment is always equal to the entire row. If the pivot table has multiple horizontal dimensions, the current row segment will include only columns with the same values as the current column in all dimension rows except for the row showing the last horizontal dimension of the inter-field sort order.

The ranking is returned as a dual value, which in the case when each column has a unique ranking will be an integer between 1 and the number of columns in the current row segment.

In the case where several columns share the same ranking, the text and number representation can be controlled with the **mode** and **format** arguments.

The second argument, **mode**, specifies the number representation of the function result:

mode examples

Value	Description
0 (default)	If all ranks within the sharing group fall on the low side of the middle value of the entire ranking, all columns get the lowest rank within the sharing group. If all ranks within the sharing group fall on the high side of the middle value of the entire ranking, all columns get the highest rank within the sharing group. If ranks within the sharing group span over the middle value of the entire ranking, all rows get the value corresponding to the average of the top and bottom ranking in the entire column segment.
1	Lowest rank on all columns in the group.
2	Average rank on all columns in the group.
3	Highest rank on all columns in the group.
4	Lowest rank on first column, then incremented by one for each column in the group.

The third argument, **format**, specifies the text representation of the function result:

format examples

Value	Description
0 (default)	Low value&' - &high value on all columns in the group (for example 3 - 4).
1	Low value on all columns in the group.
2	Low value on first column, blank on the following columns in the group.

The order of columns for **mode** 4 and **format** 2 is determined by the sort order of the chart dimensions.

Examples:

```
HRank( sum( Sales ))
HRank( sum( Sales ), 2 )
HRank( sum( Sales ), 0, 1 )
```

Optimizing with k-means: A real-world example

The following example illustrates a real world use case where the KMeans clustering and Centroid functions are applied to a dataset. The KMeans function segregates data points into clusters that share similarities. The clusters become more compact and differentiated as the KMeans algorithm is applied over a configurable number of iterations.

KMeans is used across many fields in a wide variety of use cases; some examples of clustering use cases include customer segmentation, fraud detection, predicting account attrition, targeting client incentives, cybercrime identification, and delivery route optimization. The KMeans clustering algorithm is increasingly being used where enterprises are trying to infer patterns and optimize service offerings.

Qlik Sense KMeans and Centroid functions

Qlik Sense provides two KMeans functions that group data points into clusters based on similarity. See *KMeans2D - chart function (page 1346)* and *KMeansND - chart function (page 1361)*. The **KMeans2D** function accepts two dimensions and works well for visualizing results through a **scatter plot** chart. The **KMeansND** function accepts more than two dimensions. As it is easy to conceptualize a 2D outcome on standard charts, the following demonstration applies KMeans on a **scatter plot** chart using two dimensions. KMeans clustering can be visualized through coloring by expression; or by dimension as described in this example.

Qlik Sense centroid functions determine the arithmetic mean position of all the data points in the cluster and identify a central point, or centroid for that cluster. For each chart row (or record), the centroid function displays the coordinate of the cluster this data point has been assigned to. See *KMeansCentroid2D - chart function (page 1376)* and *KMeansCentroidND - chart function (page 1377)*.

Use case and example overview

The following example stages through a simulated real world scenario. A textile company in New York state, USA, must decrease expenses by minimizing delivery costs. One way to do that is to relocate warehouses closer to their distributors. The company employs 118 distributors across the state of New York. The following demonstration simulates how an operations manager could segment distributors into five clustered geographies using the KMeans function and then identify five optimal warehouse locations central to those clusters using the centroid function. The objective is to discover mapping coordinates that can be used to identify five central warehouse locations.

The dataset

The dataset is based on randomly generated names and addresses in New York state with real latitude and longitude coordinates. The dataset contains the following ten columns: id, first_name, last_name, telephone, address, city, state, zip, latitude, longitude. The dataset is available below as a file you can download locally and then upload to Qlik Sense or inline for data load editor. The app being created is named *Distributors KMeans and Centroid* and the first sheet in the app is named *Distribution cluster analysis*.

Select the following link to download the sample data file: [DistributorData.csv](#)

Distributor dataset: Inline load for data load editor in Qlik Sense (page 1344)

Title: DistributorData

Total number of records: 118

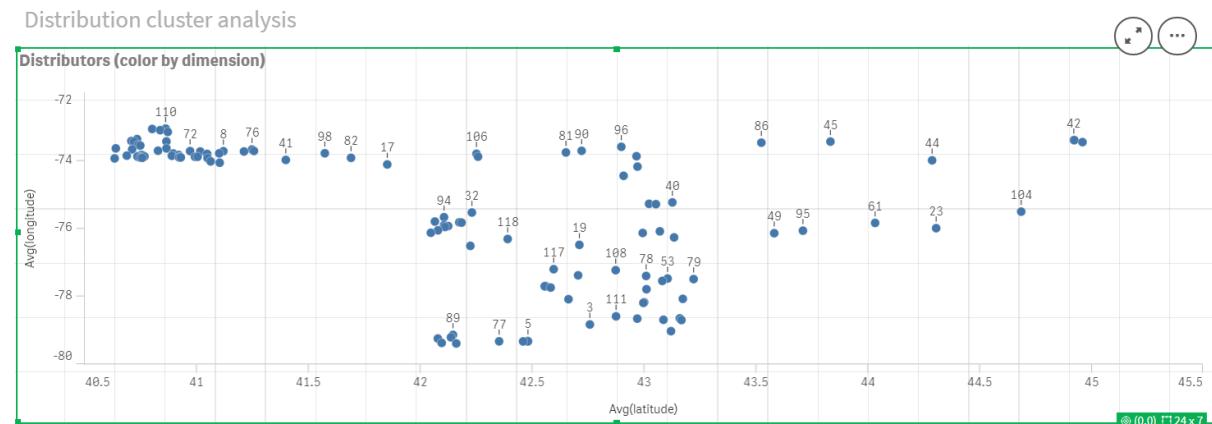
Applying the KMeans2D function

In this example, configuration of a **scatter plot** chart is demonstrated using the *DistributorData* dataset, the **KMeans2D** function is applied, and the chart is colored by dimension.

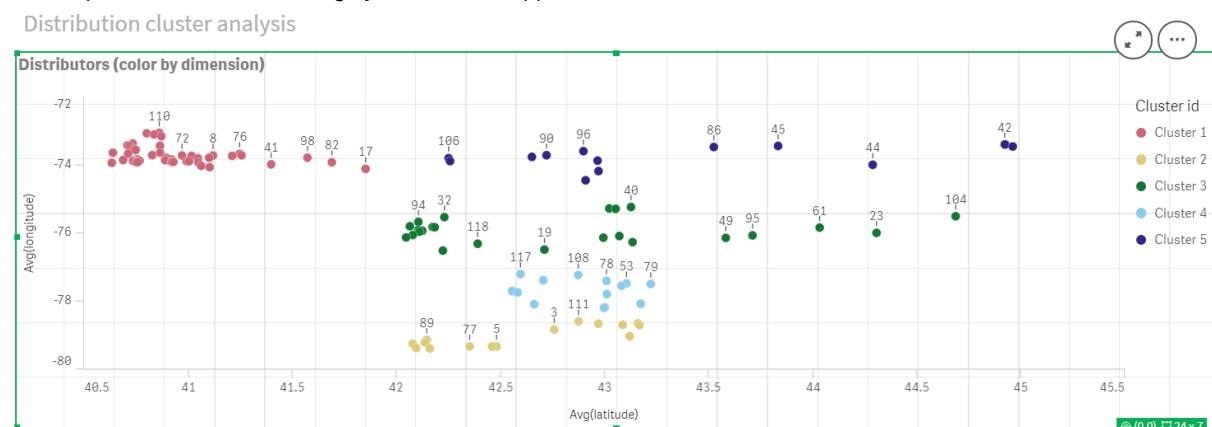
Note that Qlik Sense KMeans functions support auto-clustering using a method called depth difference (DeD). When a user sets 0 for the number of clusters, the optimal number of clusters for that dataset is determined. For this example however, a variable is created for the **num_clusters** argument (refer to *KMeans2D - chart function (page 1346)* for syntax). Therefore, the desired number of clusters (k=5) is specified by a variable.

1. A **scatter plot** chart is dragged onto the sheet and named *Distributors (by dimension)*.
2. A **variable** is created to specify the number of clusters. The **variable** is named *vDistClusters*. For the variable **Definition**, 5 is entered.
3. **Data** configuration for the chart:
 - a. Under **Dimensions**, *id* field is selected for **Bubble**. *Cluster id* is entered for the **Label**.
 - b. Under **Measures**, *Avg([latitude])* is the expression for **X-axis**.
 - c. Under **Measures**, *Avg([longitude])* is the expression for **Y-axis**.
4. **Appearance** configuration:
 - a. Under **Colors and legend**, **Custom** is chosen for **Colors**.
 - b. **By dimension** is selected for coloring the chart.
 - c. The following expression is entered: `=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')`
 - d. The checkbox for **Persistent colors** is selected.

Scatter plot before KMeans coloring by dimension is applied



Scatter plot after KMeans coloring by dimension is applied



Adding a **table**: *Distributors*

It can be helpful to have a table handy for quick access to relevant data. The **scatter plot** chart shows *ids* through a table with corresponding distributor names is added for reference.

1. A **table** named *Distributors* is dragged onto the sheet with the following **Columns** (Dimensions) added: *id*, *first_name*, and *last_name*.

Table: Distributor names

Distributors			
	id	first_name	last_name
	1	Kaiya	Snow
	2	Dean	Roy
	3	Eden	Paul
	4	Bryanna	Higgins
	5	Elisabeth	Lee
	6	Skylar	Robinson
	7	Cody	Bailey
	8	Dario	Sims
	9	Deacon	Hood

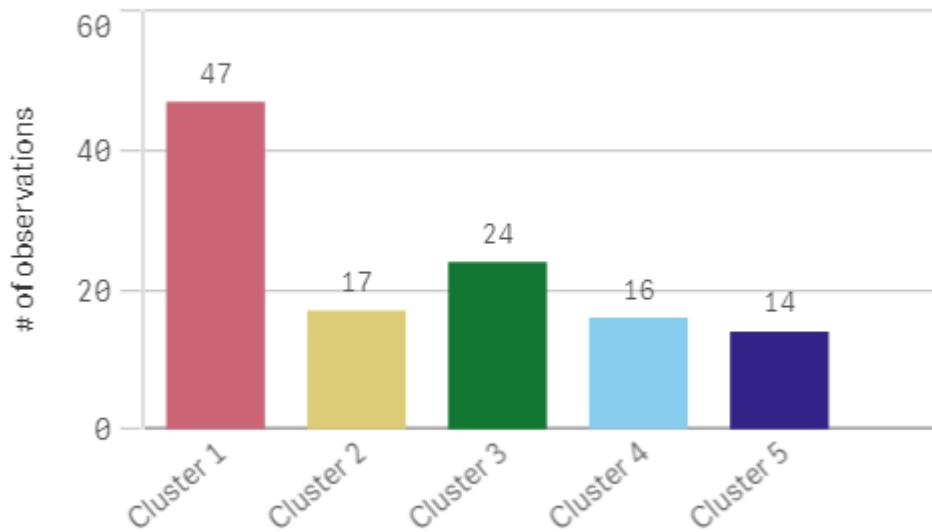
Adding a **bar chart**: # observations per cluster

For the warehouse distribution scenario, it is helpful to know how many distributors will be served by each warehouse. Therefore, a **bar chart** is created that measures how many distributors are assigned to each cluster.

1. A **bar chart** is dragged onto the sheet. The chart is named: *# observations per cluster*.
2. **Data** configuration for the **bar chart**:
 - a. A **Dimension** labeled *Clusters* is added (the label can be added after the expression is applied). The following expression is entered: `=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')`
 - b. A **Measure** labeled *# of observations* is added. The following expression is entered: `=count(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id))`
3. **Appearance** configuration:
 - a. Under **Colors and legend**, **Custom** is chosen for **Colors**.
 - b. **By dimension** is selected for coloring the chart.
 - c. The following expression is entered: `=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')`
 - d. The checkbox for **Persistent colors** is selected.
 - e. **Show legend** is turned off.
 - f. Under **Presentation**, **Value labels** is toggled to **Auto**.
 - g. Under **X-axis: Clusters**, **Labels only** is selected.

Bar chart: # observations per cluster

observations per cluster



Applying the **Centroid2D** function

A second table is added for the **Centroid2D** function that will identify the coordinates for potential warehouse locations. This table shows the central location (centroid values) for the five identified distributor groups.

1. A **Table** is dragged onto the sheet and named *Cluster centroids* with the following columns added:
 - a. A **Dimension** labeled *Clusters* is added. The following expression is entered:
`=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1,'Warehouse 1','Warehouse 2','Warehouse 3','Warehouse 4','Warehouse 5')`
 - b. A **Measure** labeled *latitude (D1)* is added. The following expression is entered:
`=only(aggr(KMeansCentroid2D(vDistClusters,0,only(latitude),only(longitude)),id))`
 Note the parameter **coordinate_no** corresponds to the first dimension(0). In this case the dimension *latitude* is plotted against the x-axis. If we were working with the **CentroidND** function and there were up to six dimensions, these parameter entries could be any of six values: 0,1,2,3,4,or 5.
 - c. A **Measure** labeled *longitude (D2)* is added. The following expression is entered:
`=only(aggr(KMeansCentroid2D(vDistClusters,1,only(latitude),only(longitude)),id))`
 The parameter **coordinate_no** in this expression corresponds to the second dimension(1). The dimension *longitude* is plotted against the y-axis.

Table: Cluster centroid calculations

Cluster centroids			
Clusters	Q	latitude (D1)	longitude (D2)
Totals		-	-
Warehouse 1		40.945422240426	-73.719966482979
Warehouse 2		42.590538729412	-79.067889217647
Warehouse 3		42.805089516667	-75.901621883333
Warehouse 4		42.8581692625	-77.6800485875
Warehouse 5		43.436770771429	-73.734622635714

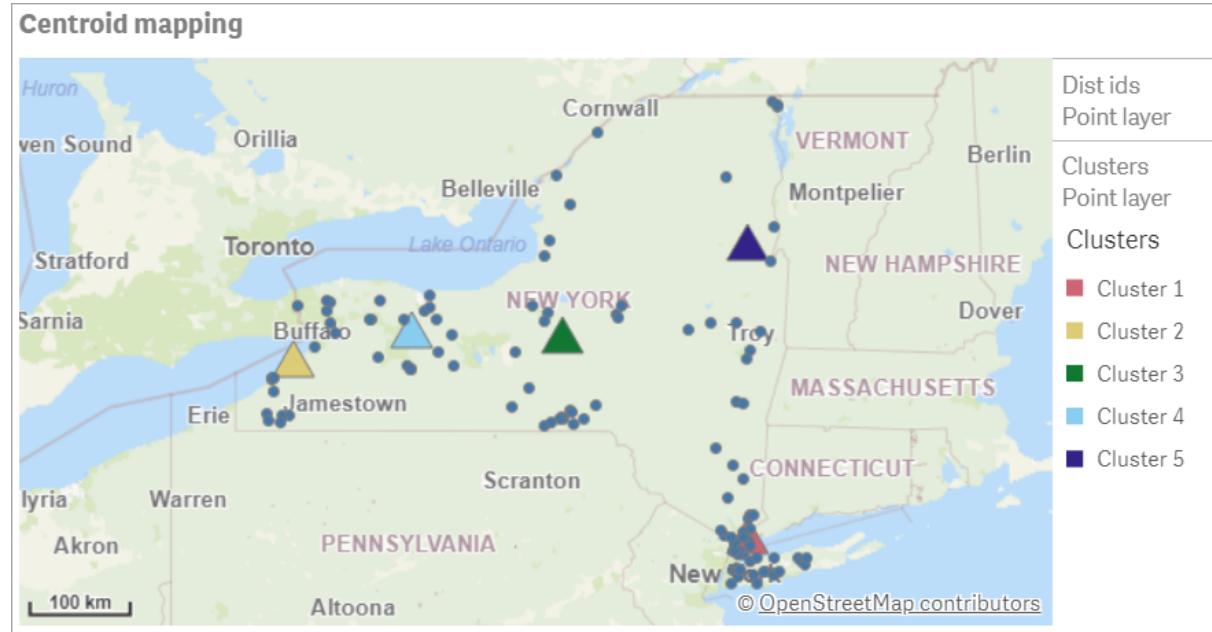
Centroid mapping

The next step is to map the centroids. It is up to the app developer if they prefer to place the visualization on separate sheets.

1. A **map** named *Centroid mapping* is dragged onto the sheet.
2. In the **Layers** section, **Add layer** is selected, then **Point layer** is selected.
 - a. The **Field id** is selected and *Dist ids Label* is added.
 - b. In the **Location** section, the checkbox for **Latitude and Longitude fields** is selected.
 - c. For **Latitude**, the *latitude* field is selected.
 - d. For **Longitude**, the *longitude* field is selected.
 - e. In the **Size & Shape** section, **Bubble** is selected for **Shape**, and the **Size** is decreased to preference on the slider.
 - f. In the **Colors** section, **Single color** is selected and blue is selected for the **Color** and grey for the **Outline** color (these choices are also a matter of preference).
3. In the **Layers** section, a second **Point layer** is added by selecting **Add layer** and then selecting **Point layer**.
 - a. The following expression is entered: `=aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)`
 - b. The **Label** *Clusters* is added.
 - c. In the **Location** section, the checkbox for **Latitude and Longitude fields** is selected.
 - d. For **Latitude** which in this case is plotted along the x-axis, the following expression is added: `=aggr(KMeansCentroid2D(vDistClusters,0,only(latitude),only(longitude)),id)`
 - e. For **Longitude** which in this case is plotted along the y-axis, the following expression is added: `=aggr(KMeansCentroid2D(vDistClusters,1,only(latitude),only(longitude)),id)`
 - f. In the **Size & Shape** section, **Triangle** is selected for **Shape**, and the **Size** is decreased on the slider to preference.
 - g. Under **Colors and legend**, **Custom** is selected for **Colors**.

- h. **By dimension** is selected for coloring the chart. The following expression is entered: `=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1,'Cluster 1','Cluster 2','Cluster 3','Cluster 4','Cluster 5')`
 - i. The dimension is labeled *Clusters*.
4. In **Map settings**, **Adaptive** is selected for **Projection**. **Metric** is selected for **Units of measurement**.

Map: Centroids mapped by cluster

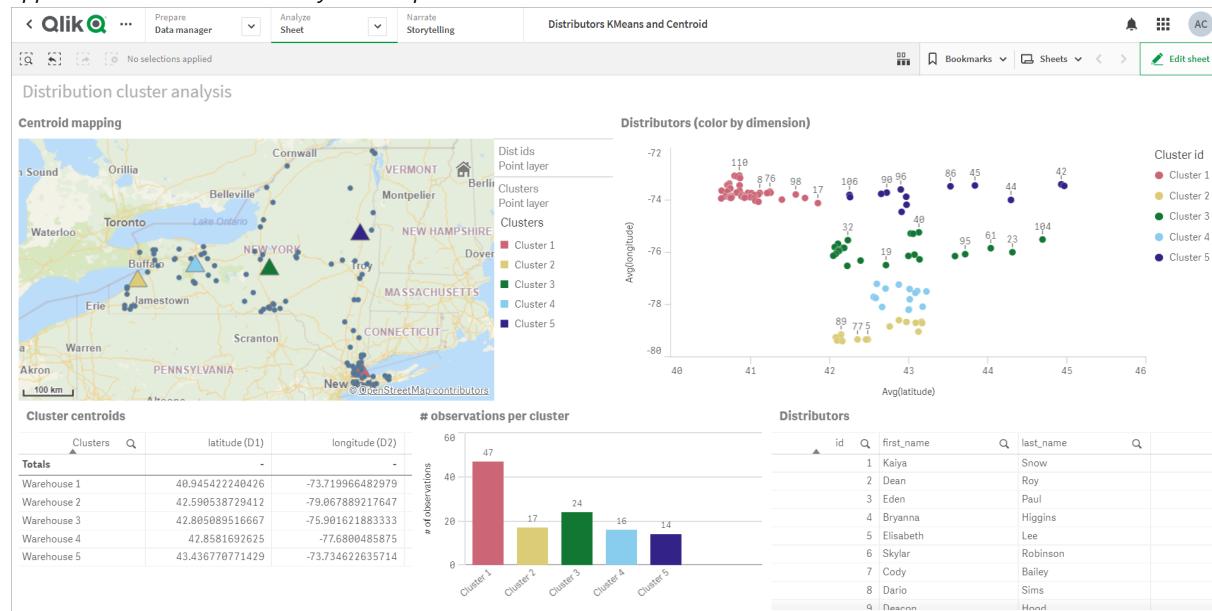


Conclusion

Using the KMeans function for this real-world scenario, distributors have been segmented into similar groups or clusters based on similarity; in this case, proximity to one another. The Centroid function was applied to those clusters to identify five mapping coordinates. Those coordinates provide an initial central location at which to build or locate warehouses. The centroid function is applied to the **map** chart, so that app users can visualize where the centroids are located relative to surrounding cluster data points. The resulting coordinates represent potential warehouse locations that could minimize delivery costs to distributors in New York state.

5 Script and chart functions

App: KMeans and centroid analysis example



Distributor dataset: Inline load for data load editor in Qlik Sense

```
DistributorData:
Load * Inline [
id,first_name,last_name,telephone,address,city,state,zip,latitude,longitude
1,Kaiya,Snow,(716) 201-1212,6231 Tonawanda Creek Rd #APT 308,Lockport,NY,14094,43.08926,-78.69313
2,Dean,Roy,(716) 201-1588,6884 E High St,Lockport,NY,14094,43.16245,-78.65036
3,Eden,Paul,(716) 202-4596,4647 Southwestern Blvd #APT 350,Hamburg,NY,14075,42.76003,-78.83194
4,Bryanna,Higgins,(716) 203-7041,418 Park Ave,Dunkirk,NY,14048,42.48279,-79.33088
5,Elisabeth,Lee,(716) 203-7043,36 E Courtney St,Dunkirk,NY,14048,42.48299,-79.31928
6,Skylar,Robinson,(716) 203-7166,26 Greco Ln,Dunkirk,NY,14048,42.4612095,-79.3317925
7,Cody,Bailey,(716) 203-7201,114 Lincoln Ave,Dunkirk,NY,14048,42.4801269,-79.322232
8,Dario,Sims,(408) 927-1606,N Castle Dr,Armonk,NY,10504,41.11979,-73.714864
9,Deacon,Hood,(410) 244-6221,4856 44th St,Woodside,NY,11377,40.748372,-73.905445
10,Zackery,Levy,(410) 363-8874,61 Executive Blvd,Farmingdale,NY,11735,40.7197457,-73.430239
11,Rey,Hawkins,(412) 344-8687,4585 Shimerville Rd,Clarence,NY,14031,42.972075,-78.6592452
12,Phillip,Howard,(413) 269-4049,464 Main St #101,Port Washington,NY,11050,40.8273756,-73.7009971
13,Shirley,Tyler,(434) 985-8943,114 Glann Rd,Apalachin,NY,13732,42.0482515,-76.1229725
14,Aniyah,Jarvis,(440) 244-1808,87 N Middletown Rd,Pearl River,NY,10965,41.0629,-74.0159
15,Alayna,Woodard,(478) 335-3704,70 W Red Oak Ln,West Harrison,NY,10604,41.0162722,-73.7234926
16,Jermaine,Lambert,(508) 561-9836,24 Kellogg Rd,New Hartford,NY,13413,43.0555739,-75.2793197
17,Harper,Gibbs,(239) 466-0238,Po Box 33,Cottekill,NY,12419,41.853392,-74.106082
18,Osvaldo,Graham,(252) 246-0816,6878 Sand Hill Rd,East Syracuse,NY,13057,43.073215,-76.081448
19,Roberto,Wade,(270) 469-1211,3936 Holley Rd,Moravia,NY,13118,42.713044,-76.481227
20,Kate,Mcguire,(270) 788-3080,6451 State 64 Rte #3,Naples,NY,14512,42.707366,-77.380489
21,Dale,Andersen,(281) 480-5690,205 W Service Rd,Champlain,NY,12919,44.9645392,-73.4470831
22,Lorelai,Burch,(302) 644-2133,1 Brewster St,Glen Cove,NY,11542,40.865177,-73.633019
23,Amiyah,Flowers,(303) 223-0055,46600 US Interstate 81 Rte,Alexandria Bay,NY,13607,44.309626,-75.988365
```

24,Mckinley,Clements,(303) 918-3230,200 Summit Lake Dr,Valhalla,NY,10595,41.101145,-73.778298
 25,Marc,Gibson,(607) 203-1233,25 Robinson St,Binghamton,NY,13901,42.107416,-75.901614
 26,Kali,Norman,(607) 203-1400,1 Ely Park Blvd #APT 15,Binghamton,NY,13905,42.125866,-75.925026
 27,Laci,Cain,(607) 203-1437,16 Zimmer Road,Kirkwood,NY,13795,42.066516,-75.792627
 28,Mohammad,Perez,(607) 203-1652,71 Endicott Ave #APT 12,Johnson City,NY,13790,42.111894,-75.952187
 29,Izabelle,Pham,(607) 204-0392,434 State 369 Rte,Port Crane,NY,13833,42.185838,-75.823074
 30,Kiley,Mays,(607) 204-0870,244 Ballyhack Rd #14,Port Crane,NY,13833,42.175612,-75.814917
 31,Peter,Trevino,(607) 205-1374,125 Melbourne St.,Vestal,NY,13850,42.080254,-76.051124
 32,Ani,Francis,(607) 208-4067,48 Caswell St,Afton,NY,13730,42.232065,-75.525674
 33,Jared,Sheppard,(716) 386-3002,4709 430th Rte,Bemus Point,NY,14712,42.162175,-79.39176
 34,Dulce,Atkinson,(914) 576-2266,501 Pelham Rd,New Rochelle,NY,10805,40.895449,-73.782602
 35,Jayla,Beasley,(716) 526-1054,5010 474th Rte,Ashville,NY,14710,42.096859,-79.375561
 36,Dane,Donovan,(718) 545-3732,5014 31st Ave,Woodside,NY,11377,40.756967,-73.909506
 37,Brendon,Clay,(585) 322-7780,133 Cummings Ave,Gainesville,NY,14066,42.664309,-78.085651
 38,Asia,Nunez,(718) 426-1472,2407 Gilmore ,East Elmhurst,NY,11369,40.766662,-73.869185
 39,Dawson,Odonnell,(718) 342-2179,5019 H Ave,Brooklyn,NY,11234,40.633245,-73.927591
 40,Kyle,Collins,(315) 733-7078,502 Rockhaven Rd,Utica,NY,13502,43.129184,-75.226726
 41,Eliza,Hardin,(315) 331-8072,502 Sladen Place,West Point,NY,10996,41.3993,-73.973003
 42,Kasen,Klein,(518) 298-4581,2407 Lake Shore Rd,Chazy,NY,12921,44.925561,-73.387373
 43,Reuben,Bradford,(518) 298-4581,33 Lake Flats Dr,Champlain,NY,12919,44.928092,-73.387884
 44,Henry,Grimes,(518) 523-3990,2407 Main St,Lake Placid,NY,12946,44.291487,-73.98474
 45,Kyan,Livingston,(518) 585-7364,241 Alexandria Ave,Ticonderoga,NY,12883,43.836553,-73.43155
 46,Kaitlyn,Short,(516) 678-3189,241 Chance Dr,Oceanside,NY,11572,40.638534,-73.63079
 47,Damaris,Jacobs,(914) 664-5331,241 Claremont Ave,Mount Vernon,NY,10552,40.919852,-73.827848
 48,Alivia,Schroeder,(315) 469-4473,241 Lafayette Rd,Syracuse,NY,13205,42.996446,-76.12957
 49,Bridget,Strong,(315) 298-4355,241 Maltby Rd,Pulaski,NY,13142,43.584966,-76.136317
 50,Francis,Lee,(585) 201-7021,166 Ross St,Batavia,NY,14020,43.0031502,-78.17487
 51,Makaila,Phelps,(585) 201-7422,58 S Main St,Batavia,NY,14020,42.99941,-78.1939285
 52,Jazlynn,Stephens,(585) 203-1087,1 Sinclair Dr,Pittsford,NY,14534,43.084157,-77.545452
 53,Ryann,Randolph,(585) 203-1519,331 Eaglehead Rd,East Rochester,NY,14445,43.10785,-77.475552
 54,Rosa,Baker,(585) 204-4011,42 Ossian St,Dansville,NY,14437,42.560761,-77.70088
 55,Marcel,Barry,(585) 204-4013,42 Jefferson St,Dansville,NY,14437,42.557735,-77.702983
 56,Dennis,Schmitt,(585) 204-4061,750 Dansville Mount Morris Rd,Dansville,NY,14437,42.584458,-77.741648
 57,Cassandra,Kim,(585) 204-4138,3 Perine Ave APT1,Dansville,NY,14437,42.562865,-77.69661
 58,Kolton,Jacobson,(585) 206-5047,4925 Upper Holly Rd,Holley,NY,14470,43.175957,-78.074465
 59,Nathanael,Donovan,(718) 393-3501,9604 57th Ave,Corona,NY,11373,40.736077,-73.864858
 60,Robert,Frazier,(718) 271-3067,300 56th Ave,Corona,NY,11373,40.735304,-73.873997
 61,Jessie,Mora,(315) 405-8991,9607 Forsyth Loop,Watertown,NY,13603,44.036466,-75.833437
 62,Martha,Rollins,(347) 242-2642,22 Main St,Corona,NY,11373,40.757727,-73.829331
 63,Emely,Townsend,(718) 699-0751,60 Sanford Ave,Corona,NY,11373,40.755466,-73.831029
 64,Kylie,Cooley,(347) 561-7149,9608 95th Ave,Ozone Park,NY,11416,40.687564,-73.845715
 65,Wendy,Cameron,(585) 571-4185,9608 Union St,Scottsville,NY,14546,43.013327,-77.7907839
 66,Kayley,Peterson,(718) 654-5027,961 E 230th St,Bronx,NY,10466,40.889275,-73.850555
 67,Camden,Ochoa,(718) 760-8699,59 Vark St,Yonkers,NY,10701,40.929322,-73.89957
 68,Priscilla,Castillo,(910) 326-7233,9359 Elm St,Chadwicks,NY,13319,43.024902,-75.26886
 69,Dana,Schultz,(913) 322-4580,99 Washington Ave,Hastings on Hudson,NY,10706,40.99265,-73.879748
 70,Blaze,Medina,(914) 207-0015,60 Elliott Ave,Yonkers,NY,10705,40.921498,-73.896682
 71,Finnegan,Tucker,(914) 207-0015,90 Hillside Drive,Yonkers,NY,10705,40.922514,-73.892911
 72,Pranav,Palmer,(914) 214-8376,5 Bruce Ave,Harrison,NY,10528,40.970916,-73.711493
 73,Kolten,Wong,(914) 218-8268,70 Barker St,Mount Kisco,NY,10549,41.211993,-73.723202
 74,Jasiah,Vazquez,(914) 231-5199,30 Broadway,Dobbs Ferry,NY,10522,41.004629,-73.879825
 75,Lamar,Pierce,(914) 232-0380,68 Ridge Rd,Katonah,NY,10536,41.256662,-73.707964
 76,Carla,Coffey,(914) 232-0469,197 Beaver Dam Rd,Katonah,NY,10536,41.247934,-73.664363

```

77,Brooklynn,Harmon,(716) 595-3227,8084 Glasgow Rd,Cassadega,NY,14718,42.353861,-79.329558
78,Raquel,Hodges,(585) 398-8125,809 County Road ,Victor,NY,14564,43.011745,-77.398806
79,Jerimiah,Gardner,(585) 787-9127,809 Houston Rd,Webster,NY,14580,43.224204,-77.491353
80,Clarence,Hammond,(720) 746-1619,809 Pierpont Ave,Piermont,NY,10968,41.0491181,-73.918622
81,Rhys,Gill,(518) 427-7887,81 Columbia St,Albany,NY,12210,42.652824,-73.752096
82,Edith,Parrish,(845) 452-7621,81 Glenwood Ave,Poughkeepsie,NY,12603,41.691058,-73.910829
83,Kobe,Mcintosh,(845) 371-1101,81 Heitman Dr,Spring Valley,NY,10977,41.103227,-74.054396
84,Ayden,Waters,(516) 796-2722,81 Kingfisher Rd,Levittown,NY,11756,40.738939,-73.52826
85,Francis,Rogers,(631) 427-7728,81 Knollwood Ave,Huntington,NY,11743,40.864905,-73.426107
86,Jaden,Landry,(716) 496-4038,12839 39th Rte,Chaffee,NY,14030,43.527396,-73.462786
87,Giancarlo,Campos,(518) 885-5717,1284 Saratoga Rd,Ballston Spa,NY,12020,42.968594,-73.862847
88,Eduardo,Contreras,(716) 285-8987,1285 Saunders Sett Rd,Niagara Falls,NY,14305,43.122963,-79.029274
89,Gabriela,Davidson,(716) 267-3195,1286 Mee Rd,Falconer,NY,14733,42.147339,-79.137976
90,Evangeline,Case,(518) 272-9435,1287 2nd Ave,Watervliet,NY,12189,42.723132,-73.703818
91,Tyrone,Ellison,(518) 843-4691,1287 Midline Rd,Amsterdam,NY,12010,42.9730876,-74.1700608
92,Bryce,Bass,(518) 943-9549,1288 Leeds Athens Rd,Athens,NY,12015,42.259381,-73.876897
93,Londyn,Butler,(518) 922-7095,129 Argersinger Rd,Fultonville,NY,12072,42.910969,-74.441917
94,Graham,Becker,(607) 655-1318,129 Baker Rd,Windsor,NY,13865,42.107271,-75.66408
95,Rolando,Fitzgerald,(315) 465-4166,17164 County 90 Rte,Mannsville,NY,13661,43.713443,-76.06232
96,Grant,Hoover,(518) 692-8363,1718 County 113 Rte,Schaghticote,NY,12154,42.900648,-73.585036
97,Mark,Goodwin,(631) 584-6761,172 Cambon Ave,Saint James,NY,11780,40.871152,-73.146032
98,Deacon,Cantu,(845) 221-7940,172 Carpenter Rd,Hopewell Junction,NY,12533,41.57388,-73.77609
99,Tristian,Walsh,(516) 997-4750,172 E Cabot Ln,Westbury,NY,11590,40.7480397,-73.54819
100,Abram,Alexander,(631) 588-3817,172 Lorenzo Cir,Ronkonkoma,NY,11779,40.837123,-73.09367
101,Lesly,Bush,(516) 489-3791,172 Nassau Blvd,Garden City,NY,11530,40.71147,-73.660753
102,Pamela,Espinoza,(716) 201-1520,172 Niagara St ,Lockport,NY,14094,43.169871,-78.70093
103,Bryanna,Newton,(914) 328-4332,172 Warren Ave,White Plains,NY,10603,41.047207,-73.79572
104,Marcelo,Schmitt,(315) 393-4432,319 Mansion Ave,Ogdensburg,NY,13669,44.690246,-75.49992
105,Layton,Valenzuela,(631) 676-2113,319 Singingwood Dr,Holbrook,NY,11741,40.801391,-73.058993
106,Roderick,Rocha,(518) 671-6037,319 Warren St,Hudson,NY,12534,42.252527,-73.790629
107,Camryn,Terrell,(315) 635-1680,3192 Olive Dr,Baldinsville,NY,13027,43.136843,-76.260303
108,Summer,Callahan,(585) 394-4195,3192 Smith Road,Canandaigua,NY,14424,42.875457,-77.228039
109,Pierre,Novak,(716) 665-2524,3194 Falconer Kimball Stand Rd,Falconer,NY,14733,42.138439,-79.211091
110,Kennedi,Fry,(315) 543-2301,32 College Rd,Selden,NY,11784,40.861624,-73.04757
111,Wyatt,Pruitt,(716) 681-4042,277 Ransom Rd,Lancaster ,NY,14086,42.87702,-78.591302
112,Lilly,Jensen,(631) 841-0859,2772 Schliegel Blvd,Amityville,NY,11701,40.708021,-73.413015
113,Tristin,Hardin,(631) 920-0927,278 Fulton Street,West Babylon,NY,11704,40.733578,-73.357321
114,Tanya,Stafford,(716) 484-0771,278 Sampson St,Jamestown,NY,14701,42.0797,-79.247805
115,Paris,Cordova,(607) 589-4857,278 Washburn Rd,Spencer,NY,14883,42.225046,-76.510257
116,Alfonso,Morse,(718) 359-5582,200 Colden St,Flushing,NY,11355,40.750403,-73.822752
117,Maurice,Hooper,(315) 595-6694,4435 Italy Hill Rd,Branchport,NY,14418,42.597957,-77.199267
118,Iris,Wolf,(607) 539-7288,444 Harford Rd,Brooktondale,NY,14817,42.392164,-76.30756
];

```

KMeans2D - chart function

KMeans2D() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the cluster id of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, respectively. These are both aggregations. The number of clusters that are created is determined by the num_clusters parameter. Data can be optionally normalized by the norm parameter.

KMeans2D returns one value per data point. The returned value is a dual and is the integer value corresponding to the cluster each data point has been assigned to.

Syntax:

```
KMeans2D (num_clusters, coordinate_1, coordinate_2 [, norm])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
num_clusters	Integer that specifies the number of clusters.
coordinate_1	The aggregation that calculates the first coordinate, usually the x-axis of the scatter chart that can be made from the chart. The additional parameter, coordinate_2, calculates the second coordinate.
norm	<p>The optional normalization method applied to datasets before KMeans clustering.</p> <p>Possible values:</p> <ul style="list-style-type: none">0 or 'none' for no normalization1 or 'zscore' for z-score normalization2 or 'minmax' for min-max normalization <p>If no parameter is supplied or if the supplied parameter is incorrect, no normalization is applied.</p> <p>Z-score normalizes data based on feature mean and standard deviation. Z-score does not ensure each feature has the same scale but it is a better approach than min-max when dealing with outliers.</p> <p>Min-max normalization ensures that the features have the same scale by taking the minimum and maximum values of each and recalculating each datapoint.</p>

Example: Chart expression

In this example, we create a scatter plot chart using the *Iris* dataset, and then use KMeans to color the data by expression.

We also create a variable for the *num_clusters* argument, and then use a variable input box to change the number of clusters.

The *Iris* data set is publicly available in a variety of formats. We have provided the data as an inline table to load using the data load editor in Qlik Sense. Note that we added an *Id* column to the data table for this example.

After loading the data in Qlik Sense, we do the following:

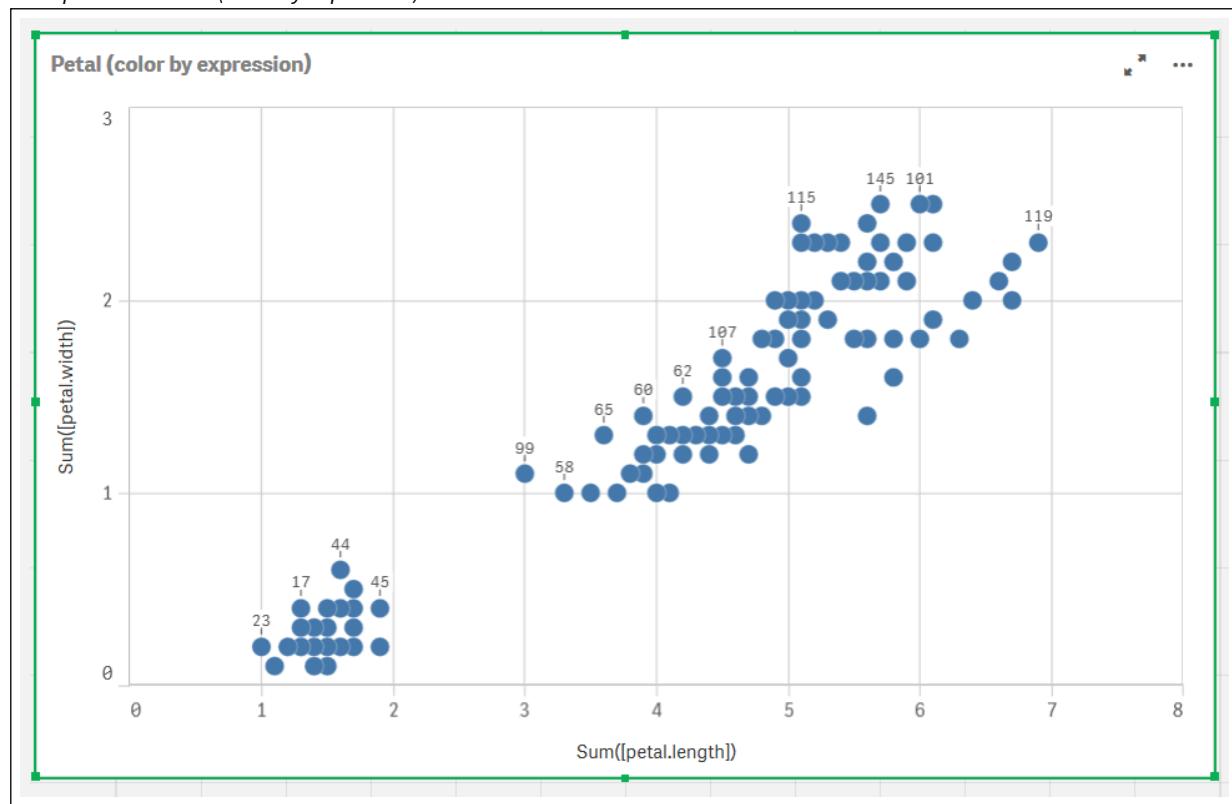
1. Drag a **Scatter plot** chart onto a new sheet. Name the chart *Petal (color by expression)*.
2. Create a variable to specify the number of clusters. For the variable **Name**, enter *KmeansPetalClusters*. For the variable **Definition**, enter *=2*.
3. Configure **Data** for the chart:
 - i. Under **Dimensions**, choose *id* for the field for **Bubble**. Enter Cluster Id for the Label.
 - ii. Under **Measures**, choose *Sum([petal.length])* for the expression for **X-axis**.
 - iii. Under **Measures**, choose *Sum([petal.width])* for the expression for **Y-axis**.

Data settings for Petal (color by expression) chart

The screenshot shows the 'Data' configuration panel for a scatter plot. It is divided into two main sections: 'Dimensions' and 'Measures'.
Dimensions: Contains a single entry 'Bubble' with the field 'Id' selected. A green box highlights this section.
Measures: Contains two entries: 'X-axis' with 'Sum [petal.length]' and 'Y-axis' with 'Sum [petal.width]'. Both entries have a green box highlighting them.
Below the dimensions section is a button labeled 'Add alternative'.

The data points are plotted on the chart.

Data points on Petal (color by expression) chart



4. Configure **Appearance** for the chart:
 - i. Under **Colors and legend**, choose **Custom** for **Colors**.
 - ii. Choose to color the chart **By expression**.
 - iii. Enter the following for **Expression**: `kmeans2d(${KmeansPetalClusters}, Sum([petal.length]), Sum([petal.width]))`
Note that `KmeansPetalClusters` is the variable that we set to 2.
Alternatively, enter the following: `kmeans2d(2, Sum([petal.length]), Sum([petal.width]))`
 - iv. Deselect the check box for **The expression is a color code**.

- v. Enter the following for **Label:** *Cluster Id*

Apearance settings for Petal (color by expression) chart

Appearance

▼ Colors and legend

Colors

Custom

By expression

Expression
kmeans2d\$(KmeansPetalC)

The expression is a color code

Label
Cluster Id

Color scheme

Sequential gradient

Sequential classes

Diverging gradient

Diverging classes

Reverse colors

Range

Auto

Show legend

Auto

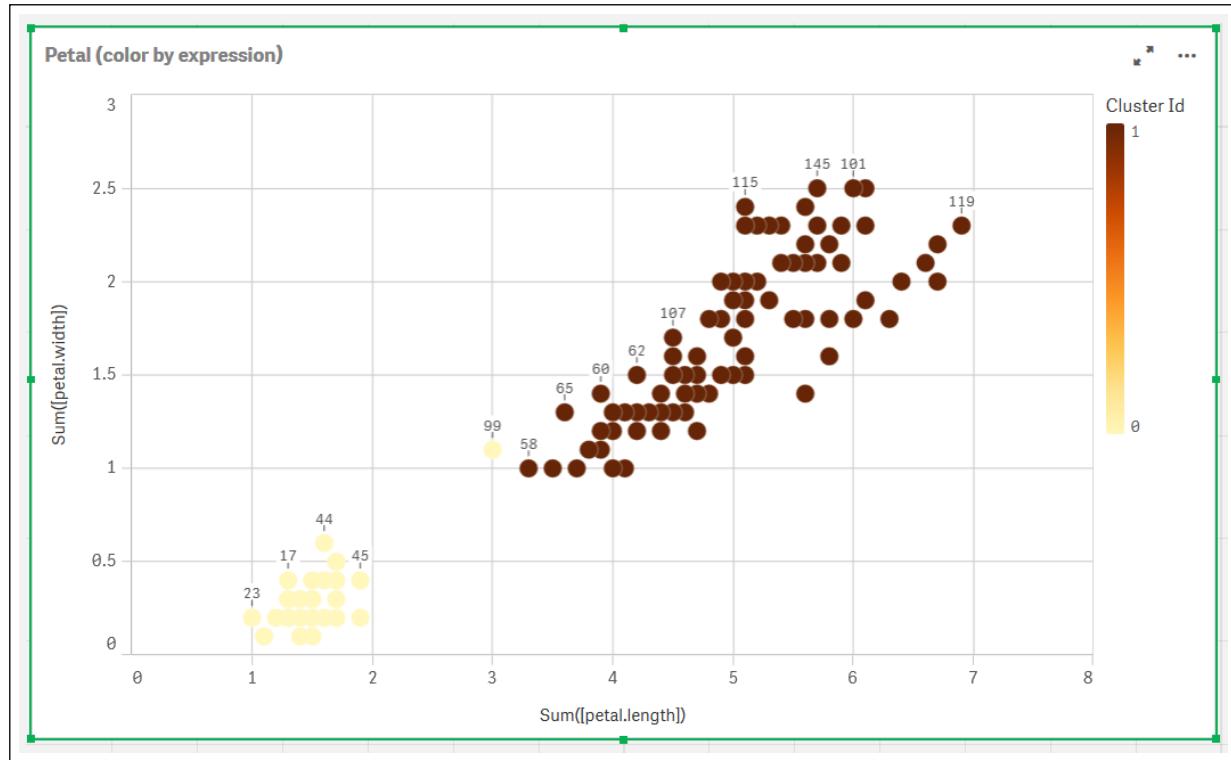
Legend position

Script syntax and chart functions - Qlik Sense, May 2023

Show legend title

The two clusters on the chart are colored by the KMeans expression.

Clusters colored by expression on Petal (color by expression) chart



5. Add a **Variable input** box for the number of clusters.

- i. Under **Custom objects** in the **Assets** panel, choose **Qlik Dashboard bundle**. If we did not have access to the dashboard bundle, we could still change the number of clusters using the variable that we created, or directly as an integer in the expression.
- ii. Drag a **Variable input** box onto the sheet.
- iii. Under **Appearance**, click **General**.
- iv. Enter the following for **Title: Clusters**
- v. Click **Variable**.
- vi. Choose the following variable for **Name: KmeansPetalClusters**.
- vii. Choose **Slider** for **Show as**.

viii. Choose **Values**, and configure the settings as required,

Appearance for Clusters variable input box

▼ General

Show titles On

Title
Clusters

Subtitle

Footnote

Disable hover menu

▼ Variable

Name
KmeansPetalClusters

Show as
Slider

Update on drag

▼ Values

Min
2

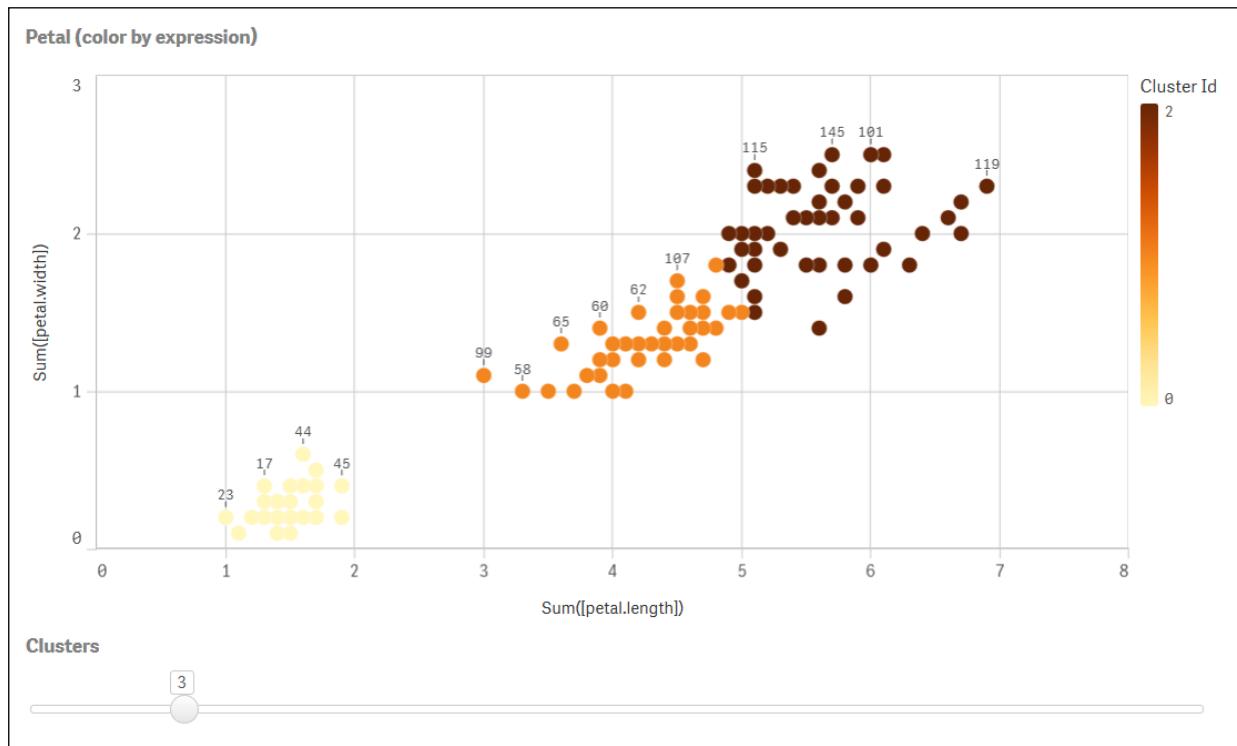
Max
10

Step
1

Slider label

When we are done editing, we can change the number of clusters using the slider in the *Clusters* variable input box.

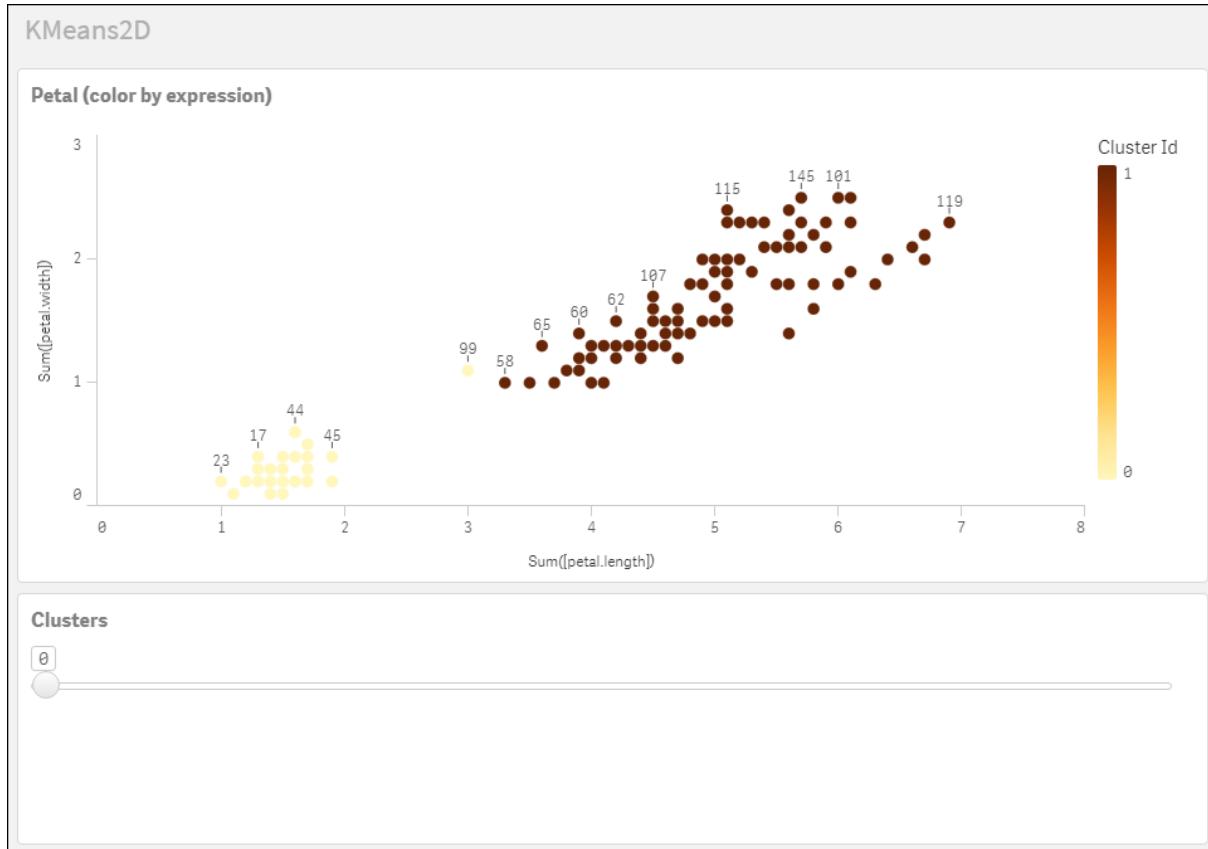
Clusters colored by expression on Petal (color by expression) chart



Auto-clustering

KMeans functions support auto-clustering using a method called depth difference (DeD). When a user sets 0 for the number of clusters, an optimal number of clusters for that dataset is determined. Note that while an integer for the number of clusters (k) is not explicitly returned, it is calculated within the KMeans algorithm. For example, if 0 is specified in the function for the value of *KmeansPetalClusters* or set through a variable input box, cluster assignments are automatically calculated for the dataset based on an optimal number of clusters.

KMeans depth difference method determines optimal number of clusters when (k) is set to 0



Iris data set: Inline load for data load editor in Qlik Sense

```
IrisData:  
Load * Inline [  
    sepal.length, sepal.width, petal.length, petal.width, variety, id  
    5.1, 3.5, 1.4, 0.2, Setosa, 1  
    4.9, 3, 1.4, 0.2, Setosa, 2  
    4.7, 3.2, 1.3, 0.2, Setosa, 3  
    4.6, 3.1, 1.5, 0.2, Setosa, 4  
    5, 3.6, 1.4, 0.2, Setosa, 5  
    5.4, 3.9, 1.7, 0.4, Setosa, 6  
    4.6, 3.4, 1.4, 0.3, Setosa, 7  
    5, 3.4, 1.5, 0.2, Setosa, 8  
    4.4, 2.9, 1.4, 0.2, Setosa, 9  
    4.9, 3.1, 1.5, 0.1, Setosa, 10  
    5.4, 3.7, 1.5, 0.2, Setosa, 11  
    4.8, 3.4, 1.6, 0.2, Setosa, 12  
    4.8, 3, 1.4, 0.1, Setosa, 13  
    4.3, 3, 1.1, 0.1, Setosa, 14  
    5.8, 4, 1.2, 0.2, Setosa, 15  
    5.7, 4.4, 1.5, 0.4, Setosa, 16  
    5.4, 3.9, 1.3, 0.4, Setosa, 17  
    5.1, 3.5, 1.4, 0.3, Setosa, 18  
    5.7, 3.8, 1.7, 0.3, Setosa, 19  
    5.1, 3.8, 1.5, 0.3, Setosa, 20  
    5.4, 3.4, 1.7, 0.2, Setosa, 21
```

5.1, 3.7, 1.5, 0.4, Setosa, 22
4.6, 3.6, 1, 0.2, Setosa, 23
5.1, 3.3, 1.7, 0.5, Setosa, 24
4.8, 3.4, 1.9, 0.2, Setosa, 25
5, 3, 1.6, 0.2, Setosa, 26
5, 3.4, 1.6, 0.4, Setosa, 27
5.2, 3.5, 1.5, 0.2, Setosa, 28
5.2, 3.4, 1.4, 0.2, Setosa, 29
4.7, 3.2, 1.6, 0.2, Setosa, 30
4.8, 3.1, 1.6, 0.2, Setosa, 31
5.4, 3.4, 1.5, 0.4, Setosa, 32
5.2, 4.1, 1.5, 0.1, Setosa, 33
5.5, 4.2, 1.4, 0.2, Setosa, 34
4.9, 3.1, 1.5, 0.1, Setosa, 35
5, 3.2, 1.2, 0.2, Setosa, 36
5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38
4.4, 3, 1.3, 0.2, Setosa, 39
5.1, 3.4, 1.5, 0.2, Setosa, 40
5, 3.5, 1.3, 0.3, Setosa, 41
4.5, 2.3, 1.3, 0.3, Setosa, 42
4.4, 3.2, 1.3, 0.2, Setosa, 43
5, 3.5, 1.6, 0.6, Setosa, 44
5.1, 3.8, 1.9, 0.4, Setosa, 45
4.8, 3, 1.4, 0.3, Setosa, 46
5.1, 3.8, 1.6, 0.2, Setosa, 47
4.6, 3.2, 1.4, 0.2, Setosa, 48
5.3, 3.7, 1.5, 0.2, Setosa, 49
5, 3.3, 1.4, 0.2, Setosa, 50
7, 3.2, 4.7, 1.4, Versicolor, 51
6.4, 3.2, 4.5, 1.5, versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53
5.5, 2.3, 4, 1.3, Versicolor, 54
6.5, 2.8, 4.6, 1.5, versicolor, 55
5.7, 2.8, 4.5, 1.3, versicolor, 56
6.3, 3.3, 4.7, 1.6, versicolor, 57
4.9, 2.4, 3.3, 1, Versicolor, 58
6.6, 2.9, 4.6, 1.3, versicolor, 59
5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61
5.9, 3, 4.2, 1.5, Versicolor, 62
6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, versicolor, 64
5.6, 2.9, 3.6, 1.3, versicolor, 65
6.7, 3.1, 4.4, 1.4, versicolor, 66
5.6, 3, 4.5, 1.5, versicolor, 67
5.8, 2.7, 4.1, 1, Versicolor, 68
6.2, 2.2, 4.5, 1.5, versicolor, 69
5.6, 2.5, 3.9, 1.1, versicolor, 70
5.9, 3.2, 4.8, 1.8, versicolor, 71
6.1, 2.8, 4, 1.3, Versicolor, 72
6.3, 2.5, 4.9, 1.5, versicolor, 73
6.1, 2.8, 4.7, 1.2, Versicolor, 74
6.4, 2.9, 4.3, 1.3, versicolor, 75
6.6, 3, 4.4, 1.4, versicolor, 76

6.8, 2.8, 4.8, 1.4, versicolor, 77
6.7, 3, 5, 1.7, Versicolor, 78
6, 2.9, 4.5, 1.5, Versicolor, 79
5.7, 2.6, 3.5, 1, Versicolor, 80
5.5, 2.4, 3.8, 1.1, versicolor, 81
5.5, 2.4, 3.7, 1, versicolor, 82
5.8, 2.7, 3.9, 1.2, versicolor, 83
6, 2.7, 5.1, 1.6, Versicolor, 84
5.4, 3, 4.5, 1.5, Versicolor, 85
6, 3.4, 4.5, 1.6, Versicolor, 86
6.7, 3.1, 4.7, 1.5, versicolor, 87
6.3, 2.3, 4.4, 1.3, versicolor, 88
5.6, 3, 4.1, 1.3, versicolor, 89
5.5, 2.5, 4, 1.3, versicolor, 90
5.5, 2.6, 4.4, 1.2, versicolor, 91
6.1, 3, 4.6, 1.4, Versicolor, 92
5.8, 2.6, 4, 1.2, versicolor, 93
5, 2.3, 3.3, 1, versicolor, 94
5.6, 2.7, 4.2, 1.3, versicolor, 95
5.7, 3, 4.2, 1.2, Versicolor, 96
5.7, 2.9, 4.2, 1.3, versicolor, 97
6.2, 2.9, 4.3, 1.3, versicolor, 98
5.1, 2.5, 3, 1.1, versicolor, 99
5.7, 2.8, 4.1, 1.3, versicolor, 100
6.3, 3.3, 6, 2.5, Virginica, 101
5.8, 2.7, 5.1, 1.9, Virginica, 102
7.1, 3, 5.9, 2.1, Virginica, 103
6.3, 2.9, 5.6, 1.8, Virginica, 104
6.5, 3, 5.8, 2.2, Virginica, 105
7.6, 3, 6.6, 2.1, Virginica, 106
4.9, 2.5, 4.5, 1.7, virginica, 107
7.3, 2.9, 6.3, 1.8, Virginica, 108
6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110
6.5, 3.2, 5.1, 2, Virginica, 111
6.4, 2.7, 5.3, 1.9, Virginica, 112
6.8, 3, 5.5, 2.1, Virginica, 113
5.7, 2.5, 5, 2, Virginica, 114
5.8, 2.8, 5.1, 2.4, Virginica, 115
6.4, 3.2, 5.3, 2.3, Virginica, 116
6.5, 3, 5.5, 1.8, Virginica, 117
7.7, 3.8, 6.7, 2.2, Virginica, 118
7.7, 2.6, 6.9, 2.3, Virginica, 119
6, 2.2, 5, 1.5, Virginica, 120
6.9, 3.2, 5.7, 2.3, virginica, 121
5.6, 2.8, 4.9, 2, Virginica, 122
7.7, 2.8, 6.7, 2, Virginica, 123
6.3, 2.7, 4.9, 1.8, Virginica, 124
6.7, 3.3, 5.7, 2.1, virginica, 125
7.2, 3.2, 6, 1.8, Virginica, 126
6.2, 2.8, 4.8, 1.8, Virginica, 127
6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129
7.2, 3, 5.8, 1.6, Virginica, 130
7.4, 2.8, 6.1, 1.9, Virginica, 131

```

7.9, 3.8, 6.4, 2, virginica, 132
6.4, 2.8, 5.6, 2.2, virginica, 133
6.3, 2.8, 5.1, 1.5, virginica, 134
6.1, 2.6, 5.6, 1.4, virginica, 135
7.7, 3, 6.1, 2.3, virginica, 136
6.3, 3.4, 5.6, 2.4, virginica, 137
6.4, 3.1, 5.5, 1.8, virginica, 138
6, 3, 4.8, 1.8, virginica, 139
6.9, 3.1, 5.4, 2.1, virginica, 140
6.7, 3.1, 5.6, 2.4, virginica, 141
6.9, 3.1, 5.1, 2.3, virginica, 142
5.8, 2.7, 5.1, 1.9, virginica, 143
6.8, 3.2, 5.9, 2.3, virginica, 144
6.7, 3.3, 5.7, 2.5, virginica, 145
6.7, 3, 5.2, 2.3, virginica, 146
6.3, 2.5, 5, 1.9, virginica, 147
6.5, 3, 5.2, 2, virginica, 148
6.2, 3.4, 5.4, 2.3, virginica, 149
5.9, 3, 5.1, 1.8, virginica, 150
];

```

KMeansND - chart function

KMeansND() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the cluster id of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, etc., up to n columns. These are all aggregations. The number of clusters that are created is determined by the num_clusters parameter.

KMeansND returns one value per data point. The returned value is a dual and is the integer value corresponding to the cluster each data point has been assigned to.

Syntax:

```
KMeansND(num_clusters, num_iter, coordinate_1, coordinate_2 [,coordinate_3 [, ...]])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
num_clusters	Integer that specifies the number of clusters.
num_iter	The number of iterations of clustering with reinitialized cluster centers.
coordinate_1	The aggregation that calculates the first coordinate, usually the x-axis (of a scatter chart that can be made from the chart). The additional parameters calculate the second, third, and fourth coordinates, etc.

Example: Chart expression

In this example, we create a scatter plot chart using the *Iris* dataset, and then use KMeans to color the data by expression.

We also create a variable for the *num_clusters* argument, and then use a variable input box to change the number of clusters.

Additionally, we create a variable for the *num_iter* argument, and then use a second variable input box to change the number of iterations.

The *Iris* data set is publicly available in a variety of formats. We have provided the data as an inline table to load using the data load editor in Qlik Sense. Note that we added an *Id* column to the data table for this example.

After loading the data in Qlik Sense, we do the following:

1. Drag a **Scatter plot** chart onto a new sheet. Name the chart *Petal (color by expression)*.
2. Create a variable to specify the number of clusters. For the variable **Name**, enter *KmeansPetalClusters*. For the variable **Definition**, enter *=2*.
3. Create a variable to specify the number of iterations. For the variable **Name**, enter *KmeansNumberIterations*. For the variable **Definition**, enter *=1*.
4. Configure **Data** for the chart:
 - i. Under **Dimensions**, choose *id* for the field for **Bubble**. Enter Cluster Id for the Label.
 - ii. Under **Measures**, choose *Sum([petal.length])* for the expression for **X-axis**.
 - iii. Under **Measures**, choose *Sum([petal.width])* for the expression for **Y-axis**.

Data settings for Petal (color by expression) chart

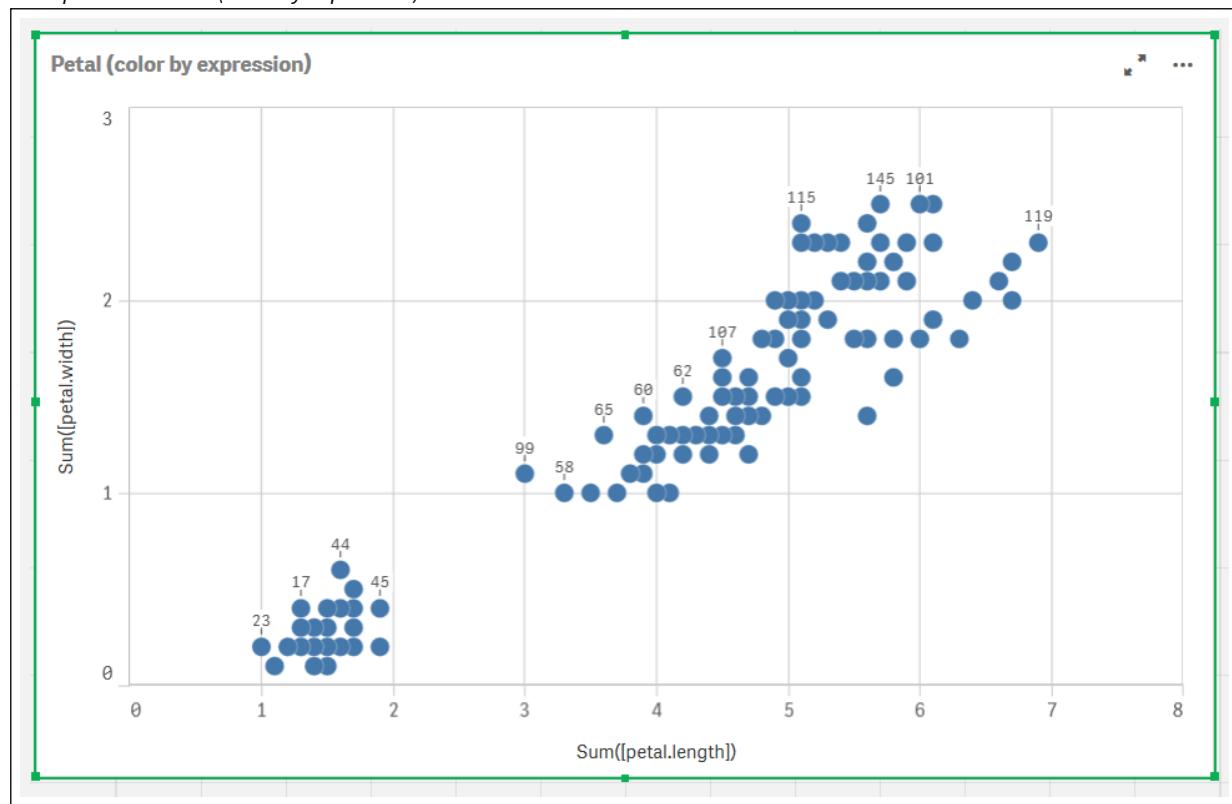
The screenshot shows the 'Data' settings panel for a chart. It is divided into two main sections: 'Dimensions' and 'Measures'.
Dimensions: Set to 'Bubble'. The 'Id' dimension is selected, indicated by a green border around the 'Id' field and its associated dropdown menu.
Measures: Set to 'X-axis' and 'Y-axis'.

- X-axis:** Sum of [petal.length].
- Y-axis:** Sum of [petal.width].

A green border highlights the entire 'Measures' section.

The data points are plotted on the chart.

Data points on Petal (color by expression) chart



5. Configure **Appearance** for the chart:

- i. Under **Colors and legend**, choose **Custom** for **Colors**.
- ii. Choose to color the chart **By expression**.
- iii. Enter the following for **Expression**: `kmeansnd
($KmeansPetalClusters),$(KmeansNumberIterations), Sum([petal.length]), Sum
([petal.width]),Sum([sepal.length]), Sum([sepal.width])`
Note that `KmeansPetalClusters` is the variable that we set to 2. `KmeansNumberIterations` is the variable that we set to 1.
Alternatively, enter the following: `kmeansnd(2, 2, Sum([petal.length]), Sum([petal.width]),Sum
([sepal.length]), Sum([sepal.width]))`
- iv. Deselect the check box for **The expression is a color code**.

- v. Enter the following for **Label:** *Cluster Id*

Apearance settings for Petal (color by expression) chart

Appearance

▼ Colors and legend

Colors

Custom

By expression

Expression

The expression is a color code

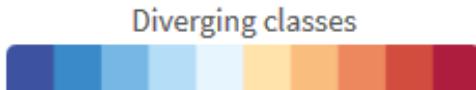
Label

Color scheme

Sequential gradient 

Sequential classes 

Diverging gradient 

Diverging classes 

Reverse colors

Range

Auto

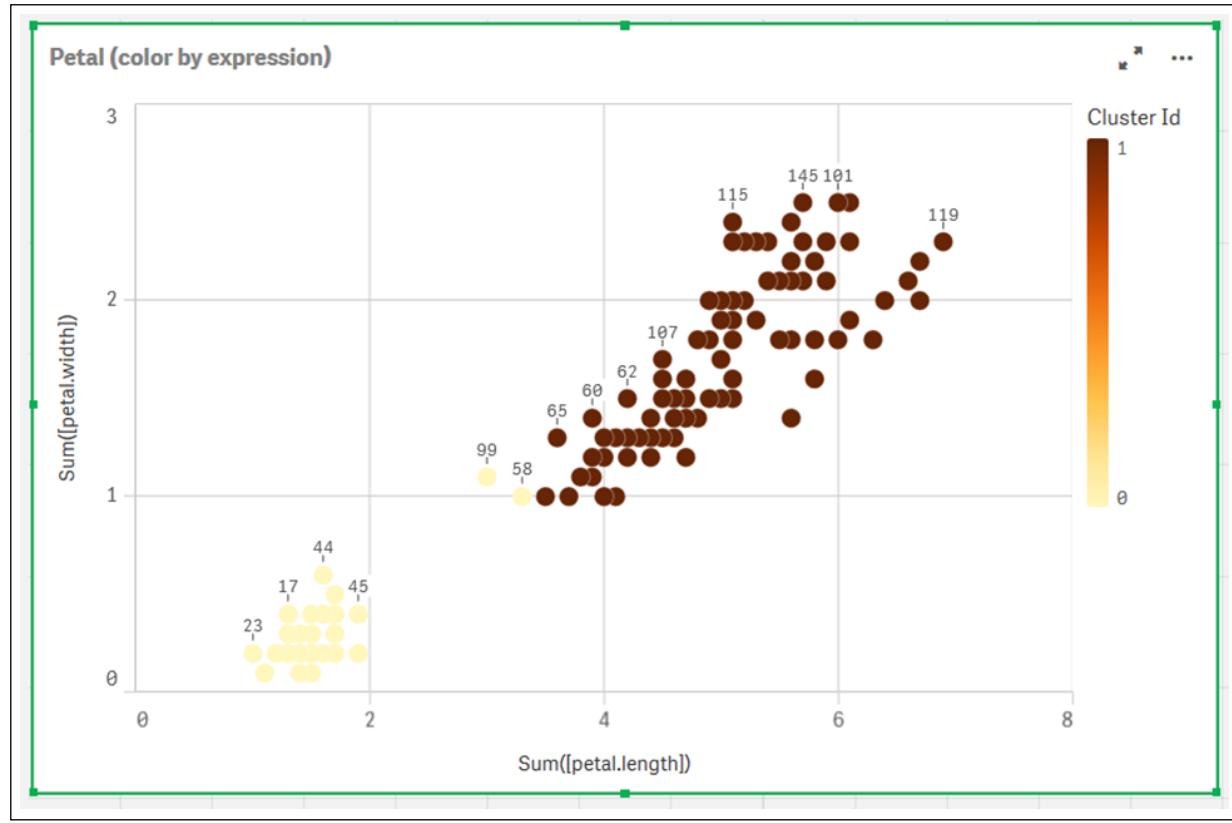
Show legend

Script syntax and chart functions - Qlik Sense, May 2023

Legend position

The two clusters on the chart are colored by the KMeans expression.

Clusters colored by expression on Petal (color by expression) chart



6. Add a **Variable input** box for the number of clusters.

- i. Under **Custom objects** in the **Assets** panel, choose **Qlik Dashboard bundle**. If we did not have access to the dashboard bundle, we could still change the number of clusters using the variable that we created, or directly as an integer in the expression.
- ii. Drag a **Variable input** box onto the sheet.
- iii. Under **Appearance**, click **General**.
- iv. Enter the following for **Title: Clusters**
- v. Click **Variable**.
- vi. Choose the following variable for **Name: KmeansPetalClusters**.
- vii. Choose **Slider** for **Show as**.

viii. Choose **Values**, and configure the settings as required,

Appearance for Clusters variable input box

▼ General

Show titles On

Title
Clusters

Subtitle

Footnote

Disable hover menu

▼ Variable

Name
KmeansPetalClusters

Show as
Slider

Update on drag

▼ Values

Min
2

Max
10

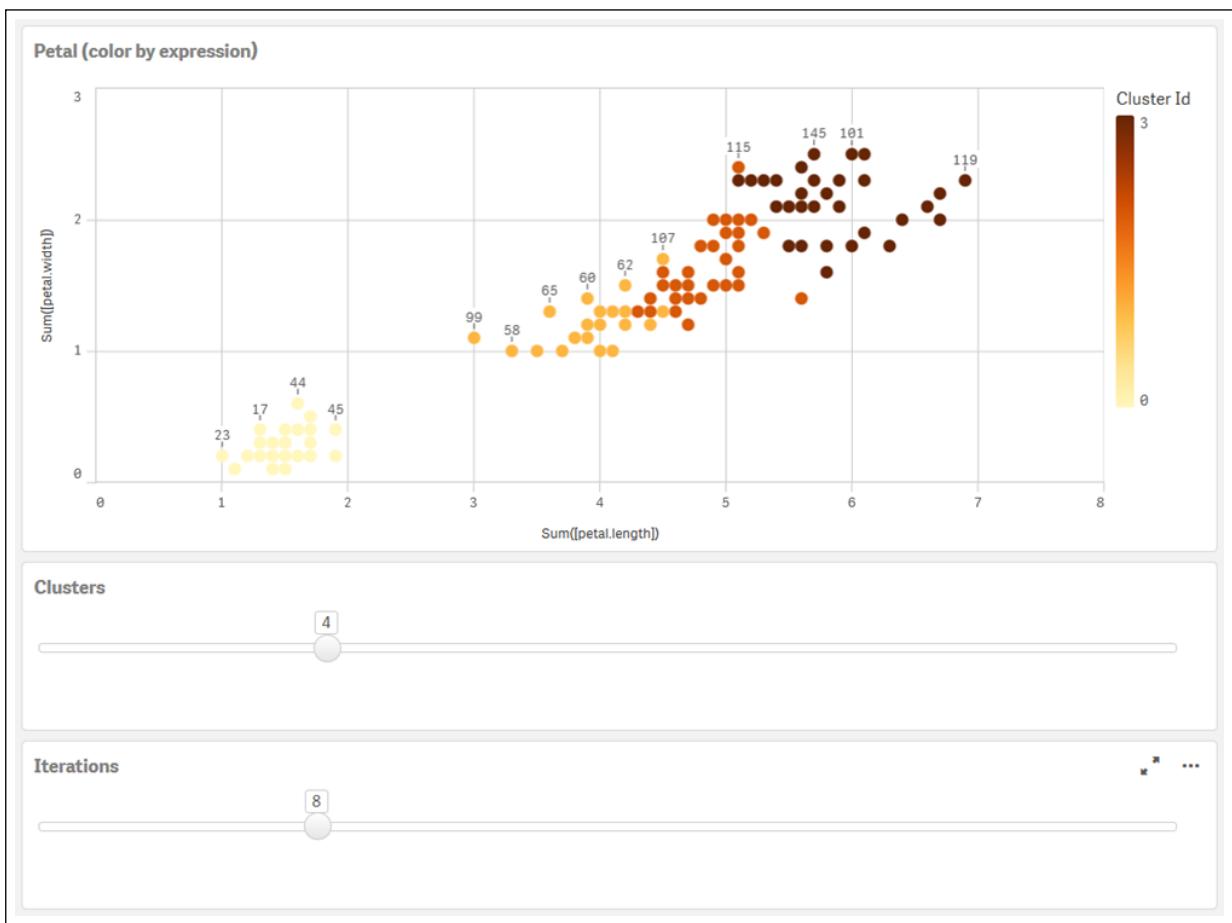
Step
1

Slider label

7. Add a **Variable input** box for the number of iterations.
 - i. Drag a **Variable input** box onto the sheet.
 - ii. Under **Appearance**, choose **General**.
 - iii. Enter the following for **Title: Iterations**
 - iv. Under **Appearance**, choose **Variable**.
 - v. Choose the following variable under **Name: KmeansNumberIterations**.
 - vi. Configure the additional settings as required,

We can now change the number of clusters and iterations using the sliders in the variable input boxes.

Clusters colored by expression on Petal (color by expression) chart

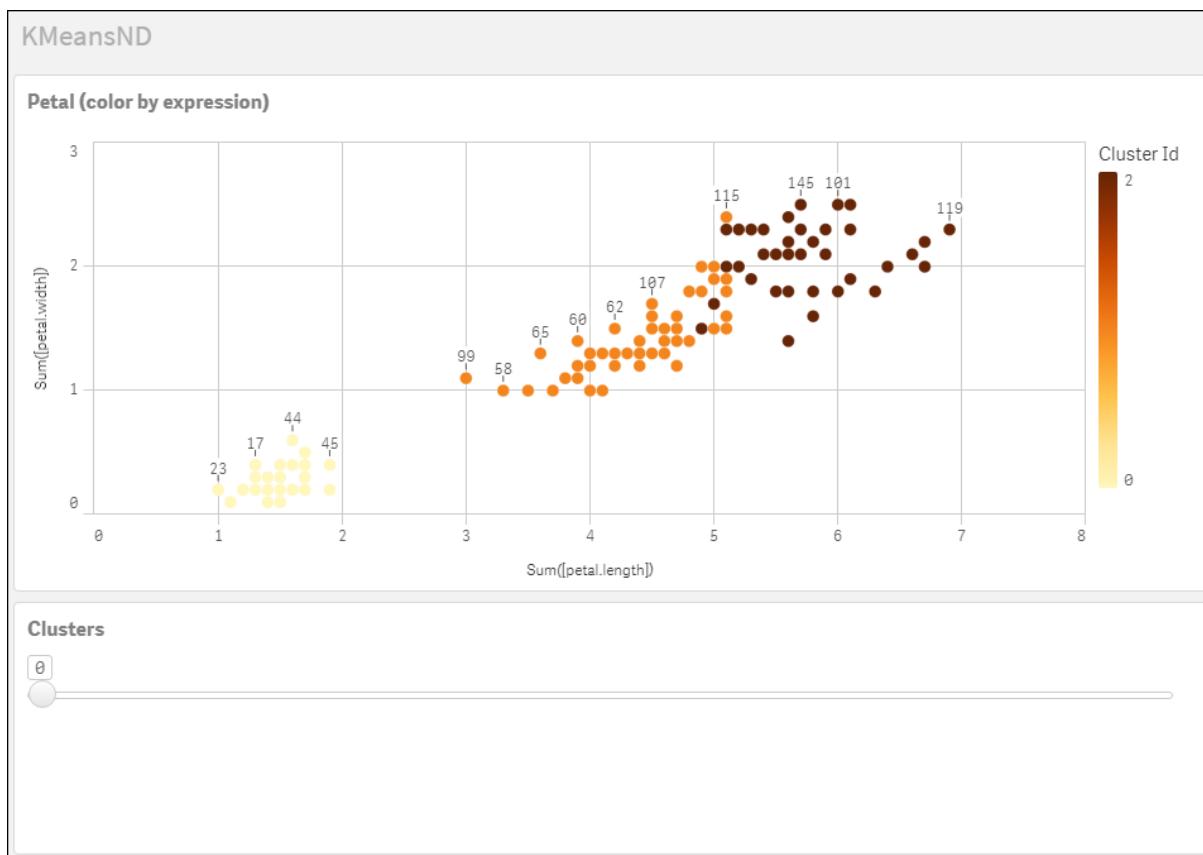


Auto-clustering

KMeans functions support auto-clustering using a method called depth difference (DeD). When a user sets 0 for the number of clusters, an optimal number of clusters for that dataset is determined. Note that while an integer for the number of clusters (k) is not explicitly returned, it is calculated within the KMeans algorithm. For example, if 0 is specified in the function for the value of *KmeansPetalClusters* or set through a variable input box, cluster assignments are automatically calculated for the dataset based on an optimal number of clusters. Given the Iris dataset, if 0 is selected for the number of clusters, the algorithm will determine (auto-cluster) an optimal number of clusters (3) for this dataset.

5 Script and chart functions

KMeans depth difference method determines optimal number of clusters when (k) is set to 0.



Iris data set: Inline load for data load editor in Qlik Sense

```
IrisData:  
Load * Inline [  
sepal.length, sepal.width, petal.length, petal.width, variety, id  
5.1, 3.5, 1.4, 0.2, Setosa, 1  
4.9, 3, 1.4, 0.2, Setosa, 2  
4.7, 3.2, 1.3, 0.2, Setosa, 3  
4.6, 3.1, 1.5, 0.2, Setosa, 4  
5, 3.6, 1.4, 0.2, Setosa, 5  
5.4, 3.9, 1.7, 0.4, Setosa, 6  
4.6, 3.4, 1.4, 0.3, Setosa, 7  
5, 3.4, 1.5, 0.2, Setosa, 8  
4.4, 2.9, 1.4, 0.2, Setosa, 9  
4.9, 3.1, 1.5, 0.1, Setosa, 10  
5.4, 3.7, 1.5, 0.2, Setosa, 11  
4.8, 3.4, 1.6, 0.2, Setosa, 12  
4.8, 3, 1.4, 0.1, Setosa, 13  
4.3, 3, 1.1, 0.1, Setosa, 14  
5.8, 4, 1.2, 0.2, Setosa, 15  
5.7, 4.4, 1.5, 0.4, Setosa, 16  
5.4, 3.9, 1.3, 0.4, Setosa, 17  
5.1, 3.5, 1.4, 0.3, Setosa, 18  
5.7, 3.8, 1.7, 0.3, Setosa, 19  
5.1, 3.8, 1.5, 0.3, Setosa, 20  
5.4, 3.4, 1.7, 0.2, Setosa, 21
```

5.1, 3.7, 1.5, 0.4, Setosa, 22
4.6, 3.6, 1, 0.2, Setosa, 23
5.1, 3.3, 1.7, 0.5, Setosa, 24
4.8, 3.4, 1.9, 0.2, Setosa, 25
5, 3, 1.6, 0.2, Setosa, 26
5, 3.4, 1.6, 0.4, Setosa, 27
5.2, 3.5, 1.5, 0.2, Setosa, 28
5.2, 3.4, 1.4, 0.2, Setosa, 29
4.7, 3.2, 1.6, 0.2, Setosa, 30
4.8, 3.1, 1.6, 0.2, Setosa, 31
5.4, 3.4, 1.5, 0.4, Setosa, 32
5.2, 4.1, 1.5, 0.1, Setosa, 33
5.5, 4.2, 1.4, 0.2, Setosa, 34
4.9, 3.1, 1.5, 0.1, Setosa, 35
5, 3.2, 1.2, 0.2, Setosa, 36
5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38
4.4, 3, 1.3, 0.2, Setosa, 39
5.1, 3.4, 1.5, 0.2, Setosa, 40
5, 3.5, 1.3, 0.3, Setosa, 41
4.5, 2.3, 1.3, 0.3, Setosa, 42
4.4, 3.2, 1.3, 0.2, Setosa, 43
5, 3.5, 1.6, 0.6, Setosa, 44
5.1, 3.8, 1.9, 0.4, Setosa, 45
4.8, 3, 1.4, 0.3, Setosa, 46
5.1, 3.8, 1.6, 0.2, Setosa, 47
4.6, 3.2, 1.4, 0.2, Setosa, 48
5.3, 3.7, 1.5, 0.2, Setosa, 49
5, 3.3, 1.4, 0.2, Setosa, 50
7, 3.2, 4.7, 1.4, Versicolor, 51
6.4, 3.2, 4.5, 1.5, versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53
5.5, 2.3, 4, 1.3, Versicolor, 54
6.5, 2.8, 4.6, 1.5, versicolor, 55
5.7, 2.8, 4.5, 1.3, versicolor, 56
6.3, 3.3, 4.7, 1.6, versicolor, 57
4.9, 2.4, 3.3, 1, Versicolor, 58
6.6, 2.9, 4.6, 1.3, versicolor, 59
5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61
5.9, 3, 4.2, 1.5, Versicolor, 62
6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, versicolor, 64
5.6, 2.9, 3.6, 1.3, versicolor, 65
6.7, 3.1, 4.4, 1.4, versicolor, 66
5.6, 3, 4.5, 1.5, versicolor, 67
5.8, 2.7, 4.1, 1, Versicolor, 68
6.2, 2.2, 4.5, 1.5, versicolor, 69
5.6, 2.5, 3.9, 1.1, versicolor, 70
5.9, 3.2, 4.8, 1.8, versicolor, 71
6.1, 2.8, 4, 1.3, Versicolor, 72
6.3, 2.5, 4.9, 1.5, versicolor, 73
6.1, 2.8, 4.7, 1.2, Versicolor, 74
6.4, 2.9, 4.3, 1.3, versicolor, 75
6.6, 3, 4.4, 1.4, Versicolor, 76

6.8, 2.8, 4.8, 1.4, versicolor, 77
6.7, 3, 5, 1.7, Versicolor, 78
6, 2.9, 4.5, 1.5, Versicolor, 79
5.7, 2.6, 3.5, 1, Versicolor, 80
5.5, 2.4, 3.8, 1.1, versicolor, 81
5.5, 2.4, 3.7, 1, versicolor, 82
5.8, 2.7, 3.9, 1.2, versicolor, 83
6, 2.7, 5.1, 1.6, Versicolor, 84
5.4, 3, 4.5, 1.5, Versicolor, 85
6, 3.4, 4.5, 1.6, Versicolor, 86
6.7, 3.1, 4.7, 1.5, Versicolor, 87
6.3, 2.3, 4.4, 1.3, versicolor, 88
5.6, 3, 4.1, 1.3, versicolor, 89
5.5, 2.5, 4, 1.3, versicolor, 90
5.5, 2.6, 4.4, 1.2, versicolor, 91
6.1, 3, 4.6, 1.4, Versicolor, 92
5.8, 2.6, 4, 1.2, versicolor, 93
5, 2.3, 3.3, 1, versicolor, 94
5.6, 2.7, 4.2, 1.3, versicolor, 95
5.7, 3, 4.2, 1.2, Versicolor, 96
5.7, 2.9, 4.2, 1.3, versicolor, 97
6.2, 2.9, 4.3, 1.3, versicolor, 98
5.1, 2.5, 3, 1.1, Versicolor, 99
5.7, 2.8, 4.1, 1.3, versicolor, 100
6.3, 3.3, 6, 2.5, Virginica, 101
5.8, 2.7, 5.1, 1.9, Virginica, 102
7.1, 3, 5.9, 2.1, Virginica, 103
6.3, 2.9, 5.6, 1.8, Virginica, 104
6.5, 3, 5.8, 2.2, Virginica, 105
7.6, 3, 6.6, 2.1, Virginica, 106
4.9, 2.5, 4.5, 1.7, virginica, 107
7.3, 2.9, 6.3, 1.8, Virginica, 108
6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110
6.5, 3.2, 5.1, 2, Virginica, 111
6.4, 2.7, 5.3, 1.9, Virginica, 112
6.8, 3, 5.5, 2.1, Virginica, 113
5.7, 2.5, 5, 2, Virginica, 114
5.8, 2.8, 5.1, 2.4, Virginica, 115
6.4, 3.2, 5.3, 2.3, Virginica, 116
6.5, 3, 5.5, 1.8, Virginica, 117
7.7, 3.8, 6.7, 2.2, Virginica, 118
7.7, 2.6, 6.9, 2.3, Virginica, 119
6, 2.2, 5, 1.5, Virginica, 120
6.9, 3.2, 5.7, 2.3, virginica, 121
5.6, 2.8, 4.9, 2, Virginica, 122
7.7, 2.8, 6.7, 2, Virginica, 123
6.3, 2.7, 4.9, 1.8, Virginica, 124
6.7, 3.3, 5.7, 2.1, virginica, 125
7.2, 3.2, 6, 1.8, Virginica, 126
6.2, 2.8, 4.8, 1.8, Virginica, 127
6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129
7.2, 3, 5.8, 1.6, Virginica, 130
7.4, 2.8, 6.1, 1.9, Virginica, 131

```

7.9, 3.8, 6.4, 2, virginica, 132
6.4, 2.8, 5.6, 2.2, virginica, 133
6.3, 2.8, 5.1, 1.5, virginica, 134
6.1, 2.6, 5.6, 1.4, virginica, 135
7.7, 3, 6.1, 2.3, virginica, 136
6.3, 3.4, 5.6, 2.4, virginica, 137
6.4, 3.1, 5.5, 1.8, virginica, 138
6, 3, 4.8, 1.8, virginica, 139
6.9, 3.1, 5.4, 2.1, virginica, 140
6.7, 3.1, 5.6, 2.4, virginica, 141
6.9, 3.1, 5.1, 2.3, virginica, 142
5.8, 2.7, 5.1, 1.9, virginica, 143
6.8, 3.2, 5.9, 2.3, virginica, 144
6.7, 3.3, 5.7, 2.5, virginica, 145
6.7, 3, 5.2, 2.3, virginica, 146
6.3, 2.5, 5, 1.9, virginica, 147
6.5, 3, 5.2, 2, virginica, 148
6.2, 3.4, 5.4, 2.3, virginica, 149
5.9, 3, 5.1, 1.8, virginica, 150
];

```

KMeansCentroid2D - chart function

KMeansCentroid2D() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the desired coordinate of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters coordinate_1, and coordinate_2, respectively. These are both aggregations. The number of clusters that are created is determined by the num_clusters parameter. Data can be optionally normalized by the norm parameter.

KMeansCentroid2D returns one value per data point. The returned value is a dual and is one of the coordinates of the position corresponding to the cluster center the data point has been assigned to.

Syntax:

```
KMeansCentroid2D(num_clusters, coordinate_no, coordinate_1, coordinate_2 [, norm])
```

Return data type: dual

Arguments:

Arguments

Argument	Description
num_clusters	Integer that specifies the number of clusters.
coordinate_no	The desired coordinate number of the centroids (corresponding, for example, to the x, y, or z axis).
coordinate_1	The aggregation that calculates the first coordinate, usually the x-axis of the scatter chart that can be made from the chart. The additional parameter, coordinate_2, calculates the second coordinate.

Argument	Description
norm	<p>The optional normalization method applied to datasets before KMeans clustering.</p> <p>Possible values:</p> <ul style="list-style-type: none"> 0 or ‘none’ for no normalization 1 or ‘zscore’ for z-score normalization 2 or ‘minmax’ for min-max normalization <p>If no parameter is supplied or if the supplied parameter is incorrect, no normalization is applied.</p> <p>Z-score normalizes data based on feature mean and standard deviation. Z-score does not ensure each feature has the same scale but it is a better approach than min-max when dealing with outliers.</p> <p>Min-max normalization ensures that the features have the same scale by taking the minimum and maximum values of each and recalculating each datapoint.</p>

Auto-clustering

KMeans functions support auto-clustering using a method called depth difference (DeD). When a user sets 0 for the number of clusters, an optimal number of clusters for that dataset is determined. Note that while an integer for the number of clusters (k) is not explicitly returned, it is calculated within the KMeans algorithm. For example, if 0 is specified in the function for the value of *KmeansPetalClusters* or set through a variable input box, cluster assignments are automatically calculated for the dataset based on an optimal number of clusters.

KMeansCentroidND - chart function

KMeansCentroidND() evaluates the rows of the chart by applying k-means clustering, and for each chart row displays the desired coordinate of the cluster this data point has been assigned to. The columns that are used by the clustering algorithm are determined by the parameters *coordinate_1*, *coordinate_2*, etc., up to *n* columns. These are all aggregations. The number of clusters that are created is determined by the *num_clusters* parameter.

KMeansCentroidND returns one value per row. The returned value is a dual and is one of the coordinates of the position corresponding to the cluster center the data point has been assigned to.

Syntax:

```
KMeansCentroidND((num_clusters, num_iter, coordinate_no, coordinate_1,
coordinate_2 [,coordinate_3 [, ...]])
```

Return data type: dual

Arguments:

Arguments	
Argument	Description
num_clusters	Integer that specifies the number of clusters.
num_iter	The number of iterations of clustering with reinitialized cluster centers.
coordinate_no	The desired coordinate number of the centroids (corresponding, for example, to the x, y, or z axis).
coordinate_1	The aggregation that calculates the first coordinate, usually the x-axis (of a scatter chart that can be made from the chart). The additional parameters calculate the second, third, and fourth coordinates, etc.

Auto-clustering

KMeans functions support auto-clustering using a method called depth difference (DeD). When a user sets 0 for the number of clusters, an optimal number of clusters for that dataset is determined. Note that while an integer for the number of clusters (k) is not explicitly returned, it is calculated within the KMeans algorithm. For example, if 0 is specified in the function for the value of *KmeansPetalClusters* or set through a variable input box, cluster assignments are automatically calculated for the dataset based on an optimal number of clusters.

STL_Trend - chart function

STL_Trend is a time series decomposition function. Along with **STL_Seasonal** and **STL_Residual**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. The **STL_Trend** function will identify a general trend, independent of seasonal patterns or cycles, from time series data.

The three STL functions are related to the input metric through a simple sum:

$$\text{STL_Trend} + \text{STL_Seasonal} + \text{STL_Residual} = \text{Input metric}$$

STL (seasonal and trend decomposition using Loess) employs data smoothing techniques, and through its input parameters, allows the user to adjust the periodicity of the calculations it performs. This periodicity determines how the time dimension of the input metric (a measure) is segmented in the analysis.

At minimum, **STL_Trend** takes an input metric (*target_measure*) and an integer value for its *period_int*, returning a floating-point value. The input metric will be in the form of an aggregation that varies along the time dimension. Optionally, you can include values for the *seasonal_smoker* and *trend_smoker* to adjust the smoothing algorithm.

Syntax:

```
STL_Trend(target_measure, period_int [,seasonal_smoker [,trend_smoker]])
```

Return data type: dual

Arguments

Argument	Description
target_measure	The measure to decompose into Seasonal and Trend components. This should be a measure such as Sum(Sales) or Sum(Passengers) that varies along the time dimension. This must not be a constant value.
period_int	The periodicity of the dataset. This parameter is an integer value representing the number of discrete steps that make up one period, or seasonal cycle, of the signal. For instance, if the time series is segmented into one section for each quarter of the year, you must set the period_int to a value of 4 to define the periodicity as Year.
seasonal_smoothen	Length of the seasonal smoother. This must be an odd integer. The seasonal smoother uses data for a particular phase in the seasonal variation, over a number of periods. One discrete step of the time dimension is used from each period. The seasonal smoother indicates the number of periods used for smoothing. For example, if the time dimension is segmented by month and the period is Year (12), the seasonal component will be computed so that each particular month of each year is calculated from data for the same month, both in that year and in adjacent years. The seasonal_smoothen value is the number of years used for smoothing.
trend_smoothen	Length of the trend smoother. This must be an odd integer. The trend smoother uses the same time scale as the period_int parameter, and its value is the number of granules used for smoothing. For example, if a time series is segmented by month, the trend smoother will be the number of months used for smoothing.

The **STL_Trend** chart function is often used in combination with the following functions:

Related functions

Function	Interaction
<i>STL_Seasonal - chart function (page 1380)</i>	This is the function used to compute the seasonal component of a time series.

Function	Interaction
<i>STL_Residual - chart function (page 1382)</i>	When breaking down an input metric into seasonal and trend component, part of the measure's variation will not fit within either of the two main components. The STL_Residual function computes this portion of the decomposition.

For a tutorial with a full example showing how to use this function, see *Tutorial - Time series decomposition in Qlik Sense (page 1384)*.

STL_Seasonal - chart function

STL_Seasonal is a time series decomposition function. Along with **STL_Trend** and **STL_Residual**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. The **STL_Seasonal** function can identify a seasonal pattern within a time series, separating this from the general trend displayed by the data.

The three STL functions are related to the input metric through a simple sum:

$$\text{STL_Trend} + \text{STL_Seasonal} + \text{STL_Residual} = \text{Input metric}$$

STL (seasonal and trend decomposition using Loess) employs data smoothing techniques, and through its input parameters, allows the user to adjust the periodicity of the calculations it performs. This periodicity determines how the time dimension of the input metric (a measure) is segmented in the analysis.

At minimum, **STL_Seasonal** takes an input metric (`target_measure`) and an integer value for its `period_int`, returning a floating-point value. The input metric will be in the form of an aggregation that varies along the time dimension. Optionally, you can include values for the `seasonal_smoothen` and `trend_smoothen` to adjust the smoothing algorithm.

Syntax:

```
STL_Seasonal(target_measure, period_int [, seasonal_smoothen [, trend_smoothen]])
```

Return data type: dual

Arguments

Argument	Description
target_measure	The measure to decompose into Seasonal and Trend components. This should be a measure such as Sum(Sales) or Sum(Passengers) that varies along the time dimension. This must not be a constant value.
period_int	The periodicity of the dataset. This parameter is an integer value representing the number of discrete steps that make up one period, or seasonal cycle, of the signal. For instance, if the time series is segmented into one section for each quarter of the year, you must set the period_int to a value of 4 to define the periodicity as Year.
seasonal_smoothen	Length of the seasonal smoother. This must be an odd integer. The seasonal smoother uses data for a particular phase in the seasonal variation, over a number of periods. One discrete step of the time dimension is used from each period. The seasonal smoother indicates the number of periods used for smoothing. For example, if the time dimension is segmented by month and the period is Year (12), the seasonal component will be computed so that each particular month of each year is calculated from data for the same month, both in that year and in adjacent years. The seasonal_smoothen value is the number of years used for smoothing.
trend_smoothen	Length of the trend smoother. This must be an odd integer. The trend smoother uses the same time scale as the period_int parameter, and its value is the number of granules used for smoothing. For example, if a time series is segmented by month, the trend smoother will be the number of months used for smoothing.

The **STL_Seasonal** chart function is often used in combination with the following functions:

Related functions

Function	Interaction
<i>STL_Trend - chart function (page 1378)</i>	This is the function used to compute the trend component of a time series.

Function	Interaction
<i>STL_Residual - chart function (page 1382)</i>	When breaking down an input metric into seasonal and trend component, part of the measure's variation will not fit within either of the two main components. The STL_Residual function computes this portion of the decomposition.

For a tutorial with a full example showing how to use this function, see *Tutorial - Time series decomposition in Qlik Sense (page 1384)*.

STL_Residual - chart function

STL_Residual is a time series decomposition function. Along with **STL_Seasonal** and **STL_Trend**, this function is used to decompose a time series into seasonal, trend, and residual components. In the context of the STL algorithm, time series decomposition is used to identify both a recurring seasonal pattern and a general trend, given an input metric and other parameters. In performing this operation, part of the variation in the input metric will neither fit within the seasonal nor the trend component, and will be defined as the residual component. The **STL_Residual** chart function captures this portion of the calculation.

The three STL functions are related to the input metric through a simple sum:

$$\text{STL_Trend} + \text{STL_Seasonal} + \text{STL_Residual} = \text{Input metric}$$

STL (seasonal and trend decomposition using Loess) employs data smoothing techniques, and through its input parameters, allows the user to adjust the periodicity of the calculations it performs. This periodicity determines how the time dimension of the input metric (a measure) is segmented in the analysis.

Since time series decomposition primarily looks for seasonality and general variations in data, the information in the residual is considered the least significant of the three components. However, a skewed or periodic residual component can help identify issues in the calculation, such as incorrect periodicity settings.

At minimum, **STL_Residual** takes an input metric (`target_measure`) and an integer value for its `period_int`, returning a floating-point value. The input metric will be in the form of an aggregation that varies along the time dimension. Optionally, you can include values for the `seasonal_smoothen` and `trend_smoothen` to adjust the smoothing algorithm.

Syntax:

```
STL_Residual(target_measure, period_int [,seasonal_smoothen [,trend_smoothen]])
```

Return data type: dual

Arguments

Argument	Description
target_measure	The measure to decompose into Seasonal and Trend components. This should be a measure such as <code>Sum(Sales)</code> or <code>Sum(Passengers)</code> that varies along the time dimension. This must not be a constant value.
period_int	The periodicity of the dataset. This parameter is an integer value representing the number of discrete steps that make up one period, or seasonal cycle, of the signal. For instance, if the time series is segmented into one section for each quarter of the year, you must set the period_int to a value of 4 to define the periodicity as Year.
seasonal_smoothen	Length of the seasonal smoother. This must be an odd integer. The seasonal smoother uses data for a particular phase in the seasonal variation, over a number of periods. One discrete step of the time dimension is used from each period. The seasonal smoother indicates the number of periods used for smoothing. For example, if the time dimension is segmented by month and the period is Year (12), the seasonal component will be computed so that each particular month of each year is calculated from data for the same month, both in that year and in adjacent years. The seasonal_smoothen value is the number of years used for smoothing.
trend_smoothen	Length of the trend smoother. This must be an odd integer. The trend smoother uses the same time scale as the period_int parameter, and its value is the number of granules used for smoothing. For example, if a time series is segmented by month, the trend smoother will be the number of months used for smoothing.

The **STL_Residual** chart function is often used in combination with the following functions:

Related functions	
Function	Interaction
<i>STL_Seasonal - chart function (page 1380)</i>	This is the function used to compute the seasonal component of a time series.
<i>STL_Trend - chart function (page 1378)</i>	This is the function used to compute the trend component of a time series.

For a tutorial with a full example showing how to use this function, see *Tutorial - Time series decomposition in Qlik Sense (page 1384)*.

Tutorial - Time series decomposition in Qlik Sense

This tutorial demonstrates using three chart functions to decompose a time series using the STL algorithm.

This tutorial uses time series data for the number of passengers using an airline per month to demonstrate the functionality of the STL algorithm. The **STL_Trend**, **STL_Seasonal**, and **STL_Residual** chart functions will be used to create the visualizations. For more information about time series decomposition in Qlik Sense, see *Time series decomposition functions (page 1331)*.

Create an app

Start by creating a new app and importing the dataset into it.

Download this dataset:

[Tutorial - Time series decomposition](#)

This file contains data regarding an airline's number of passengers per month.

Do the following:

1. From the hub, click **Create new app**.
2. Open the app and drop the *Tutorial - Time series decomposition.csv* file into it.

Prepare and load the data

In order for Qlik Sense to interpret the YearMonth field correctly, you might need to use Data manager to recognize the field as a date field, not a field with string values. Typically this step is handled automatically, but in this case the dates are presented in the slightly uncommon YYYY-MM format.

1. In Data manager, select the table and click .
2. With the *YearMonth* field selected, click  and set the **Field type** to **Date**.
3. Under **Input format**, enter *YYYY-MM*.
4. Under **Display format**, enter *YYYY-MM* and click **OK**.
The field should now show the calendar icon.
5. Click **Load data**.

Now you are ready to start using the STL functions to visually represent your data.

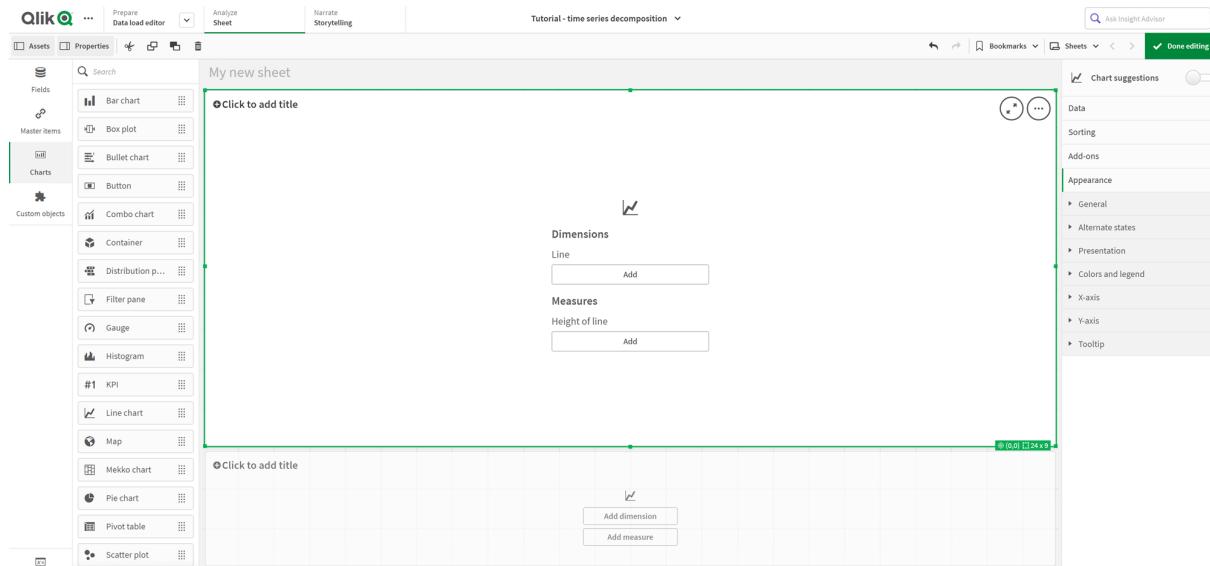
Create the visualizations

Next, you will create two line charts to demonstrate the functionality of the **STL_Trend**, **STL_Seasonal**, and **STL_Residual** chart functions.

Open a new sheet and give it a title.

Add two line charts to the sheet. Resize and reposition the charts to match the following image.

Qlik SenseGrid outline of blank app sheet



First line chart: Trend and seasonal components

Do the following:

1. Add the title *Seasonal and Trend* to the first line chart.
 2. Add *YearMonth* as a dimension, and label it *Date*.
 3. Add the following measure and label it *Passengers per month*:
 $=Sum(Passenger)$
 4. Under **Data**, expand the *Passengers per month* measure and click **Add trend line**.
 5. Set the **Type** to **Linear**.
- You will compare this trend line to the smoothed output of the trend component.

6. Add the following measure to plot the trend component and label it *Trend*:
 $=STL_Trend(SUM(Passengers), 12)$
7. Next, add the following measure to plot the seasonal component and label it *Seasonal*:
 $=STL_Seasonal(SUM(Passengers), 12)$
8. Under **Appearance > Presentation**, set **Scroll bar to None**.
9. Keep the default colors, or change them to fit your preferences.

Second line chart: Residual component

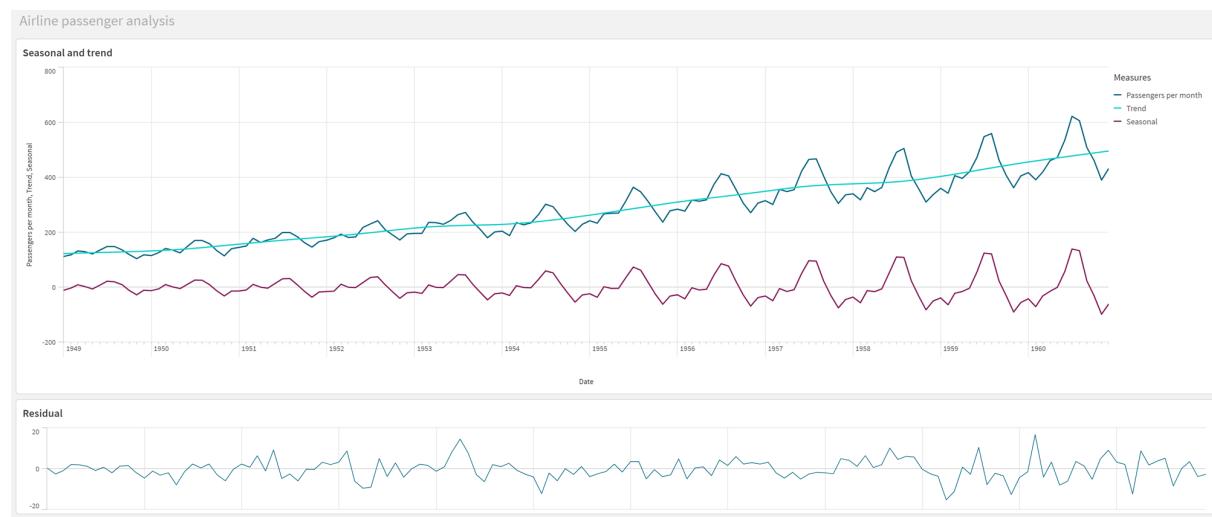
Next, configure the second line chart. This visualization will display the residual component of the time series.

Do the following:

1. Drag a line chart onto the sheet. Add the title *Residual*.
2. Add *Date* as a dimension.
3. Add the following measure and label it *Residual*:
 $=STL_Residual(SUM(Passengers), 12)$
4. Under **Appearance > Presentation**, set **Scroll bar to None**.

Your sheet should now look like the one below.

Qlik Sense sheet for airline passenger analysis



Interpreting and explaining the data

With the STL chart functions, we can gain a number of insights from our time series data.

Trend component

The statistical information in the trend component is deseasonalized. This makes it easier to see general, non-repeating fluctuations over time. Compared to the straight, linear trend line for *Passenger per month*, the STL trend component does capture changing trends. It displays some clear deviations while still presenting the information in a readable fashion. The smoothing behaviors in the STL algorithm helped to capture this.

The drops in number of airline passengers that are visible in the STL trend graph can be explained as part of the economic impact of recessions that occurred during the 1950s.

Seasonal component

The detrended seasonal component isolated recurring fluctuations throughout the time series, and removed general trend information from that part of the analysis. We started with a dataset consisting of year-month aggregations. With this data, it is implicit that we are segmenting the data into one-month granules. By defining a period value of 12, we set the chart to model seasonal patterns over the course of one-year (twelve-month) cycles.

In the data, there is a repeated seasonal pattern of surges in airline passengers in the summer months, followed by declines for the winter months. This is aligned with the idea that summer is typically a popular time to take vacations and travel. We also see that over the course of the time series, these seasonal cycles increase drastically in amplitude.

Residual component

The chart for the residual component shows all the information that was not captured in the trend and seasonal decomposition. The residual component includes statistical noise, but it can also indicate an incorrect setting of the STL trend and seasonal function arguments. Generally, if there are periodic oscillations in the residual component of the signal, or the information displayed is clearly not random, it is usually a sign that there is information in the time series not currently captured in the seasonal or trend components. In this case, you need to revisit your definitions of each function argument and possibly change the periodicity.

Smoother values

Since we did not specify any values for the trend and seasonal smoothers, the function will use the default values for these parameters. In Qlik Sense, the default smoother values in the STL algorithm produce effective results. As a result, in most cases, these arguments can be left out of the expressions.



Setting the seasonal or trend smoother arguments as 0 in either of the three STL functions makes the algorithm use default values, rather than values of 0.

The trend smoother value uses the dimension that is specified in the chart. Since the *YearMonth* field presents data by months, the trend smoother value will be the number of months. The seasonal smoother will reflect the periodicity defined. In this case, since we defined one period as lasting twelve months (one year), the seasonal smoother value is the number of years. This may sound confusing, but it really means that to find the seasonality, we need to look across a number of seasons. This number is the seasonal smoother.

Other useful information

Given that the seasonal cycles increase in amplitude over time, a more advanced analytics approach could make use of logarithmic functions to create a multiplicative decomposition. In practice, a simple measure of relative amplitude can be created in Qlik Sense by dividing the seasonal by the trend component. When this is done, we notice that over time, the summer peaks of each cycle grow larger in relative amplitude. The amplitude of the winter low points, however, do not increase over time.

5.23 Statistical distribution functions

Statistical distribution functions return the probabilities of occurrence of different possible outcomes for a given input variable. You can use these functions to calculate the potential values of your data points.

The three groups of statistical distribution functions described below are all implemented in Qlik Sense using the Cephes function library. For references and details on algorithms used, accuracy, and so on, see: [Cephes library](#). The Cephes function library is used by permission.

- The probability functions calculate the probability at the point in the distribution given by the supplied value.
 - The Frequency functions are used for discrete distributions.
 - The Density functions are used for continuous functions.
- The Dist functions calculate the accumulated probability of the distribution at the point in the distribution given by the supplied value.
- The Inv functions calculate the inverse value, given the accumulated probability of the distribution.

All functions can be used in both the data load script and in chart expressions.

Statistical distribution functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

BetaDensity

BetaDensity() returns the probability of the Beta distribution.

```
BetaDensity (value, alpha, beta)
```

BetaDist

BetaDist() returns the accumulated probability of the Beta distribution.

```
BetaDist (value, alpha, beta)
```

BetaInv

BetaINV() returns the inverse of the accumulated probability of the Beta distribution.

```
BetaInv (prob, alpha, beta)
```

BinomDist

BinomDist() returns the accumulated probability of the Binomial distribution.

```
BinomDist (value, trials, trial_probability)
```

BinomFrequency

BinomFrequency() returns the Binomial probability distribution.

BinomFrequency (value, trials, trial_probability)

BinomInv

BinomInv() returns the inverse of the accumulated probability of the Binomial distribution.

BinomInv (prob, trials, trial_probability)

ChiDensity

ChiDensity() returns the one-tailed probability of the chi² distribution. The chi² density function is associated with a chi² test.

ChiDensity (value, degrees_freedom)

ChiDist

ChiDist() returns the one-tailed probability of the chi² distribution. The chi² distribution is associated with a chi² test.

ChiDist (value, degrees_freedom)

ChiInv

ChiInv() returns the inverse of the one-tailed probability of the chi² distribution.

ChiInv (prob, degrees_freedom)

FDensity

FDensity() returns the probability of the F distribution.

FDensity (value, degrees_freedom1, degrees_freedom2)

FDist

FDist() returns the accumulated probability of the F distribution.

FDist (value, degrees_freedom1, degrees_freedom2)

FInv

FInv() returns the inverse of the accumulated probability of the F distribution.

FInv (prob, degrees_freedom1, degrees_freedom2)

GammaDensity

GammaDensity() returns the probability of the Gamma distribution.

GammaDensity (value, k, θ)

GammaDist

GammaDist() returns the accumulated probability of the Gamma distribution.

GammaDist (value, k, θ)

GammaInv

GammaInv() returns the inverse of the accumulated probability of the Gamma distribution.

GammaInv (prob, k, θ)

NormDist

NormDist() returns the cumulative normal distribution for the specified mean and standard deviation. If mean = 0 and standard_dev = 1, the function returns the standard normal distribution.

```
NormDist (value, mean, standard_dev)
```

NormInv

NormInv() returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

```
NormInv (prob, mean, standard_dev)
```

PoissonDist

PoissonDist() returns the accumulated probability of the Poisson distribution.

```
PoissonDist (value, mean)
```

PoissonFrequency

PoissonFrequency() returns the Poisson probability distribution.

```
PoissonFrequency (value, mean)
```

PoissonInv

PoissonInv() returns the inverse of the accumulated probability of the Poisson distribution.

```
PoissonInv (prob, mean)
```

TDensity

TDensity() returns the value for the student's t density function where a numeric value is a calculated value of t for which the probability is to be computed.

```
TDensity (value, degrees_freedom, tails)
```

TDist

TDist() returns the probability for the student's t distribution where a numeric value is a calculated value of t for which the probability is to be computed.

```
TDist (value, degrees_freedom, tails)
```

TInv

TInv() returns the t value of the student's t distribution as a function of the probability and the degrees of freedom.

```
TInv (prob, degrees_freedom)
```

See also:

- *Statistical aggregation functions (page 382)*

BetaDensity

`BetaDensity()` returns the probability of the Beta distribution.

Syntax:

```
BetaDensity(value, alpha, beta)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must be between 0 and 1.
alpha	A positive number defining the first shape parameter. It is the exponent of the random variable
beta	A positive number defining the second shape parameter. It states the number of denominator degrees of freedom.

BetaDist

`BetaDist()` returns the accumulated probability of the Beta distribution.

Syntax:

```
BetaDist(value, alpha, beta)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must be between 0 and 1.
alpha	A positive number defining the first shape parameter. It is the exponent of the random variable
beta	A positive number defining the second shape parameter. It is the exponent that controls the shape of the distribution.

This function is related to the `BetaInv` function in the following way:

If `prob = BetaDist(value, alpha, beta)`, then `BetaInv(prob, alpha, beta) = value`

BetaInv

`BetaInv()` returns the inverse of the accumulated probability of the Beta distribution.

Syntax:

```
BetaInv(prob, alpha, beta)
```

Return data type: number

Arguments

Argument	Description
prob	A probability associated with the Beta-probability distribution. It must be a number between 0 and 1.
alpha	A positive number defining the first shape parameter. It is the exponent of the random variable
beta	A positive number defining the second shape parameter. It is the exponent that controls the shape of the distribution.

This function is related to the `Betadist` function in the following way:

```
If prob = Betadist(value, alpha, beta), then BetaInv(prob, alpha, beta) = value
```

BinomDist

`BinomDist()` returns the accumulated probability of the Binomial distribution.

Syntax:

```
BinomDist(value, trials, trial_probability)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must be an integer not smaller than zero and not greater than the number of trials.
trials	A positive integer that states the number of trials.
trial_probability	The success probability for each trial. It is always a number between 0 and 1.

This function is related to the `BinomInv` function in the following way:

```
If prob = BinomDIST(value, trials, trial_probability), then BinomInv(prob, trials, trial_probability) = value
```

BinomFrequency

`BinomFrequency()` returns the Binomial probability distribution.

Syntax:

```
BinomFrequency(value, trials, trial_probability)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must be an integer not smaller than zero and not greater than the number of trials.
trials	A positive integer that states the number of trials
trial_probability	The success probability for each trial. It is always a number between 0 and 1.

BinomInv

`BinomInv()` returns the inverse of the accumulated probability of the Binomial distribution.

Syntax:

```
BinomInv(prob, trials, trial_probability)
```

Return data type: number

Arguments

Argument	Description
prob	A probability associated with the Binomial-probability distribution. It must be a number between 0 and 1.
trials	A positive integer that states the number of trials.
trial_probability	The success probability for each trial. It is always a number between 0 and 1.

This function is related to the `BinomDist` function in the following way:

If `prob = BinomDist(value, trials, trial_probability)`, then `BinomInv(prob, trials, trial_probability) = value`

ChiDensity

`ChiDensity()` returns the one-tailed probability of the χ^2 distribution. The χ^2 density function is associated with a χ^2 test.

Syntax:

```
ChiDensity(value, degrees_freedom)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom	A positive integer stating the number of numerator degrees of freedom.

ChiDist

chiDist() returns the one-tailed probability of the chi² distribution. The chi² distribution is associated with a chi² test.

Syntax:

```
CHIDIST(value, degrees_freedom)
```

Return data type: number

Arguments:

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom	A positive integer stating the number of degrees of freedom.

This function is related to the **Chilnv** function in the following way:

If prob = CHIDIST(value,df), then CHIINV(prob, df) = value

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
CHIDIST(8, 15)	Returns 0.9238

Chilnv

chiInv() returns the inverse of the one-tailed probability of the chi² distribution.

Syntax:

```
CHIINV(prob, degrees_freedom)
```

Return data type: number

Arguments:

Arguments	
Argument	Description
prob	A probability associated with the χ^2 distribution. It must be a number between 0 and 1.
degrees_freedom	An integer stating the number of degrees of freedom.

This function is related to the **ChiDist** function in the following way:

If `prob = CHIDIST(value,df)`, then `CHIINV(prob, df) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>CHIINV(0.9237827, 15)</code>	Returns 8.0000

FDensity

`FDensity()` returns the probability of the F distribution.

Syntax:

```
FDensity(value, degrees_freedom1, degrees_freedom2)
```

Return data type: number

Arguments	
Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom1	A positive integer stating the number of numerator degrees of freedom.
degrees_freedom2	A positive integer stating the number of denominator degrees of freedom.

FDist

`FDist()` returns the accumulated probability of the F distribution.

Syntax:

```
FDist(value, degrees_freedom1, degrees_freedom2)
```

Return data type: number

Arguments:

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom1	A positive integer stating the number of numerator degrees of freedom.
degrees_freedom2	A positive integer stating the number of denominator degrees of freedom.

This function is related to the **FInv** function in the following way:

If `prob = FDIST(value, df1, df2)`, then `FINV(prob, df1, df2) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>FDIST(15, 8, 6)</code>	Returns 0.0019

FInv

FInv() returns the inverse of the accumulated probability of the F distribution.

Syntax:

```
FInv(prob, degrees_freedom1, degrees_freedom2)
```

Return data type: number

Arguments:

Arguments

Argument	Description
prob	A probability associated with the F-probability distribution and must be a number between 0 and 1.
degrees_freedom	An integer stating the number of degrees of freedom.

This function is related to the **FDist** function in the following way:

If `prob = FDIST(value, df1, df2)`, then `FINV(prob, df1, df2) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>FINV(0.0019369, 8, 6)</code>	Returns 15.0000

GammaDensity

`GammaDensity()` returns the probability of the Gamma distribution.

Syntax:

```
GammaDensity(value, k, θ)
```

Return data type: number

Arguments

Argument	Description
<code>value</code>	The value at which you want to evaluate the distribution. The value must not be negative.
<code>k</code>	A positive number defining the shape parameter.
<code>θ</code>	A positive number defining the scale parameter.

GammaDist

`GammaDist()` returns the accumulated probability of the Gamma distribution.

Syntax:

```
GammaDist(value, k, θ)
```

Return data type: number

Arguments

Argument	Description
<code>value</code>	The value at which you want to evaluate the distribution. The value must not be negative.
<code>k</code>	A positive number defining the shape parameter.
<code>θ</code>	A positive number defining the scale parameter.

This function is related to the `GammaInv` function in the following way:

If `prob = GammaDist(value, k, θ)`, then `GammaInv(prob, k, θ) = value`

Gammaln

GammaInv() returns the inverse of the accumulated probability of the Gamma distribution.

Syntax:

```
GammaInv(prob, k, θ)
```

Return data type: number

Arguments

Argument	Description
prob	A probability associated with the Gamma-probability distribution. It must be a number between 0 and 1.
k	A positive number defining the shape parameter.
θ	A positive number defining the scale parameter.

This function is related to the GammaDist function in the following way:

If `prob = GammaDist(value, k, θ)`, then `GammaInv(prob, k, θ) = value`

NormDist

NormDist() returns the cumulative normal distribution for the specified mean and standard deviation. If `mean = 0` and `standard_dev = 1`, the function returns the standard normal distribution.

Syntax:

```
NORMDIST(value, [mean], [standard_dev], [cumulative])
```

Return data type: number

Arguments:

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution.
mean	Optional value stating the arithmetic mean for the distribution. If you do not state this argument, the default value is 0.
standard_dev	Optional positive value stating the standard deviation of the distribution. If you do not state this argument, the default value is 1.

Argument	Description
cumulative	You can optionally select to use a standard normal distribution or a cumulative distribution. 0 = standard normal distribution 1 = cumulative distribution (default)

This function is related to the **NormInv** function in the following way:

If `prob = NORMDIST(value, m, sd)`, then `NORMINV(prob, m, sd) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>NORMDIST(0.5, 0, 1)</code>	Returns 0.6915

NormInv

`NormInv()` returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

Syntax:

```
NORMINV(prob, mean, standard_dev)
```

Return data type: number

Arguments:

Arguments

Argument	Description
prob	A probability associated with the normal distribution. It must be a number between 0 and 1.
mean	A value stating the arithmetic mean for the distribution.
standard_dev	A positive value stating the standard deviation of the distribution.

This function is related to the **NormDist** function in the following way:

If `prob = NORMDIST(value, m, sd)`, then `NORMINV(prob, m, sd) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>NORMINV(0.6914625, 0, 1)</code>	Returns 0.5000

PoissonDist

`PoissonDist()` returns the accumulated probability of the Poisson distribution.

Syntax:

```
PoissonDist(value, mean)
```

Return data type: number

Arguments

Argument	Description
<code>value</code>	The value at which you want to evaluate the distribution. The value must not be negative.
<code>mean</code>	A positive number defining the average outcome.

This function is related to the `PoissonInv` function in the following way:

If `prob = PoissonDist(value, mean)`, then `PoissonInv(prob, mean) = value`

PoissonFrequency

`PoissonFrequency()` returns the Poisson probability distribution.

Syntax:

```
PoissonFrequency(value, mean)
```

Return data type: number

Arguments

Argument	Description
<code>value</code>	The value at which you want to evaluate the distribution. The value must not be negative.
<code>mean</code>	A positive number defining the average outcome.

PoissonInv

`PoissonInv()` returns the inverse of the accumulated probability of the Poisson distribution.

Syntax:

```
PoissonInv(prob, mean)
```

Return data type: number

Arguments

Argument	Description
prob	A probability associated with the Poisson-probability distribution. It must be a number between 0 and 1.
mean	A positive number defining the average outcome.

This function is related to the `PoissonDIST` function in the following way:

If `prob = PoissonDist(value, mean)`, then `PoissonInv(prob, mean) = value`

TDensity

`TDensity()` returns the value for the student's t density function where a numeric value is a calculated value of t for which the probability is to be computed.

Syntax:

```
TDensity(value, degrees_freedom)
```

Return data type: number

Arguments

Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom	A positive integer stating the number of degrees of freedom.

TDist

`TDist()` returns the probability for the student's t distribution where a numeric value is a calculated value of t for which the probability is to be computed.

Syntax:

```
TDist(value, degrees_freedom, tails)
```

Return data type: number

Arguments:

Arguments	
Argument	Description
value	The value at which you want to evaluate the distribution. The value must not be negative.
degrees_freedom	A positive integer stating the number of degrees of freedom.
tails	Must be either 1 (one-tailed distribution) or 2 (two-tailed distribution).

This function is related to the **TInv** function in the following way:

If `prob = TDIST(value, df, 2)`, then `TINV(prob, df) = value`

Limitations:

All arguments must be numeric, else NULL will be returned.

Examples and results:

Example	Result
<code>TDIST(1, 30, 2)</code>	Returns 0.3253

TInv

`TINV()` returns the t value of the student's t distribution as a function of the probability and the degrees of freedom.

Syntax:

`TINV(prob, degrees_freedom)`

Return data type: number

Arguments:

Arguments	
Argument	Description
prob	A two-tailed probability associated with the t-distribution. It must be a number between 0 and 1.
degrees_freedom	An integer stating the number of degrees of freedom.

Limitations:

All arguments must be numeric, else NULL will be returned.

This function is related to the **TDist** function in the following way:

If `prob = TDIST(value, df, 2)`, then `TINV(prob, df) = value`.

Examples and results:

Example	Result
<code>TINV(0.3253086, 30)</code>	Returns 1.0000

5.24 String functions

This section describes functions for handling and manipulating strings.

All functions can be used in both the data load script and in chart expressions, except for **Evaluate** which can only be used in the data load script.

String functions overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Capitalize

Capitalize() returns the string with all words in initial uppercase letters.

`Capitalize (text)`

Chr

Chr() returns the Unicode character corresponding to the input integer.

`Chr (int)`

Evaluate

Evaluate() finds if the input text string can be evaluated as a valid Qlik Sense expression, and if so, returns the value of the expression as a string. If the input string is not a valid expression, NULL is returned.

`Evaluate (expression_text)`

FindOneOf

FindOneOf() searches a string to find the position of the occurrence of any character from a set of provided characters. The position of the first occurrence of any character from the search set is returned unless a third argument (with a value greater than 1) is supplied. If no match is found, **0** is returned.

`FindOneOf (text, char_set[, count])`

Hash128

Hash128() returns a 128-bit hash of the combined input expression values. The result is a 22-character string.

Hash128 (expr{, expression})

Hash160

Hash160() returns a 160-bit hash of the combined input expression values. The result is a 27-character string.

Hash160 (expr{, expression})

Hash256

Hash256() returns a 256-bit hash of the combined input expression values. The result is a 43-character string.

Hash256 (expr{, expression})

Index

Index() searches a string to find the starting position of the nth occurrence of a provided substring. An optional third argument provides the value of n, which is 1 if omitted. A negative value searches from the end of the string. The positions in the string are numbered from 1 and up.

Index (text, substring[, count])

IsJson

IsJson() tests whether a specified string contains valid JSON (JavaScript Object Notation) data. You can also validate a specific JSON data type.

IsJson (json [, type])

JsonGet

JsonGet() returns the path of a JSON (JavaScript Object Notation) data string. The data must be valid JSON but can contain extra spaces or newlines.

JsonGet (json, path)

JsonSet

JsonSet() modifies a string containing JSON (JavaScript Object Notation) data. It can set or insert a JSON value with the new location specified by the path. The data must be valid JSON but can contain extra spaces or newlines.

JsonSet(json, path, value)

KeepChar

KeepChar() returns a string consisting of the first string , 'text', less any of the characters NOT contained in the second string, "keep_chars".

KeepChar (text, keep_chars)

Left

Left() returns a string consisting of the first (leftmost) characters of the input string, where the number of characters is determined by the second argument.

Left (text, count)

Len

Len() returns the length of the input string.

Len (`text`)

LevenshteinDist

LevenshteinDist() returns the Levenshtein distance between two strings. It is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into the other. The function is useful for fuzzy string comparisons.

LevenshteinDist (`text1, text2`)

Lower

Lower() converts all the characters in the input string to lower case.

Lower (`text`)

LTrim

LTrim() returns the input string trimmed of any leading spaces.

LTrim (`text`)

Mid

Mid() returns the part of the input string starting at the position of the character defined by the second argument, 'start', and returning the number of characters defined by the third argument, 'count'. If 'count' is omitted, the rest of the input string is returned. The first character in the input string is numbered 1.

Mid (`text, start[, count]`)

Ord

Ord() returns the Unicode code point number of the first character of the input string.

Ord (`text`)

PurgeChar

PurgeChar() returns a string consisting of the characters contained in the input string ('text'), excluding any that appear in the second argument ('remove_chars').

PurgeChar (`text, remove_chars`)

Repeat

Repeat() forms a string consisting of the input string repeated the number of times defined by the second argument.

Repeat (`text[, repeat_count]`)

Replace

Replace() returns a string after replacing all occurrences of a given substring within the input string with another substring. The function is non-recursive and works from left to right.

Replace (`text, from_str, to_str`)

Right

Right() returns a string consisting of the last (rightmost) characters of the input string, where the number of characters is determined by the second argument.

```
Right (text, count)
```

RTrim

RTrim() returns the input string trimmed of any trailing spaces.

```
RTrim (text)
```

SubField

SubField() is used to extract substring components from a parent string field, where the original record fields consist of two or more parts separated by a delimiter.

```
SubField (text, delimiter[, field_no ])
```

SubStringCount

SubStringCount() returns the number of occurrences of the specified substring in the input string text. If there is no match, 0 is returned.

```
SubStringCount (text, substring)
```

TextBetween

TextBetween() returns the text in the input string that occurs between the characters specified as delimiters.

```
TextBetween (text, delimiter1, delimiter2[, n])
```

Trim

Trim() returns the input string trimmed of any leading and trailing spaces.

```
Trim (text)
```

Upper

Upper() converts all the characters in the input string to upper case for all text characters in the expression. Numbers and symbols are ignored.

```
Upper (text)
```

Capitalize

Capitalize() returns the string with all words in initial uppercase letters.

Syntax:

```
Capitalize(text)
```

Return data type: string

Example: Chart expressions

Example	Result
Capitalize ('star trek')	Returns 'Star Trek'
Capitalize ('AA bb cc Dd')	Returns 'Aa Bb Cc Dd'

Example: Load script

```
Load
String,
Capitalize(String)
Inline
[String
rHode iSland
washiNgTon d.C.
new york];
```

Result

String	Capitalize(String)
rHode iSland	Rhode Island
washiNgTon d.C.	Washington D.C.
new york	New York

Chr

Chr() returns the Unicode character corresponding to the input integer.

Syntax:

```
Chr(int)
```

Return data type: string

Examples and results:

Example	Result
Chr(65)	Returns the string 'A'
Chr(163)	Returns the string '£'
Chr(35)	Returns the string '#'

Evaluate

Evaluate() finds if the input text string can be evaluated as a valid Qlik Sense expression, and if so, returns the value of the expression as a string. If the input string is not a valid expression, NULL is returned.

Syntax:

```
Evaluate(expression_text)
```

Return data type: dual



This string function cannot be used in chart expressions.

Examples and results:

Function example	Result
Evaluate (5 * 8)	Returns '40'

Load script example

```
Load
Evaluate(String) as Evaluated,
String
Inline
[String
4
5+3
0123456789012345678
Today()
];
```

Result

String	Evaluated
4	4
5+3	8
0123456789012345678	0123456789012345678
Today()	2022-02-02

FindOneOf

FindOneOf() searches a string to find the position of the occurrence of any character from a set of provided characters. The position of the first occurrence of any character from the search set is returned unless a third argument (with a value greater than 1) is supplied. If no match is

found, **0** is returned.

Syntax:

```
FindOneOf(text, char_set[, count])
```

Return data type: integer

Arguments:

Arguments

Argument	Description
text	The original string.
char_set	A set of characters to search for in text.
count	Defines which occurrence of any of the character to search for. For example, a value of 2 searches for the second occurrence.

Example: Chart expressions

Example	Result
FindOneOf('my example text string', 'et%s')	Returns '4' because 'e' is the fourth character in the example string.
Findoneof('my example text string', 'et%s', 3)	Returns '12' because the search is for any of the characters e, t, % or s, and "t" is the third occurrence in position 12 of the example string.
Findoneof('my example text string', '%&')	Returns '0' because none of the characters %, &, or & exist in the example string.

Example: Load script

```
Load *
Inline
[SearchFor, occurrence
et%s,1
et%s,3
¤%,1]
```

Result

SearchFor	Occurrence	FindOneOf('my example text string', SearchFor, Occurrence)
et%s	1	4
et%s	3	12
¤%,	1	0

Hash128

Hash128() returns a 128-bit hash of the combined input expression values. The result is a 22-character string.

Syntax:

```
Hash128(expr{, expression})
```

Return data type: string

Example: Chart expressions

Example	Result
Hash128 ('abc', 'xyz', '123')	Returns 'MA&5]6+3=:>;G%S<U*S2+'.
Hash128 (Region, Year, Month)	Returns 'G7*=6GKPJ(Z+)^KM?<\$'A+'.
Note: Region, Year, and Month are table fields.	

Example: Load script

```
Hash_128:  
Load *,  
Hash128(Region, Year, Month) as Hash128;  
Load * inline [  
Region, Year, Month  
abc, xyz, 123  
EU, 2022, 01  
UK, 2022, 02  
US, 2022, 02 ];
```

Result

Region	Year	Month	Hash128
abc	xyz	123	MA&5]6+3=:>;G%S<U*S2+
EU	2022	01	B40^K&[T@!;VB'XR]<5=/\$
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!
US	2022	02	C6@#]4#_G-()J7EQY#KRW0

Hash160

Hash160() returns a 160-bit hash of the combined input expression values. The result is a 27-character string.

Syntax:

```
Hash160(expr{, expression})
```

Return data type: string

Example: Chart expressions

Example	Result
Hash160 ('abc', 'xyz', '123')	Returns 'MA&5]6+3=:>;>G%S<U*S2I:`=X*.'
Hash160 (Region, Year, Month) Note: Region, Year, and Month are table fields.	Returns 'G7*=6GKPJ (Z+)^KM?<\$'AI.)?U\$'.

Example: Load script

```
Hash_160:  
Load *,  
Hash160(Region, Year, Month) as Hash160;  
Load * inline [  
Region, Year, Month  
abc, xyz, 123  
EU, 2022, 01  
UK, 2022, 02  
US, 2022, 02];
```

Result

Region	Year	Month	Hash160
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2I:`=X*'
EU	2022	01	B40^K&[T@!;VB'XR]<5//_F853
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZ
US	2022	02	C6@#]4#_G-(JJ7EQY#KRW`@KF+W

Hash256

Hash256() returns a 256-bit hash of the combined input expression values. The result is a 43-character string.

Syntax:

```
Hash256(expr{, expression})
```

Return data type: string

Example: Chart expressions

Example	Result
Hash256 ('abc', 'xyz', '123')	Returns 'MA&5]6+3=:=>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ'.
Hash256 (Region, Year, Month) Note: Region, Year, and Month are table fields.	Returns 'G7*=6GKPJ(Z+)^KM?<\$'AI.)?U\$#X2RB [:0ZP=+Z` F!'. Note: Region, Year, and Month are table fields.

Example: Load script

```
Hash_256:  
Load *,  
Hash256(Region, Year, Month) as Hash256;  
Load * inline [  
Region, Year, Month  
abc, xyz, 123  
EU, 2022, 01  
UK, 2022, 02  
US, 2022, 02];
```

Result

Region	Year	Month	Hash256
abc	xyz	123	MA&5]6+3=:=>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ
EU	2022	01	B40^K&[T@!:VB'XR]<5=//_F853?BE6'G&,YH*T'MF)
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZT=4#\#V` M%6_\0C>4
US	2022	02	C6@#]4#_G-(]J7EQY#KRW` @KF+W-0` [Z8R+#"")=+0

Index

Index() searches a string to find the starting position of the nth occurrence of a provided substring. An optional third argument provides the value of n, which is 1 if omitted. A negative value searches from the end of the string. The positions in the string are numbered from **1** and up.

Syntax:

```
Index(text, substring[, count])
```

Return data type: integer

Arguments:

Arguments

Argument	Description
text	The original string.
substring	A string of characters to search for in text.
count	Defines which occurrence of substring to search for. For example, a value of 2 searches for the second occurrence.

Examples and results:

Example	Result
Index('abcdefg', 'cd')	Returns 3
Index('abcdabcd', 'b', 2)	Returns 6 (the second occurrence of 'b')
Index('abcdabcd', 'b', -2)	Returns 2 (the second occurrence of 'b' starting from the end)
Left(Date, Index(Date, '-') -1) where Date = 1997-07-14	Returns 1997
Mid(Date, Index(Date, ' - ', 2) -2, 2) where Date = 1997-07-14	Returns 07

Example: Script

```
T1:
Load
 *,
index(String, 'cd') as Index_CD,          // returns 3 in Index_CD
index(String, 'b') as Index_B,           // returns 2 in Index_B
index(String, 'b', -1) as Index_B2;      // returns 2 or 6 in Index_B2
Load * inline [
String
abcdefg
abcdabcd ];
```

IsJson

IsJson() tests whether a specified string contains valid JSON (JavaScript Object Notation) data. You can also validate a specific JSON data type.

Syntax:

```
value IsJson(json [, type])
```

Return data type: dual

Arguments

Argument	Description
json	String to test. It can contain extra spaces or newlines.
type	Optional argument that specifies the JSON data type to test for. <ul style="list-style-type: none"> • 'value' (default) • 'object' • 'array' • 'string' • 'number' • 'Boolean' • 'null'

Example: Valid JSON and type

Example	Result
<code>IsJson('null')</code>	Returns -1 (true)
<code>IsJson('"abc"', 'value')</code>	Returns -1 (true)
<code>IsJson('"abc"', 'string')</code>	Returns -1 (true)
<code>IsJson(123, 'number')</code>	Returns -1 (true)

Example: Invalid JSON or type

Example	Result	Description
<code>IsJson('text')</code>	Returns 0 (false)	'text' is not a valid JSON value
<code>IsJson('"text"', 'number')</code>	Returns 0 (false)	""text"" is not a valid JSON number
<code>IsJson('"text"', 'text')</code>	Returns 0 (false)	'text' is not a valid JSON type

JsonGet

JsonGet() returns the path of a JSON (JavaScript Object Notation) data string. The data must be valid JSON but can contain extra spaces or newlines.

Syntax:

```
value JsonGet(json, path)
```

Return data type: dual

Arguments

Argument	Description
json	String containing JSON data.
path	The path must be specified according to RFC 6901 . This will allow lookup of properties inside JSON data without using complex substring or index functions.

Example: Valid JSON and path

Example	Result
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '')	Returns '{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}'
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/a')	Returns '{"foo":"bar"}'
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/a/foo')	Returns '"bar"
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/b')	Returns '[123, "abc", "ABC"]'
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/b/0')	Returns '123'
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/b/1')	Returns "abc"
JsonGet('{"a":{"foo":"bar"}, "b": [123, "abc", "ABC"]}', '/b/2')	Returns "ABC"

Example: Invalid JSON or path

Example	Result	Description
JsonGet('{"a": "b"}', '/b')	Returns null	The path does not point to a valid part of the JSON data.
JsonGet('{"a"}', '/a')	Returns null	The JSON data is not valid JSON (member "a" does not have a value).

JsonSet

JsonSet() modifies a string containing JSON (JavaScript Object Notation) data. It can set or insert a JSON value with the new location specified by the path. The data must be valid JSON but can contain extra spaces or newlines.

Syntax:

```
value JsonSet(json, path, value)
```

Return data type: dual

Arguments

Argument	Description
json	String containing JSON data.
path	The path must be specified according to RFC 6901 . This allows buildup of properties inside JSON data without using complex substring or index functions and concatenation.
value	The new string value in JSON format.

Example: Valid JSON, path, and value

Example	Result
<code>JsonSet('{}",'/a',"b")</code>	Returns '{"a":"b"}'
<code>JsonSet('[]','/0',"x")</code>	Returns '[{"x"}]
<code>JsonSet('"abc"','','123')</code>	Returns 123

Example: Invalid JSON, path, or value

Example	Result	Description
<code>JsonSet('"abc"','/x',123)</code>	Returns null	The path does not point to a valid part of the JSON data.
<code>JsonSet('{"a": {"b": "c"}},'a/b',"x")</code>	Returns null	The path is invalid.
<code>JsonSet('{"a": "b"}','/a','abc')</code>	Returns null	The value is not valid JSON. A string must be enclosed in quotes.

KeepChar

KeepChar() returns a string consisting of the first string , 'text', less any of the characters NOT contained in the second string, "keep_chars".

Syntax:

```
KeepChar(text, keep_chars)
```

Return data type: string

Arguments:

Arguments	
Argument	Description
text	The original string.
keep_chars	A string containing the characters in text to be kept.

Example: Chart expressions

Example	Result
KeepChar ('a1b2c3','123')	Returns '123'.
KeepChar ('a1b2c3','1234')	Returns '123'.
KeepChar ('a1b22c3','1234')	Returns '1223'.
KeepChar ('a1b2c3','312')	Returns '123'.

Example: Load script

```
T1:  
Load  
*,  
keepchar(String1, String2) as KeepChar;  
Load * inline [  
String1, String2  
'a1b2c3', '123'  
];
```

Results

Qlik Sense table showing the output from using the *KeepChar* function in the load script.

String1	String2	KeepChar
a1b2c3	123	123

See also:

- [PurgeChar \(page 1425\)](#)

Left

Left() returns a string consisting of the first (leftmost) characters of the input string, where the number of characters is determined by the second argument.

Syntax:

```
Left(text, count)
```

Return data type: string

Arguments:

Argument	Description
text	The original string.
count	Defines the number of characters to included from the left-hand part of the string text .

Example: Chart expression

Example	Result
<code>Left('abcdef', 3)</code>	Returns 'abc'

Example: Load script

T1:
Load
*,
`left(Text,Start) as Left;`
Load * inline [
Text, Start
'abcdef', 3
'2021-07-14', 4
'2021-07-14', 2
];

Result

Qlik Sense table showing the output from using the *Left* function in the load script.

Text	Start	Left
abcdef	3	abc
2021-07-14	4	2021
2021-07-14	2	20

□ See also *Index* (page 1412), which allows more complex string analysis.

Len

Len() returns the length of the input string.

Syntax:

`Len(text)`

Return data type: integer

Example: Chart expression

Example	Result
Len('Peter')	Returns '5'

Example: Load script

```
T1:
Load String, First&Second as NewString;
Load *, mid(String,len(First)+1) as Second;
Load *, upper(left(String,1)) as First;
Load * inline [
String
this is a sample text string
capitalize first letter only ];
```

Result

String	NewString
this is a sample text string	This is a sample text string
capitalize first letter only	Capitalize first letter only

LevenshteinDist

LevenshteinDist() returns the Levenshtein distance between two strings. It is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into the other. The function is useful for fuzzy string comparisons.

Syntax:

```
LevenshteinDist(text1, text2)
```

Return data type: integer

Example: Chart expression

Example	Result
LevenshteinDist('Kitten','sitting')	Returns '3'

Example: Load script

Load script

```
T1:
Load *, recno() as ID;
Load 'silver' as String_1,* inline [
String_2
```

```

Sliver
SSiver
SSiveer ];

T1:
Load *, recno()+3 as ID;
Load 'Gold' as String_1,* inline [
String_2
Bold
Bool
Bond ];

T1:
Load *, recno()+6 as ID;
Load 'Ove' as String_1,* inline [
String_2
Ove
Uve
Üve ];

T1:
Load *, recno()+9 as ID;
Load 'ABC' as String_1,* inline [
String_2
DEFG
abc
ЀЀЀ ];

set nullinterpret = '<NULL>';
T1:
Load *, recno()+12 as ID;
Load 'X' as String_1,* inline [
String_2
 ''
<NULL>
1 ];

R1:
Load
ID,
String_1,
String_2,
LevenshteinDist(String_1, String_2) as LevenshteinDistance
resident T1;

Drop table T1;

```

Result

ID	String_1	String_2	LevenshteinDistance
1	Silver	Sliver	2

ID	String_1	String_2	LevenshteinDistance
2	Silver	SSiver	2
3	Silver	SSiveer	3
4	Gold	Bold	1
5	Gold	Bool	3
6	Gold	Bond	2
7	Ove	Ove	0
8	Ove	Uve	1
9	Ove	Üve	1
10	ABC	DEFG	4
11	ABC	abc	3
12	ABC	ビビビ	3
13	X		1
14	X	-	1
15	X	1	1

Lower

Lower() converts all the characters in the input string to lower case.

Syntax:

```
Lower(text)
```

Return data type: string

Example: Chart expression

Example	Result
Lower('abcd')	Returns 'abcd'

Example: Load script

```
Load
String,
Lower(String)
Inline
[String
rHode island
washingTon d.c.
new york];
```

Result

String	Lower(String)
rHode iSlAnd	rhode island
washingTon d.C.	washington d.c.
new york	new york

LTrim

LTrim() returns the input string trimmed of any leading spaces.

Syntax:

```
LTrim(text)
```

Return data type: string

Example: Chart expressions

Example	Result
LTrim(' abc')	Returns 'abc'
LTrim('abc ')	Returns 'abc '

Example: Load script

```
Set verbatim=1;
T1:
```

```
Load *,
Len(LtrimString) as LtrimStringLength;
Load *,
Ltrim(String) as LtrimString;
Load *,
Len(String) as StringLength;
Load * Inline [
String
' abc
' def '];
```



The "Set verbatim=1" statement is included in the example to ensure that the spaces are not automatically trimmed before the demonstration of the ltrim function. See Verbatim (page 197) for more information.

Result

String	StringLength	LtrimStringLength
def	6	5
abc	10	7

See also:

□ [RTrim \(page 1428\)](#)

Mid

Mid() returns the part of the input string starting at the position of the character defined by the second argument, 'start', and returning the number of characters defined by the third argument, 'count'. If 'count' is omitted, the rest of the input string is returned. The first character in the input string is numbered 1.

Syntax:

```
Mid(text, start[, count])
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	The original string.
start	Integer defining the position of the first character in text to include.
count	Defines the string length of the output string. If omitted, all characters from the position defined by start are included.

Example: Chart expressions

Example	Result
Mid('abcdef',3)	Returns 'cdef'
Mid('abcdef',3, 2)	Returns 'cd'

Example: Load script

```
T1:
Load *, 
mid(Text,Start) as Mid1,
mid(Text,Start,Count) as Mid2;
Load * inline [
```

```
Text, Start, Count
'abcdef', 3, 2
'abcdef', 2, 3
'210714', 3, 2
'210714', 2, 3
];
```

Result

Qlik Sense table showing the output from using the *Mid* function in the load script.

Text	Start	Mid1	Count	Mid2
abcdef	2	bcd	3	bcd
abcdef	3	cde	2	cd
210714	2	10714	3	107
210714	3	0714	2	07

See also:

- [Index \(page 1412\)](#)

Ord

Ord() returns the Unicode code point number of the first character of the input string.

Syntax:

```
Ord(text)
```

Return data type: integer

Examples and results:

Example: Chart expression

Example	Result
ord('A')	Returns the integer 65.
ord('Ab')	Returns the integer 65.

Example: Load script

```
//Guqin (Chinese: 古琴) – 7-stringed zithers
T2:
Load *,
ord(Chinese) as Ordunicode,
ord(Western) as OrdASCII;
Load * inline [
Chinese, Western
```

古琴, Guqin];

Result:

Chinese	Western	OrdASCII	OrdUnicode
古琴	Guqin	71	21476

PurgeChar

PurgeChar() returns a string consisting of the characters contained in the input string ('text'), excluding any that appear in the second argument ('remove_chars').

Syntax:

```
PurgeChar(text, remove_chars)
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	The original string.
remove_chars	A string containing the characters in text to be removed.

Return data type: string

Example: Chart expressions

Example	Result
PurgeChar ('a1b2c3', '123')	Returns 'abc'.
PurgeChar ('a1b2c3', '312')	Returns 'abc'.

Example: Load script

```
T1:  
Load  
*,  
purgechar(String1, String2) as PurgeChar;  
Load * inline [  
String1, String2  
'a1b2c3', '123'  
];
```

Results

Qlik Sense table showing the output from using the *PurgeChar* function in the load script.

String1	String2	PurgeChar
a1b2c3	123	abc

See also:

- [KeepChar \(page 1416\)](#)

Repeat

Repeat() forms a string consisting of the input string repeated the number of times defined by the second argument.

Syntax:

```
Repeat(text[, repeat_count])
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	The original string.
repeat_count	Defines the number of times the characters in the string text are to be repeated in the output string.

Example: Chart expression

Example	Result
Repeat(' * ', rating) when rating = 4	Returns '****'

Example: Load script

T1:

```
Load *,  
repeat(String,2) as Repeat;  
Load * inline [  
String  
hello world!  
how are you? ];
```

Result

String	Repeat
hello world!	hello world!hello world!
hOw aRe you?	hOw aRe you?hOw aRe you?

Replace

Replace() returns a string after replacing all occurrences of a given substring within the input string with another substring. The function is non-recursive and works from left to right.

Syntax:

```
Replace(text, from_str, to_str)
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	The original string.
from_str	A string that may occur one or more times within the input string text .
to_str	The string that will replace all occurrences of from_str within the string text .

Examples and results:

Example	Result
Replace('abccde', 'cc', 'xyz')	Returns 'abxyzde'

See also:

Right

Right() returns a string consisting of the last (rightmost) characters of the input string, where the number of characters is determined by the second argument.

Syntax:

```
Right(text, count)
```

Return data type: string

Arguments:

Arguments	
Argument	Description
text	The original string.
count	Defines the number of characters to be included from the rightmost part of the string text .

Example: Chart expression

Example	Result
Right('abcdef', 3)	Returns 'def'

Example: Load script

T1:

```
Load
 *,
right(Text,Start) as Right;
Load * inline [
Text, Start
'abcdef', 3
'2021-07-14', 4
'2021-07-14', 2
];
```

Result

Qlik Sense table showing the output from using the *Right* function in the load script.

Text	Start	Right
abcdef	3	def
2021-07-14	4	7-14
2021-07-14	2	14

RTrim

RTrim() returns the input string trimmed of any trailing spaces.

Syntax:

```
RTrim(text)
```

Return data type: string

Example: Chart expressions

Example	Result
RTrim(' abc')	Returns ' abc'
RTrim('abc ')	Returns 'abc'

Example: Load script

```
Set verbatim=1;
```

T1:

```
Load *, len(RtrimString) as RtrimStringLength;
Load *, rtrim(String) as RtrimString;
Load *, len(String) as StringLength;
Load * Inline [
String
'    abc      '
' def   '];
```



The "Set verbatim=1" statement is included in the example to ensure that the spaces are not automatically trimmed before the demonstration of the rtrim function. See Verbatim (page 197) for more information.

Result

String	StringLength	RtrimStringLength
def	6	4
abc	10	6

See also:

- [LTrim \(page 1422\)](#)

SubField

SubField() is used to extract substring components from a parent string field, where the original record fields consist of two or more parts separated by a delimiter.

The **Subfield()** function can be used, for example, to extract first name and surname from a list of records consisting of full names, the component parts of a path name, or for extracting data from comma-separated tables.

If you use the **Subfield()** function in a **LOAD** statement with the optional field_no parameter left out, one full record will be generated for each substring. If several fields are loaded using **Subfield()** the Cartesian products of all combinations are created.

Syntax:

```
SubField(text, delimiter[, field_no ])
```

Return data type: string

Arguments:

Arguments

Argument	Description
text	The original string. This can be a hard-coded text, a variable, a dollar-sign expansion, or another expression.
delimiter	A character within the input text that divides the string into component parts.
field_no	The optional third argument is an integer that specifies which of the substrings of the parent string text is to be returned. Use the value 1 to return the first substring, 2 to return the second substring, and so on. <ul style="list-style-type: none">• If field_no is a positive value, substrings are extracted from left to right.• If field_no is a negative value, substrings are extracted from right to left.



SubField() can be used instead of using complex combinations of functions such as Len(), Right(), Left(), Mid(), and other string functions.

Examples: Script and chart expressions using SubField

Examples - script and chart expressions

Basic examples

Example	Result
SubField(\$, ';' ,2)	Returns 'cde' if \$ is 'abc;cde;efg'.
SubField(\$, ';' ,1)	Returns an empty string if \$ is an empty string.
SubField(\$, ';' ,1)	Returns an empty string if \$ is ':'.

Example	Result
Suppose you have a variable that holds a path name vMyPath, <pre>Set vMyPath=\Users\ext_jrb\Documents\Qlik\Sense\Apps;;</pre>	In a text & image chart, you can add a measure such as: SubField(vMyPath, '\',-3), which results in 'Qlik', because it is the substring third from the right-hand end of the variable vMyPath.

Script example 1

Load script

Load the following script expressions and data in the data load editor.

FullName:

```
LOAD * inline [
Name
'Dave Owen'
'Joe Tem'
];
```

SepNames:

```
Load Name,
SubField(Name, ' ',1) as FirstName,
SubField(Name, ' ',-1) as Surname
Resident FullName;
Drop Table FullName;
```

Create a visualization

Create a table visualization in a Qlik Sense sheet with **Name**, **FirstName**, and **SurName** as dimensions.

Result

Name	FirstName	SurName
Dave Owen	Dave	Owen
Joe Tem	Joe	Tem

Explanation

The **SubField()** function extracts the first substring of **Name** by setting the **field_no** argument to 1. Since the value of **field_no** is positive, a left to right order is followed for extracting the substring. A second function call extracts the second substring by setting the **field_no** argument to -1, which extracts the substring following a right to left order.

Script example 2

Load script

Load the following script expressions and data in the data load editor.

```
LOAD DISTINCT
Instrument,
SubField(Player,',') as Player,
SubField(Project,',') as Project;

Load * inline [
Instrument|Player|Project
Guitar|Neil,Mike|Music,Video
Guitar|Neil|Music,OST
Synth|Neil,Jen|Music,Video,OST
Synth|Jo|Music
Guitar|Neil,Mike|Music,OST
] (delimiter is '|');
```

Create a visualization

Create a table visualization in a Qlik Sense sheet with **Instrument**, **Player**, and **Project** as dimensions.

Result

Instrument	Player	Project
Guitar	Mike	Music
Guitar	Mike	Video
Guitar	Mike	OST
Guitar	Neil	Music
Guitar	Neil	Video
Guitar	Neil	OST
Synth	Jen	Music
Synth	Jen	Video
Synth	Jen	OST
Synth	Jo	Music
Synth	Neil	Music
Synth	Neil	Video
Synth	Neil	OST

Explanation

This example shows how using multiple instances of the **Subfield()** function, each with the `field_no` parameter left out, from within the same **LOAD** statement creates Cartesian products of all combinations. The **DISTINCT** option is used to avoid creating duplicate records.

SubStringCount

SubStringCount() returns the number of occurrences of the specified substring in the input string text. If there is no match, 0 is returned.

Syntax:

```
SubStringCount(text, sub_string)
```

Return data type: integer

Arguments:

Argument	Description
text	The original string.
sub_string	A string which may occur one or more times within the input string text .

Example: Chart expressions

Example	Result
SubStringCount ('abcdefgcdxyz', 'cd')	Returns '2'
SubStringCount ('abcdefgcdxyz', 'dc')	Returns '0'

Example: Load script

T1:

```
Load *,  
substringcount(upper(Strings),'AB') as SubStringCount_AB;  
Load * inline [  
Strings  
ABC:DEF:GHI:AB:CD:EF:GH  
aB/cd/ef/gh/Abc/abandoned];
```

Result

Strings	SubStringCount_AB
aB/cd/ef/gh/Abc/abandoned	3
ABC:DEF:GHI:AB:CD:EF:GH	2

TextBetween

TextBetween() returns the text in the input string that occurs between the characters specified as delimiters.

Syntax:

```
TextBetween(text, delimiter1, delimiter2[, n])
```

Return data type: string

Arguments:

Argument	Description
text	The original string.
delimiter1	Specifies the first delimiting character (or string) to search for in text .
delimiter2	Specifies the second delimiting character (or string) to search for in text .
n	Defines which occurrence of the delimiter pair to search between. For example, a value of 2 returns the characters between the second occurrence of delimiter1 and the second occurrence of delimiter2.

Example: Chart expressions

Example	Result
<code>TextBetween('<abc>', '<', '>')</code>	Returns 'abc'
<code>TextBetween('<abc><de>', '<', '>', 2)</code>	Returns 'de'
<code>TextBetween('abc', '<', '>')</code> <code>TextBetween('<a<b', '<', '>')</code>	Both examples return NULL. If any of the delimiter is not found in the string, NULL is returned.
<code>TextBetween('<>', '<', '>')</code>	Returns a zero-length string.
<code>TextBetween('<abc>', '<', '>', 2)</code>	Returns NULL, as n is greater than the number of occurrences of the delimiters.

Example: Load script

```
Load *,
textbetween(Text, '<', '>') as TextBetween,
textbetween(Text, '<', '>', 2) as SecondTextBetween;
Load * inline [
Text
<abc><de>
<def><ghi><jkl> ];
```

Result

Text	TextBetween	SecondTextBetween
<abc><de>	abc	de
<def><ghi><jkl>	def	ghi

Trim

Trim() returns the input string trimmed of any leading and trailing spaces.

Syntax:

```
Trim(text)
```

Return data type: string

Examples and results:

Example: Chart expression

Example	Result
Trim(' abc')	Returns 'abc'
Trim('abc ')	Returns 'abc'
Trim(' abc ')	Returns 'abc'

Example: Load script

```
Set verbatim=1;

T1:
Load *, len(TrimString) as TrimStringLength;
Load *, trim(String) as TrimString;
Load *, len(String) as StringLength;
Load * inline [
String
'    abc
' def  '](delimiter is '\t');
```



The "Set verbatim=1" statement is included in the example to ensure that the spaces are not automatically trimmed before the demonstration of the trim function. See Verbatim (page 197) for more information.

Result:

String	StringLength	TrimStringLength
def	6	3
abc	10	3

Upper

Upper() converts all the characters in the input string to upper case for all text characters in the expression. Numbers and symbols are ignored.

Syntax:

```
Upper(text)
```

Return data type: string

Example: Chart expression

Example	Result
Upper(' abcD ')	Returns 'ABCD'

Example: Load script

```
Load  
String,Upper(String)  
Inline  
[String  
rHode iSland  
washiNGTon d.C.  
new york];
```

Result

String	Upper(String)
rHode iSland	RHODE ISLAND
washiNGTon d.C.	WASHINGTON D.C.
new york	NEW YORK

5.25 System functions

System functions provide functions for accessing system, device and Qlik Sense app properties.

System functions overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

Author()

This function returns a string containing the author property of the current app. It can be used in both the data load script and in a chart expression.



Author property can not be set in the current version of Qlik Sense. If you migrate a QlikView document, the author property will be retained.

ClientPlatform()

This function returns the user agent string of the client browser. It can be used in both the data load script and in a chart expression.

Example:

Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36

ComputerName

This function returns a string containing the name of the computer as returned by the operating system. It can be used in both the data load script and in a chart expression.



If the name of the computer has more than 15 characters, the string will only contain the first 15 characters.

ComputerName()

DocumentName

This function returns a string containing the name of the current Qlik Sense app, without path but with extension. It can be used in both the data load script and in a chart expression.

DocumentName()

DocumentPath

This function returns a string containing the full path to the current Qlik Sense app. It can be used in both the data load script and in a chart expression.

DocumentPath()



This function is not supported in standard mode. .

DocumentTitle

This function returns a string containing the title of the current Qlik Sense app. It can be used in both the data load script and in a chart expression.

DocumentTitle()

EngineVersion

This function returns the full Qlik Sense engine version as a string.

EngineVersion ()

GetCollationLocale

This script function returns the culture name of the collation locale that is used. If the variable CollationLocale has not been set, the actual user machine locale is returned.

GetCollationLocale()

GetObjectField

GetObjectField() returns the name of the dimension. **Index** is an optional integer denoting the dimension that should be returned.

```
GetObjectField - chart function([index])
```

GetRegistryString

This function returns the value of a key in the Windows registry. It can be used in both the data load script and in a chart expression.

```
GetRegistryString(path, key)
```



This function is not supported in standard mode. .

IsPartialReload

This function returns -1 (True) if the current reload is partial, otherwise 0 (False).

```
IsPartialReload()
```

InObject

The **InObject()** chart function evaluates whether or not the current object is contained inside another object with the ID specified in the function argument. The object can be a sheet or a visualization.

```
InObject - chart function(id_str)
```

ObjectId

The **ObjectId()** chart function returns the ID of the object in which the expression is evaluated. The function takes an optional argument specifying which type of object the function concerns. The object can be a sheet or a visualization. This function is only available in chart expressions.

```
ObjectId - chart function([object_type_str])
```

OSUser

This function returns a string containing the name of the user that is currently connected. It can be used in both the data load script and in a chart expression.

```
OSUser()
```



In Qlik Sense Desktop and Qlik Sense Mobile Client Managed, this function always returns 'Personal\Me'.

ProductVersion

This function returns the full Qlik Sense version and build number as a string.

This function is deprecated and replaced by **EngineVersion()**.

```
ProductVersion()
```

ReloadTime

This function returns a timestamp for when the last data load finished. It can be used in both the data load script and in a chart expression.

`ReloadTime()`

StateName

StateName() returns the name of the alternate state of the visualization in which it is used. StateName can be used, for example, to create visualizations with dynamic text and colors to reflect when the state of a visualization is changed. This function can be used in chart expressions, but cannot be used to determine the state that the expression refers to.

`StateName - chart function()`

EngineVersion

This function returns the full Qlik Sense engine version as a string.

Syntax:

`EngineVersion()`

InObject - chart function

The **InObject()** chart function evaluates whether or not the current object is contained inside another object with the ID specified in the function argument. The object can be a sheet or a visualization.

This function can be used to show the hierarchy of objects in a sheet, from the top-level sheet object to visualizations nested within other visualizations. This function can be used alongside the **if** and **ObjectId** functions to create custom navigation in your apps.

Syntax:

`InObject(id_str)`

Return data type: Boolean

In Qlik Sense, the Boolean true value is represented by -1, and the false value is represented by 0.

Arguments

Argument	Description
<code>id_str</code>	A string value representing the ID of the object being evaluated.

The sheet ID can be obtained from the app URL. For visualizations, use the **Developer** options to identify the object ID and the text string of the object type.

Do the following:

1. In analysis mode, add the following text to your URL:
`/options/developer`

2. Right-click a visualization and click  **Developer**.
3. Under **Properties**, obtain the object ID from the dialog header, and the object type from the "**qType**" property.

Limitations:

This function can give unexpected results when invoked in an object (for example, a button) inside a container which is a master item. This limitation also applies to filter pane master items, which are containers for a number of listboxes. This is because of how master items use the object hierarchy.

InObject() is often used in combination with the following functions:

Related functions

Function	Interaction
<i>if</i> (page 542)	The if and ObjectId functions can be used together to create conditional expressions. For example, visualizations might achieve conditional coloring through expressions using these functions.
<i>ObjectId</i> - chart function (page 1443)	Similar to if , ObjectId is also used with InObject to create conditional expressions.

Example 1 – Basic functionality

Chart expression and results

The following basic example demonstrates how to determine whether an object is contained inside another object. In this case, we will be checking if a **Text & image** object resides in a sheet object using the ID of the sheet as an argument.

Do the following:

1. Open a new sheet and drag a **Text & image** chart onto the sheet.
2. In the properties panel, click **Add measure**.
3. Click **fx** to open the expression editor.
4. Paste the following expression into the dialog:
=InObject()
5. Modify the expression to include the ID of your sheet as a string between the parentheses.
For example, for a sheet with ID 1234-5678, you would use the following:
6. =Inobject('1234-5678')
7. Click **Apply**.

The value -1 is displayed in the chart, indicating that the expression was evaluated to be true.

Example 2 – Objects with conditional colors

Chart expression and results

Overview

The following example demonstrates how to create custom navigation buttons showing different coloring to indicate the sheet that is currently open.

Start by creating a new app and opening the Data load editor. Paste the following load script into a new tab. Note that the data itself is a placeholder and will not be used in the example content.

Load script

Transactions:

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'4/7/2022',40.39
```

```
8195,'5/16/2022',87.21  
8196,'6/15/2022',95.93  
8197,'7/26/2022',45.89  
8198,'8/9/2022',36.23  
8199,'9/22/2022',25.66  
8200,'11/23/2022',82.77  
8201,'12/27/2022',69.98  
8202,'1/1/2023',76.11  
8203,'2/8/2022',25.12  
8204,'3/19/2022',46.23  
8205,'6/26/2022',84.21  
8206,'9/14/2022',96.24  
8207,'11/29/2022',67.67  
];
```

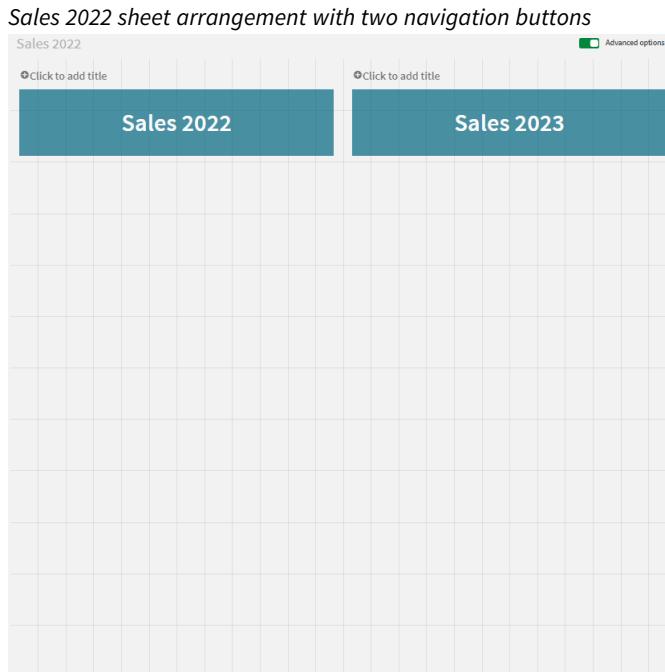
Creating the visualizations

Load the data and create two new sheets. Title them *Sales 2022* and *Sales 2023* respectively.

Next, build two button objects that will be used to navigate between the two sheets.

Do the following:

1. Add two **Button** objects to the sheet.
2. Under **Appearance > General**, set the **Label** of each button to *Sales 2022* and *Sales 2023*, respectively.
3. Arrange the buttons to match the following image.



4. Select the *Sales 2022* button, and expand **Actions and navigation** in the properties panel.
5. Click **Add action** and under **Navigation**, select **Go to a sheet**.
6. Under **Sheet**, select *Sales 2022*.

7. Repeat this button action setup to link the **Sales 2023** button to the *Sales 2023* sheet.
8. Convert the buttons to master items by right-clicking them and selecting  **Add to master items**.

You can now copy each button and paste it in the *Sales 2023* sheet, using the same size and arrangement on the sheet.

Creating conditional colors

Next, configure the buttons so that they will be blue if they are linked to the currently open sheet, and light gray if linked to the sheet that is not open.

Do the following:

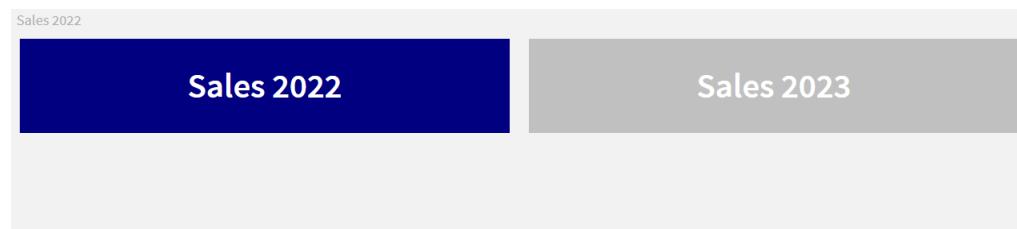
1. Open the *Sales 2022* sheet and obtain the sheet ID from the URL. Keep the *Sales 2022* sheet open.
2. Click the **Sales 2022** button master item and select **Edit** in the properties panel.
3. Under **Appearance > Background**, select to color the button **By expression**.
4. In **Expression**, paste the following text:
`=if(InObject("), Blue(), LightGray())`
5. Between the parentheses in the above expression, paste the sheet ID for the *Sales 2022* sheet.

The button is now configured to turn blue when the *Sales 2022* sheet is open, and light gray when it is not open.

Repeat the above instructions for the *Sales 2023* sheet, linking the **Sales 2023** button master item to the *Sales 2023* sheet ID.

Each sheet should now have two buttons indicating the currently open sheet with the color blue.

Sales 2022 sheet with blue coloring to indicate that Sales 2022 is currently displayed



IsPartialReload

This function returns -1 (True) if the current reload is partial, otherwise 0 (False).

Syntax:

```
IsPartialReload()
```

ObjectId - chart function

The **ObjectId()** chart function returns the ID of the object in which the expression is evaluated. The function takes an optional argument specifying which type of object the function concerns. The object can be a sheet or a visualization. This function is only available in chart expressions.

Syntax:

```
ObjectId([object_type_str])
```

Return data type:

string

The function's only argument, **object_type_str**, is optional and refers to a string value representing the type of the object.

Arguments

Argument	Description
object_type_str	A string value representing the type of the object being evaluated.

If no argument is specified in the function expression, **ObjectId()** returns the ID of the object in which the expression is used. To return the ID of the sheet object within which the visualization appears, use *ObjectId('sheet')*.

In the case of visualization objects nested within other visualization objects, specify the desired object type in the function argument for different results. For example, for a **Text & image** chart within a container, use 'text-image' to return the **Text & image** object and 'container' to return the ID of the container.

Do the following:

1. In analysis mode, add the following text to your URL:
/options/developer
2. Right-click a visualization and click  **Developer**.
3. Under **Properties**, obtain the object ID from the dialog header, and the object type from the "**qType**" property.

Limitations:

This function can give unexpected results when invoked in an object (for example, a button) inside a container which is a master item. This limitation also applies to filter pane master items, which are containers for a number of listboxes. This is because of how master items use the object hierarchy.

The chart expression *ObjectId('sheet')* will return an empty string in those cases, whereas *ObjectId('masterobject')* will show the identifier of the owning master item.

ObjectId() is often used in combination with the following functions:

Related functions

Function	Interaction
<i>if</i> (page 542)	The if and ObjectId functions can be used together to create conditional expressions. For example, visualizations might achieve conditional coloring through expressions using these functions.
<i>InObject</i> - chart function (page 1439)	Similar to if , InObject is also used with ObjectId to create conditional expressions.

Example 1 – Return chart object ID

Chart expression and results

The following basic example demonstrates how to return the ID of a visualization.

Do the following:

1. Open a new sheet and drag a **Text & image** chart onto the sheet.
2. In the properties panel, click **Add measure**.
3. Click **fx** to open the expression editor.
4. Paste the following expression into the dialog:
=ObjectId()
5. Click **Apply**.

The ID of the **Text & image** object is displayed in the visualization.

The same result can be achieved with the following expression:

```
=objectId('text-image')
```

Example 2 – Return sheet ID

Chart expression and results

The following basic example demonstrates how to return the ID of the sheet in which a visualization appears.

Do the following:

1. Open a new sheet and drag a **Text & image** chart onto the sheet.
2. In the properties panel, click **Add measure**.
3. Click  to open the expression editor.
4. Paste the following expression into the dialog:
`=objectId('sheet')`
5. Click **Apply**.

The ID of the sheet is displayed in the visualization.

Example 3 – Nested expression

Chart expression and results

The following example shows how the **ObjectId()** function can be nested inside other expressions.

Do the following:

1. Open a new sheet and drag a **Text & image** chart onto the sheet.
2. In the properties panel, click **Add measure**.
3. Click  to open the expression editor.
4. Paste the following expression into the dialog:
`=if(InObject(ObjectId('text-image')), 'In Text & image', 'Not in Text & image')`
5. Click **Apply**.

The text *In Text & image* appears in the chart, indicating that the object referenced in the expression is a **Text & image** chart.

For a more detailed example using conditional coloring, see the example on *InObject - chart function (page 1439)*.

ProductVersion

This function returns the full Qlik Sense version and build number as a string. This function is deprecated and replaced by **EngineVersion()**.

Syntax:

```
ProductVersion()
```

StateName - chart function

StateName() returns the name of the alternate state of the visualization in which it is used. StateName can be used, for example, to create visualizations with dynamic text and colors to reflect when the state of a visualization is changed. This function can be used in chart expressions, but cannot be used to determine the state that the expression refers to.

Syntax:

```
StateName ()
```

Example 1:

```
Dynamic Text  
='Region - ' & if(StateName() = '$', 'Default', StateName())
```

Example 2:

```
Dynamic Colors  
if(StateName() = 'Group 1', rgb(152, 171, 206),  
    if(StateName() = 'Group 2', rgb(187, 200, 179),  
        rgb(210, 210, 210)  
    )  
)
```

5.26 Table functions

The table functions return information about the data table which is currently being read. If no table name is specified and the function is used within a **LOAD** statement, the current table is assumed.

All functions can be used in the data load script, while only **NoOfRows** can be used in a chart expression.

Table functions overview

Some of the functions are described further after the overview. For those functions, you can click the function name in the syntax to immediately access the details for that specific function.

FieldName

The **FieldName** script function returns the name of the field with the specified number within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
FieldName (field_number ,table_name)
```

FieldNumber

The **FieldNumber** script function returns the number of a specified field within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
FieldNumber (field_name ,table_name)
```

NoOfFields

The **NoOfFields** script function returns the number of fields in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
NoOfFields (table_name)
```

NoOfRows

The **NoOfRows** function returns the number of rows (records) in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

```
NoOfRows (table_name)
```

NoOfTables

This script function returns the number of tables previously loaded.

```
NoOfTables ()
```

TableName

This script function returns the name of the table with the specified number.

```
TableName (table_number)
```

TableNumber

This script function returns the number of the specified table. The first table has number 0.

If table_name does not exist, NULL is returned.

```
TableNumber (table_name)
```

Example:

In this example, we want to create a table with information about the tables and fields that have been loaded.

First we load some sample data. This creates the two tables that will be used to illustrate the table functions described in this section.

Characters:

```
Load Chr(RecNo()+Ord('A')-1) as Alpha, RecNo() as Num autogenerate 26;
```

ASCII:

```
Load  
  if(RecNo()>=65 and RecNo()<=90,RecNo()-64) as Num,  
  Chr(RecNo()) as AsciiAlpha,  
  RecNo() as AsciiNum  
  autogenerate 255  
  where (RecNo()>=32 and RecNo()<=126) or RecNo()>=160 ;
```

Next, we iterate through the tables that have been loaded, using the **NoOfTables** function, and then through the fields of each table, using the **NoOfFields** function, and load information using the table functions.

```
//Iterate through the loaded tables
For t = 0 to NoOfTables() - 1

//Iterate through the fields of table
For f = 1 to NoOfFields(TableName($(t)))
Tables:
Load
    TableName($(t)) as Table,
    TableNumber(TableName($(t))) as TableNo,
    NoOfRows(TableName($(t))) as TableRows,
    FieldName($(f),TableName($(t))) as Field,
    FieldNumber(FieldName($(f),TableName($(t))),TableName($(t))) as FieldNo
    Autogenerate 1;
Next f
Next t;
```

The resulting table Tables will look like this:

Load table

Table	TableNo	TableRows	Field	FieldNo
Characters	0	26	Alpha	1
Characters	0	26	Num	2
ASCII	1	191	Num	1
ASCII	1	191	AsciiAlpha	2
ASCII	1	191	AsciiNum	3

FieldName

The **FieldName** script function returns the name of the field with the specified number within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
FieldName(field_number ,table_name)
```

Arguments:

Arguments

Argument	Description
field_number	The field number of the field you want to reference.
table_name	The table containing the field you want to reference.

Example:

```
LET a = FieldName(4,'tab1');
```

FieldNumber

The **FieldNumber** script function returns the number of a specified field within a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
FieldNumber(field_name ,table_name)
```

Arguments:

Arguments

Argument	Description
field_name	The name of the field.
table_name	The name of the table containing the field.

If the field field_name does not exist in table_name, or table_name does not exist, the function returns 0.

Example:

```
LET a = FieldNumber('Customer','tab1');
```

NoOfFields

The **NoOfFields** script function returns the number of fields in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
NoOfFields(table_name)
```

Arguments:

Arguments

Argument	Description
table_name	The name of the table.

Example:

```
LET a = NoOfFields('tab1');
```

NoOfRows

The **NoOfRows** function returns the number of rows (records) in a previously loaded table. If the function is used within a **LOAD** statement, it must not reference the table currently being loaded.

Syntax:

```
NoOfRows(table_name)
```

Arguments:

Arguments	
Argument	Description
table_name	The name of the table.

Example:

```
LET a = NoOfRows('tab1');
```

5.27 Trigonometric and hyperbolic functions

This section describes functions for performing trigonometric and hyperbolic operations. In all of the functions, the arguments are expressions resolving to angles measured in radians, where **x** should be interpreted as a real number.

All angles are measured in radians.

All functions can be used in both the data load script and in chart expressions.

cos

Cosine of **x**. The result is a number between -1 and 1.

```
cos( x )
```

acos

Inverse cosine of **x**. The function is only defined if $-1 \leq x \leq 1$. The result is a number between 0 and π .

```
acos( x )
```

sin

Sine of **x**. The result is a number between -1 and 1.

```
sin( x )
```

asin

Inverse sine of **x**. The function is only defined if $-1 \leq x \leq 1$. The result is a number between $-\pi/2$ and $\pi/2$.

```
asin( x )
```

tan

Tangent of **x**. The result is a real number.

```
tan( x )
```

atan

Inverse tangent of **x**. The result is a number between $-\pi/2$ and $\pi/2$.

```
atan( x )
```

atan2

Two-dimensional generalization of the inverse tangent function. Returns the angle between the origin and the point represented by the coordinates **x** and **y**. The result is a number between $-\pi$ and $+\pi$.

```
atan2( y, x )
```

cosh

Hyperbolic cosine of **x**. The result is a positive real number.

```
cosh( x )
```

sinh

Hyperbolic sine of **x**. The result is a real number.

```
sinh( x )
```

tanh

Hyperbolic tangent of **x**. The result is a real number.

```
tanh( x )
```

acosh

Inverse hyperbolic cosine of **x**. The result is a positive real number.

```
acosh( x )
```

asinh

Inverse hyperbolic sine of **x**. The result is a real number.

```
asinh( x )
```

atanh

Inverse hyperbolic tangent of **x**. The result is a real number.

```
atanh( x )
```

Examples:

The following script code loads a sample table, and then loads a table containing the calculated trigonometric and hyperbolic operations on the values.

```
SampleData:  
LOAD * Inline  
[Value
```

```
-1  
0  
1];  
  
Results:  
Load *,  
cos(value),  
acos(value),  
sin(value),  
asin(value),  
tan(value),  
atan(value),  
atan2(value, value),  
cosh(value),  
sinh(value),  
tanh(value)  
RESIDENT SampleData;  
  
Drop Table SampleData;
```

6 File system access restriction

For security reasons, Qlik Sense in standard mode does not support paths in the data load script or functions and variables that expose the file system.

However, since file system paths were supported in QlikView, it is possible to disable standard mode and use legacy mode in order to reuse QlikView load scripts.



Disabling standard mode can create a security risk by exposing the file system.

Disabling standard mode (page 1459)

6.1 Security aspects when connecting to file based ODBC and OLE DB data connections

ODBC and OLE DB data connections using file-based drivers will expose the path to the connected data file in the connection string. The path can be exposed when the connection is edited, in the data selection dialog, or in certain SQL queries. This is the case both in standard mode and legacy mode.



If exposing the path to the data file is a concern, it is recommended to connect to the data file using a folder data connection if it is possible.

6.2 Limitations in standard mode

Several statements, variables and functions cannot be used or have limitations in standard mode. Using unsupported statements in the data load script produces an error when the load script runs. Error messages can be found in the script log file. Using unsupported variables and functions does not produce error messages or log file entries. Instead, the function returns NULL.

There is no indication that a variable, statement or function is unsupported when you are editing the data load script.

System variables

System variables

Variable	Standard mode	Legacy mode	Definition
Floppy	Not supported	Supported	Returns the drive letter of the first floppy drive found, normally <i>a:</i>

6 File system access restriction

Variable	Standard mode	Legacy mode	Definition
CD	Not supported	Supported	Returns the drive letter of the first CD-ROM drive found. If no CD-ROM is found, then c: is returned.
QvPath	Not supported	Supported	Returns the browse string to the Qlik Sense executable.
QvRoot	Not supported	Supported	Returns the root directory of the Qlik Sense executable.
QvWorkPath	Not supported	Supported	Returns the browse string to the current Qlik Sense app.
QvWorkRoot	Not supported	Supported	Returns the root directory of the current Qlik Sense app.
WinPath	Not supported	Supported	Returns the browse string to Windows.
WinRoot	Not supported	Supported	Returns the root directory of Windows.
\$(include=...)	Supported input: Path using library connection	Supported input: Path using library connection or file system	The Include/Must_Include variable specifies a file that contains text that should be included in the script and evaluated as script code. It is not used to add data. You can store parts of your script code in a separate text file and reuse it in several apps. This is a user-defined variable.

Regular script statements

Regular script statements

Statement	Standard mode	Legacy mode	Definition
Binary	Supported input: Path using library connection	Supported input: Path using library connection or file system	The binary statement is used for loading data from another app.
Connect	Supported input: Path using library connection	Supported input: Path using library connection or file system	The CONNECT statement is used to define Qlik Sense access to a general database through the OLE DB/ODBC interface. For ODBC, the data source first needs to be specified using the ODBC administrator.
Directory	Supported input: Path using library connection	Supported input: Path using library connection or file system	The Directory statement defines which directory to look in for data files in subsequent LOAD statements, until a new Directory statement is made.
Execute	Not supported	Supported input: Path using library connection or file system	The Execute statement is used to run other programs while Qlik Sense is loading data. For example, to make conversions that are necessary.
LOAD from ...	Supported input: Path using library connection	Supported input: Path using library connection or file system	The LOAD statement loads fields from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent SELECT statement or by generating data automatically.

Statement	Standard mode	Legacy mode	Definition
Store into ...	Supported input: Path using library connection	Supported input: Path using library connection or file system	The Store statement creates a QVD, or text file.

Script control statements

Script control statements

Statement	Standard mode	Legacy mode	Definition
For each... filelist mask/dirlist mask	Supported input: Path using library connection Returned output: Library connection	Supported input: Path using library connection or file system Returned output: Library connection or file system path, depending on input	The filelist mask syntax produces a comma separated list of all files in the current directory matching the filelist mask . The dirlist mask syntax produces a comma separated list of all directories in the current directory matching the directory name mask.

File functions

File functions

Function	Standard mode	Legacy mode	Definition
Attribute()	Supported input: Path using library connection	Supported input: Path using library connection or file system	Returns the value of the meta tags of different media files as text.
ConnectionString()	Returned output: Library connection name	Library connection name or actual connection, depending on input	Returns the active connect string for ODBC or OLE DB connections.
FileDir()	Returned output: Library connection	Returned output: Library connection or file system path, depending on input	The FileDir function returns a string containing the path to the directory of the table file currently being read.

6 File system access restriction

Function	Standard mode	Legacy mode	Definition
FilePath()	Returned output: Library connection	Returned output: Library connection or file system path, depending on input	The FilePath function returns a string containing the full path to the table file currently being read.
FileSize()	Supported input: Path using library connection	Supported input: Path using library connection or file system	The FileSize function returns an integer containing the size in bytes of the file filename or, if no filename is specified, of the table file currently being read.
FileTime()	Supported input: Path using library connection	Supported input: Path using library connection or file system	The FileTime function returns a timestamp in UTC format of the last modification of a specified file. If a file is not specified, the function returns a timestamp in UTC of the last modification of the currently read table file.
GetFolderPath()	Not supported	Returned output: Absolute path	The GetFolderPath function returns the value of the Microsoft Windows <i>SHGetFolderPath</i> function. This function takes as input the name of a Microsoft Windows folder and returns the full path of the folder.
QvdCreateTime()	Supported input: Path using library connection	Supported input: Path using library connection or file system	This script function returns the XML-header timestamp from a QVD file, if any is present, otherwise it returns NULL. In the timestamp, time is provided in UTC.

Function	Standard mode	Legacy mode	Definition
QvdFieldName()	Supported input: Path using library connection	Supported input: Path using library connection or file system	This script function returns the name of field number fieldno in a QVD file. If the field does not exist NULL is returned.
QvdNoOfFields()	Supported input: Path using library connection	Supported input: Path using library connection or file system	This script function returns the number of fields in a QVD file.
QvdNoOfRecords()	Supported input: Path using library connection	Supported input: Path using library connection or file system	This script function returns the number of records currently in a QVD file.
QvdTableName()	Supported input: Path using library connection	Supported input: Path using library connection or file system	This script function returns the name of the table stored in a QVD file.

System functions

System functions

Function	Standard mode	Legacy mode	Definition
DocumentPath()	Not supported	Returned output: Absolute path	This function returns a string containing the full path to the current Qlik Sense app.
GetRegistryString()	Not supported	Supported	Returns the value of a named registry key with a given registry path. This function can be used in chart and script alike.

6.3 Disabling standard mode

You can disable standard mode, or in other words, set legacy mode, in order to reuse QlikView load scripts that refer to absolute or relative file paths as well as library connections.



Disabling standard mode can create a security risk by exposing the file system.

Qlik Sense

For Qlik Sense, standard mode can be disabled in QMC using the **Standard mode** property.

Qlik Sense Desktop

In Qlik Sense Desktop, you can set standard/legacy mode in *Settings.ini*.

If you installed Qlik Sense Desktop using the default installation location, *Settings.ini* is located in *C:\Users\{user}\Documents\Qlik\Sense\Settings.ini*. If you installed Qlik Sense Desktop to a folder that you selected, *Settings.ini* is located in the *Engine* folder of the installation path.

Do the following:

1. Open *Settings.ini* in a text editor.
2. Change *StandardReload=1* to *StandardReload=0*.
3. Save the file and start Qlik Sense Desktop.

Qlik Sense Desktop now runs in legacy mode.

Settings

The available settings for StandardReload are:

- 1 (standard mode)
- 0 (legacy mode)

6 Chart level scripting

When modifying chart data, you use a sub-set of the Qlik Sense script which consists of a number of statements. A statement can be either a regular script statement or a script control statement. Certain statements can be preceded by prefixes.

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by a semicolon or the end-of-line.

Prefixes may be applied to applicable regular statements but never to control statements.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

In this section you can find an alphabetical listing of all script statements, control statements and prefixes available in the sub-set of the script used when modifying chart data.

6.4 Control statements

When modifying chart data, you use a sub-set of the Qlik Sense script which consists of a number of statements. A statement can be either a regular script statement or a script control statement.

Control statements are typically used for controlling the flow of the script execution. Each clause of a control statement must be kept inside one script line and may be terminated by semicolon or end-of-line.

Prefixes are never applied to control statements.

All script keywords can be typed with any combination of lower case and upper case characters.

Chart modifier control statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

```
Call name ( [ paramlist ] )
```

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

```
Do..loop [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop [ ( while | until ) condition ]
```

End

The **End** script keyword is used to close **If**, **Sub** and **Switch** clauses.

Exit

The **Exit** script keyword is part of the **Exit Script** statement, but can also be used to exit **Do**, **For** or **Sub** clauses.

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

```
Exit script[ (when | unless) condition ]
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

```
For..next counter = expr1 to expr2 [ stepexpr3 ]
[statements]
[exit for [ (when | unless) condition ]
[statements]
Next [counter]
```

For each ..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

```
For each..next var in list
[statements]
[exit for [ (when | unless) condition ]
[statements]
next [var]
```

If..then

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.



Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

```
If..then..elseif..else..end if condition then
[ statements ]
{ elseif condition then
[ statements ]
[ else
[ statements ]
end if
```

Next

The **Next** script keyword is used to close **For** loops.

Sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

```
Sub..end sub name [ ( paramlist )] statements end sub
```

Switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

```
Switch..case..default..end switch expression {case valuelist [ statements ]}  
[default statements] end switch
```

To

The **To** script keyword is used in several script statements.

Call

The **call** control statement calls a subroutine which must be defined by a previous **sub** statement.

Syntax:

```
Call name ( [ paramlist ] )
```

Arguments:

Arguments

Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of the actual parameters to be sent to the subroutine. Each item in the list may be a field name, a variable or an arbitrary expression.

The subroutine called by a **call** statement must be defined by a **sub** encountered earlier during script execution.

Parameters are copied into the subroutine and, if the parameter in the **call** statement is a variable and not an expression, copied back out again upon exiting the subroutine.

Limitations:

- Since the **call** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.
- When you define a subroutine with **sub..end sub** inside a control statement, for example **if..then**, you can only call the subroutine from within the same control statement.

Do..loop

The **do..loop** control statement is a script iteration construct which executes one or several statements until a logical condition is met.

Syntax:

```
Do [ ( while | until ) condition ] [statements]
[exit do [ ( when | unless ) condition ] [statements]
loop [ ( while | until ) condition ]
```



Since the **do..loop** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**do**, **exit do** and **loop**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
while / until	The while or until conditional clause must only appear once in any do..loop statement, i.e. either after do or after loop . Each condition is interpreted only the first time it is encountered but is evaluated for every time it encountered in the loop.
exit do	If an exit do clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the loop clause denoting the end of the loop. An exit do clause can be made conditional by the optional use of a when or unless suffix.

End

The **End** script keyword is used to close **If**, **Sub** and **Switch** clauses.

Exit

The **Exit** script keyword is part of the **Exit Script** statement, but can also be used to exit **Do**, **For** or **Sub** clauses.

Exit script

This control statement stops script execution. It may be inserted anywhere in the script.

Syntax:

```
Exit Script [ ( when | unless ) condition ]
```

Since the **exit script** statement is a control statement and as such is ended with either a semicolon or end-of-line, it must not cross a line boundary.

Arguments:

Arguments	
Argument	Description
condition	A logical expression evaluating to True or False.
when / unless	An exit script statement can be made conditional by the optional use of when or unless clause.

Examples:

```
//Exit script
Exit Script;

//Exit script when a condition is fulfilled
Exit Script when a=1
```

For..next

The **for..next** control statement is a script iteration construct with a counter. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the counter variable between specified low and high limits.

Syntax:

```
For counter = expr1 to expr2 [ step expr3 ]
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
Next [counter]
```

The expressions *expr1*, *expr2* and *expr3* are only evaluated the first time the loop is entered. The value of the counter variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.



Since the **for..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for..to..step**, **exit for** and **next**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
counter	A variable name. If <i>counter</i> is specified after next it must be the same variable name as the one found after the corresponding for .
expr1	An expression which determines the first value of the <i>counter</i> variable for which the loop should be executed.
expr2	An expression which determines the last value of the <i>counter</i> variable for which the loop should be executed.
expr3	An expression which determines the value indicating the increment of the <i>counter</i> variable each time the loop has been executed.
condition	a logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.

For each..next

The **for each..next** control statement is a script iteration construct which executes one or several statements for each value in a comma separated list. The statements inside the loop enclosed by **for** and **next** will be executed for each value of the list.

Syntax:

Special syntax makes it possible to generate lists with file and directory names in the current directory.

```
for each var in list
[statements]
[exit for [ ( when | unless ) condition ]
[statements]
next [var]
```

Arguments:

Arguments

Argument	Description
var	A script variable name which will acquire a new value from list for each loop execution. If var is specified after next it must be the same variable name as the one found after the corresponding for each .

The value of the **var** variable may be changed by statements inside the loop, but this is not good programming practice.

If an **exit for** clause is encountered inside the loop, the execution of the script will be transferred to the first statement after the **next** clause denoting the end of the loop. An **exit for** clause can be made conditional by the optional use of a **when** or **unless** suffix.



Since the **for each..next** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its three possible clauses (**for each**, **exit for** and **next**) must not cross a line boundary.

Syntax:

```
list := item { , item }
item := constant | (expression) | filelist mask | dirlist mask |
fieldvaluelist mask
```

Arguments

Argument	Description
constant	Any number or string. Note that a string written directly in the script must be enclosed by single quotes. A string without single quotes will be interpreted as a variable, and the value of the variable will be used. Numbers do not need to be enclosed by single quotes.
expression	An arbitrary expression.
mask	A filename or folder name mask which may include any valid filename characters as well as the standard wildcard characters, * and ?. You can use absolute file paths or lib:// paths.
condition	A logical expression evaluating to True or False.
statements	Any group of one or more Qlik Sense script statements.
filelist mask	This syntax produces a comma separated list of all files in the current directory matching the filename mask. This argument supports only library connections in standard mode.
dirlist mask	This syntax produces a comma separated list of all folders in the current folder matching the folder name mask. This argument supports only library connections in standard mode.
fieldvaluelist mask	This syntax iterates through the values of a field already loaded into Qlik Sense.



The Qlik Web Storage Provider Connectors and other DataFiles connections do not support filter masks that use wildcard (and ?) characters.*

Example 1: Loading a list of files

```
// LOAD the files 1.csv, 3.csv, 7.csv and xyz.csv
for each a in 1,3,7,'xyz'
    LOAD * from file$(a).csv;
next
```

Example 2: Creating a list of files on disk

This example loads a list of all Qlik Sense related files in a folder.

```
sub DoDir (Root)
    for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'

        for each File in filelist (Root&'/*.' &Ext)

            LOAD
                '$(File)' as Name,
                FileSize( '$(File)' ) as size,
                FileTime( '$(File)' ) as FileTime
            autogenerate 1;

        next File

    next Ext
    for each Dir in dirlist (Root&'/*' )

        call DoDir (Dir)

    next Dir

end sub

call DoDir ('lib://DataFiles')
```

Example 3: Iterating through a the values of a field

This example iterates through the list of loaded values of FIELD and generates a new field, NEWFIELD. For each value of FIELD, two NEWFIELD records will be created.

```
load * inline [
FIELD
one
two
three
];

FOR Each a in FieldvalueList('FIELD')
```

```
LOAD '$(a)' & '-'&RecNo() as NEWFIELD AutoGenerate 2;
NEXT a
```

The resulting table looks like this:

Example table

NEWFIELD
one-1
one-2
two-1
two-2
three-1
three-2

If..then..elseif..else..end if

The **if..then** control statement is a script selection construct forcing the script execution to follow different paths depending on one or several logical conditions.

Control statements are typically used to control the flow of the script execution. In a chart expression, use the **if** conditional function instead.

Syntax:

```
If condition then
  [ statements ]
{ elseif condition then
  [ statements ] }
[ else
  [ statements ] ]
end if
```

Since the **if..then** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
condition	A logical expression which can be evaluated as True or False.
statements	Any group of one or more Qlik Sense script statements.

Example 1:

```
if a=1 then
```

```
LOAD * from abc.csv;
SQL SELECT e, f, g from tab1;

end if
```

Example 2:

```
if a=1 then; drop table xyz; end if;
```

Example 3:

```
if x>0 then
    LOAD * from pos.csv;
elseif x<0 then
    LOAD * from neg.csv;
else
    LOAD * from zero.txt;
end if
```

Next

The **Next** script keyword is used to close **For** loops.

Sub..end sub

The **sub..end sub** control statement defines a subroutine which can be called upon from a **call** statement.

Syntax:

```
Sub name [ ( paramlist )] statements end sub
```

Arguments are copied into the subroutine and, if the corresponding actual parameter in the **call** statement is a variable name, copied back out again upon exiting the subroutine.

If a subroutine has more formal parameters than actual parameters passed by a **call** statement, the extra parameters will be initialized to NULL and can be used as local variables within the subroutine.

Arguments:

Arguments

Argument	Description
name	The name of the subroutine.
paramlist	A comma separated list of variable names for the formal parameters of the subroutine. These can be used as any variable inside the subroutine.
statements	Any group of one or more Qlik Sense script statements.

Limitations:

- Since the **sub** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its two clauses (**sub** and **end sub**) must not cross a line boundary.
- When you define a subroutine with **sub..end sub** inside a control statement, for example **if..then**, you can only call the subroutine from within the same control statement.

Example 1:

```
Sub INCR (I,J)
I = I + 1
Exit Sub when I < 10
J = J + 1
End Sub
Call INCR (X,Y)
```

Example 2: - parameter transfer

```
Sub ParTrans (A,B,C)
A=A+1
B=B+1
C=C+1
End Sub
A=1
X=1
C=1
Call ParTrans (A, (X+1)*2)
```

The result of the above will be that locally, inside the subroutine, A will be initialized to 1, B will be initialized to 4 and C will be initialized to NULL.

When exiting the subroutine, the global variable A will get 2 as value (copied back from subroutine). The second actual parameter “(X+1)*2” will not be copied back since it is not a variable. Finally, the global variable C will not be affected by the subroutine call.

Switch..case..default..end switch

The **switch** control statement is a script selection construct forcing the script execution to follow different paths, depending on the value of an expression.

Syntax:

```
Switch expression {case valuelist [ statements ]} [default statements] end
switch
```



Since the **switch** statement is a control statement and as such is ended with either a semicolon or end-of-line, each of its four possible clauses (**switch**, **case**, **default** and **end switch**) must not cross a line boundary.

Arguments:

Arguments

Argument	Description
expression	An arbitrary expression.
valuelist	A comma separated list of values with which the value of expression will be compared. Execution of the script will continue with the statements in the first group encountered with a value in valuelist equal to the value in expression. Each value in valuelist may be an arbitrary expression. If no match is found in any case clause, the statements under the default clause, if specified, will be executed.
statements	Any group of one or more Qlik Sense script statements.

Example:

```

Switch I
Case 1
LOAD '$(I)': CASE 1' as case autogenerate 1;
Case 2
LOAD '$(I)': CASE 2' as case autogenerate 1;
Default
LOAD '$(I)': DEFAULT' as case autogenerate 1;
End Switch

```

To

The **To** script keyword is used in several script statements.

6.5 Prefixes

Prefixes may be applied to applicable regular statements but never to control statements.

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Chart modifier prefixes overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

Add

The **Add** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that it should add records to another table. It also specifies that this statement should be run in a partial reload. The **Add** prefix can also be used in a **Map** statement.

```

Add [only] [Concatenate[(tablename )]] (loadstatement | selectstatement)
Add [ Only ] mapstatement

```

Replace

The **Replace** prefix can be added to any **LOAD** or **SELECT** statement in the script to specify that the loaded table should replace another table. It also specifies that this statement should be run in a partial reload. The **Replace** prefix can also be used in a **Map** statement.

```
Replace [only] [Concatenate[(tablename) ]] (loadstatement | selectstatement)  
Replace [only] mapstatement
```

Add

In a chart modifying context, the **Add** prefix is used with **LOAD** to append values to the *HC1* table, representing the hypercube computed by the Qlik associative engine. You can specify one or several columns. Missing values are automatically filled by the Qlik associative engine.

Syntax:

```
Add loadstatement
```

Example:

This example adds two rows to the columns *Dates* and *Sales* from the inline statement

```
Add Load  
x as Dates,  
y as Sales  
Inline  
[  
Dates,Sales  
2001/09/1,1000  
2001/09/10,-300  
]
```

Replace

In a chart modifying context, the **Replace** prefix changes all values of the *HC1* table with a computed value defined by the script.

Syntax:

```
Replace loadstatement
```

Example:

This example overwrites all values in column *z* with the sum of *x* and *y*.

```
Replace Load  
x+y as z  
Resident HC1;
```

6.6 Regular statements

Regular statements are typically used for manipulating data in one way or another. These statements may be written over any number of lines in the script and must always be terminated by a semicolon, ";".

All script keywords can be typed with any combination of lower case and upper case characters. Field and variable names used in the statements are however case sensitive.

Chart modifier regular statements overview

Each function is described further after the overview. You can also click the function name in the syntax to immediately access the details for that specific function.

LOAD

In a chart modifying context, the **LOAD** statement loads additional data to the hypercube from data defined in the script, or from a previously loaded table. It is also possible to load data from analytic connections.



*The **LOAD** statement must have either **Replace** or **Add** prefix, or it will be rejected.*

```
Add | Replace Load [ distinct ] fieldlist
(
  inline data [ format-spec ] |
  resident table-label
) | extension pluginname.functionname([script] tabledescription)
[ where criterion | while criterion ]
[ group by groupbyfieldlist ]
[order by orderbyfieldlist ]
```

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '=' at script run time before it is assigned to the variable.

```
Let variablename=expression
```

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

```
Set variablename=string
```

Put

The **Put** statement is used to set some numeric value in the hypercube.

HCValue

The **HCValue** statement is used to retrieve values in a row of a specified column.

Load

In a chart modifying context, the **LOAD** statement loads additional data to the hypercube from data defined in the script, or from a previously loaded table. It is also possible to load data from analytic connections.



*The **LOAD** statement must have either **Replace** or **Add** prefix, or it will be rejected.*

Syntax:

```
Add | Replace LOAD fieldlist
(
  inline data [ format-spec ] |
  resident table-label
) | extension pluginname.functionname([script] tabledescription)
[ where criterion | while criterion ]
[ group by groupbyfieldlist ]
[order by orderbyfieldlist ]
```

Arguments:

Arguments

Argument	Description
fieldlist	<p><i>fieldlist ::= (* field {, * field})</i> A list of the fields to be loaded. Using * as a field list indicates all fields in the table.</p> <p><i>field ::= (fieldref expression) [as aliasname]</i> The field definition must always contain a literal, a reference to an existing field, or an expression.</p> <p><i>fieldref ::= (fieldname @fieldnumber @startpos:endpos [I U R B T])</i> <i>fieldname</i> is a text that is identical to a field name in the table. Note that the field name must be enclosed by straight double quotation marks or square brackets if it contains e.g. spaces. Sometimes field names are not explicitly available. Then a different notation is used:</p> <p>@<i>fieldnumber</i> represents the field number in a delimited table file. It must be a positive integer preceded by "@". The numbering is always made from 1 and up to the number of fields.</p> <p>@<i>startpos:endpos</i> represents the start and end positions of a field in a file with fixed length records. The positions must both be positive integers. The two numbers must be preceded by "@" and separated by a colon. The numbering is always made from 1 and up to the number of positions. In the last field, n is used as end position.</p> <ul style="list-style-type: none"> If @<i>startpos:endpos</i> is immediately followed by the characters I or U, the bytes read will be interpreted as a binary signed (I) or unsigned (U) integer (Intel byte order). The number of positions read must be 1, 2 or 4. If @<i>startpos:endpos</i> is immediately followed by the character R, the bytes read will be interpreted as a binary real number (IEEE 32-bit or 64 bit floating point). The number of positions read must be 4 or 8. If @<i>startpos:endpos</i> is immediately followed by the character B, the bytes read will be interpreted as a BCD (Binary Coded Decimal) numbers according to the COMP-3 standard. Any number of bytes may be specified. <p><i>expression</i> can be a numeric function or a string function based on one or several other fields in the same table. For further information, see the syntax of expressions.</p> <p>as is used for assigning a new name to the field.</p>

Argument	Description
inline	<p>inline is used if data should be typed within the script, and not loaded from a file.</p> <p><i>data ::= [text]</i></p> <p>Data entered through an inline clause must be enclosed by double quotation marks or by square brackets. The text between these is interpreted in the same way as the content of a file. Hence, where you would insert a new line in a text file, you should also do it in the text of an inline clause, i.e. by pressing the Enter key when typing the script. The number of columns are defined by the first line.</p> <p><i>format-spec ::= (fspec-item {, fspec-item})</i></p> <p>The format specification consists of a list of several format specification items, within brackets.</p>
resident	<p>resident is used if data should be loaded from a previously loaded table.</p> <p><i>table label</i> is a label preceding the LOAD statement that created the original table. The label should be given with a colon at the end.</p>
extension	<p>You can load data from analytic connections. You need to use the extension clause to call a function defined in the server-side extension (SSE) plugin, or evaluate a script.</p> <p>You can send a single table to the SSE plugin, and a single data table is returned. If the plugin does not specify the names of the fields that are returned, the fields will be named Field1, Field2, and so on.</p> <pre>Extension pluginname.functionname(tabledescription);</pre> <ul style="list-style-type: none"> • Loading data using a function in an SSE plugin <i>tabledescription ::= (table {,tablefield})</i> If you do not state table fields, the fields will be used in load order. • Loading data by evaluating a script in an SSE plugin <i>tabledescription ::= (script, table {,tablefield})</i> <p>Data type handling in the table field definition</p> <p>Data types are automatically detected in analytic connections. If the data has no numeric values and at least one non-NULL text string, the field is considered as text. In any other case it is considered as numeric.</p> <p>You can force the data type by wrapping a field name with String() or Mixed().</p> <ul style="list-style-type: none"> • String() forces the field to be text. If the field is numeric, the text part of the dual value is extracted, there is no conversion performed. • Mixed() forces the field to be dual. <p>String() or Mixed() cannot be used outside extension table field definitions, and you cannot use other Qlik Sense functions in a table field definition.</p>

Argument	Description
where	where is a clause used for stating whether a record should be included in the selection or not. The selection is included if <i>criterion</i> is True. <i>criterion</i> is a logical expression.
while	while is a clause used for stating whether a record should be repeatedly read. The same record is read as long as <i>criterion</i> is True. In order to be useful, a while clause must typically include the IterNo() function. <i>criterion</i> is a logical expression.
group by	group by is a clause used for defining over which fields the data should be aggregated (grouped). The aggregation fields should be included in some way in the expressions loaded. No other fields than the aggregation fields may be used outside aggregation functions in the loaded expressions. <i>groupbyfieldlist ::= (fieldname { ,fieldname })</i>
order by	order by is a clause used for sorting the records of a resident table before they are processed by the load statement. The resident table can be sorted by one or more fields in ascending or descending order. The sorting is made primarily by numeric value and secondarily by national collation order. This clause may only be used when the data source is a resident table. The ordering fields specify which field the resident table is sorted by. The field can be specified by its name or by its number in the resident table (the first field is number 1). <i>orderbyfieldlist ::= fieldname [sortorder] { , fieldname [sortorder] }</i> <i>sortorder</i> is either <i>asc</i> for ascending or <i>desc</i> for descending. If no <i>sortorder</i> is specified, <i>asc</i> is assumed. <i>fieldname, path, filename</i> and <i>aliasname</i> are text strings representing what the respective names imply. Any field in the source table can be used as <i>fieldname</i> . However, fields created through the <i>as</i> clause (<i>aliasname</i>) are out of scope and cannot be used inside the same load statement.

Let

The **let** statement is a complement to the **set** statement, used for defining script variables. The **let** statement, in opposition to the **set** statement, evaluates the expression on the right side of the '*=*' at script run time before it is assigned to the variable.

Syntax:

```
Let variablename=expression
```

Examples and results:

Example	Result
Set x=3+4; Let y=3+4; z=\$(y)+1;	\$(x) will be evaluated as ' 3+4 ' \$(y) will be evaluated as ' 7 ' \$(z) will be evaluated as ' 8 ' Note the difference between the Set and Let statements. The Set statement assigns the string '3+4' to the variable, whereas the Let statement evaluates the string and assigns 7 to the variable.
Let T=now();	\$(T) will be given the value of the current time.

Set

The **set** statement is used for defining script variables. These can be used for substituting strings, paths, drives, and so on.

Syntax:

```
Set variableName=string
```

Example 1:

```
Set FileToUse=Data1.csv;
```

Example 2:

```
Set Constant="My string";
```

Example 3:

```
Set BudgetYear=2012;
```

Put

The **put** statement is used to set some numeric value in the hypercube.

Access to the columns can be done by labels. You can also access columns and rows by declaration order. See the examples below for more details.

Syntax:

```
put column(position)=value
```

Example 1:

Access to the columns can be done by labels.

This example will set a value of 1 in the first position of the column labeled *Sales*.

```
Put Sales(1) = 1;
```

Example 2:

You can access measure columns by declaration order using the #hc1.measure format for measures.

This example will set the value 1000 in the tenth position of the final sorted hypercube.

```
Put #hc1.measure.2(10) = 1000;
```

Example 3:

You can access the dimension rows by declaration order using the #hc1.dimension format for dimensions.

This example puts the value of the constant Pi in the fifth row of the third declared dimension.

```
Put #hc1.dimension.3(5) = Pi();
```



If there are no such dimensions or expressions, in value or labels, an error is returned indicating that the column was not found. If the index for the column is out of bounds, no error is displayed.

HCValue

The **HCValue** function it is used to retrieve values in a row of a specified column.

Syntax:

```
HCValue(column,position)
```

Example 1:

This example returns the value at the first position of the column with label ‘Sales’.

```
HCValue(Sales,1)
```

Example 2:

This example returns the value at the tenth position of the sorted hypercube.

```
HCValue(#hc1.measure.2,10)
```

Example 3:

This example returns the value at the fifth row in the third dimension.

```
HCValue(#hc1.dimension.3,5)
```



If there are no such dimensions or expressions, in value or labels, an error is returned indicating that the column was not found. If the index for the column is out of bounds, NULL is returned.

7 QlikView functions and statements not supported in Qlik Sense

Most functions and statements that can be used in QlikView load scripts and chart expressions are also supported in Qlik Sense, but there are some exceptions, as described here.

7.1 Script statements not supported in Qlik Sense

QlikView script statements that are not supported in Qlik Sense

Statement	Comments
Command	Use SQL instead.
InputField	

7.2 Functions not supported in Qlik Sense

This list describes QlikView script and chart functions that are not supported in Qlik Sense.

- **GetCurrentField**
- **GetExtendedProperty**
- **Input**
- **InputAvg**
- **InputSum**
- **MsgBox**
- **NoOfReports**
- **ReportComment**
- **ReportId**
- **ReportName**
- **ReportNumber**

7.3 Prefixes not supported in Qlik Sense

This list describes QlikView prefixes that are not supported in Qlik Sense.

- **Bundle**
- **Image_Size**
- **Info**

8 Functions and statements not recommended in Qlik Sense

Most functions and statements that can be used in QlikView load scripts and chart expressions are also supported in Qlik Sense, but some of them are not recommended for use in Qlik Sense. There are also functions and statements available in previous versions of Qlik Sense that have been deprecated.

For compatibility reasons they will still work as intended, but it is advisable to update the code according to the recommendations in this section, as they may be removed in coming versions.

8.1 Script statements not recommended in Qlik Sense

This table contains script statements that are not recommended for use in Qlik Sense.

Script statements that are not recommended

Statement	Recommendation
Command	Use SQL instead.
CustomConnect	Use Custom Connect instead.

8.2 Script statement parameters not recommended in Qlik Sense

This table describes script statement parameters that are not recommended for use in Qlik Sense.

Script statement parameters that are not recommended

Statement	Parameters
Buffer	Use Incremental instead of: <ul style="list-style-type: none">• Inc (not recommended)• Incr (not recommended)

8 Functions and statements not recommended in Qlik Sense

Statement	Parameters
LOAD	<p>The following parameter keywords are generated by QlikView file transformation wizards. Functionality is retained when data is reloaded, but Qlik Sense does not provide guided support/wizards for generating the statement with these parameters:</p> <ul style="list-style-type: none">• Bottom• Cellvalue• Col• Colmatch• Colsplit• Colxtr• Compound• Contain• Equal• Every• Expand• Filters• Intarray• Interpret• Length• Longer• Numerical• Pos• Remove• Rotate• Row• Rowrnd• Shorter• Start• Strrnd• Top• Transpose• Unwrap• XML: XMLSAX and Pattern is Path

8.3 Functions not recommended in Qlik Sense

This table describes script and chart functions that are not recommended for use in Qlik Sense.

8 Functions and statements not recommended in Qlik Sense

Functions that are not recommended

Function	Recommendation
NumAvg	Use Range functions instead. <i>Range functions (page 1290)</i>
NumCount	
NumMax	
NumMin	
NumSum	
Color()	Use other color functions instead. QliktechBlue() can be replaced by RGB(8, 18, 90) and QliktechGray can be replaced by RGB(158, 148, 137) to get the same colors.
QliktechBlue	 <i>Color functions (page 531)</i>
QliktechGray	
QlikViewVersion	Use EngineVersion instead. <i>EngineVersion (page 1439)</i>
ProductVersion	Use EngineVersion instead. <i>EngineVersion (page 1439)</i>
QVUser	
Year2Date	Use YearToDate instead.
Vrank	Use Rank instead.
WildMatch5	Use WildMatch instead.

ALL qualifier

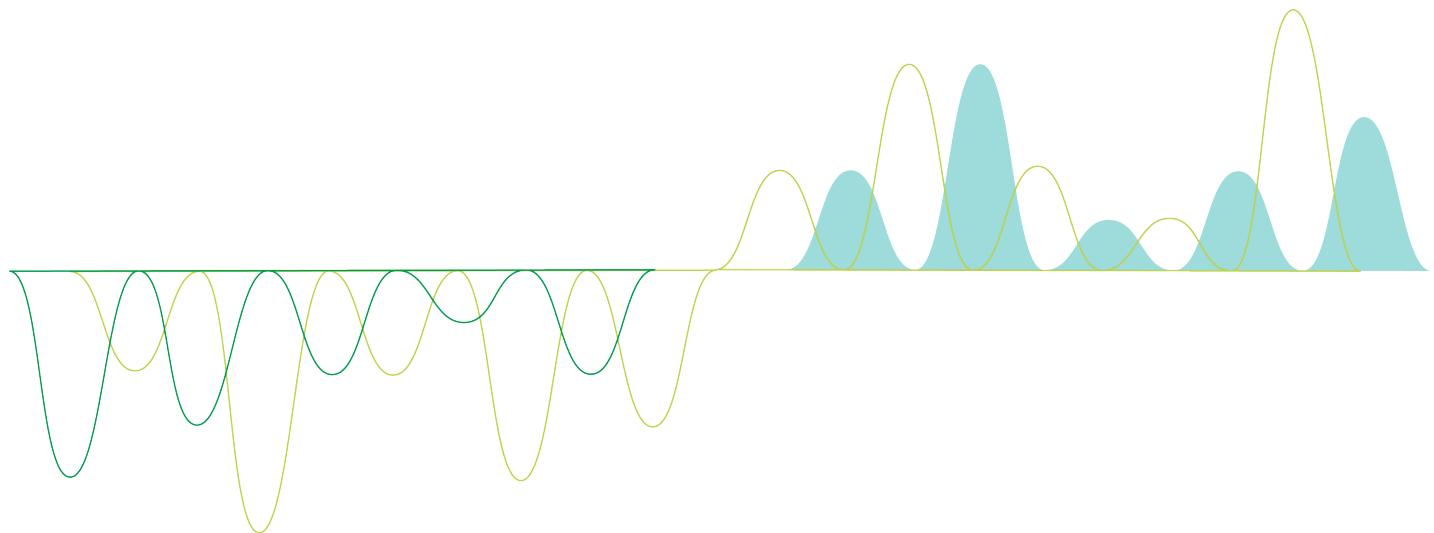
In QlikView, the **ALL** qualifier may occur before an expression. This is equivalent to using **{1} TOTAL**. In such a case the calculation will be made over all the values of the field in the document, disregarding the chart dimensions and current selections. The same value is always returned regardless of the logical state in the document. If the **ALL** qualifier is used, a set expression cannot be used, since the **ALL** qualifier defines a set by itself. For legacy reasons, the **ALL** qualifier will still work in this version of Qlik Sense, but may be removed in coming versions.

Explore, discover and analyze

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 About this document	5
2 Discover and analyze	6
2.1 Routine analysis	6
2.2 Exploratory analysis	6
3 Exploring data with visualizations	7
3.1 Selection preview	7
No selection	7
A selection is made	7
A second selection is made	8
3.2 Types of selections in visualizations	9
Limitations	10
Click selection	10
Draw selection	11
Label selection	13
Lasso selection	13
Legend selection	14
Range selection	14
3.3 The associative selection model	15
Selection states	15
3.4 Viewing data of visualizations	20
Visualizations where data viewing is available	21
Changing between visualization and data view	21
3.5 Visual exploration	21
Using the visual exploration menu to change properties	22
Using the visual exploration menu to change data	23
Using the visual exploration menu on mobile devices	23
3.6 Scrolling in visualizations	24
Using lasso selection with scrolling	24
Visualizations where lasso selection needs to be enabled	25
3.7 Canceling data retrieval	26
4 Bookmarking selections	27
4.1 Creating bookmarks	27
Creating a bookmark	27
States and set expressions	28
Bookmark options	28
Searching for bookmarks	28
Changing the title and description of a bookmark	29
4.2 Setting a default bookmark to create an app landing page	29
Set a default bookmark	30
4.3 Deleting bookmarks	30
Deleting a bookmark in sheet view	30
Deleting a bookmark from app overview	31
5 Exploring with selections	32
5.1 Selection options	34
Select all	34
Select possible	34

Contents

Select alternative	34
Select excluded	34
5.2 Searching within selections or visualizations	34
Search types	36
Text search	37
Numeric search	43
Fuzzy search	45
Expression search	45
Compound search	47
5.3 Editing the selections	50
5.4 Locking and unlocking selections	50
Locking selections	50
Unlocking selections	51
5.5 Stepping back and forward in selections	51
5.6 Using the selections tool	52
Making and clearing selections	53
Searching in the App dimensions section	53
Scrolling in the selections tool	53
Generating insights	54
5.7 Discovering your data with associative insights	54
Limitations	54
Associative insights selections view	54
Insights card details view	57
Insights card KPI view	59
Generating insights	60
Associative insights example: No data left behind	60
Disabling associative insights in an app	63
Troubleshooting associative insights	64
6 Using smart search	65
6.1 What happens when you search	65
A: Search field	66
B: Apply a selection	66
C: Color-coded search result terms	66
6.2 Using search results to change selections	67
Interacting with search results for data	67
6.3 Keyboard shortcuts used in smart search	69
7 Troubleshooting - Discover	70
7.1 My search does not produce any results	70
7.2 My search using Insight Advisor does not produce any results	70
7.3 Incomplete visualization	70

1 About this document

Read and learn how to make discoveries in your data, using different tools.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

You find the online help, additional guides and much more at help.qlik.com/sense.

2 Discover and analyze

When you have created your app and loaded data into it you can start using it for data discovery and analysis.

2.1 Routine analysis

It is typical in routine analysis to follow up on key metrics on a regular basis. Here are some examples of KPIs you might want to keep a close watch on:

- Total sales versus quota each morning
- Total sales versus total sales the same period last year
- Orders placed but not delivered at the end of the week
- Sales per region on a certain day each month

2.2 Exploratory analysis

Sometimes when you are analyzing data, you might find that something is missing in the app that you have access to. Even though Qlik Sense allows for efficiently filtering the data by making multiple selections, you might want to adapt the existing visualizations, dimensions or measures to be able to explore the data for new insights.

3 Exploring data with visualizations

You make selections by clicking and drawing lines in the different visualizations.

When you make a selection, all associated visualizations are updated immediately to reflect the selection. You confirm the selection by clicking ✓, or by clicking anywhere on the sheet outside the visualization, including in another visualization, (in which case you generate a new selection). You can also press Enter to confirm.

You cancel a selection by clicking ✗. You can also press Esc to undo.

By default, new selections in a visualization are added to the previous ones. You deselect an item by clicking it. On a computer, you can hold down Ctrl while you make a selection, to automatically clear previous selections in a visualization, and only keep the new selection.

3.1 Selection preview

The following images show how the visualizations are updated immediately when a selection is made.

No selection

Sheet view with no selections made.



A selection is made

In this image, a selection is made (in the filter pane *Region*) and is reflected in all associated visualizations.

A selection made in the Region filter pane.



A second selection is made

In this image, a second selection is made (in the bar chart *Total Sales*). It automatically confirms the first selection and presents a preview of the new selection.

A second selection has been made in the bar chart Total Sales.



When making selections in filter panes there is a difference between **Cancel selection** (X) and **Clear selection** (C-X). With X, you only clear the latest selection, but C-X clears all selections.

3.2 Types of selections in visualizations

When you analyze your data, you have different ways of making selections. The charts and tables have different selection patterns. Some selection types are particularly useful for certain visualizations. The following table displays which kinds of selections that are supported in the visualizations.

Selections that are supported in the visualizations

Visualizations	Click selection	Draw selection	Range selection	Lasso selection	Legend selection	Label selection
Bar chart	Yes	Yes	Yes	Yes	Yes	Yes
Box plot	Yes	Yes	Yes	Yes	-	Yes
Combo chart	Yes	Yes	Yes	Yes	Yes	Yes
Distribution plot	Yes	Yes	Yes	Yes	Yes	Yes
Filter pane	Yes	Yes	-	-	-	-

Visualizations	Click selection	Draw selection	Range selection	Lasso selection	Legend selection	Label selection
Gauge	-	-	-	-	-	-
Histogram	Yes	Yes	Yes	Yes	-	-
KPI	-	-	-	-	-	-
Line chart	Yes	Yes	Yes	Yes	Yes	Yes
Map	Yes	Yes	-	Yes	Yes	-
Pie chart	Yes	Yes	-	Yes	Yes	Yes
Pivot table	Yes	Yes	-	-	-	-
Scatter plot	Yes	Yes	Yes	Yes	-	-
Table	Yes	Yes	-	-	-	-
Text & image	-	-	-	-	-	-
Treemap	Yes	Yes	-	Yes	-	-
Waterfall chart	-	-	-	-	-	-

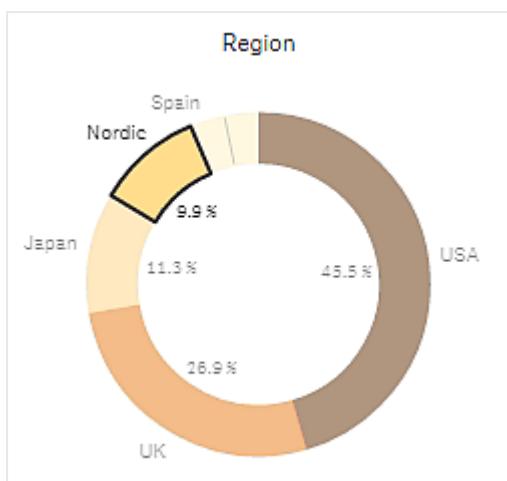
Limitations

- You cannot make selections in **Gauge**, **KPI**, **Waterfall chart**, and **Text & image** visualizations.
- Legend selection is not available in a visualization when coloring by expression.
- Range selection is only available on the dimension axis for stacked bar charts or combo charts, and box plots.
- You cannot select a measure by name.

Click selection

You click to select single values/data points, one at a time. If you want to deselect a value/data point, click it.

Pie chart with the sector Nordic selected.

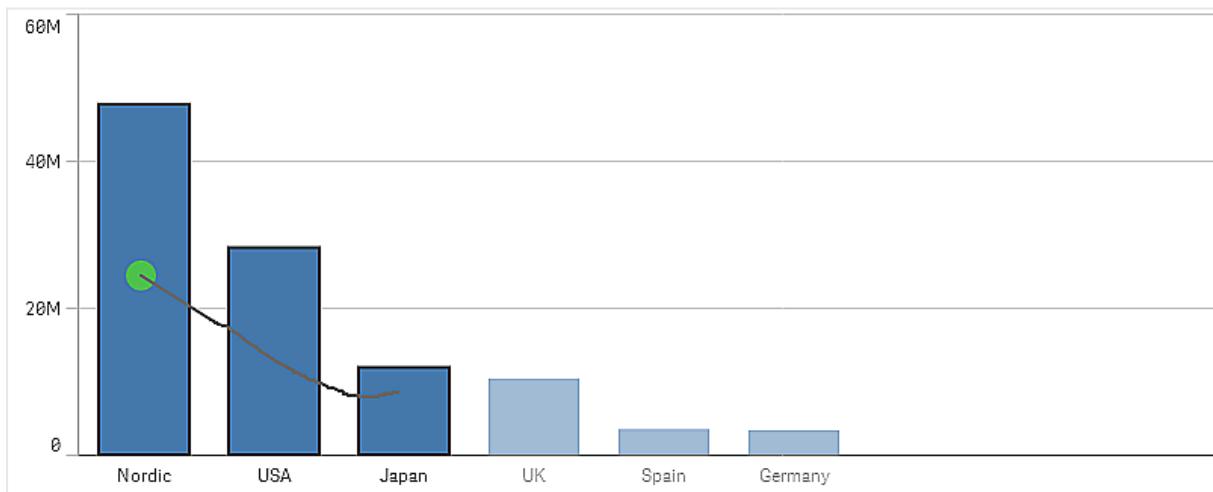


Draw selection

When you want to make a draw or lasso selection, you must first click inside the visualization and turn on lasso selection by clicking the lasso icon  at the top of the visualization. On a computer, you can also press Shift and make the selection.

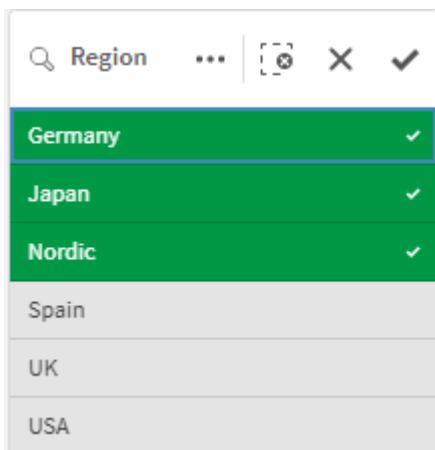
You draw a freehand line to select several values/data points at a time. You cannot draw to deselect values/data points.

Bar chart with selections Nordic, USA and Japan made with draw selection.



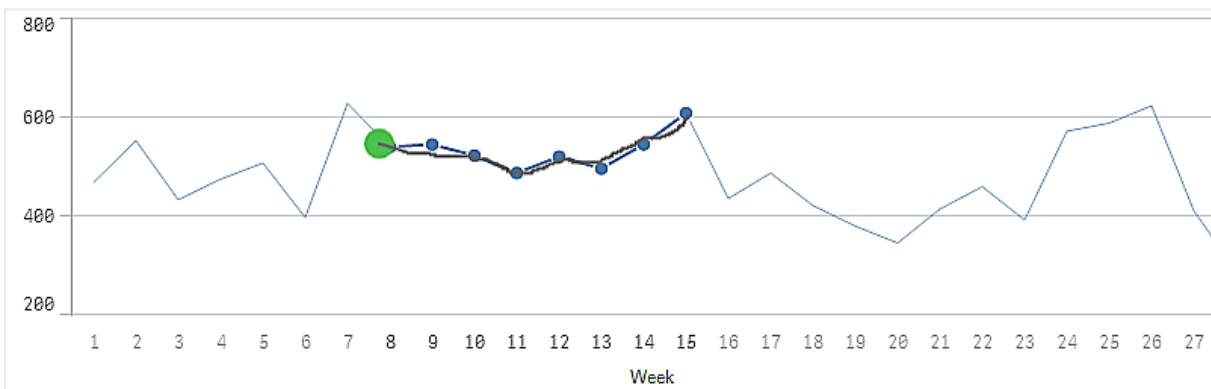
In a table or a filter pane, you draw across several values to select them.

Filter pane with selections Germany, Japan and Nordic made with draw selection.



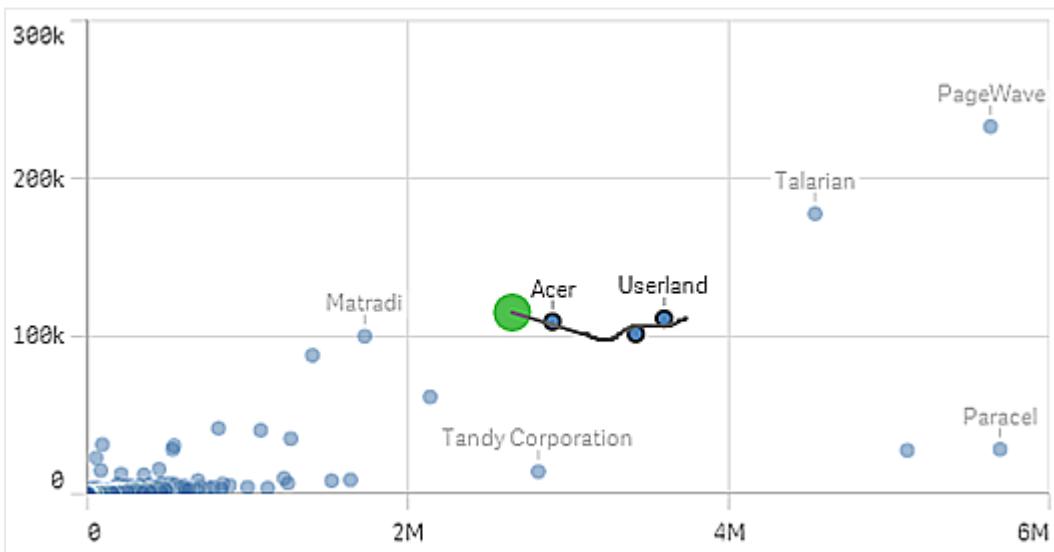
In a line chart you draw along a line to select a number of data points.

Line chart with weeks selected with draw selection.



In a scatter plot you draw across a number of data points to select them.

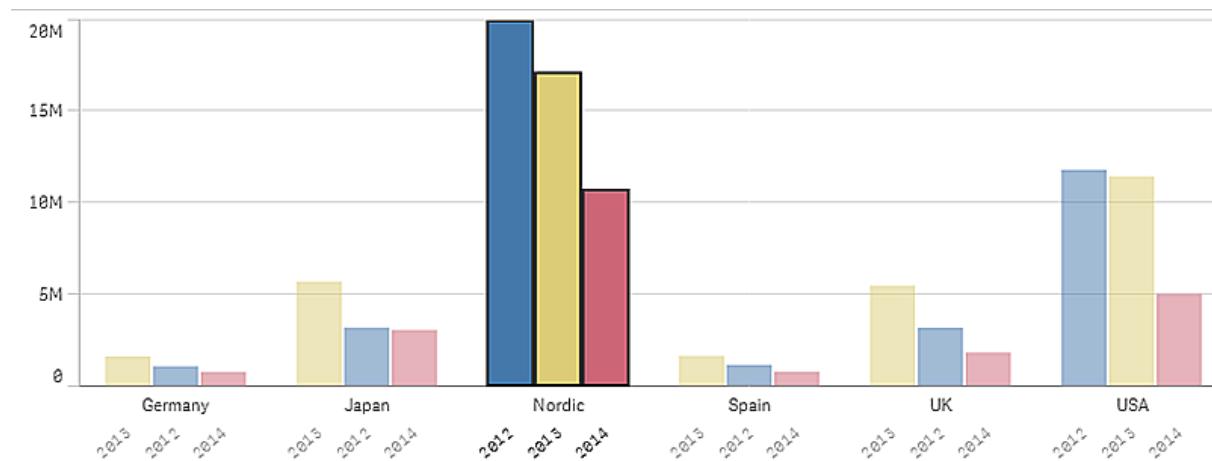
Scatter plot with selections made with draw selection.



Label selection

You can click the dimension labels to make selections. When dimensions are grouped or stacked, the whole group or stack is selected.

Bar chart with label selection of 2011, 2012, and 2013. Clicking any of the years selects the whole group.

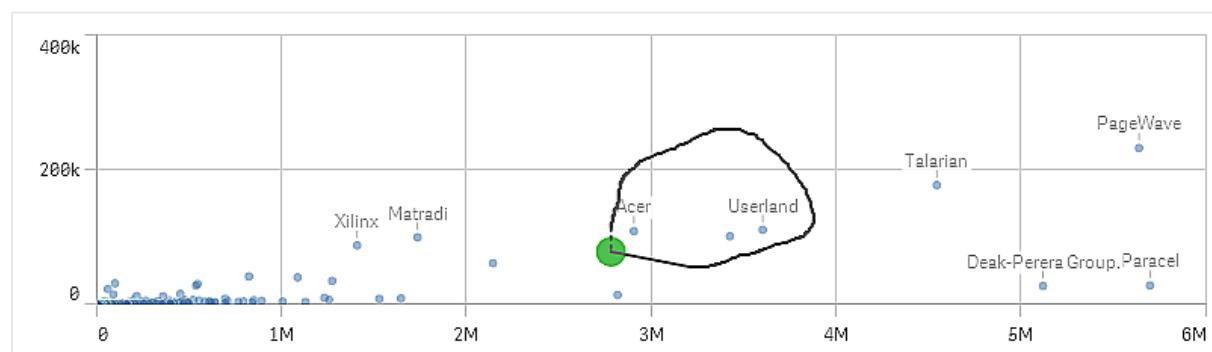


Lasso selection

When you want to make a draw or lasso selection, you must first click inside the visualization and turn on lasso selection by clicking the lasso icon  at the top of the visualization. On a computer, you can also press Shift and make the selection.

You draw a freehand circle to capture and select data points.

Selection of values made in a scatter plot using lasso selection.



Your lasso selections only include visible data points. For charts using a continuous axis, data points not visible in the chart will be excluded, even if they are within the area being selected.

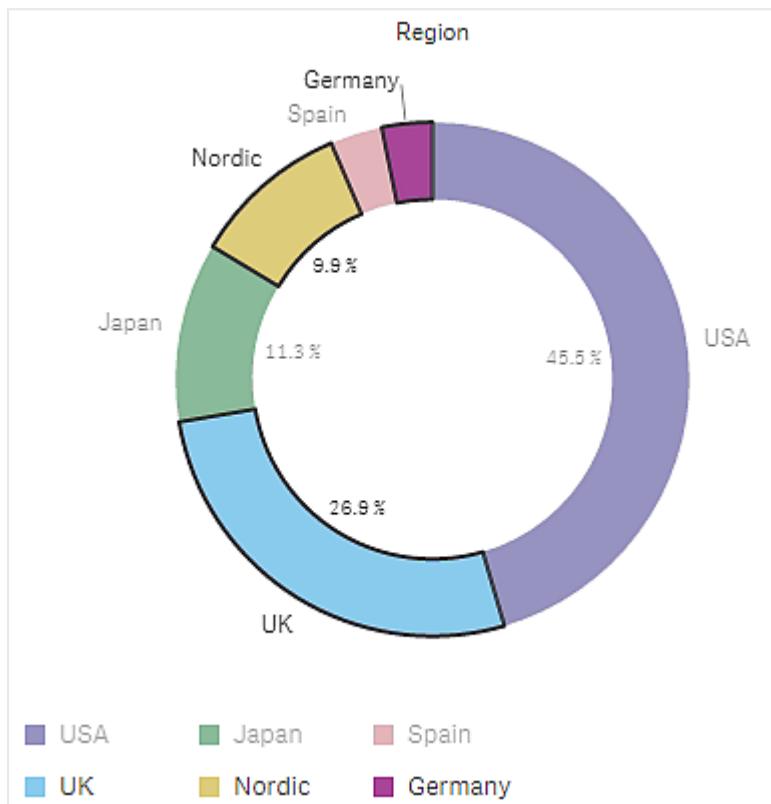
Legend selection

You can click any legend item, except the item representing **Others**, to select the associated values.



Legend selection is not available in a visualization when coloring by expression.

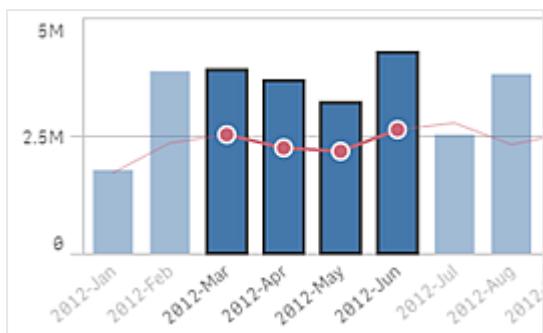
Pie chart with sectors Nordic, Germany and UK selected, using legend selection.



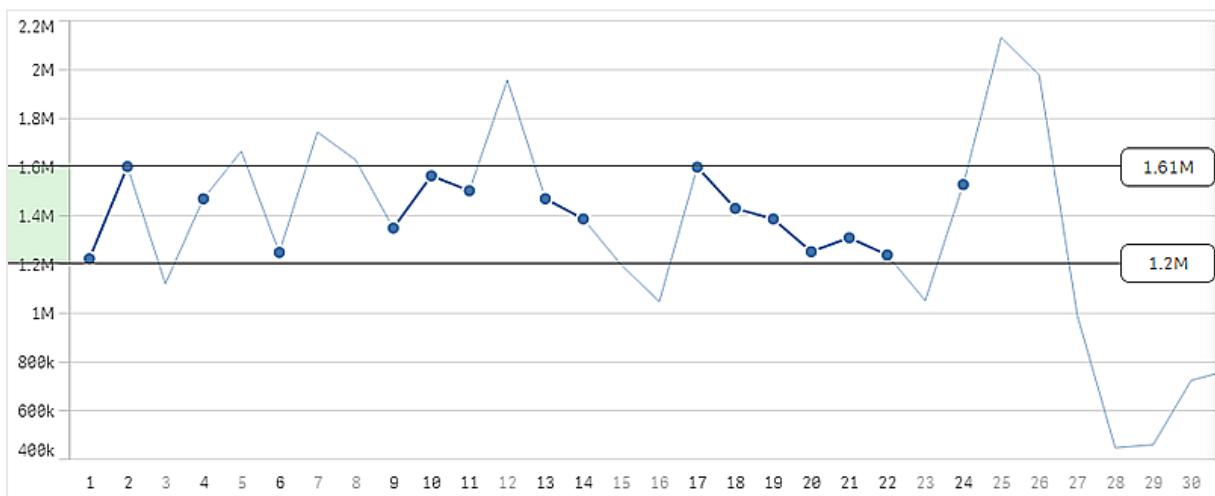
Range selection

You draw your selections on the y-axis or the x-axis. For an axis with measure values, you can click on the range bubble to enter a specific numeric value.

Combo chart with selections made with range selection.



Line chart with selections made with range selection.



3.3 The associative selection model

Making selections is the main interaction method in Qlik Sense. Selections filter out a subset of the data that is loaded into Qlik Sense. You use selections to focus on something you want to know more about. Qlik Sense responds by color coding values according to their different states.

You can think of your selections as an input for Qlik Sense. Qlik Sense evaluates the input and displays the color codes on data values as the output.

- The input state: the selection that you have made – whether the field value is selected or not.
- The output state: whether the field value is possible or not, given the logical inference of the selection.

Selection states

When you make selections, the colors of the values change accordingly. Color-coding is used in filter panes, selections list items, and the selections tool, with the characteristic Qlik Sense colors green, white, and gray. The colors bring you information about which field values are selected, alternative, possible and excluded, respectively.

The following table lists which colors are used for the different states.

Colors used for different states

Color	State
Green, with a check mark as a selection indicator	Selected
White	Possible
Light gray	Alternative
Dark gray	Excluded
Dark gray with a check mark as a selection indicator	Selected excluded

The selected state

When you select one or more values in a filter pane and the values turn green, they are in the selected state. In the following image, the value *1910s* has been selected. The selection filters out a subset of the data that is loaded, and the filter panes *Decade* and *Year* are updated according to the selection.

1910s in the *Decade* filter pane has been selected.

Decade	Year
1910s	1914
1920s	1915
1930s	1916
1940s	1917
1950s	1918
1960s	1919
1970s	1920
1980s	1921
1990s	1922

The filter panes have four states altogether. Apart from the selected state (green), there are possible values (white), light gray values (alternative), and dark gray values (excluded). These states are explained in the following sections.

The possible state

In the *Year* filter pane, the years *1914* up to *1919* are white (possible), because these values are all years from the *1910s*, the selected value in *Decade*. All possible values are 'associated' with the value *1910*. You could refine your selection by selecting one or more of the possible values.

1914 to *1919* in the *Year* filter pane are possible values.

Decade	Year
1910s	1914
1920s	1915
1930s	1916
1940s	1917
1950s	1918
1960s	1919
1970s	1920
1980s	1921
1990s	1922

In the following image, such a refinement has been made. The value *1918* has been selected in the *Year* filter pane.

1918 has been selected in the *Year* filter pane.

Decade	Year
1910s	1918
1920s	1914
1930s	1915
1940s	1916
1950s	1917
1960s	1919
1970s	1920
1980s	1921
1990s	1922

With selections in two filter panes, the possible values are only those that are associated both with *1910s* and *1918*. There is a logical AND condition between selections from different filter panes. A possible value must then be associated both with *1910s* and *1918*.

In the *Year* filter pane, there are no longer any values in the state possible, because none of the values are associated with both *1910s* and *1918*.

The alternative state

In the *Decade* filter pane, the value *1910s* has been selected, and all the other fields in the filter panes have a certain state, depending on their relationship to the selected value.

1910s has been selected and all other fields in the filter panes have different states depending on their relationship with 1910s.

Decade	Year
1910s	1914
1920s	1915
1930s	1916
1940s	1917
1950s	1918
1960s	1919
1970s	1920
1980s	1921
1990s	1922

All the other values in the filter pane *Decade* are light gray, meaning that they are alternative values. The alternative state is used for values that would have been possible if a selection had not already been made in that field. Before *1910s* was selected, all the values in the filter pane *Decade* were possible values.

Logically, the alternative values are excluded, but they are only excluded by a single selection (of one or more values), in the same filter pane. If you would clear the selection of *1910s* in *Decade*, all the values would have the state possible.

Even if a value is alternative, you can still select it, but that means that you are, partly, making a new selection rather than refining your original selection. What is useful with alternative values is that you know that there are alternatives available for the same set of selections. If you have a list of sales persons, the alternative values constitute sales persons that may be able to help or replace the selected person.

The excluded state

When a selection is made, values in other filter panes may automatically be excluded, because they are not associated. In the following image, *1910s* has been selected, and as a consequence the values *1920*, *1921*, and *1922* have been excluded. This is an obvious exclusion, because the years *1920*, *1921*, and *1922* are not part of the *1910s*. The other values in *Decade* are alternative, that is, they are excluded but you can still select them and thereby expand the selection. If you were to select *1920s* the value would turn green and have the state selected.

1920, 1921, and 1922 in the Year filter pane have been excluded.

Decade	Year
1910s	1914
1920s	1915
1930s	1916
1940s	1917
1950s	1918
1960s	1919
1970s	1920
1980s	1921
1990s	1922

But if you select one of the possible values in the filter pane *Year*, all the values in *Decade* that were alternative become excluded instead. When only *1910s* was selected they were alternative, but with selections in two filter panes, values that do not match the condition *1910sAND1918* are excluded.

The values that are alternative in *Year* are only excluded by the selection *1918*. They are all associated with the value *1910s* and had the state possible until *1918* was selected.

Decade	Year
1910s	1918
1920s	1914
1930s	1915
1940s	1916
1950s	1917
1960s	1919
1970s	1920
1980s	1921
1990s	1922

The selected excluded state

When you make selections in more than one filter pane, you might run into a fifth state: selected excluded.

As mentioned previously, there are two different states for each field value:

- The input state: the selection that you have made – whether the field value is selected or not.
- The output state: whether the field value is possible or not, given the logical inference of the selection.

A value enters the selected excluded state because the value was first selected, and then excluded by a selection in another field.

For the selected excluded state, the check mark is an indicator that the value was first selected and then excluded, in contrast to excluded values that have never been selected. A dark gray field with a check mark indicates that the value was previously a selected value, but a new selection has then rendered it selected excluded.

Example:

In the following image, the first selection was of the values *1910s* and *1920s*. The values *1910s* and *1920s* were both selected (green) and all the values in the filter pane *Year* were white (possible), since they are all years from the 1910s or 1920s and therefore logically possible values after the first selection. The second selection is of the years *1914*, *1915*, and *1916*. Now, *1920s* is no longer a part of the active selection, since the second selection logically excludes *1920s*. However, *1920s* is still a selected value and therefore it makes sense to denote it as a value that is selected excluded. It was originally selected, but a later selection excluded it. The check mark distinguishes it from the excluded values that have never been selected.

The dark gray value with a check mark is selected excluded.

Decade	Year
1910s	✓
1920s	✓
1930s	
1940s	
1950s	
1960s	
1970s	
1980s	
1990s	
	1914
	1915
	1916
	1917
	1918
	1919
	1920
	1921
	1922

3.4 Viewing data of visualizations

You can change between a visualization and a view of the data it represents.

When working with a visualization in analysis mode, you might need to take a look at the data behind a selection. For most visualizations, a table containing their data is available through the context menu.

Visualizations where data viewing is available

In the following visualizations you can change between the visualization and a view of its data in table form:

- Bar chart
- Box plot (with one or more dimensions)
- Combo chart
- Distribution plot
- Histogram
- Line chart
- Pie chart
- Pivot table
- Scatter plot
- Treemap

Changing between visualization and data view

Do the following:

1. Right-click on the visualization or click the hover menu ***.
2. Select **View data**. If a selection has been made only the selected data will be shown.
3. Right-click on the view of data and select **View chart** to go back to the visualization.



When you enter edit mode, or if you navigate to another sheet, any visualizations changed to data view will change back to the original visualizations.

3.5 Visual exploration

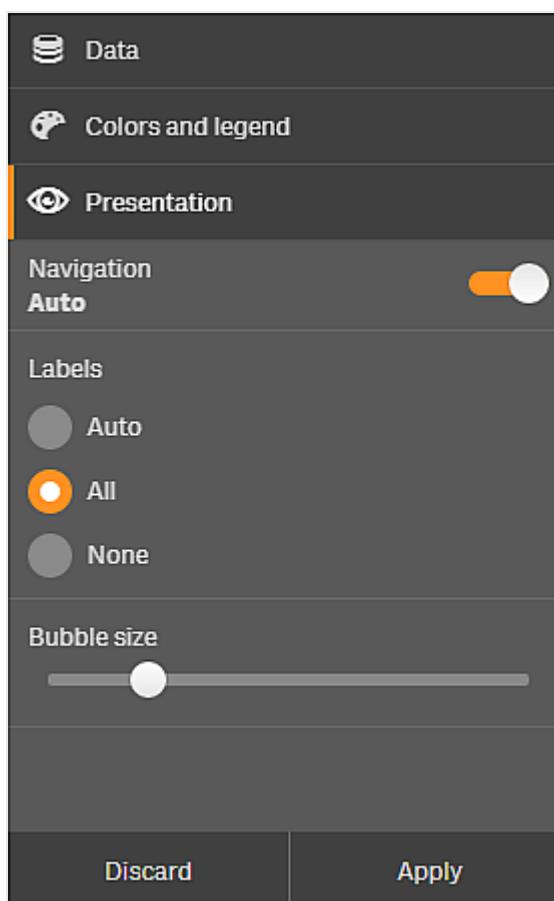
The visual exploration menu lets you change certain visualization properties, without making selections or editing the sheet. For example, you can change data, sort data, color by dimension or measure, and change how labels are displayed.

For a visual demo about how to use the visual exploration menu, see [Visual explorations](#).



The visual exploration menu is available for the following visualization: bar chart, line chart, pie chart, scatter plot, treemap, box plot, distribution plot, map, and combo chart.

Example of visual exploration menu for a scatter plot visualization.



Using the visual exploration menu to change properties

Do the following:

1. When analyzing, hover over the visualization you want to change.
2. Click at the top right of the visualization or right-click on the visualization and select **Exploration menu**.
3. Update the properties you want to change.
4. To close the menu and save your changes, click . The changes are saved during this session.
To save your changes for future sessions (and have them updated in the properties panel), click **Apply**.
This button is only available for unpublished sheets, visualizations that are not master items or linked to master items, and for users with rights to edit the sheet.



If you do not click **Apply** to save the changes or **Discard** to discard the changes and later click **Edit** to edit the sheet, you will be prompted to select whether to apply or discard the changes you made when analyzing the sheet.

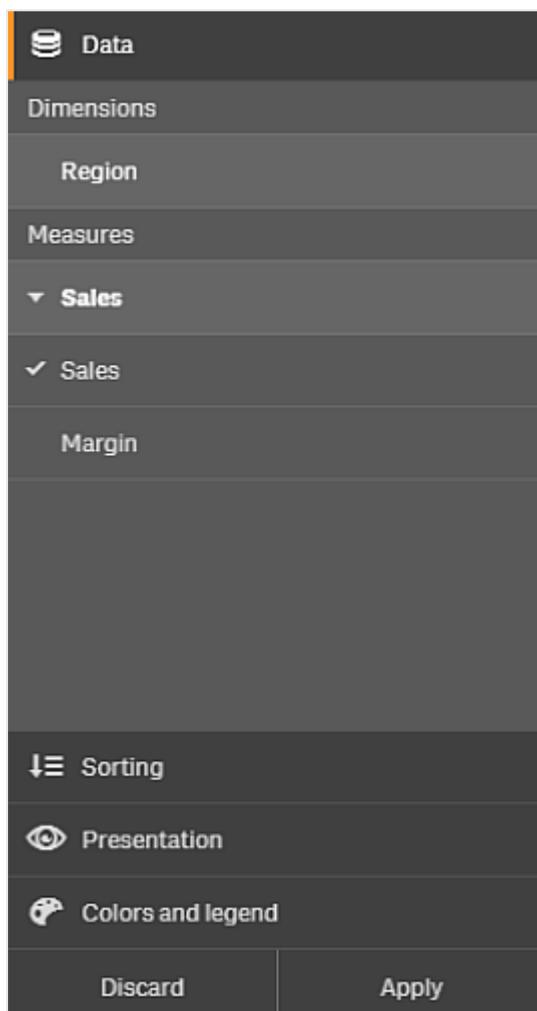
Using the visual exploration menu to change data

The visual exploration menu can change visualization data, if you have alternative dimensions or measures. Simply click on the dimension or measure you want the visualization to display. The selection is marked with a ✓.



Alternative dimensions and measures are dimensions and measures that are added to a visualization, but are not displayed until a user chooses to switch which dimensions and measures are being displayed during visual exploration.

Example of visual exploration menu when changing data



Using the visual exploration menu on mobile devices

When you are using Qlik Sense on a very small screen (640 pixels wide or smaller), you access the visual exploration menu by doing the following:

1. Tap the visualization you want to change to open it in full screen.
2. Tap at the top of the visualization or long-touch on the visualization and select **Exploration menu**.
3. Update the properties you want to change.
4. To get a preview of what the changes will look like, long-touch outside of the menu on the visualization, and the menu will slide to the side. Release to open the menu again and continue doing your changes.
5. To close the menu and save your changes, tap at the top of the visualization, or long-touch and select **Close exploration menu**.

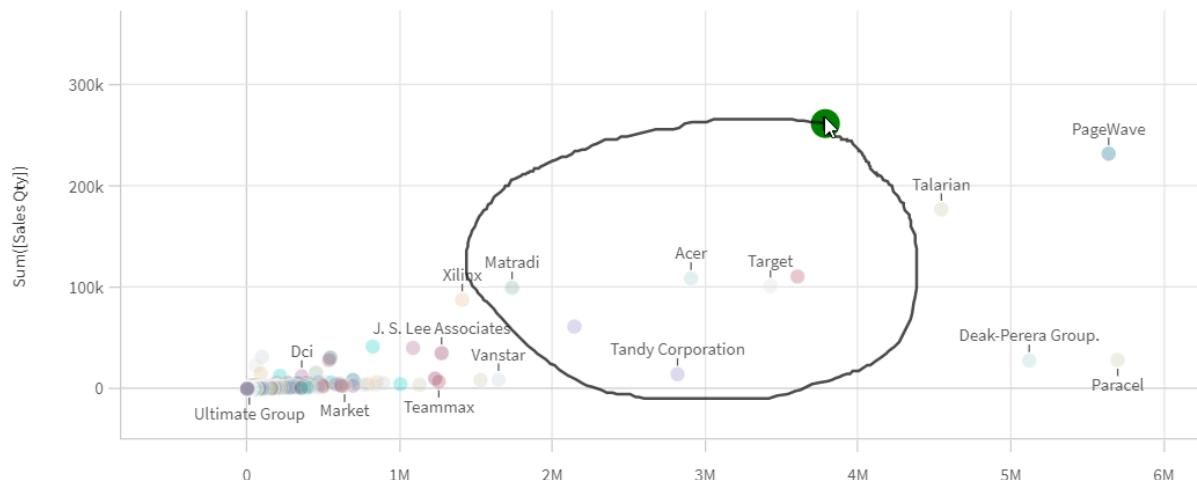
3.6 Scrolling in visualizations

You can swipe/drag to pan and scroll in visualizations and then turn on lasso selection to draw and select.

When working with visualizations, you often need to scroll to find the data you are looking for. Especially on a touch device, the most natural way of scrolling is to swipe. You scroll by swiping to the data that you want to select and then make your selection. When you scroll, draw selection and lasso selection are disabled, so as not to interrupt the scrolling and to avoid making accidental selections. The other selection options are available as usual.

Scatter plot with selections made with lasso selection.

Customer Sales and Quantity



Using lasso selection with scrolling

When you make a lasso selection, the interaction differs depending on what device you are using.

Touch device interaction

Do the following:

1. Tap  to turn on lasso selection.
2. Draw to make a selection.
You can make consecutive selections.
3. Confirm the selection.

Use two-finger-swipe if you need to scroll and pan between selections.

Computer (mouse) interaction

Do the following:

1. Press Shift and draw to make a selection.
You can make consecutive selections. Lasso selection is turned on for as long as Shift is pressed.
2. Confirm the selection.

Alternative procedure

Do the following:

1. Click inside the visualization without making a selection.
Selection options are displayed at the top of the visualization.
2. Click  to turn on lasso selection.
3. Make and confirm the selection.

You can click  to turn lasso selection on and off if you need to scroll and pan between selections.

Visualizations where lasso selection needs to be enabled

In the following visualizations you need to activate lasso selection:

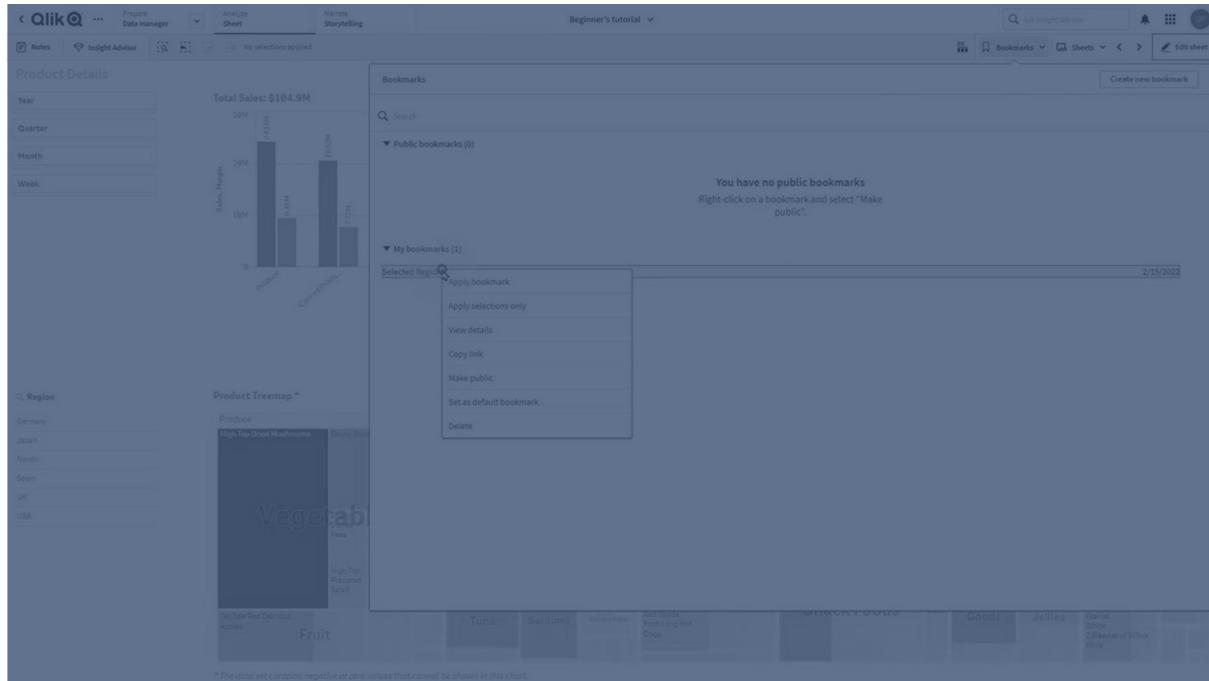
- Bar chart
- Box plot
- Combo chart
- Distribution plot
- Line chart
- Map
- Pie chart
- Scatter plot
- Treemap

3.7 Canceling data retrieval

When you use Qlik Sense to explore your data on a server, you might experience a delay. If the waiting time is long, a Cancel button displays on each visualization. Use the Cancel buttons to stop the data retrieval. You must stop each visualization separately. When you stop a data retrieval, a Retry button appears so you can try again.

4 Bookmarking selections

When you analyze data, you might find something interesting that you want to return to, or share with others. Using bookmarks is an easy way to save a specific selection state on a sheet.



Bookmarks save your selections and a particular sheet location. Bookmarks can later on be opened to restore the selections to a former state. You can apply bookmarked selections to any sheet that has the same data as the sheet used to create the bookmark. All bookmark tools can be found under  in the toolbar.

If you use alternate states in the app and create a bookmark, the bookmark will capture the selections of all states.

To manage bookmarks you will need to know how to:

[Create bookmarks](#)

[Set a default bookmark to create an app landing page](#)

[Delete bookmarks](#)

4.1 Creating bookmarks

Bookmarks let you save specific selection states. This allows you to review them later, and share them with other users.

Creating a bookmark

Do the following:

1. Make the selections on the sheet that you want to save as a bookmark.
2. Click **Bookmarks** in the toolbar.
3. Click **Create new bookmark**.
4. **Title:** By default, the name of the sheet and a summary of the selections are used as the title of the bookmark. You can change this if needed.



Do not use a name already used by an alternate state.

5. **Description:** You have the option to enter a description of the bookmark.
6. **Save sheet location:** Toggle this on if you want the bookmark to switch to the sheet that was open when it was created. Toggling it off means that the user will stay in their current sheet when the bookmark is applied.
7. **Save layout:** Toggle this on if you want to save chart layouts, sorting, or expansions.
8. Click **Create**.

States and set expressions

When creating or editing a bookmark, you will see the possible alternate states that have been bookmarked. If a state includes selections, then the set expression for the selections is also shown.

You can copy the set expression by clicking **Copy**.



If the bookmark selection includes a calculated dimension, the set expression will show MISSING VALUES and cannot be used.

Bookmark options

In sheet view, if you right-click on a bookmark, you will see the following options:

- **Apply bookmark:** The selections saved in the bookmark are applied and the sheet which the bookmark originates from is displayed. Any previous selections are cleared.
- **Apply selections only:** The selections saved in the bookmark are applied. Any previous selections are cleared.
- **View details:** Displays location, layout state, and set expressions.
- **Copy link:** Copies the bookmark location so it can be shared.
- **Make public:** Making a bookmark public means that anyone can use it. You will no longer be the owner of the bookmark.
- **Set as default bookmark:** When the app is opened, the layout state and the selection state of the default bookmark are used, instead of the app overview page. See *Setting a default bookmark to create an app landing page (page 29)*.
- **Delete:** The bookmark is deleted.

Searching for bookmarks

Do the following:

1. In sheet view, click **Bookmarks** in the toolbar.
2. Type your search criteria in the search field.
The list is filtered as you type.



When you search in bookmarks, Qlik Sense looks for matches in titles and descriptions.

Changing the title and description of a bookmark

You can change the title and description of a bookmark.

Do the following:

1. In sheet view, click **Bookmarks** in the toolbar.
2. Click next to the bookmark you want to edit.
3. Click .
4. Make your changes to **Title** and **Description**.
5. Click to stop editing.

The changes are automatically saved.



You can also edit bookmarks from the app overview using the same procedure.

4.2 Setting a default bookmark to create an app landing page

You can select a sheet to be the landing page of your app by setting a default bookmark. When you open the app, the layout state and the selection state of the default bookmark are used.

If you do not set a default bookmark, the app overview is displayed when the app is opened.

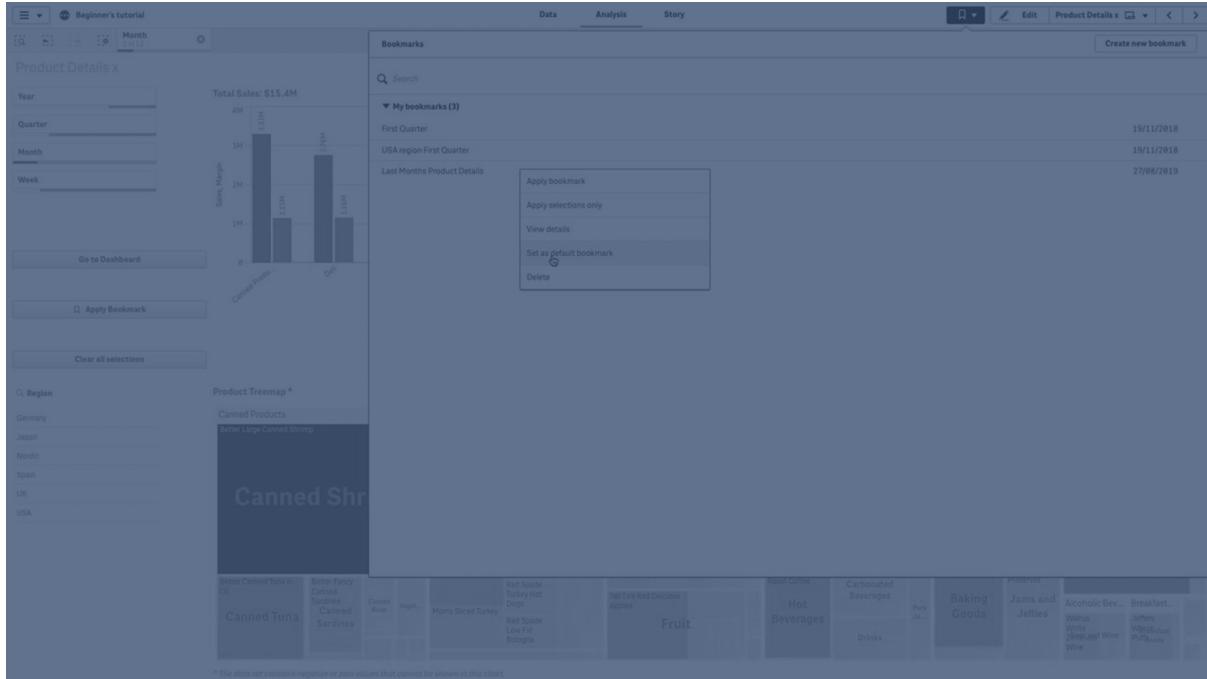
Consider the following when using default bookmarks:

- You can only set or remove a default bookmark in an unpublished app.
- You can only set one default bookmark for an app.
- You can clear the selection of the default bookmark when viewing a sheet. However, the bookmark will be applied when you re-open or reload the sheet.
- In a published app, the app consumer can clear the selection of the default bookmark and make other selections. However, the selection state of the default bookmark will be re-applied when the app consumer re-opens or reloads a sheet.
- When you duplicate an app, the default bookmark is duplicated with the app.
- If you have the same app open in an active session in another tab, the default bookmark will not override the selections made in the other tab.

You can use a default bookmark in conjunction with the **Always one selected value** for a field to highlight specific selections for your app consumer.



In previous versions of Qlik Sense, the default bookmark would only apply the selection state of the bookmark.



Set a default bookmark

Do the following:

1. Click **Bookmarks** in the toolbar of a sheet. The **Bookmarks** window opens.
2. Right-click an existing bookmark, and then click **Set as default bookmark**.
3. Click the bookmark. The **Bookmarks** window closes. The bookmark is shown in the top toolbar, and the selections are shown in the app.
4. To test that the default bookmark is working properly, close and then reopen the app. The default bookmark sheet and selections should be shown.

To remove a bookmark as the default, open the **Bookmarks** window in an unpublished app, and then right-click the default bookmark. Click **Remove as default bookmark**. The next time you open the app, the app overview is shown.

4.3 Deleting bookmarks

There are several different ways to delete bookmarks.

Deleting a bookmark in sheet view

Do the following:

1. In sheet view, click **Bookmarks** in the toolbar.
2. Click  next to the bookmark you want to delete.
3. Click .
4. Click .
5. To confirm that you want to delete the bookmark, click **Delete** in the dialog.

The bookmark is deleted.

Deleting a bookmark from app overview

Do the following:

1. From the app overview, click **Bookmarks** to view the bookmarks.
2. Click  next to the bookmark you want to delete.
3. Click .
4. Click .
5. To confirm that you want to delete the bookmark, click **Delete** in the dialog.

The bookmark is deleted.



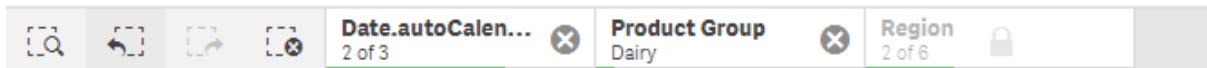
*In the bookmarks dialog, you can right-click on a bookmark and select **Delete**.*

5 Exploring with selections

During analysis, your selections are displayed above the sheet.

Each selection item has a small bar at the bottom that reflects the selection states for that dimension. Three states are displayed in the bars: selected (green), alternative (light gray), and excluded (dark gray). Locked values are indicated by a lock icon.

Selections bar with the selections Year, Product Group and Region made. Region is locked.

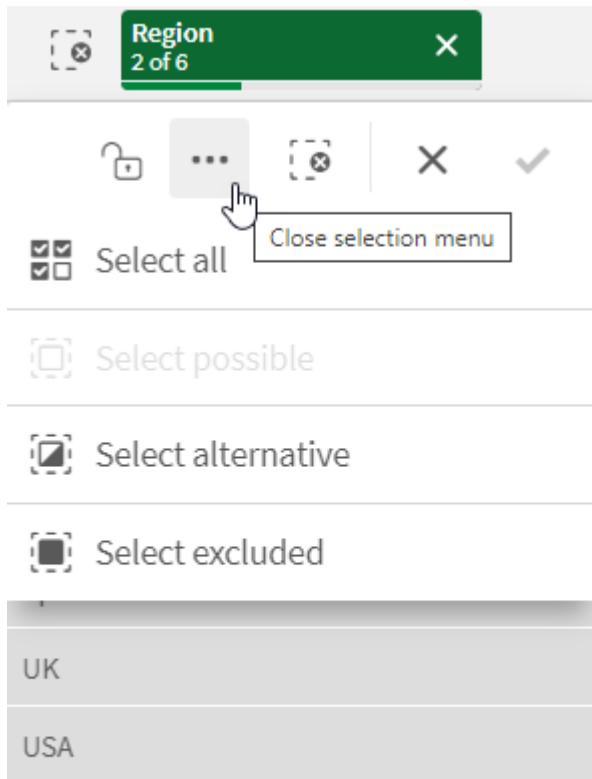


When an app is saved using Qlik Sense, the current selections and locks are not saved. Selections and locks must be made every time the app is opened.

When you click a selection item, a pop-up appears. This lets you view, edit, or clear that selection. You can also search for dimension values or lock the selection. In the following image the selection menu is open. Some options may not be available, depending on what selections were made previously,

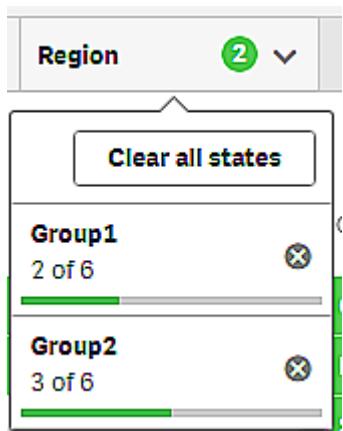
5 Exploring with selections

Selection Region with selection pop-up menu.



If you use alternate states in the app, you can see selections that are made in the states in the selection bar. The field that is used in the state is displayed with the number of alternate states. You can click on the field to show the states, and then click on a state to see the selections. You can also clear the selections of a state or clear all selections.

Selection Region with alternate state pop-up menu for the states Group1 and Group2.



Sheets can have actions that trigger when you navigate to the sheet. Actions can change your selections or states.

5.1 Selection options

Select all

All values are selected (marked ✓). Alternative values change state to selected (green). Excluded values change state to selected excluded. They are still dark gray, but are now selected (marked ✓). If you clear the selections that made these values excluded, they will change state to selected (green).

Select possible

All possible values (white) are selected. This option is never available in the selections item, because when a selection is made, the other values are either alternative or excluded. In a filter pane, however, you can have possible values as a result of another selection.

Select alternative

When a selection has already been made in a field, alternative values, when present, have a light gray color. These are values that would have been possible values (white), if a selection had not already been made in that field.

By selecting alternative values, the values that previously were selected become alternative.

Select excluded

If there are alternative values, they will be selected (green) and the values that previously were selected will change state to alternative. Excluded values will change state to excluded.

If there are no alternative values, the excluded values are selected (green), and the previously selected values change state to alternative.

5.2 Searching within selections or visualizations

You can search for values and make selections from the resulting filtered list. You can search selection items in the selection bar, and within visualizations such as filter panes and tables. Search is not case sensitive.

Click a selection item, and in the selection pop-up, type your search string. The list is filtered as you type, to display matching values.

5 Exploring with selections

The result of searching for "Food" in the selection Product Group.

A screenshot of the Qlik Sense search interface. At the top, there is a green header bar with the text "Product Group" and "ALL". Below this is a search bar with a magnifying glass icon containing the text "Food". To the right of the search bar are icons for a lock, three dots, a delete button, and a checkmark. The main area shows a list of search results:

Category	Item	Status
Breakfast	Foods	✓
Frozen	Foods	✓
Starchy	Foods	✓

The result of search for "Mich" in the Manager filter pane.

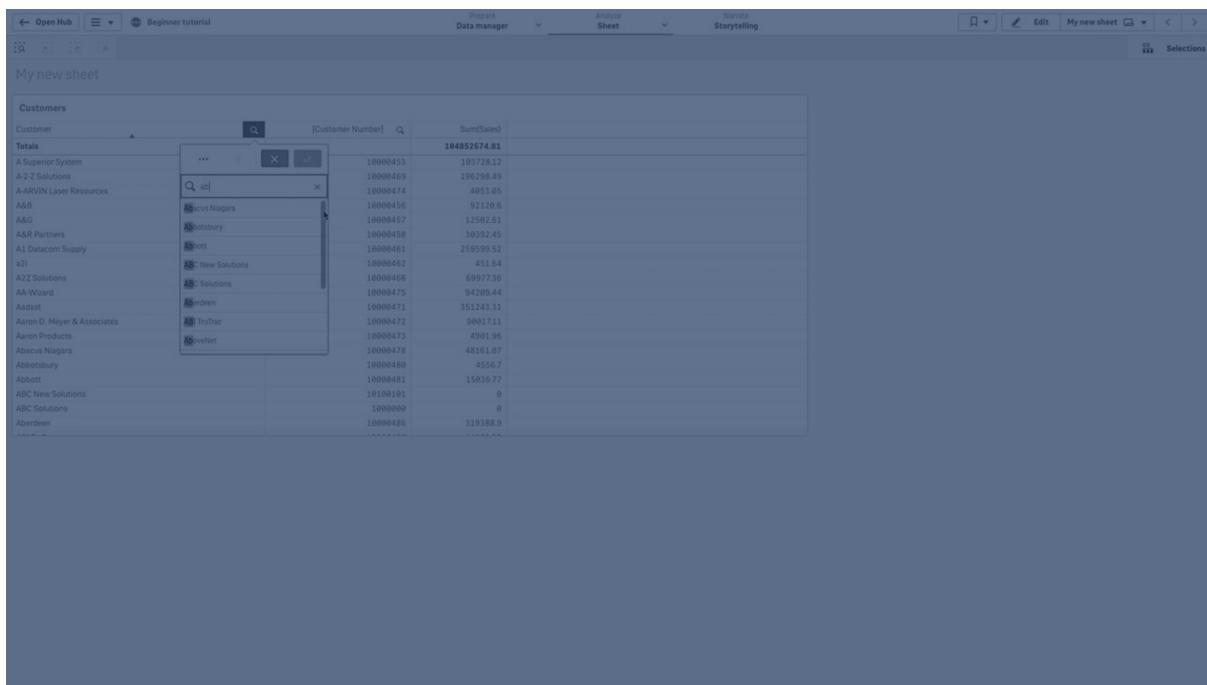
A screenshot of the Qlik Sense search interface. At the top, there is a search bar with a magnifying glass icon containing the text "Manager". Below this is another search bar with a magnifying glass icon containing the text "mich". To the right of these bars are icons for a lock, three dots, a delete button, and a checkmark. The main area shows a list of search results:

Category	Item
	Micheal Williams



You can search for strings that contain up to 5,000 characters.

For a PDF containing examples with versions in several languages, see [Qlik Sense Cheat Sheet version 2.0](#).



Search types

Qlik Sense supports the following methods of searching in selections and visualizations:

Search methods

Search type	Description	Example	Supported data types in search field
Text search	<p>Use text, including quotation marks, wildcards, and modifiers (minus and plus). Text search is separated into two different search modes: normal search and wildcard search.</p> <p>For more information, see <i>Text search</i> (page 37).</p>	*company	Text string, numeric value, dual value
Numeric search	Relational symbols (">", ">=", "<" or "<=") allow values greater than, less than, and so on, to be found. For more information, see <i>Numeric search</i> (page 43).	>=5<20	Numeric value, dual value
Fuzzy search	The tilde character ("~") as a prefix allows inexact matches to be found. For more information, see <i>Fuzzy search</i> (page 45).	~beast company	Text string, numeric value, dual value

Search type	Description	Example	Supported data types in search field
Expression search	An equals sign ("=") indicates an expression. Field values that match the expression are selected. For more information, see <i>Expression search (page 45)</i> .	=sum (Sales)> 1000000	Text string, numeric value, dual value
Compound search	Use search operators to combine multiple searches within a single line. For more information, see <i>Compound search (page 47)</i> .	(*ABC* & ?*Inc*)	Text string, numeric value, dual value

Text search

Text search is the main search method you can use in Qlik Sense. Use quotation marks, wildcards, and modifiers to search for values in a field. As you type your search string, Qlik Sense filters the field values and displays the matching items.

Text search can be split up into two separate search modes:

- Normal search
- Wildcard search

Search modes within text search

Search type	Description	Example	Supported types of data
Normal search	Use text including plus and minus modifiers	ACME -Inc	Character or text string, numeric value, dual value
Wildcard search	Use text including wildcards, excluding plus and minus modifiers	*company	Character or text string, numeric value, dual value

Normal search

Normal text cannot contain wildcards, but can contain plus and minus modifiers. Normal search can only be used interactively and cannot be used in search inside set analysis expressions.

If you perform a normal search, strings that match the search string are displayed. If you use several strings, separated by blanks, each of these is interpreted as a separate search string and displays all field values that contain any of the strings.

5 Exploring with selections

Normal text search using a single search string (no quotation marks)

The screenshot shows a search interface with a search bar containing 'Customer'. Below the search bar is a list of results for the query 'ab'. The results are: ABC New Solution, ABC Solution, Neat ABC Company, and Solutions ABC Inc. Each result item has a yellow background.

Normal text search using two search strings separated by a space (no quotation marks)

The screenshot shows a search interface with a search bar containing 'Customer'. Below the search bar is a list of results for the query 'abc solution'. The results are: ABC Solution, ABC New Solution, Solutions ABC Inc, Neat ABC Company, and Solution Company Inc. Each result item has a yellow background.

The following table contains additional examples of normal searches, with explanations of the results.

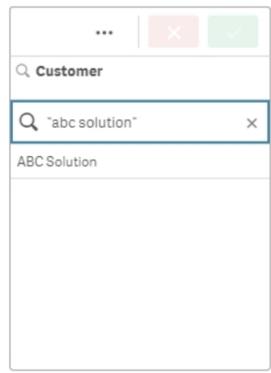
Normal text search examples

Example	Result
"orange juice"	Only finds field values that contain the whole string "orange juice". See <i>Quotation marks</i> (page 38).
orange juice	Without the quotation marks, all fields that contain either "orange" or "juice" would be displayed.
+orange +juice	Finds matches such as "orange juice", "orange and apple juice" and "juice from oranges". See <i>Plus modifier (+)</i> (page 39).
-orange -juice	Excludes results containing orange or juice. See <i>Minus modifier (-)</i> (page 39).

Quotation marks

If you want separate search strings to be interpreted as only one string, use quotation marks (" ") to link the strings together.

Text search using a single search string (contained within quotation marks)



Modifiers

Modifiers allow you to refine the results of your searches by including or excluding values meeting specific conditions.

There are two types of modifiers available:

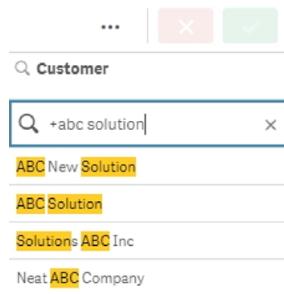
- Plus modifier (+)
- Minus modifier (-)

Plus modifier (+)

By using the plus modifier, you set the condition that strings with a plus sign must be included in the matching items. However, the strings do not necessarily need to be next to each other, nor in the same order as they were entered.

In the following example, the plus modifier is placed next to the value *abc*. The results must contain this value, but order does not matter. The second string, *solution*, is not bound by this condition so it can appear in results but does not need to be present.

Search containing plus modifier (+)



Minus modifier (-)

Placing the minus modifier before a search term excludes results containing that text.

In this example, the minus sign before the word "solution" removes all values containing this string from the results.

Search string with minus modifier (-)

The screenshot shows a search interface with a search bar containing '-solution'. Below the search bar, there is a list of results:

- Best Company
- Company Great Prices
- Neat ABC Company
- Products and Ideas

Combining modifiers

You can combine minus and plus modifiers in a single search. In this example, values containing the string "solution" will be excluded, and the string "abc" must be present in order for the value to appear in the results.

Search combining minus and plus modifiers

The screenshot shows a search interface with a search bar containing '-solution +abc'. Below the search bar, there is a list of results:

- NeatABC Company

Wildcard search

Wildcard search allows you to make your text searches more flexible. Wildcard search can be used in interactive situations and with set analysis. Wildcard search must not include plus or minus modifiers.

You can use one or several wildcards in a search string. The following wildcards can be used:

- * wildcard
- ? wildcard
- ^ wildcard

Wildcard text search examples

Example	Result
a*	Finds all values that begin with the letter "a", including strings with several words where the first word begins with an "a".
*b	Finds all values that end with the letter "b", including strings with several words where the last word ends with a "b".
c	Finds all values that contain the letter "c", including strings with several words.

Example	Result
^ab	Returns all values that have words that begin with “ab”. Equivalent to a normal search for “ab”, but unlike the normal search it can be made more complex using wildcards. It can also be used in a programmatic search, such as in Set Analysis.
r?ck	Finds all values that have four letters and start with an “r”, followed by any character, and ending with “ck”, for example, “rack”, “rick”, “rock”, and “ruck”.
r?? ???d	Finds all values that consist of a three-letter word beginning with an “r” and a five-letter word ending with a “d”.



If you use wildcards, only those records that match the entire search string are displayed; that is, a blank does not imply a logical OR. The search string ‘*creamed’ does not get a match on “Rocky’s creamed corn” since the value does not end with “creamed”. Neither does “creamed*” result in a match on “Rocky’s creamed corn”, since the value does not start with “creamed”.



Space in a search string makes a difference. If you search for “*corn” you get matches on strings ending with, for example, “popcorn” as well as “corn”. If you use a space in your search string, “* corn”, you only get matches that end with “corn”.

* wildcard

The * wildcard is used in the place of zero or more characters, including spaces. This wildcard is flexible and matches any character or any block of characters in a specific position.

In this example, all values starting with the string “company” are listed in the results.

Search string with * wildcard after defined characters

A screenshot of a Qlik Sense search interface. At the top, there are three buttons: an ellipsis (...), a red X, and a green checkmark. Below them is a search bar with a magnifying glass icon containing the text "Customer". Underneath the search bar is another search bar with a magnifying glass icon containing the text "company*". To the right of this search bar is a small 'X' button. Below these search bars, the results are displayed in a table. The first row shows "Company Great Prices" in yellow, indicating it is the active selection.

Here, all values ending with the string “company” are listed in the results.

5 Exploring with selections

Search string with * wildcard before defined characters

The screenshot shows a search interface with a search bar containing the query '*company'. Below the search bar, there are two results listed: 'Best Company' and 'Neat ABC Company'. The results are displayed in a simple list format with horizontal lines separating them.

By placing * wildcards before and after a string, the results will include all values containing this string.

Search string with * wildcard before and after defined characters

The screenshot shows a search interface with a search bar containing the query '*company*'. Below the search bar, there are four results listed: 'Best Company', 'Company Great Prices', 'Neat ABC Company', and 'Solution Company Inc.'. The results are displayed in a simple list format with horizontal lines separating them.

? wildcard

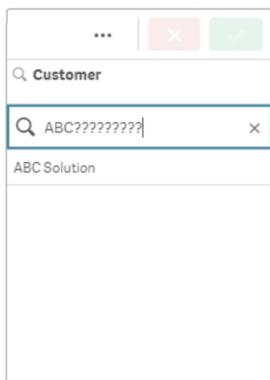
The ? wildcard is used in the place of a single character, including spaces. This wildcard is useful when you suspect that a string may be misspelled, when you are unsure of the spelling, or when the string contains special characters that may be difficult to reproduce correctly.

This wildcard can be substituted for all characters in the string, or used in combination with defined characters.

Search string with ? wildcard symbols for all characters

The screenshot shows a search interface with a search bar containing the query '??????????'. Below the search bar, there are two results listed: 'ABC Solution' and 'Best Company'. The results are displayed in a simple list format with horizontal lines separating them.

Search string with ? wildcard symbols after three defined characters

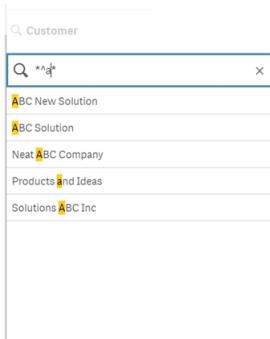


^ wildcard

The ^ wildcard is used to filter values by the character or characters at the beginning of a word within a field value. This wildcard is only used in conjunction with other wildcards.

In this example, the search string "*^a*" will return all values containing a string starting with the letter "a".

*Search string with ^ and * wildcards*



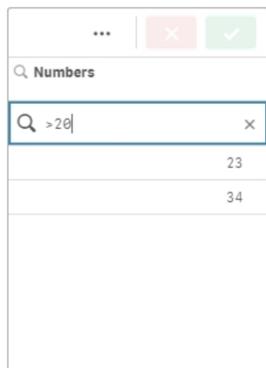
Numeric search

Numeric search is very similar to text search. The only difference is that the search string must begin with one of the relational operators ">" (greater than), ">=" (greater than or equal to), "<" (less than), or "<=" (less than or equal to).

Only values that fulfill the numeric requirement will be matched.

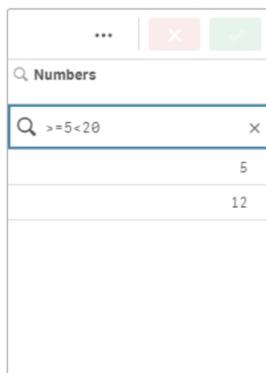
5 Exploring with selections

Numeric search for field values with one comparison (greater than 20)



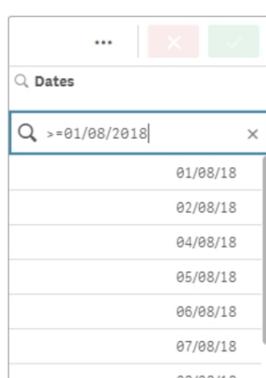
You can combine multiple numeric operators in a single search.

Numeric search for field values with multiple comparisons (greater than or equal to five, and less than 20)



Numeric searches can also be used to filter date fields.

Numeric search for dates on and after January 8, 2018



The following table contains additional examples of numeric searches, with explanations of the results.

Numeric search examples

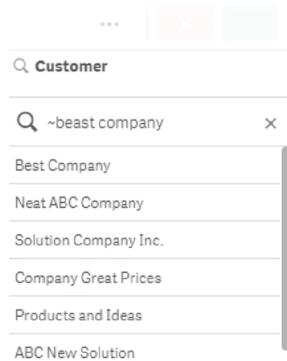
Example	Result
>900	Finds all values greater than 900.
<=900	Finds all values less than or equal to 900.
>900<1000	Finds all values greater than 900 and less than 1000.
<900>1000	Finds all values less than 900 or greater than 1000.

Fuzzy search

Fuzzy search returns a list of values which may not be identical to the search input.

Fuzzy search is similar to a text search, with the difference that it compares and sorts all field values according to their degree of resemblance to the search string. Fuzzy search is especially useful when items may be misspelled. Fuzzy search can also help you find multiple values that are nearly identical.

Fuzzy search



The screenshot shows a search interface with a search bar containing the text "customer". Below the search bar, a dropdown menu displays search suggestions starting with "~beast company". The suggestions listed are: Best Company, Neat ABC Company, Solution Company Inc., Company Great Prices, Products and Ideas, and ABC New Solution. The interface includes standard window controls (minimize, maximize, close) at the top left.

Expression search

With an expression search, you can search for values across all fields associated with the search field.

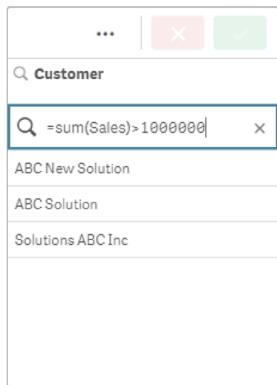
An expression search always begins with an equals sign ("="). The expression is evaluated for every value in every field associated with the search field. All values for which the search expression returns a non-zero value are selected.

In a filter pane with Sales values, you can use a search such as: " $=Sum(Sales) > 1000000$ " to find values larger than 1,000,000. This is a simple search and you could get the same result by using the numeric search: " >1000000 ". Often, an expression search is the only choice. For example, if you want to search for values in associated fields, you have to use an expression search.

In the example below, the search " $=Sum(Sales) > 1000000$ " in the Customer field returns every value in the Customer field for which the Sales column value is greater than 1000000.

5 Exploring with selections

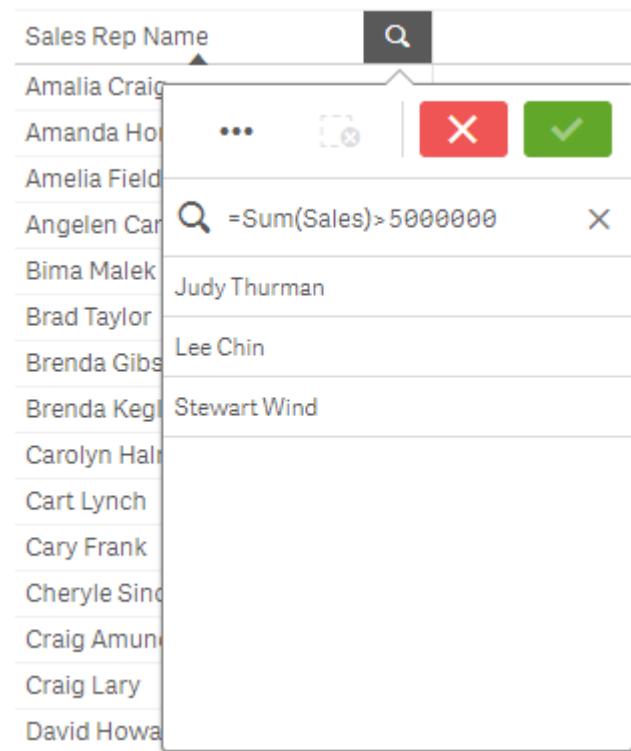
Expression search



Example:

Let us assume that you have a filter pane for sales representatives. You can then use an expression search for the sales representatives who have sales larger than, for example, 5,000,000. The search string is similar to the previous one: " $=\text{Sum}(\text{Sales}) > 5000000$ ". Because the sales values are associated with the sales representatives, you can perform the search in the Sales Rep filter pane.

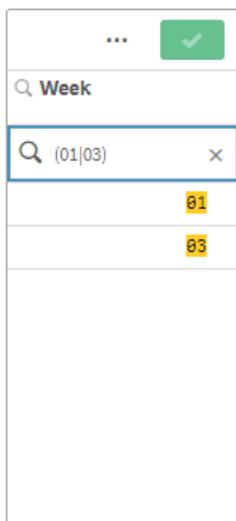
Expression search in a filter pane using a search string



Compound search

With a compound search, you can combine two or more searches with search operators. This allows customized and specific filtering of data. A compound search is triggered by enclosing the search within a set of parentheses. You can use multiple search operators in a single compound search.

Compound search in Qlik Sense



In its most basic form, a compound search can contain a single value. Unlike other search methods, a compound search for a single term will only return values that are an exact match with the search term. This can help you create more specific searches of your data.

Compound search using a single search term

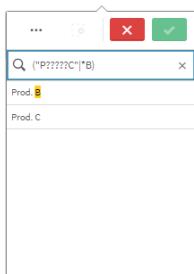
Sales per Region		Sales	# of Invoices
Region	...	674.81	38314
	X	936.92	2915
	...	615.03	7884
	X	965.91	6613
Spain	...	601.72	1957
	X	182.23	8230
	...	691373	10715

5 Exploring with selections

A compound search can contain one or more wildcards. If search content contains spaces, use quotation marks to enclose the value as it appears within the compound search.

For more information about quotation marks and wildcards, see *Text search (page 37)*.

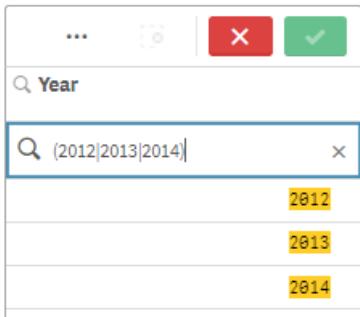
Compound search using multiple wildcards and quotation marks



OR search operator ("|")

Using the OR operator, the compound search will return values that match any of the included searches.

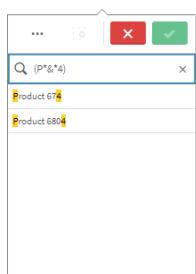
Compound search using the OR search operator



AND search operator ("&")

Using the AND operator, the compound search will return values that match all of the search items included in the statement. Since compound searches return exact matches only, this operator is typically only used with one or more wildcards.

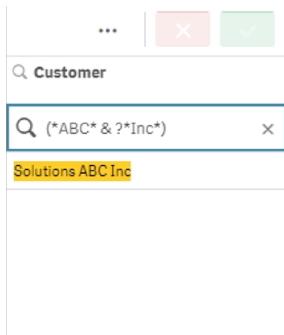
Compound search using the AND search operator



This additional example uses the AND operator and multiple wildcards in a compound search.

5 Exploring with selections

Additional example of compound search using the AND search operator

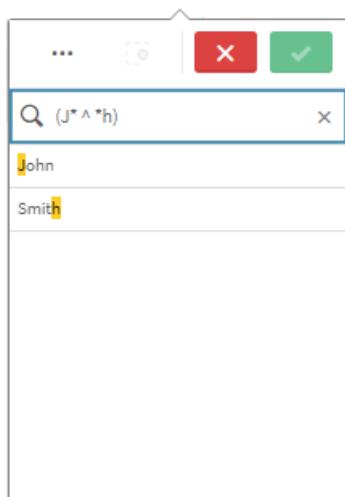


XOR search operator ("^")

Using the XOR operator, the compound search will return values that match either the first or second search, but not both.

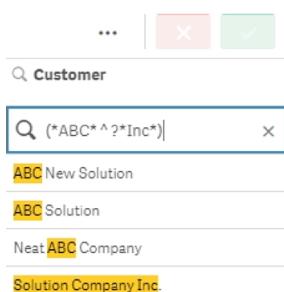
In the example below, the search returns values that either start with the letter "j" or end with the letter "h", but values that match both these criteria (such as "Josh") are not included.

Compound search using the XOR search operator



This example uses the XOR operator and multiple wildcards.

Additional example of compound search using the XOR search operator



5.3 Editing the selections

You can change selections during data analysis. You change the selections in the selections bar.

Do the following:

1. Switch to sheet view.
2. In the selections bar above the sheet, click the selection that you want to edit.

A popup window with the selection appears.

3. In the popup window, select the values that you want to add or clear.

You can search and filter your selections using the special characters, operators, wildcards, and methods described in *Searching within selections or visualizations (page 34)*.

4. Confirm your selection.

The selection is updated.

5.4 Locking and unlocking selections

With the lock option, you can protect your selections.

Locking selections

You can lock a selection by clicking the lock icon  in the selection popup. The lock prevents any changes from being made to that selection. You can neither change nor clear a locked selection. If you have locked a selection and then try to select excluded field values, the selection item will flash to indicate that the locked selection prevents the selection from being made.

Locked values Germany and Japan within the selection Region.

The screenshot shows the Qlik Sense selection history for the 'Region' dimension. At the top, it says 'Region 2 of 6'. Below that is a button labeled 'Click to unlock'. A search bar is present. The list of regions is as follows:

Region	Status
Germany	Selected (Green)
Japan	Selected (Green)
Nordic	Not Selected (Grey)
Spain	Not Selected (Grey)
UK	Not Selected (Grey)
USA	Not Selected (Grey)



It is possible to step back in the selection history to a state before the dimension was locked.

Unlocking selections

You can unlock a selection by clicking the lock icon in the selection popup. When you have unlocked the selection, you can make changes to it, or clear it.



When an app is saved using Qlik Sense, the current selections and locks are not saved. Selections and locks must be made every time the app is opened.

5.5 Stepping back and forward in selections

When you make selections, these are saved as items in the selections bar above the sheet.

Selections bar with the options to step back and forward in the selections history and to clear all selections.

To the left in the selections bar, there are three options, one for stepping back in the selections history, one for stepping forward, and one for clearing all selections. In the screen shot you can see that the option to step back is available, but not the forward option. This is the normal case when you have not stepped back in the selection history.

Clicking brings you one step back in the selection history. You can move back all the way to the first selection in the session. Even if a selection has been locked, you can move back to a state before the selection was made. A locked selection has a before the dimension name. In the screen shot, the dimension *Region* is locked.

Clicking brings you one step forward in the selection history.

Clicking clears all selections, except the ones that are locked.

5.6 Using the selections tool

The selections tool gives an overview of every dimension and field in an app. It also gives a more detailed view of selected data, so that you can explore associations in dimensions that have not been used.

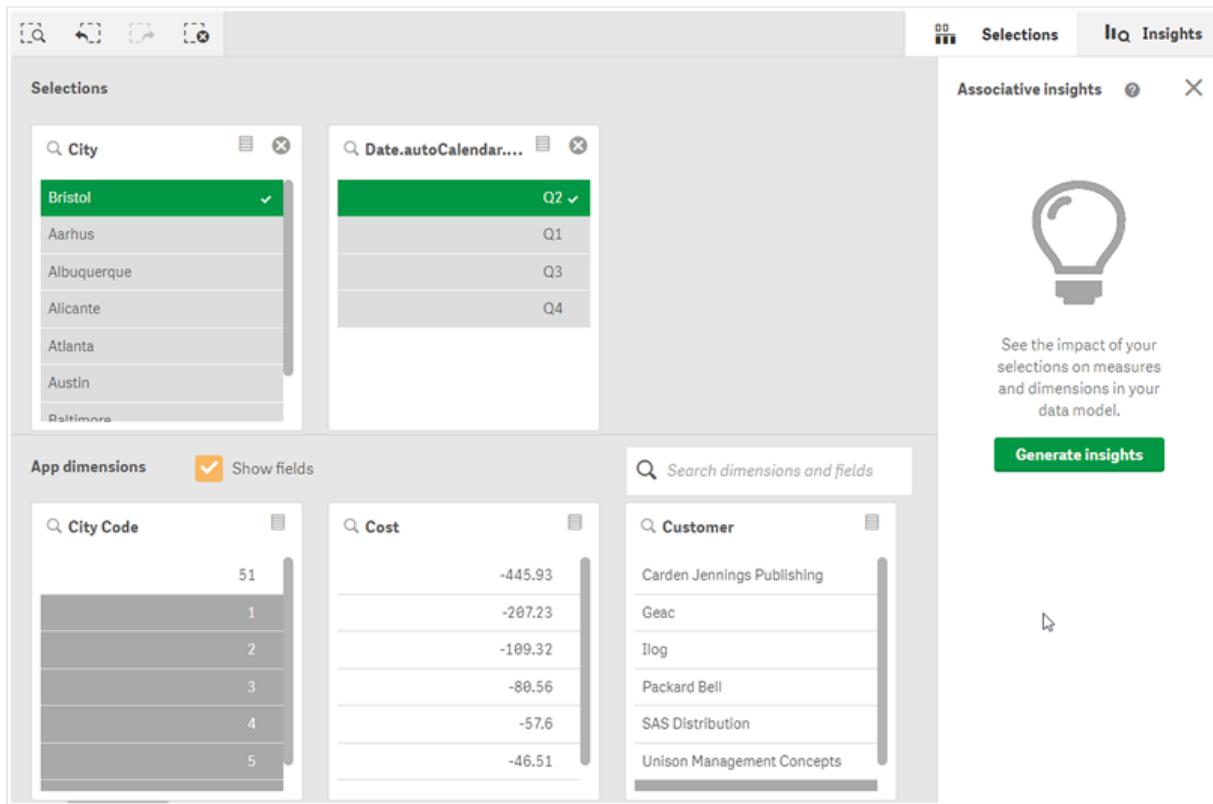
During analysis, click **Selections** to open selections view.

The selections tool is divided into two sections: **Selections** and **App dimensions**. The selections section displays the fields that have active selections. The app dimensions section displays all dimensions without an active selection. Both sections are sorted alphabetically. By default, app dimensions only displays master items. Select **Show fields** to show all the fields that have been loaded into the app, but are not used as dimensions.



Calculated dimensions are not shown in the selections tool.

Selections tool displaying the two sections Selections and App dimensions.



Making and clearing selections

You can make several selections consecutively. Click ✓ or click outside the list, but within the selections tool area, to confirm your selections. These dimensions will move up to **Selections**. Click the toolbar to close the selections tool.

In **Selections**, you can clear a selection in a field by clicking ✕. The field is then moved down to **App dimensions**.

When you are in the selections tool, you can still use the options in the selections bar: step back, step forward, and clear all selections. In each dimension you have the usual list options: selection menu, clear selection, cancel selection, confirm selection, and search.

Searching in the App dimensions section

App dimensions has a search box that is useful when you have many dimensions and fields. You can search by field or dimension title. Your search string can consist of one or more words, or only a part of word. The search is not case sensitive, but only exact string matches are displayed. A search for "numbers" will not display fields with the string "number", but a search for "mbe" will.

Scrolling in the selections tool

You can drag the scroll bar to scroll horizontally through a section. When you scroll in one section the other section is unaffected.

The dimension lists also have scroll bars, for vertical scrolling.

Generating insights

Associative insights reveals relationships in your data. The Qlik cognitive engine searches your data model for excluded values in a subset of fields. It then highlights those values, so you can explore them.

Once you have made a selection, click **Generate insights**. Cards are shown in the right hand panel. The cards show the effect of your selection on dimensions and measures in your data.

5.7 Discovering your data with associative insights

Associative insights reveals relationships in your data. The Qlik cognitive engine searches your data model for excluded values in a subset of fields. It then highlights those values, so you can explore them.

You can compare the contributions of your selections and excluded values against your measures. In any data set, particularly in complex ones, this can help you uncover blind spots and reveal relationships that you may have missed.

For a visual demo and practical example about discovering your data with associative insights, see the following:

- [Discovering your data with associative insights](#)
- [Qlik associative insights - A simple yet practical example](#)

Limitations

App consumers with the appropriate access rights for an app can use associative insights. They must connect to a Qlik Sense Enterprise or Qlik Core server. They cannot:

- Use the **Alternate states** feature.
- Set **Always one selected value** for a field in an app
- Lock their selections..

Associative insights selections view

When generating insights, Qlik Sense looks at your selections and analyzes the excluded values in your data model. It then highlights data that may be of interest for further exploration. That data is displayed in cards, which can be clicked to provide a more detailed view.

Associative insights selections view.

5 Exploring with selections

The screenshot shows the Qlik Sense interface with the 'Analyze Sheet' tab selected. In the top left, there's a 'Selections' card for 'City' with 'Aarhus' highlighted. Below it, the 'App dimensions' section lists various fields like XKEY, City Code, Cost, Customer, Customer Number, Date, and Date.autoCalendar.C. To the right, an 'Associative insights' panel displays insights for 'City' (Aarhus), 'Sales Rep Name', and 'Desc'. A search bar at the top right says 'Search dimensions and fields'.

Selections

Your currently applied selection or selections. The top list box shows data associated with your current selection. The other list boxes show data excluded from your current selection.

Selections card in associative insights.

This is a close-up view of the 'Selections' card. It shows a list of cities under the 'City' dimension. 'Aarhus' is highlighted with a green background, while the other cities (Albuquerque, Alicante, Atlanta, Austin, Baltimore, Barcelona, Bergen, Berlin, Birmingham) are shown in grey.

App dimensions

The selections that are available to you. By default, only master items are displayed. Select **Show fields** to show all fields.

5 Exploring with selections

App dimensions in associative insights.

App dimensions		Show fields	Search dimensions and fields		
🔍 %KEY	86	🔍 City Code	78	🔍 Cost	-188.09
96	1	2	-41.29	3	-38.07
120	3	4	-8.79	Molson Breweries	-6.45
913	5	6	-6.26	Razorfish	-6.26
1796	7	8	-0.84	Saturnus Multi produkten	-0.84
1964	A Superior System	0.01	Unilys	Zitel	Bitstream
1966					Edify
2214					Home Automation (HAI)
2856					Molson Breweries

Measure

The measure that Qlik Sense has selected for insights. You can change the measure in the drop-down menu.

App dimensions in associative insights.

App dimensions		Show fields	Search dimensions and fields		
🔍 %KEY	86	🔍 City Code	78	🔍 Cost	-188.09
96	1	2	-41.29	3	-38.07
120	3	4	-8.79	Molson Breweries	-6.45
913	5	6	-6.26	Razorfish	-6.26
1796	7	8	-0.84	Saturnus Multi produkten	-0.84
1964	A Superior System	0.01	Unilys	Zitel	Bitstream
1966					Edify
2214					Home Automation (HAI)
2856					Molson Breweries

Included values

This card represents included values.

Values included in associative insights.



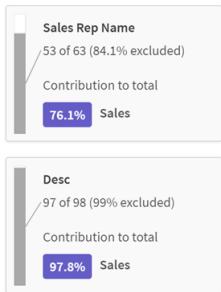
Insights from excluded values

Insight cards show the impact of the excluded value on your measure. In this case, it shows that one employee (*EmployeeName*), who is not in Canada (CAN), contributed 37.8% of yearly sales (*YearlySales*). Purple indicates an insight. Click on a card to reveal the insights details view.

5 Exploring with selections

Insights from excluded values in associative insights.

Insights from excluded values:



Add a dimension drop-down

You can use this drop-down to add cards to the **Insights from excluded values** section.

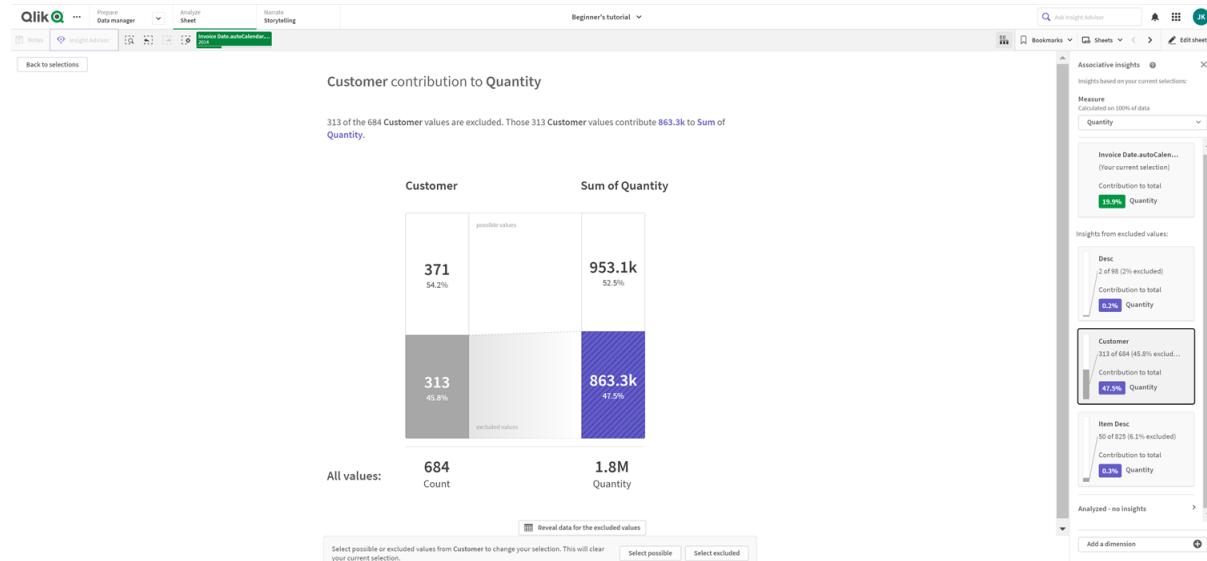
Add dimensions drop-down.

[Add a dimension](#)

Associative insights card details view

Details view will display a detailed chart if you have a simple sum() measure.

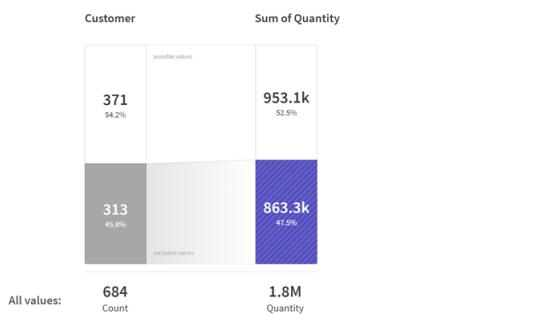
Associative insights card details view.



Details view

A detailed view of the currently selected card. Click the chart to see the data in a table. Purple indicates an insight.

Details view for currently selected card.



Measure and included values

You can use the drop-down to change the current measure. The card beneath it represents included values.

Measure selection and included values.

Associative insights ? X

Insights based on your current selections:

Measure
Calculated on 100% of data

Quantity

Invoice Date.autoCalen...
(Your current selection)

Contribution to total
19.9% Quantity

Insights from excluded values

These cards represent excluded values. You can click them to change the chart in details view.

Add dimensions drop-down.

Associative insights ? X

Insights based on your current selections:

Measure
Calculated on 100% of data

Quantity

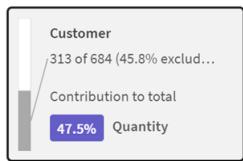
Invoice Date.autoCalen...
(Your current selection)

Contribution to total
19.9% Quantity

Currently selected dimension card

The dimension you have currently selected will have a dark border around it.

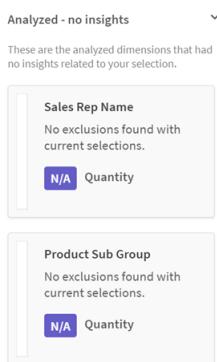
The currently selected card.



Analyzed - no insights

These values have been analyzed, but have no interesting insights.

List of cards with no available insights.



Add a dimension drop-down

You can use this drop-down to add cards to the **Insights from excluded values** section.

Drop-down menu for adding dimension.



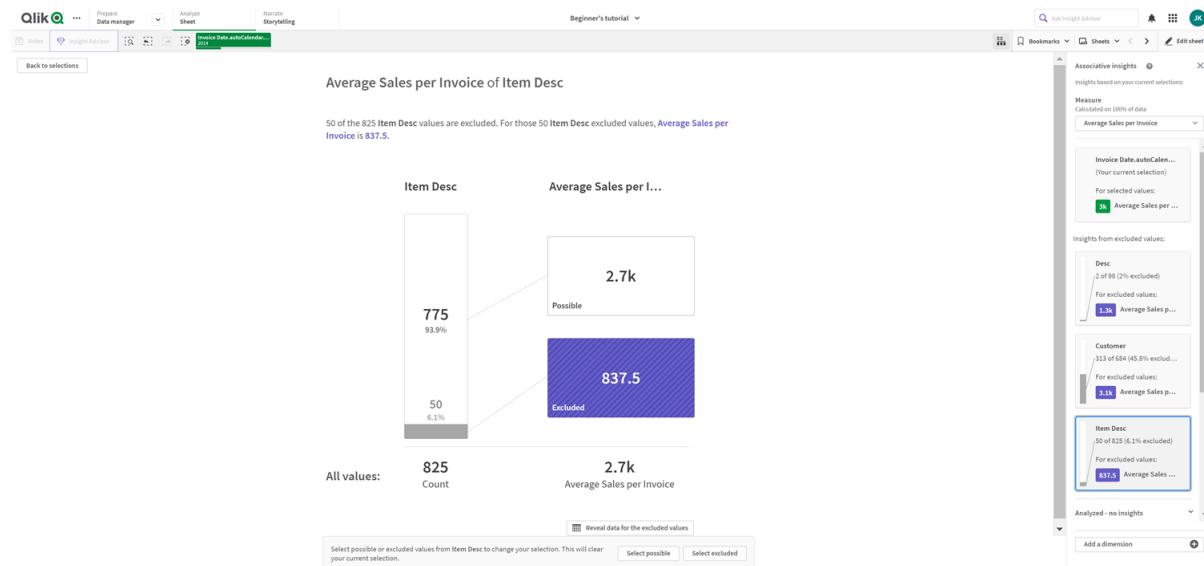
Insights card KPI view

Details view will display a KPI chart under two conditions:

- Measure aggregation is other than sum().
- Measure aggregation is sum(), but the data model is not appropriate for standard stacked charts.

5 Exploring with selections

Associative insights card details KPI layout.



Generating insights

Do the following:

1. In a sheet in your app, click **Selections**. The **Selections** window opens. The window contains the **Associative insights** panel.
2. Select a value in **App dimensions**.
By default, only master items are displayed. Select **Show fields** to show all fields. You can then select a value from the fields that are shown.
3. Click **Generate insights**.
4. Cards are shown in the right hand panel. The cards show the effect of your selection on dimensions and measures in your data.
5. You can change which measure is assessed. Choose a new measure from the **Measure** drop-down menu.
6. Click a card to see more information about your associative insight.

You can make or change your selections in the **Associative insights** window at any time. Make your selections and then click **Refresh**. The cards will be updated based on your new selections.

Associative insights example: No data left behind

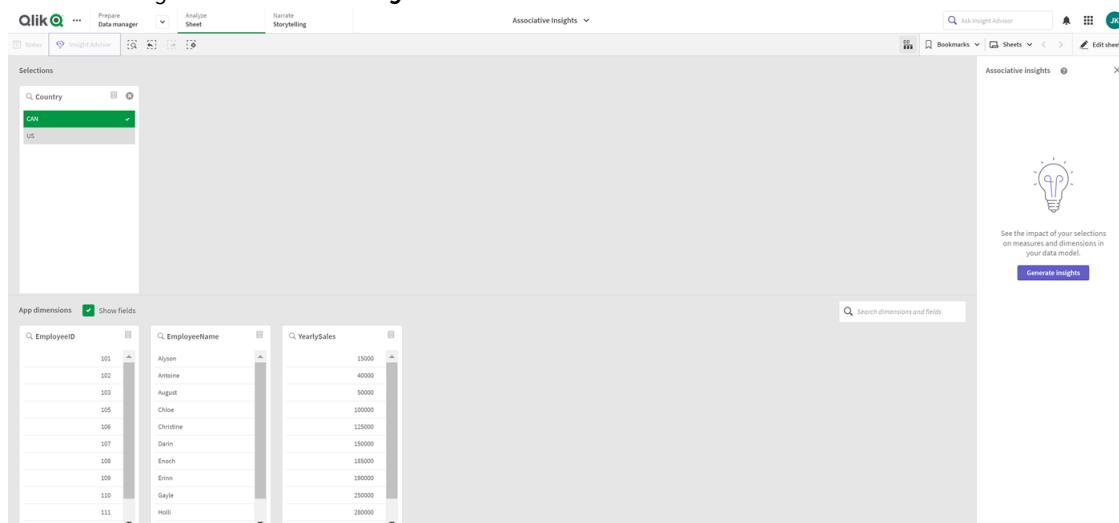
In this example, we use a simple data set to uncover an insight.

Do the following:

1. Here is a view of our app with the **Selections** window open. We have selected *CAN* in the *Country* field. *US* is colored light gray because it is an alternate selection. The values in the *EmployeeID*, *EmployeeName*, and *YearlySales* fields are white because they are possible values. In other words, they are associated with our selection of *CAN*.

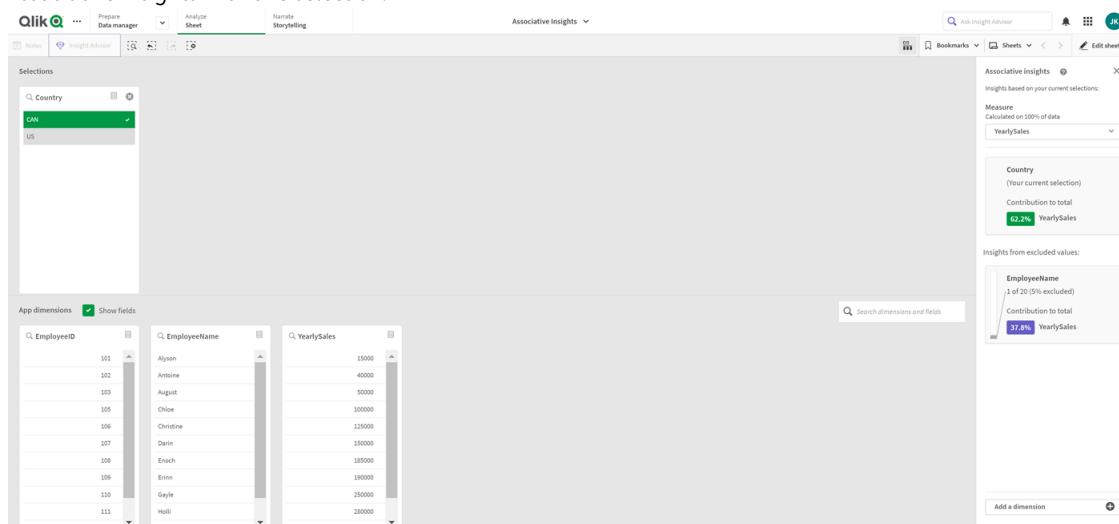
5 Exploring with selections

Associative insights with **Generate insights** button.



- When we click **Generate insights**, Qlik Sense selects interesting data to show in the associative insights cards. We see that CAN contributes 62.2% to our yearly sales. We also see in the bottom card that one of our employees (or 5% of all *EmployeeName*) is excluded from this selection. This employee contributes 37.8% to our yearly sales. Purple indicates an insight.

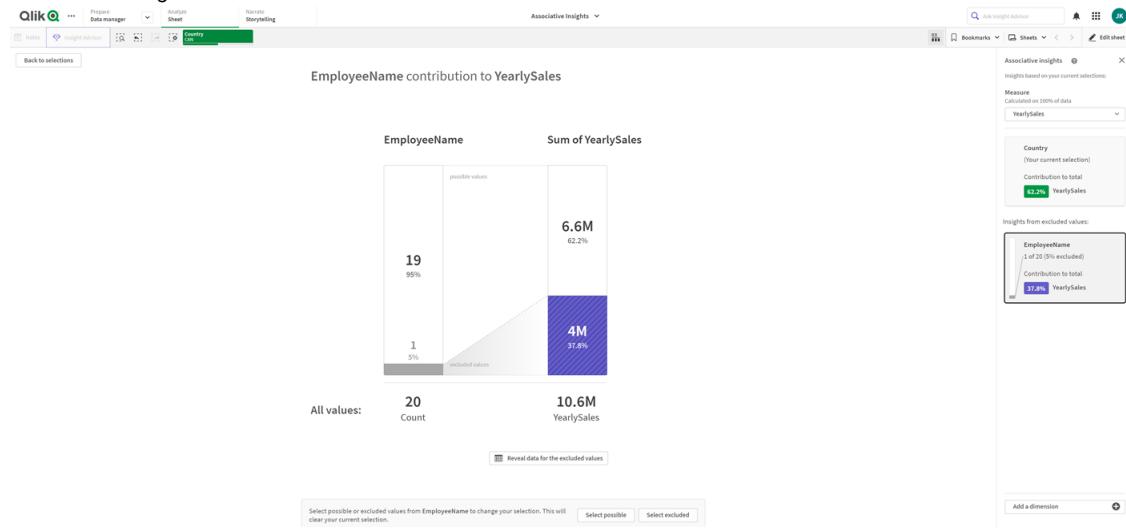
Associative insights with one selection.



- When we click on card, Qlik Sense shows us a detailed view of the data. It shows how much this excluded employee contributed to yearly sales. Who is this employee? We know they are excluded from our selection of CAN. Click **Reveal data for the excluded values**.

5 Exploring with selections

Associative insights with detailed view.



4. We now see a table showing information about *Kasie*, the excluded value from the *EmployeeName* field. Click **Select excluded** to apply *Kasie* as a selection.

Associative insights with detailed view of excluded values.

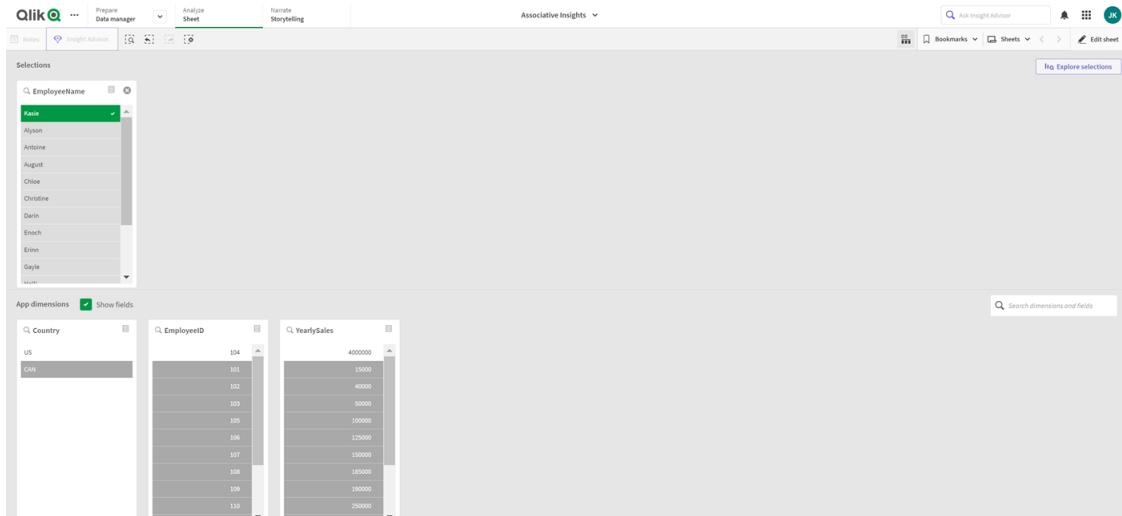


5. We are brought back to the **Selections** window. *Kasie* is selected. Values in our former selection of *CAN* are colored dark gray, because they are now excluded.

Click **Explore selections**, and then **Generate insights**.

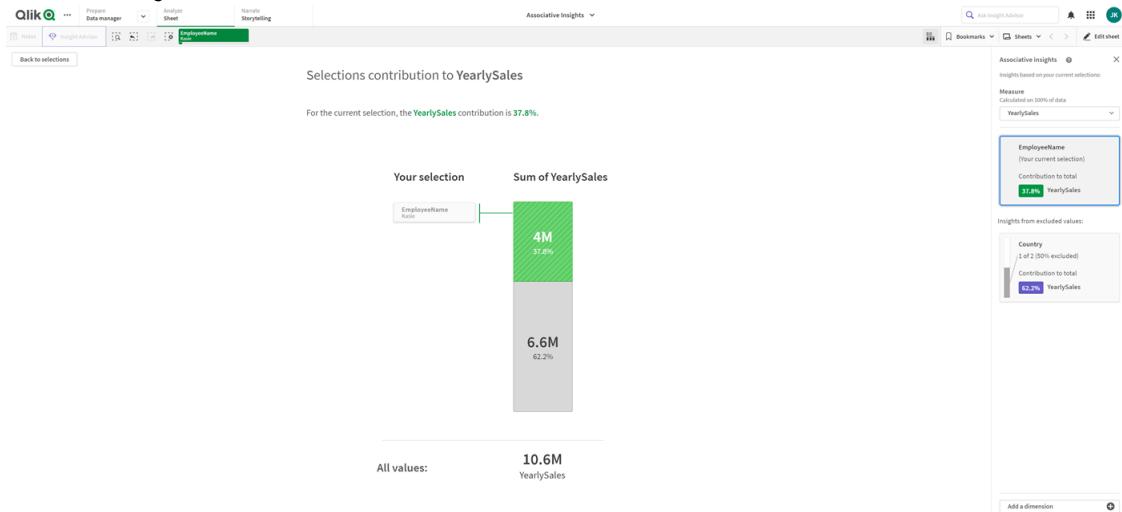
5 Exploring with selections

Associative insights with previously excluded values included.



6. Click the *EmployeeName* card, and you will see an updated insight card details view.

Associative insights with Kasie as selected value.



Disabling associative insights in an app

Associative insights can be disabled in an app by adding a variable and then refreshing the app.

Do the following:

1. In the sheet edit mode, in the assets panel, click .
2. Click **Create new**.
3. In **Name**, type *DISABLE_SELECTION_INSIGHTS*.
4. In **Definition**, type any value.
5. Click **Close**.
6. Refresh the app.

Troubleshooting associative insights

I cannot see associative insights

Possible cause

A script variable is disabling this feature.

Proposed action

Contact your administrator.

Insights are not available

Possible cause

Qlik Sense is offline or not connected to a Qlik Sense server.

Proposed action

Verify that you are online. Log into your Qlik Sense server and try again. If this does not solve the problem, there might be an issue with engine.

I am missing fields under **My dimensions**

Possible cause

You can only see master dimensions, because **Show fields** is not selected

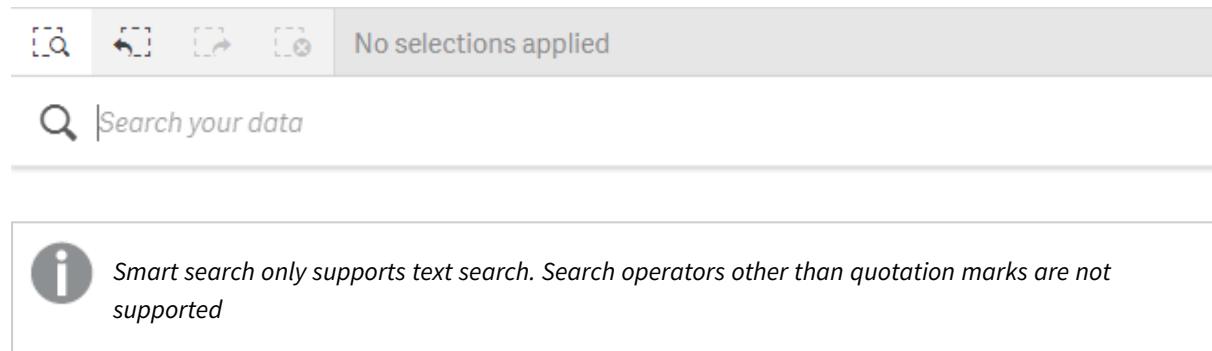
Proposed action

On the **Selections** screen, select **Show fields**.

6 Using smart search

Smart search is the global search tool in Qlik Sense, enabling you to search the entire data set in your app from any sheet in the app. Smart search is available from the selections bar in a sheet by clicking .

Smart search field where you can search the entire data set in your app from any sheet.



The screenshot shows the Qlik Sense interface with the 'Smart search' feature. At the top, there's a toolbar with icons for search, refresh, and other functions, followed by a message 'No selections applied'. Below this is a search bar with a magnifying glass icon and the placeholder text 'Search your data'. A tooltip box with an 'i' icon provides information: 'Smart search only supports text search. Search operators other than quotation marks are not supported'.

If you click on a result under **Apply a selection**, the results disappear and the selection is applied.

Smart search is available when you are analyzing data on a sheet. The data results help you to find associations and make selections in your data.

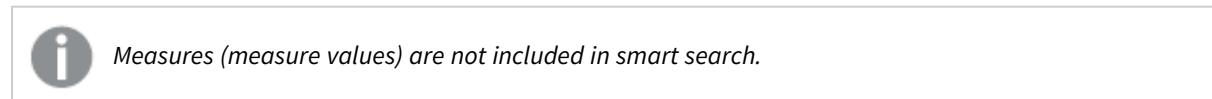
You can search for data in visualizations in your sheets with Insight Advisor, which you can access by clicking **Insight Advisor** in the sheet view. Insight Advisor can also generate new visualizations based on your data searches.

You can also search within selections and visualizations such as tables and filter panes.

6.1 What happens when you search

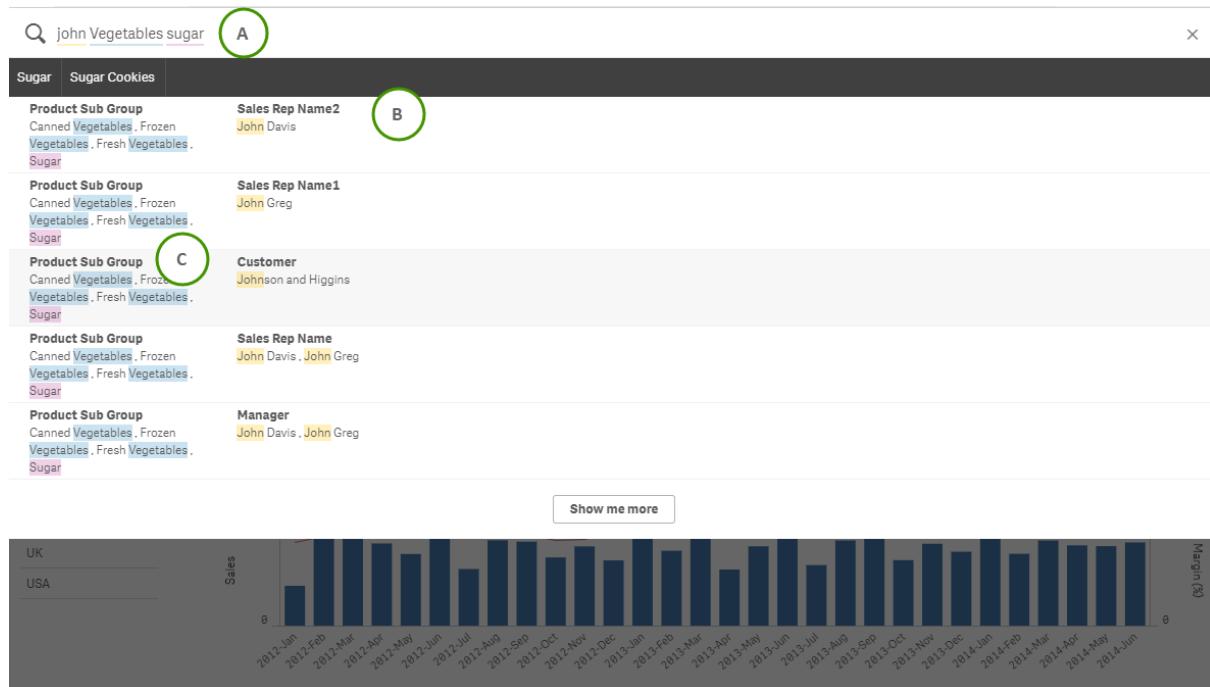
As you type your search query, Qlik Sense searches data items. Smart search filters the field values and displays the matching items. Smart search looks for:

- Field values
- Dimension values (also dimension values that are created as master items)



A tooltip box with an 'i' icon states: 'Measures (measure values) are not included in smart search.'

The results of searching for 'John Vegetables sugar', which generates one search query for each term.



A: Search field

Separate search terms with spaces. To link words into one search term, use quotation marks, for example, "mountain bike". Suggested search strings are displayed beneath the search field

B: Apply a selection

The search results from the app data, shown with one result per row.

Click **Show me more** to see more results.

C: Color-coded search result terms

A color code is assigned to each search term found. It shows partial matches as well as full matches.

You can clear the search field by clicking the cancel icon to the right in the search field. Click the search icon to close smart search.

The search terms are always compared against the beginning of the words in the database. Searching for "read" does not present "bread" as a match, whereas "reader" and "Reading" would both be matches. The search terms are each given a color to support the identification of the matches. When there are more than six search terms, the colors are reused.

6.2 Using search results to change selections

Interacting with search results for data

When you search data, the results show the combinations of matches found in the entire Qlik Sense database. The results are based on field associations. They are sorted by the number of matched search terms, in descending order. If there is more than one match, the results are ranked by relevance. Click a match to insert it into the search field.

When you select a result, you make an actual selection of the values, and your current selections and the visualizations containing the selected data are updated.

The screenshot illustrates the categories that are available in the database: *john*, *Vegetables*, and *sugar*.

The screenshot shows the Qlik Sense search interface with the following details:

- Search Bar:** Shows the query `john vegetables sugar`.
- Selection Bar:** Displays the current selections: `Sugar` and `Sugar Cookies`.
- Search Results:** The results are grouped by dimension and field name. Each group has a "Show me more" button at the bottom.
 - Product Sub Group:** Sales Rep Name2
 - Canned Vegetables. (highlighted in blue)
 - Frozen Vegetables.
 - Fresh Vegetables.
 - Sugar (highlighted in pink)
 - Product Sub Group:** Sales Rep Name1
 - Canned Vegetables. (highlighted in blue)
 - Frozen Vegetables.
 - Fresh Vegetables.
 - Sugar (highlighted in pink)
 - Product Sub Group:** Customer
 - Canned Vegetables. (highlighted in blue)
 - Frozen Vegetables.
 - Fresh Vegetables.
 - Sugar (highlighted in pink)
 - Product Sub Group:** Sales Rep Name
 - Canned Vegetables. (highlighted in blue)
 - Frozen Vegetables.
 - Fresh Vegetables.
 - Sugar (highlighted in pink)
 - Product Sub Group:** Manager
 - Canned Vegetables. (highlighted in blue)
 - Frozen Vegetables.
 - Fresh Vegetables.
 - Sugar (highlighted in pink)
- Show me more:** A button located at the bottom of each group of results.



If you select a dimension value in the search result, the field name (not the dimension name) will be displayed in the selections bar.

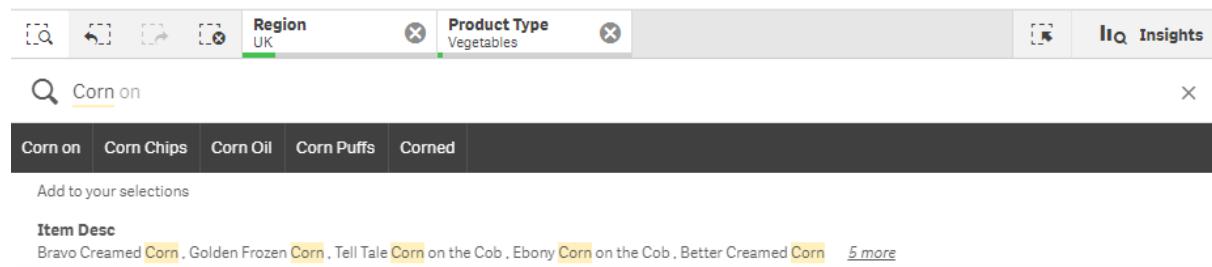
You can clear the search field by clicking the cancel icon  to the right in the search field. Click the search icon  to close smart search.

The search terms are always compared to the beginnings of the words in the database. Searching for “read” does not present “bread” as a match, whereas “reader” and “Reading” would both match. The search terms are colored to show the matches. When there are more than six search terms, the colors are reused.

Using smart search to change the current selection

When you use smart search to search the data in an app, and you make a selection from the results, you can search within that selection. Then you can click on search results to change your current selections. Smart search searches within your selections automatically, all you need to do is add search strings and perform a new search. You can do this over and over to filter your search results.

Using smart search to search within the selections Region and Product type.



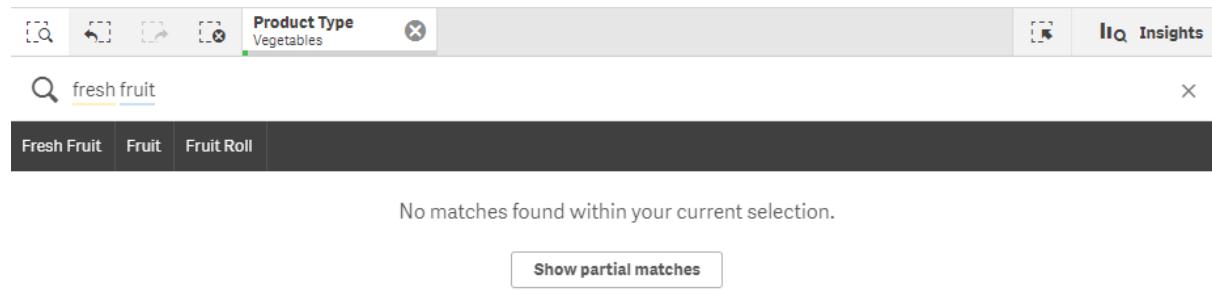
The screenshot shows the Qlik Sense interface with two selected dimensions at the top: "Region UK" and "Product Type Vegetables". A search bar contains the query "Corn on". Below the search bar, a list of search results is displayed, including "Corn on", "Corn Chips", "Corn Oil", "Corn Puffs", and "Corned". A tooltip "Add to your selections" appears over the "Corn on" result. Below the results, a section titled "Item Desc" lists items like "Bravo Creamed Corn", "Golden Frozen Corn", "Tell Tale Corn on the Cob", "Ebony Corn on the Cob", and "Better Creamed Corn", with a link to "5 more".

When you search within your current selection, smart search will find results associated with your current selection. If the terms you are searching for are not associated with the current selection, you will see an option to clear the current selection. If you clear the selection you will see the result for the searched terms without having to re-enter them.

If you search within a selection, and your search terms are excluded because of the selections (dark gray), there will be no results.

If you search within a selection using multiple terms and your query does not produce a result for all of your search terms, you can view partial matches by clicking **Show partial matches**.

Searching for the multiple terms 'fresh fruit' did not produce a result for all search terms, making the option 'Show partial matches' available.



The screenshot shows the Qlik Sense interface with a selected dimension "Product Type Vegetables". A search bar contains the query "fresh fruit". Below the search bar, a list of search results is displayed, including "Fresh Fruit", "Fruit", and "Fruit Roll". A message "No matches found within your current selection." is shown. A button labeled "Show partial matches" is visible.

If you search within a selection and your query does not produce a result, you will get the message “**No matches found within your current selection.**”. If the selection is locked, you can consider to unlock the selection and perform a new search.

No result searching for 'sugar' in the selection Product Type.



6.3 Keyboard shortcuts used in smart search



This description of keyboard shortcuts assumes you are working in Windows. For macOS use Cmd instead of Ctrl.

Keyboard shortcuts used in smart search

Keyboard navigation	Description
Ctrl+F	Open smart search. You can then enter the values or characters you want to search for.
Tab or Enter	Add the first suggested search string to the search field, if none is highlighted.
Down/up arrow keys	Move between the search field, suggested search string list, and search results. Move between rows in the search results.
Right/left arrow keys	Move between the entries in the suggested search string list.
Tab	Add the highlighted entry from the suggested search string list to the search field.
Enter	Make a selection of the highlighted search result. Make a selection from the highlighted suggested search string list.
Esc	Clear the search field. Close smart search (if the search field is empty).
Ctrl+F	Close smart search.

7 Troubleshooting - Discover

This section describes problems that can occur when discovering and analyzing in Qlik Sense.

7.1 My search does not produce any results

Possible cause

You have locked selections.

Proposed action

Unlock the selections and then perform a new search.

Do the following:

1. Click on the selection with .
2. Click  to unlock.
3. Perform a new search.

7.2 My search using Insight Advisor does not produce any results

Possible cause

You are searching for a field which is not a master item.

Proposed action

Search for a master item instead. Additionally you can use smart search to find fields which are not master items.

7.3 Incomplete visualization

A visualization is not displayed, instead an error message **Incomplete visualization** is displayed.

Possible cause

The visualization contains data fields that you do not have access to.

Proposed action

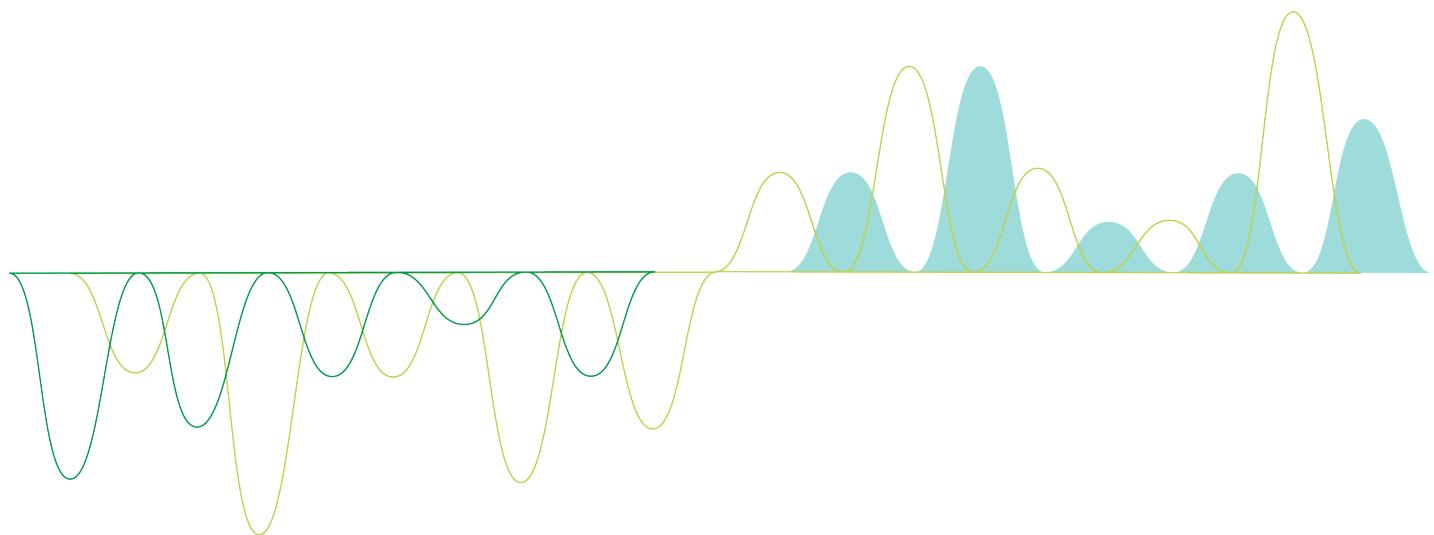
Contact your Qlik Sense administrator to see if you can get access to the omitted data fields, to be able to use the visualization.

Collaborate in Qlik Sense

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 About this document	5
2 Sharing insights with data storytelling	6
2.1 The story	6
2.2 Snapshots	6
2.3 Live data sheets	7
2.4 Collecting insights for stories using snapshots	7
Taking a snapshot	8
Viewing snapshots	8
Editing annotations	8
Deleting snapshots	9
2.5 Building stories	10
Working with stories	10
Working with slides	10
Working with stories	10
Working with slides	13
2.6 Making stories compelling	18
Adding emphasis	19
Styling with text and shapes	21
Adding bookmarks to a slide	24
Adding images to a slide	24
Adding links to a slide	25
Changing a snapshot's appearance	26
2.7 Presenting stories	28
Entering play	28
Showing data point information	28
Interacting with a live data sheet	28
Downloading stories	29
Closing play	29
Accessing the live data of a snapshot	29
Accessing a sheet from a slide	30
Selecting in a live data sheet	30
2.8 Troubleshooting - Using data storytelling	31
I cannot edit a story	31
A sheet is missing	31
I cannot change a snapshot's appearance	31
I cannot take a snapshot	32
I cannot re-order slides in a story	32
The data in my snapshot does not exist in the visualization	32
3 Publishing	34
3.1 Distribute your insights	34
3.2 Publishing to Qlik Sense Enterprise streams	35
Streams	35
Sheets and stories	36
Interacting with apps	38
Publishing an app from the hub	39
Republishing an app from the hub	40
Managing app properties	40

Contents

Publishing a sheet	41
Unpublishing a sheet	41
Adding sheets to the public sheets of an app	42
Removing sheets from the public sheets of an app	42
Publishing a story	42
Unpublishing a story	43
Adding stories to the public stories of an app	43
Removing stories from the public stories of an app	44
Publishing bookmarks	44
Unpublishing bookmarks	45
Copying links to bookmarks in published apps	45
3.3 Publishing to Qlik Sense Enterprise streams for Qlik Sense Mobile Client Managed	45
3.4 Publishing from Qlik Sense Enterprise on Windows to other hubs	46
Staged apps	46
Tags	46
Streams and tags	46
Publishing apps to cloud hubs with tags	47
3.5 Publishing between Qlik Sense platforms	49
3.6 Qlik NPrinting reports in Qlik Sense	49
Limitations	49
Distributing Qlik NPrinting reports to Qlik Sense	49
Connecting to Qlik Sense apps in Qlik NPrinting	50
3.7 QlikView documents in Qlik Sense	51
Requirements	51
Publishing links to QlikView documents in the Qlik Sense hub	52
4 Downloading and printing	53
4.1 Downloading data from a visualization	53
Downloading data from a table	53
4.2 Downloading a sheet	54
About aspect ratio	54
Downloading sheets	55
Limitations	55
4.3 Downloading a visualization	55
Downloading visualizations from desktop devices	55
Downloading visualizations on mobile devices	56
Limitations	57
4.4 Downloading a story	57
Downloading as a PowerPoint presentation	58
Downloading as a PDF	59
4.5 Troubleshooting - Downloading	59
Anonymous users cannot download app data	59
I cannot download a visualization as an image	60
I have blank characters in PDF files	60
I downloaded a sheet but the data view tables changed back to visualizations	61

1 About this document

This guide will introduce you to working with data storytelling, publishing, and exporting and printing, to make content available to other users.

Qlik Sense is developed with collaboration in mind and provides tools to help create a common understanding to support decisions and influence others.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

You find the online help, additional guides and much more at help.qlik.com/sense.

2 Sharing insights with data storytelling

Data storytelling provides a way to share your data insights, whether they belong to a larger discussion or the main topic.

The purpose of data storytelling is to turn data discoveries into a story. Emphasizing important elements helps create convincing stories and supports stakeholders in decision-making.

Data storytelling lets you combine reporting, presentation, and exploratory analysis techniques to create and collaborate. You take snapshots of your discovered data to use in stories composed of slides. Snapshots can be enhanced with effects. This lets you emphasize the data insights you want your audience to focus on.

As you tell the story, you can answer questions by switching to a snapshot's source and accessing live data. This guides your story to new directions, triggering new conversations and deeper insights.

Storytelling becomes interactive by inserting live data sheets in slides, and making selections while presenting.

For a visual demo about storytelling, see [Sharing insights with data storytelling](#).

2.1 The story

In data storytelling, you use a story to collect and present insights and ideas to your audience. A story is presented as a timeline of slides. It can be based on traditional data storytelling structures, such as a three-act play or a hero's journey.

Stories are contained within an app. A story is connected to its app, so you can return to the live data anytime, to discover new and hidden stories.

To build a story you use time-based snapshots of your data visualizations and live data sheets and place them on the story's timeline.

You can, for instance add text and shapes, put emphasis on certain insights with visual effects, apply styling, and so on, to make the story compelling and engaging, and its purpose very clear.



You can download the story if you want to present it outside of Qlik Sense.

2.2 Snapshots

A snapshot is a graphical representation of the state (type and data) of a data object at a certain point in time that you can use when you build stories. The snapshot you take is a copy of the state. This means that the state of the snapshot does not change when the state of the corresponding data object gets updated.

Snapshots capture individual objects on a sheet during the analysis process. They store the visualization and data as you see it at that time enabling you to use them at a later point in time to tell a story. Each snapshot contains a bookmark back to the original context so that you quickly get access to the live data.

2 Sharing insights with data storytelling

When you take a snapshot you can make an annotation for your snapshot. The annotation helps you distinguish between the different snapshots in the snapshot library when you build your story. The annotation is not visible when you play the story.



A snapshot's state and selections will not be updated at a data reload. It will always reflect the data that existed at the point in time the snapshot was taken.

2.3 Live data sheets

Using live data sheets in stories you can make sheet selections while playing the story. This way, you can show your insights without going to the app itself.

Playing a story, you can make and reset selections in the live data sheets. It works the same way as in any sheet, in sheet view.



The reset gives you the possibility to always return to the same selections as when you inserted the live data sheet on the slide.

- *Collecting insights for stories using snapshots (page 7) --> Working with stories (page 10)*
- *Working with stories (page 10) --> Creating a slide (page 14)*
- *Creating a slide (page 14) --> Editing slides (page 15)*
- *Editing slides (page 15) --> Organizing slides (page 18)*
- *Creating a slide (page 14) --> Adding emphasis (page 19)*
- *Adding emphasis (page 19) --> Styling with text and shapes (page 21)*
- *Styling with text and shapes (page 21) --> Adding images to a slide (page 24)*
- *Adding emphasis (page 19) --> Presenting stories (page 28)*

2.4 Collecting insights for stories using snapshots

To get your app's full picture, you need a way to collect the insights you encounter. To do this, you take static snapshots of your visualizations.

A snapshot is a graphical representation of the state (type and data) of a data object at a certain point in time that you can use when you build stories. The snapshot you take is a copy of the state. This means that the state of the snapshot does not change when the state of the corresponding data object gets updated.

When you take a snapshot you can make an annotation for your snapshot. The annotation helps you distinguish between the different snapshots in the snapshot library when you build your story. The annotation is not visible when you play the story.

After taking snapshots of a specific visualization, you can open the **Snapshot library** from the visualization's shortcut menu. From there you can decide which snapshots to keep and use in your stories. You can also edit the annotation and delete the snapshots.



When the snapshot library is opened from storytelling view, all the snapshots from all the visualizations in the app are displayed.

If the visualization you want to take a snapshot of is an extension, make sure to enable the snapshot functionality in its main script. To do that, set the snapshot property to true.

Taking a snapshot

When you are in sheet view, you can take snapshots of the visualizations and use them to build a story in data storytelling.

You can take snapshots of all the different visualization types but not the filter panes.

Do the following:

1. In sheet view, go to the visualization.
2. Right-click on the visualization or click the hover menu .
3. Click **Take Snapshot**.
4. The snapshot is taken and saved to the snapshot library. It will have the same title as the visualization.

Viewing snapshots

When you are working on a story you can view all your snapshots from the snapshot library.

Do the following:

1. In storytelling view, click in the story tools panel.
The snapshot library is opened with your snapshots sorted by date with the most recent snapshot on top.
2. Click to close the snapshot library.

Editing annotations

When taking a snapshot you can choose to add an annotation. From the snapshot library, you can change the annotations or add annotations to snapshots without annotations.

Do the following:

1. In storytelling view, click in the story tools panel.
The snapshot library is opened with a list of all your snapshots.



*In sheet view, right-click the visualization you want to view snapshots for, click ••• and select **Snapshot library** in the shortcut menu.*

2. Click .
3. Click the annotation text area, to the right of the snapshot name.



You cannot edit annotations for snapshots that belong to a published version of an app.

4. Edit the annotation.
5. Click or outside the text area.
The annotation is saved.
6. Click outside the library to close it.

Deleting snapshots

You can delete a snapshot from the snapshot library, which contains all snapshots you have taken in the app.
You cannot delete snapshots that belong to a published version of an app.

Do the following:

1. In storytelling view, click in the story tools panel.
The snapshot library is opened with a list of all your snapshots.



*In sheet view, right-click the visualization you want to view snapshots for, click ••• and select **Snapshot library** in the shortcut menu.*

2. Click .
- Editing is enabled.
3. Select the snapshots you want to delete.



You cannot delete snapshots that belong to a published version of an app.

4. Click .
- The selected snapshots are deleted.
5. Click to stop editing.
6. Click outside the library to close it.

2.5 Building stories

The purpose of a story is to gather insight and build narratives from your data. You structure the story to convince your audience, by taking snapshots of your visualizations and put them in slides. To make your slides stand out you can apply visual effects, add text and shapes, and can embed sheets to make your story interactive. A story is connected to its app, so you can return to the live data anytime, to discover new and hidden stories.



Working with stories

The first step, if you have not done yet, is to create snapshots to populate your story.

Then you will create and configure a story, that you can then populate with your data.

[Building your story](#)

Working with slides

Slides create the structure of your story, and can contain data snapshots or other information such as images, titles, and live data sheets.

Working with stories

In this section you will find how to work with stories. For instance, you can understand how to create, duplicate and change settings for a story.

Creating new stories

Do the following:

1. Click  to view the stories, from the app overview or sheet view.
2. Click  or **Create new story**.
A new story is created, with the title **My new story**.
3. Give your story a meaningful title and add a description, if you want.
4. Click outside the text area to save the title and description.

The new story is saved.

Editing stories

When you load data into an app you analyze the data to get new insights. You can present your new insights in a new story or in a edited version of an existing story. .

A snapshot's state and selections will not be updated at a data reload. It will always reflect the data that existed at the point in time the snapshot was taken. However, live data sheets are not static and will be affected by a data reload.

You can replace a snapshot of a visualization on a slide by going to its source, in the app. The visualization will now have updated data. You can take a new snapshot to use in your story.

When you take a snapshot you can make an annotation for your snapshot. The annotation helps you distinguish between the different snapshots in the snapshot library when you build your story. The annotation is not visible when you play the story.



If you want to keep the original story, make a duplicate of it before you start editing.



You can download the story if you want to present it outside of Qlik Sense.

Changing the titles and descriptions of stories

You can change the title and description of your stories.

Note the following:

- To change the name or description of a story, the app must be in your personal cloud or group workspace.

Do the following:

1. In the app overview, click  **Stories**.
2. Do one of the following:
 - If you are in grid view, , click the story title followed by clicking .
 - If you are in list view, , click .
3. Edit **Title** and **Description**.
4. Click outside the text area.

The changes you made are saved.

2 Sharing insights with data storytelling



You can also change a story's title and description in the story navigator at the top right.

Changing the thumbnails of stories

You can replace the default thumbnail of a story with another thumbnail, to make it easier to distinguish between stories in the app overview and in the story navigator. You can use one of the default images, or an image of your own.

Do the following:

1. In the app overview, click **Stories**.
2. Do one of the following:
 - If you are in grid view, , click the story title followed by clicking .
 - If you are in list view, , click .
3. Click on the default thumbnail.
The **Media library** opens.
4. Click on a folder in the media library, for example **In app** or **Default**.
5. Select the image you want to use as a thumbnail for the story and click **Insert**.
6. Click to stop editing.

The image you selected is now used as a thumbnail for the story, and is visible in the story navigator and in the app overview.



You can also change a story's thumbnail in the story navigator at the top right.



The optimal aspect ratio of a thumbnail is 8:5 (width:height).

The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

For Qlik Sense Desktop: You can place images in the following folder on your computer:

C:\Users\<user>\Documents\Qlik\Sense\Content\Default. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.



You can only add or change the thumbnail of an unpublished story.

Duplicating stories

You can duplicate any story, regardless of whether it belongs to the app or if you created it yourself. The purpose of duplicating stories is to save time by reusing content, and to modify the duplicate to fit your needs.

A duplicated story contains the same content as the original story, and is linked to the same snapshots. The duplicated story will not be updated if the original story is updated. Duplicated stories appear under **My stories** in app overview and in the story navigator.

Duplicating a story from app overview

Do the following:

1. Click  on the left-hand side to show the stories of the app.
2. Right-click a story.
The shortcut menu opens.
3. Click **Duplicate**.

The new story is created. It is located under **My stories**.



You can also duplicate a story when you are in storytelling view, using the story navigator .

Deleting stories

Do the following:

1. Click  to view the stories, from the app overview.
2. Right-click the story you want to delete.
The shortcut menu opens.
3. Click **Delete**.
4. Click **Delete**, to confirm that you want to delete the story.

The story is deleted from the app.



You can also delete a story in the story navigator .

Working with slides

The purpose of slides in a story is to create a structure to your story.

In this section you will find how to work with slides, for instance by adding, reordering and deleting slides. You will find how to add snapshots and live data sheets to slides and how to reorder, resize and copy items on slides.

Learn about the following:

- Creating a slide
- Editing slides
- Organizing slides

Creating a slide

In this section you will find how to add slides to a story, and add data to your slides from multiple sources.

Adding a slide to a story

You can extend your story by adding one or more slides to it.

Do the following:

1. From the app overview or sheet view, click  **Stories**.
2. Click on the story you want to add the slide to.
The story opens in storytelling view with slide thumbnails visible in the story timeline.
3. Click  in the story timeline.

A new slide is added. You can now add contents to the slide, such as snapshots.

Adding a snapshot to a slide

You can add snapshots to a slide from your **Snapshot library**.

Do the following:

1. In storytelling view, click  in the story tools panel.
The **Snapshot library** is opened. The date stamp and any annotations helps you to distinguish between your snapshots.
2. Locate the snapshot you want to add and drag it onto the slide.
The snapshot snaps to the grid. You can use the keyboard arrows to move it freely.

The snapshot is placed on the slide.

Adding a live data sheet to a slide

You can add live data sheets to a story slide. When you insert a live data sheet to a story, the current selections of the app are saved with the selected sheet. Every time a live data sheet is viewed in a story, these selections will be applied.

Do the following:

1. In storytelling view, click  in the story tools panel.
A dialog opens.
2. Click the sheet you want to use in the story.

The live data sheet is added to a new slide.

Editing slides

In this section you will find how to edit and customize slides, by replacing, configuring and reordering slide elements.

Replacing a snapshot on a slide

You can replace a snapshot of a visualization on a slide. This is useful when you want to use another snapshot of the visualization, for example when new data is loaded into the app, if the visualization is changed or if new selections are made.

Do the following:

1. In storytelling view, select a snapshot on a slide, and then click  in the hover menu. The **Replace snapshot** dialog opens.



*You can navigate to the sheet and visualization which the snapshot originated from, by clicking **Go to source**. By doing this you get access to the live data of the snapshot, where you can make new selections and take new snapshots.*

2. Select the snapshot you want to use.

The snapshot is updated and adapts to the size of the previous snapshot, using free resize which causes the visualization to use progressive disclosure.



Progressive disclosure means the following: If the size of a visualization (or a unlocked snapshot) is increased, its information is disclosed progressively. If the size of a visualization (or a unlocked snapshot) is decreased, its information is reduced, which allows you to focus on the essential information and avoid cluttering the visualization with too much information in too little space.

The snapshot is replaced on the slide.

Replacing a live data sheet on a slide

You can replace a live data sheet on a story slide.

Do the following:

1. In storytelling view, select a slide with a live data sheet in the story timeline.
A dialog opens.
2. Click the active sheet.
A new dialog opens.
3. Select a new sheet to insert.

The selected sheet is added as a live data sheet on the slide.

Copying and moving items on story slides

You can copy and move items on the same story slide or between story slides. You can do this in different ways:

- Using the toolbar on the slide (and).
- With the keyboard shortcuts Ctrl+C, Ctrl+X and Ctrl+V.



*You can copy items between stories in the same app, but not between stories in different apps.
Switch between stories using in the toolbar.*

Copying items

Do the following:

1. In storytelling view, click on the item you want to copy.
The item is highlighted.
2. Click .
3. To insert the item on another slide, switch to that slide.
4. Click .

The copied item is placed in front of any other items.

Moving an item to another slide

To move an item between two story slides you first cut it out from one slide and then paste it onto another slide.

Do the following:

1. In storytelling view, click on the item you want to move.
The item is highlighted.
2. Click .
3. Switch to the slide you want to move the item to.
4. Click .

The moved item is placed in front of any other items on the slide.

Reordering items on a slide

When you are editing a story slide you can arrange and stack items on top of each other and decide the order in which the stacked items appear. This is useful when items overlap.

Reordering by one level

Do the following:

1. Right-click the item you want to reorder.
The item's shortcut menu is opened.
2. Select **Bring forward** or **Send backward**.

2 Sharing insights with data storytelling

The selected item is brought one level forward or sent one level backward.

Reordering to the front or the back

Do the following:

1. Right-click the item you want to reorder.
The item's shortcut menu is opened.
2. Select **Bring to front** or **Send to back**.

The selected item is sent behind or brought in front of all the other items on the slide.

Resizing items on a slide

You can resize an item on a slide by using its resizing handles. When you resize items such as images and text titles, they keep their aspect ratio. Paragraph texts and certain shapes can be resized more freely, both horizontally and vertically.

Snapshots can be resized in two different ways:

- Locked: keeping the aspect ratio.
- Unlocked: allowing free resizing and progressive disclosure.



Progressive disclosure means the following: If the size of a visualization (or a unlocked snapshot) is increased, its information is disclosed progressively. If the size of a visualization (or a unlocked snapshot) is decreased, its information is reduced, which allows you to focus on the essential information and avoid cluttering the visualization with too much information in too little space.

Resizing in general

Do the following:

1. In storytelling view, click on the item you want to resize.
2. Drag one of the resizing handles to resize the item.
The item snaps to the grid.
3. Release the resizing handle.

The item is resized.

Deleting items from a slide

Do the following:

1. In storytelling view, click on the item you want to delete.
The item is highlighted.
2. Click .

The item is deleted from the slide.



*You can also delete an item by selecting **Delete** from the item's shortcut menu.*

Organizing slides

In this section you will find how to reorganize and delete slides, to structure your story as needed.

Duplicating a slide in a story

You can duplicate a slide in a story.

Do the following:

1. Click the story you want edit.
The story opens with slide thumbnails to the left.
2. Right-click the slide that you want to duplicate.
3. Select **Duplicate**.

The duplicate slide is placed below the original slide.

Reordering slides on the timeline

When you are editing a story you can move slides around on the timeline.

Do the following:

1. Long-touch/click and hold the slide you want to move on the timeline.
2. Drag the slide to a new location.

A gap will open up between the slides at the new location and the slide will be placed there.

Deleting slides from stories

Do the following:

1. Right-click the slide you want to delete on the timeline, while in storytelling view.
The slide's shortcut menu is opened.
2. Click **Delete slide**.

The slide is deleted from the story and the timeline.

2.6 Making stories compelling

To make your story convincing, you can add emphasis to your insights.

Make sure that you add text, images, and shapes to your story slides. You can even apply effects to your snapshots to get your data to stand out.

Learn about the following:

[Adding emphasis](#)

[Styling with text and shapes](#)

[Adding bookmarks to a slide](#)

[Adding bookmarks to a slide](#)

[Adding links to a slide](#)

[Changing a snapshot's appearance](#)

Adding emphasis

You can add emphasis to your insights to make them clear and engaging for your audience.

You do this by adding effects to your snapshots to highlight certain data points, and at the same time suppressing information that might be irrelevant. In this way, the effects help you reduce information overload for your audience so that they can focus on the key insights you want to present to them.

In this section you find topics that will help you emphasize your insights, for instance how to apply various effects to a snapshot and how to highlight data points in a snapshot.

Another way to emphasize an insight and to reduce information overflow is to hide information. For example you could change a snapshot's appearance so that elements such as titles, axes, data point lines, footnotes, legends, and so on, are hidden.

Applying an effect to a snapshot

You can add different visual effects to your snapshots to make certain values stand out.



You can only use effects on bar charts, line charts, and pie charts.

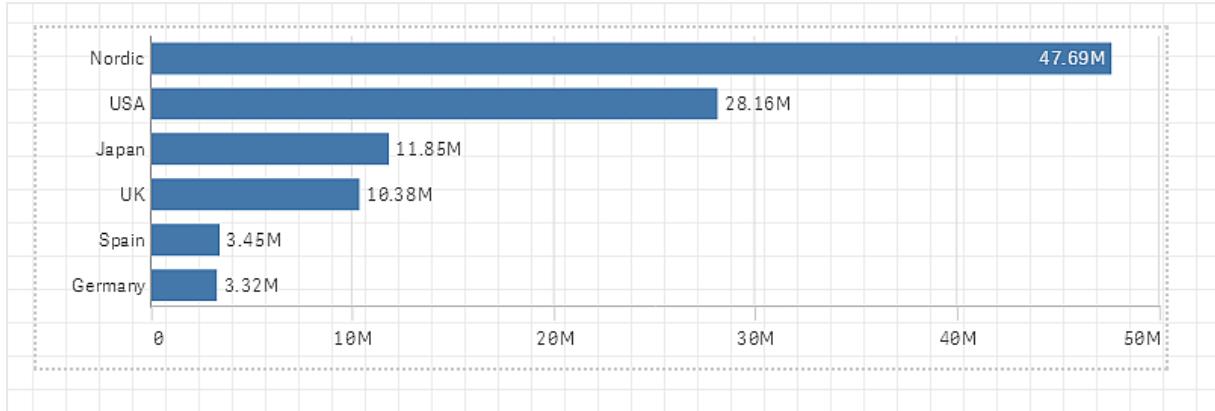
Do the following:

1. In storytelling view, click . The effects library is opened.
2. Locate the effect you want to use and drag it onto the snapshot.

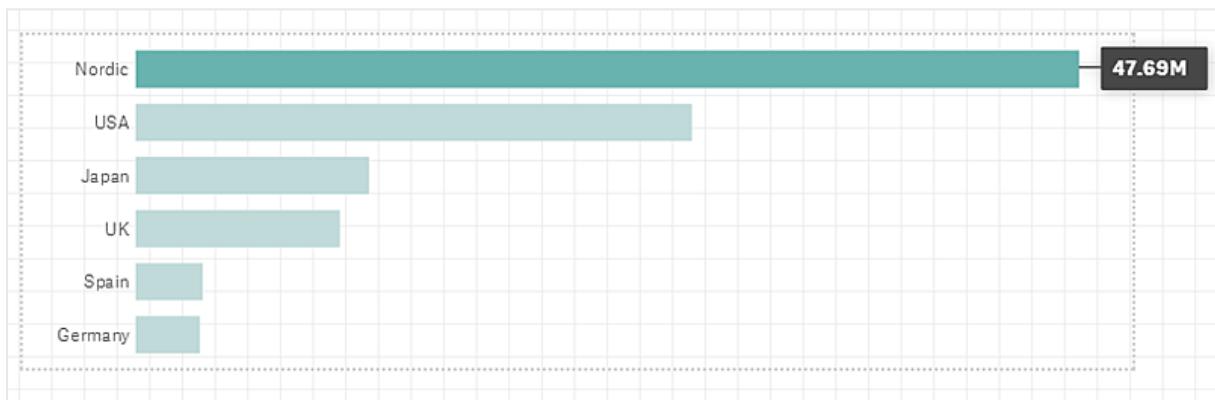
The snapshot is shown with the visual effect.

Snapshot of a bar chart without effects.

2 Sharing insights with data storytelling



Snapshot of a bar chart with the effect **Highest value** applied to it.



Highlighting data points in a snapshot

For a snapshot with the effect **Any value** applied to it, it is possible to select a certain data point to highlight.

Do the following:

1. In storytelling view, on a slide, select a snapshot with the applied effect **Any value**.
2. Click in the snapshot.
A dialog with a list of data points appears.
3. Click one of the data points in the list.



You can also click directly on the data points in the snapshot.

4. Click outside the snapshot.

The selected data point is now highlighted in the snapshot.

Removing an effect from a snapshot

Do the following:

1. In storytelling view, right-click on the snapshot with the effect you want to remove.
A shortcut menu appears.

2. In the shortcut menu, click **Remove effect**.

The effect is removed from the snapshot.

Styling with text and shapes

To make your stories compelling you can add stylistic elements. Text, hyper links and shapes are good examples.

In this section you will find topics that help you style your insights. You can add and format elements by size, color, font style and more.

Adding text to slides

You can add different styles of text to a story slide.

Do the following:

1. In storytelling view, click **A**.
The **Text objects** dialog is opened.
2. Locate the text style you want to add and drag it onto the slide.
The text object is placed on the slide and snaps to the grid. You can use the keyboard arrows to move it freely.
3. Double-click on the text object or click .
4. Type your text.
5. Click outside the text object or click .

The text is saved.

Formatting text on slides

On a story slide you can change the format of your texts with the text formatting toolbar. You can change color, size, font style, and text alignment. Additionally, you have options for adding links and bookmarks.

Text formatting toolbar for paragraph text object.



To enable a text object for formatting, you double-click on it.

Changing font styles

Do the following:

1. Double-click on the text object you want to format, while in storytelling view. Or, click on the text object and click . The text formatting toolbar appears.
2. Select the text you want to style.
3. Click one or more of **B**, **I** and **U**.
4. Click outside the text object or click .

The font style of the text is changed.

Changing font size of paragraphs

Do the following:

1. Double-click the **Paragraph** text object you want to format, while in storytelling view. Or, click on the text object and click . The text formatting toolbar appears.
2. Select the text you want to resize.
3. Click ▾ next to the font size indicator and select a size: XS, S, M, L or XL.
4. Click outside the text object or click .

The font size of the text is changed.

Changing font size of titles

Do the following:

1. Click the **Title** text object you want to format, while in storytelling view.
2. Drag one of the corners to change the font size.
3. Click outside the text object or click .

The font size of the text is changed.

Changing text alignment of paragraphs

Do the following:

1. Double-click the **Paragraph** text object you want to format, while in storytelling view. Or, click on the text object and click . The text formatting toolbar appears.
2. Click in the text paragraph you want to align.
3. Click ,  or .
4. Click outside the text object or click .

The text alignment is changed.



By default, the text is left-aligned.

Changing text color

Do the following:

1. Double-click the text object you want to format, while in storytelling view. Or, click on the text object and click .
The text formatting toolbar appears.
2. Select the text you want to change color on.
3. Click ▾ next to the color indicator in the toolbar.
A color palette appears.
4. Click on a color in the palette.
5. Click outside the text object or click .

The color is applied to the text.

Adding a shape to a slide

You can add different types of shapes on a slide.

Do the following:

1. In storytelling view, click .
The shapes library is opened.
2. Locate the shape you want to use and drag it onto the slide.
The shape is placed on the slide and snaps to the grid. You can use the keyboard arrows to move it freely.

The shape is placed on the slide.

Formatting shapes on a slide

Using the shapes formatting toolbar you have the following options to format your shapes:

- Color

Changing the color of a shape

Do the following:

1. In storytelling view, click the shape you want to format.
A toolbar with color options appears.
2. Click the color square in the toolbar.
A color palette appears.
3. Click a color.
4. Click outside the shape.

The color is applied to the shape.

Adding bookmarks to a slide

Via the text formatting toolbar, there is a link option where you can add bookmarks inside a text object.

Adding bookmarks

You can mark a text paragraph and use it as a bookmark.

Do the following:

1. In storytelling view, double-click the text object in which you want to add the bookmark.
The text formatting toolbar appears.
2. Select the text paragraph that you want to use as a bookmark.
3. Click .
A dialog opens.
4. Click **Bookmarks** on the right and select a bookmark from the list.
5. Click outside the text.

The bookmark is added.

Adding images to a slide

You can add an image to a story slide. You can use one of the default images, or an image of your own.

Do the following:

1. In storytelling view, click .
The **Media library** opens.
The following formats are supported: .png, .jpg, .jpeg, and .gif.

For Qlik Sense: You can upload images to the **In app** folder in the media library. You need to use the Qlik Management Console to upload images to the default folder.

For Qlik Sense Desktop: You can place images in the following folder on your computer:
`C:\Users\<user>\Documents\Qlik\Sense\Content\Default`. Images will be available in the **default** folder in the media library. When moving an app between installations, the images that you use in the app are saved in the qvf file together with the app. When you open the app in a new location, the images will be in the **In app** folder in the media library for the app.

2. Click on a folder in the media library, for example **In app** or **Default**.
3. Select the image that you want to add to the slide.
A preview of the image is shown.

4. Click **Insert**.



Alternatively, right-click the image file you want to add and select **Insert**.

The image is added and snaps to the grid. You can use the keyboard arrows to move it freely.



If the image is larger than the width or height of a slide, it will be re-sized to fit on the slide.

Adding links to a slide

Via the text formatting toolbar, there is an option for adding a link inside a text object.

Adding links

You can mark a text paragraph and use it for a link.

Do the following:

1. In storytelling view, double-click the text object in which you want to add the link.
The text formatting toolbar appears.
2. Select the text paragraph that you want to use for the link.
3. Click A link dialog opens.
4. Type the web address that you want to link to, in the **Link to** field.



If you do not add a prefix, http:// is added automatically, assuming that you are adding a web address.

5. Click or press Enter, to apply the new link.
6. Click outside the text.

The link is added.

Removing links

You can remove a link from a text paragraph.

Do the following:

1. Double-click the text object you want to remove the link from, while in storytelling view.
The text formatting toolbar appears.
2. Click the link so that the cursor is somewhere inside it.
3. Click A link dialog opens.
4. Click
5. Click outside the text.

The link is removed, but the text paragraph is kept.

Changing a snapshot's appearance

One way to make insights stand out and help you getting your message through in a clear way is the possibility to hide, or show, certain parts of a snapshot.

Changing snapshot properties

The following list summarizes the editable properties:

- Titles (main title, subtitle and footnote)
- Grid line spacing
- Labels (data point labels, leaf labels, dimension labels)
- Legend
- X-axis with title and labels
- Y-axis with title and labels

Do the following:

1. In storytelling view, click the snapshot you want to edit.
2. Click A dialog where you can change the snapshot's appearance is opened.
3. For **Show titles** click either **On** or **Off** to show or hide main title, subtitle and footnote.
4. For **Grid line spacing** you can set the spacing of the grid lines to **Auto** or **Custom** and select one of the options **No lines**, **Wide**, **Medium** or **Narrow** from the drop-down list.
5. For **Value labels** click either **Auto** or **Off**.
6. For **Show legend** click either **Auto** or **Off**.
7. For **x-axis and y-axis Labels and title**, select one of the options **Labels and title**, **Labels only**, **Title only** or **None** from the drop-down list.
8. Click **Done**.

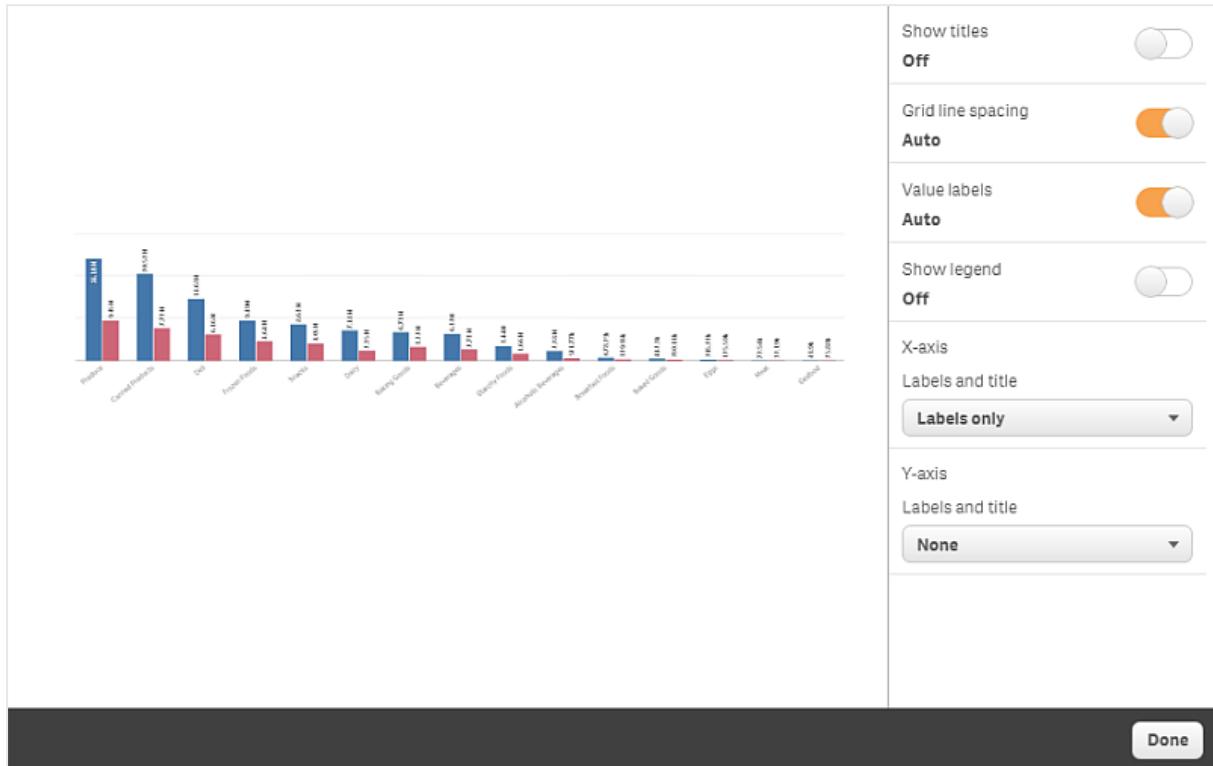
The snapshot's appearance has changed.



If you select auto and the property is not shown, you need to increase the size of the snapshot.

A snapshot dialog with grid lines and only the x-axis and the data labels visible.

2 Sharing insights with data storytelling



Changing the aspect ratio when resizing a snapshot

When you resize a snapshot you can choose to unlock the aspect ratio to enable free resize. Then the resize causes the visualization to use progressive disclosure.



Progressive disclosure means the following: If the size of a visualization (or a unlocked snapshot) is increased, its information is disclosed progressively. If the size of a visualization (or a unlocked snapshot) is decreased, its information is reduced, which allows you to focus on the essential information and avoid cluttering the visualization with too much information in too little space.

Do the following:

1. In storytelling view, click the snapshot you want to resize.
2. Click to unlock the aspect ratio for free resize.
 is displayed.
3. Use the handles on the edges to resize the visualization.
The resize causes the visualization to use progressive disclosure.
4. Click to lock the aspect ratio.

You now have changed the aspect ratio in the snapshot and resized the visualization.



If you resize the visualization when is displayed, the image is resized without progressive disclosure.

2.7 Presenting stories

While building your story you can see how looks and behaves for an audience, by playing it. It will occupy the whole browser view. You can see what your snapshots, live data sheets, texts, shapes, and visual effects look like.

If your audience asks questions, you can access the live data of your snapshots and your live data sheets. This way, you can find hidden stories to present, steering your story to new directions.

You can play a story as it would be presented to an audience. There are controls for you to navigate back and forth through the slides. On a slide with a live data sheet inserted, you must first click the sheet before you can start making selections in it.

Entering play

Do the following:

- In storytelling view, click ► **Play story**.

The story is opened in is ready to play starting with the current slide.



You can also start playing a story by right-clicking it in the app overview or the story navigator, and selecting **Play** from the shortcut menu.



On a touch device you can zoom in and out on the slide by using the pinch touch gesture. When you zoom in, you cannot make selections in live data sheets.

Showing data point information

Do the following:

- Hover over a data point to display a tool tip with data point information.

Interacting with a live data sheet

To interact and make selections in a live data sheet you need to click somewhere on the sheet first.

Do the following:

- Click on the live data sheet.

You can now make selection in the sheet.



You disable the possibility to make selections by clicking outside the live data sheet, or by navigating to another slide.

Downloading stories

Do the following:

1. Click at the bottom of the slide.
2. Click and select your format.

For more information, see *Downloading a story (page 57)*.

Closing play

Do the following:

- To close play either click or press Esc.

Accessing the live data of a snapshot

From a snapshot (on a story slide) you can navigate to the sheet and visualization it originated from. By doing this you get access to the live data of the snapshot where you can make new selections to take your story in new directions.

Do the following:

1. In storytelling view, navigate to a story slide with a snapshot.
2. Right-click the snapshot.
The snapshot's shortcut menu is opened.
3. Select **Go to source**.

You are directed to the sheet where the snapshot originated from. And the visualization which the snapshot was taken of, is initially highlighted.



A snapshot shows data according to the access rights of the user who takes the snapshot, and the snapshot can then be shared in a story. However, when users return to a visualization from a story to see the live data in the app, they are restricted by their own access rights.



If the sheet that the snapshot came from is no longer available, you are still directed to the originating sheet. If the whole sheet where the visualization was placed on is no longer available, you stay in storytelling view and get an error message stating **Sheet is missing**.



You can also access a snapshot's live data while playing the story.

Accessing a sheet from a slide

When you play a story, you can navigate from a live data sheet to the sheet in sheet view. By doing this you get access to the original sheet with your selections applied to it. You can then make new selections to take your story in new directions.

Do the following:

1. While playing the story, navigate to a story slide with a live data sheet.
2. Click **Go to sheet**.

You are directed to the sheet in sheet view carrying with you the selections you have made on the live data sheet.

Selecting in a live data sheet

When you play a story you can make selections in live data sheet just as you can in a sheet in sheet view.

Enable selections

Do the following:

1. In storytelling view, click ► above the timeline.
2. Navigate to a slide with a live data sheet.
3. Click on the sheet to enable it for selections.
The sheet is possible to interact with.
4. Start making selections in the visualizations.

The visualizations respond to selections and filter out subsets of the data.



You disable the possibility to make selections by clicking outside the sheet, or by navigating to another story slide.

When you are playing a story on a small screen you must go to sheet view to view the sheet and make selections.

Reset selections

You can reset the selections you have made on the live data sheet.

Do the following:

- While playing the story and with some selections made on a live data sheet, click **Reset selections**.

The sheet's selections will be reset to the same selections it had at the time you inserted the live data sheet on the slide.



The reset gives you the possibility to always return to the same selections as when you inserted the live data sheet on the slide.

2.8 Troubleshooting - Using data storytelling

This section describes problems that can occur when using data storytelling in Qlik Sense.

I cannot edit a story

I want to make changes to a story but I cannot edit it.

Possible cause

The story is published.

Proposed action

If you created the story, unpublish the story to enable editing.

For more information, see *Unpublishing a story (page 43)*.

Proposed action

If you did not create the story, make a duplicate of the story to enable editing. The duplicated story will not be updated if the original story is updated.

For more information, see *Working with stories (page 10)*.

A sheet is missing

When I play a story a sheet is missing.

Possible cause

The sheet has been deleted.

The sheet is private (located in another user's **My sheets**).

Proposed action

If the sheet is private, ask the owner of the sheet to publish it.

For more information, see *Publishing (page 34)* and *Publishing a sheet (page 41)*.

I cannot change a snapshot's appearance

I want to use auto for a snapshot property but the snapshot's appearance is not affected.

Possible cause

The snapshot is too small.

Proposed action

In storytelling view, increase the size of the snapshot.

For more information, see *Editing slides (page 15)*.

I cannot take a snapshot

I want to take a snapshot but it is not possible.

Possible cause

The visualization you are trying to take a snapshot of is incomplete.

The visualization you are trying to take a snapshot of is an extension. Extensions do not have enabled snapshot functionality by default.

The visualization you are trying to take a snapshot of is a filter pane. Snapshots of filter panes are not supported.

Proposed action

Complete the visualization.

Enable snapshot functionality in the main script of the visualization extension. To do this you have to set the snapshot property to true. You need permission to edit the script or need to get help from a developer.

I cannot re-order slides in a story

I want to move a slide in the story timeline but it is not possible.

Possible cause

You are using the mouse on a hybrid device.

Proposed action

Do one of the following:

- Turn touch device support off by tapping the navigation button (***) and toggling off **Touch screen mode**.
- Hold the mouse button pressed for a while before moving the slide.
- Alternatively, use the touch screen instead and move the item by long-touch and drag.

The data in my snapshot does not exist in the visualization

When I go to the snapshot source, I cannot see the same data in the visualization as in the snapshot.

2 Sharing insights with data storytelling

Possible cause

The user taking the snapshot has access to data you do not. This is because of section access rights defined in the data load script.

Proposed action

Request the same section access rights for you in the script.

3 Publishing

Publishing is a way to share the content of an app. You publish an app to a stream.

Those who have access rights to a stream can analyze data by interacting with the visualizations in the app.

You can also publish an app to a collection, making it available to users outside of your Qlik Sense Enterprise deployment.

Typically, you publish an app when you have stopped working on the design. Designing an app includes creating the visualizations, and organizing the presentation of the app. You can republish a published app to update it with new content.

When you publish an app, it is locked. Others cannot edit your published sheets and stories, but they can use them to interact with and analyze the data. New sheets and stories can be approved for inclusion in the public sheets and stories of a published app. This enables the collaborative design content in a published app.

In Qlik Sense Enterprise, your administrator can publish apps from the Qlik Management Console. The owners of an app can also publish their apps from the hub and move their published apps between streams. All published apps are evaluated by distribution policies set up by your Qlik Sense administrator. These policies determine whether or not the app will be distributed for consumption in a Qlik Sense Enterprise cloud hub.

Additionally, in Qlik Sense Enterprise, reports can be distributed from Qlik NPrinting to the Qlik Sense Enterprise hub. A link to a QlikView document can also be published in the Qlik Sense Enterprise hub.

3.1 Distribute your insights

Depending on your Qlik Sense Enterprise deployment, there are different ways to develop and publish apps.

You can develop an app by yourself and then publish it to a stream to make it available to consumers. If you are developing apps collaboratively, you can publish an app to a stream where reviewers and collaborators can add sheets and stories. You can then include these sheets and stories in the public content of your app before publishing it to a stream for consumers.

The following is a sample workflow for developing and publishing apps.

Publish your app to a Qlik Sense Enterprise review stream.

When you have created a data-model and created visualizations in your app, you can publish to a stream dedicated for reviews so that other Qlik Sense Enterprise hub users can access, collaborate, and provide feedback.

Manage the collaborative development of the published app.

Once the app is in the review stream, other developers can add sheets and stories to the app. Sheets and stories are private by default. However, developers can publish them as community sheets and stories to make them available to other users.

As the app owner, you can add community sheets and stories to the public sheets and stories of your app. This enables the sheets and stories to be included when the app is duplicated or exported.

Manage your app properties.

Qlik Sense administrators can create custom properties that are used for tasks such as limiting access to the app to certain users or groups. You can add these custom properties to your app in the hub.

Move your published app to a consumption stream.

When you have completed the review and collaborative development of the app, you can move your published app to a consumption stream, where the target consumers can access the app.

Republish the app.

Optionally, republish the app to make any necessary updates to the published app, such as new public sheets or updates to the data model.

Retire the app to an archive stream.

When the app is no longer required, you can remove it from the consumption stream to an archive stream.

3.2 Publishing to Qlik Sense Enterprise streams

The Qlik Sense Enterprise administrator publishes apps to a stream from the Qlik Management Console. The sheets and stories in the app are then available to the users that have access to the stream. You can publish your own apps from the hub, to a stream which you have publish access to. You can also move your published apps between streams from the hub.

An app that has been published to a stream is locked, although the owner of a published app can edit the thumbnail, app name, and the description of their published app. Others can interact with the visualizations and analyze the data. However, no one can edit the public sheets and public stories that were published with the app. If you have the correct access rights, you can add your private sheets and stories to a published app.

If you own a published app, you can add your private sheets and stories to a published app, and approve your own and community sheets for inclusion in the public sheets of an app. You cannot edit the public sheets and public stories of the app, but you can make them private sheets and stories if you want to edit them.

If a published app needs to be changed, it can be republished, updating the published app with content from a duplicate.

Streams

The content in the hub is organized in streams. A stream is a collection of apps that a group of users has specific access to. The users of the stream can have different access rights. Some users might only be able to read the content in the stream, while others might have the rights to publish their content to the stream.

By default, Qlik Sense includes a stream called Everyone, which all users have both read and publish rights to.

All users have their own spaces, **Work** and **Published**, for the content they own. **Work** contains personal unpublished apps. **Published** contains links to your published apps. These links enable you to keep track of your published work and perform tasks such as moving or duplicating a published app more easily. You can also mark your favorite apps for easy access. The **Favorites** space appears after you tag at least one app as a favorite, and then refresh the page.

The read and publish rights on the streams are defined in the Qlik Management Console.

You can use a duplicate of a published app to use as a template for a new one.

You can move your own apps from the hub between streams to which you have publish access.

Sheets and stories

You can create private sheets and stories as a part of a published app. These sheets and stories can be published as part of the app. Published sheets and stories can be added to public sheets of an app by the app owner. You can also remove public sheets and stories from your app and make them published sheets and stories.

When working with a published app, sheets and stories are organized in sections in the app overview depending on their status.

Sample app with overview showing four sections: Public sheets, Community, Published by me, and My sheets.

The screenshot shows the Qlik Sense Sample app interface. At the top, there's a header with the app name "Sample app" and some metadata: "Data last loaded: Oct 18, 2022, 12:17 PM", "Published: Nov 1, 2022, 11:45 AM", and "Published to: Documentation". Below the header, there are three navigation tabs: "Sheets" (which is selected), "Bookmarks", and "Stories".

The main content area is divided into four sections:

- Public sheets (1)**: Contains one sheet named "My new sheet".
- Community (1)**: Contains one sheet named "Community sheet".
- Published by me (1)**: Contains one sheet named "My new sheet".
- My sheets (3)**: Contains three sheets named "Sheet 1", "Sheet 2", and "Sheet 3". To the right of these, there's a button labeled "Create new sheet" with a plus sign icon.

Sheet view sections

Section	Description
Public sheets / Public stories	Sheets and stories that were included in the app when it was published or republished, or added to the public sheets and stories of the app after it was published. All users of the app have access to these.
Community	Sheets, bookmarks, and stories that someone else has created and published to the app that you have access to. Sheets and stories in this section can be added to public sheets and stories.
Published by me	Sheets, bookmarks, and stories that you have created and then published so that all the users of the app can access them. The other users will find these in their Community section. Sheets in this section can be approved for inclusion in public sheets.
My sheets / My stories	Sheets and stories that you have created but not published. No one else can see these.

Interacting with apps

The tasks that you can perform in an app depend on whether or not that app has been published.

Unpublished app

You can:

- Edit the load script and reload the data.
- View the data structure and details about the data from the data model viewer.
- Create, edit, and delete sheets, visualizations, and bookmarks.
- Create master items (dimensions, measures, and visualizations) for reuse.
- Navigate between sheets, stories, and bookmarks.
- Make and clear selections.
- Apply/recall bookmarks.

Published app

You can:

- Navigate between sheets.
- Explore sheets and visualizations that were included in the app from the beginning.
- View stories that were included in the app from the beginning.
- Create and edit your own private sheets, visualizations and stories based on the data in the app.
- Publish sheets, stories, and bookmarks that you have created so that others can use them.
- Unpublish sheets, stories, and bookmarks that you have published so that they become private again.
- Update your private sheets
- Add published sheets and stories to the public sheets and stories.
- Remove sheets and stories from the public sheets and stories.
- Copy and paste visualizations between sheets.
- Use master items (dimensions, measures and visualizations) that were included in the app from the beginning.
- Create and edit your own private bookmarks.
- Make and clear selections.
- Apply/recall bookmarks.
- Access the data model viewer, if you are the owner of the app with the default permissions.

You cannot:

- Edit the load script or reload the data.
- Access the data model viewer if you do not own the app.
- Edit visualizations, sheets, stories and bookmarks that were included in the app from the beginning.
- Edit sheets and stories that have been published.

Publishing an app from the hub

You can publish an app that you have created to any stream to which you have publish access. If you have published an app to a stream, you can move your app between streams to which you have publish access.

When you publish an app to a stream, the app is added to that stream. A link to that app is also added to **Published** and marked with  to indicate the app is in a stream. This enables you to keep track of your published work and perform tasks such as moving or duplicating a published app more easily. When publishing an app, you can also add or remove app properties created by your Qlik Sense administrator to your app. For information on app properties, see *Managing app properties (page 40)*.

When you publish an app or move a published app between streams, the sheets and stories of the app will become available to the other users that have access to the stream that your published app belongs to.



*The **Variables** overview is not available in published apps. If you need to add or change variables in an published app, use the variable input control available with the Dashboard bundle.*

A published app can only be deleted from the QMC.



To avoid exposing restricted data, remove all attached files with section access settings before publishing the app. Attached files are included when the app is published. If the published app is copied, the attached files are included in the copy. However, if section access restrictions have been applied to the attached data files, the section access settings are not retained when the files are copied, so users of the copied app will be able to see all the data in the attached files.

Do the following:

1. In **Work** in the hub, right-click the app and do one of the following:
 - Select **Publish**.
 - Select **Move**.
2. Select the stream you want to publish the app to, in the **Stream name** drop-down list.



There is no drop-down if you only have access to one stream.

3. Type a name in the **App name** field (optional). The field displays the name of the app you selected from the hub.



It is possible to publish many apps with identical names in a stream. Qlik Sense will indicate when there are published apps with the same name in a stream.

4. If you want to add app properties, click **Manage**, select app properties, select the values, and click

Apply.

5. Click **Publish or Move**.

The published app is now in the selected stream. An entry for the published app is added to **Published** that links to the published app in its new stream.

If you publish or move an app into a stream they are currently viewing, they will be notified and can update the app list for their stream.

Republishing an app from the hub

After you publish an app to a stream, you may need to make changes to the base content of the app without removing the app from the stream.

For example, you might want to continue working on and improving an already published app without losing the community and private content in the published app. To republish an app, you need to duplicate your published app. The duplicate app contains a link to the original app. You then make your changes to the duplicate app and republish it back to the same stream as the published app.

All content in the base section is overwritten with the content from the duplicate when republishing. Content in the original app, such as private and community sheets and stories, are kept when you republish an app.

You can also republish an app from QMC.

Do the following:

1. In **Work**, right-click the duplicated app and select **Publish**.
2. Select **Replace the existing app**.
3. Click **Republish**.

Managing app properties

Your Qlik Sense administrator can create custom properties in QMC for apps.

These properties can be used for tasks such as creating a distribution policy limiting access to apps. An administrator might create a custom property for users or groups and then add the names of the users or groups as values to that property. You can apply these custom properties and specific values from these properties to apps in **Work** and in streams, as well as when you are publishing or moving apps.



You can add app properties to an unpublished app, but they may not be applied depending on the custom properties settings, which are controlled by your Qlik Sense administrator.

You can view the app properties in the **Manage properties** dialog and from the app details.

Do the following:

1. In the hub, right-click on an app and select **Manage properties**.

You can also access **Manage properties** from the app details. Click  on an app and then click **Manage**.

2. Do one of the following:
 - To add properties to your app, select an app property and then select values from that property.
You can search for values
 - To remove app properties, clicking  on the property value.

3. Click **Apply**.

Publishing a sheet

You can publish sheets that you have created, so that other users can view them.



You can only publish sheets as a part of an app that is already published.

Do the following:

1. From the app overview, click  to view the sheets.
2. Right-click the sheet you want to publish and select **Publish**.
The **Publish sheet** dialog appears.
3. Click **Publish**.

The sheet is published and moved from **My sheets** to **Published by me**.

Other users, with access to the same app, will find the published sheet in their **Community** section.



*You can also publish a sheet in the sheet view by clicking  and selecting **Publish sheet**, or by right-clicking a sheet in the sheet navigator.*

Unpublishing a sheet

You can revert a sheet that you have published to being unavailable to others by unpublishing it.

Do the following:

1. From the app overview, click  to view the sheets.
2. Find the published sheet under **Published by me**.
3. Right-click the sheet and select **Unpublish**.
The **Unpublish sheet** dialog appears.
4. Click **Unpublish**.
The sheet is unpublished and moved from **Published by me** to **My sheets**.

The sheet is no longer available to other users of the app.



*You can also unpublish a sheet in the sheet view by clicking and selecting **Unpublish sheet**, or by right-clicking a sheet in the sheet navigator.*

Adding sheets to the public sheets of an app

You can approve published sheets, both published sheets owned by you or community sheets, to add them to the public sheets of the app.

Sheets added to the public sheets, unlike published sheets, are included when an app is duplicated or exported. Adding sheets to your public sheets enables you to collaboratively develop the public sheets of your app.

Do the following:

- In your app, right-click on a community sheet or one of your published sheets and select **Approve**.

The sheet is now included in the public sheets of the app.



*You can also add a sheet in the sheet view by clicking and selecting **Approve**, or by right-clicking a sheet in the sheet navigator.*

Removing sheets from the public sheets of an app

You can remove public sheets from your app, both those originally in the app and those approved for inclusion in the public sheets.

When you remove a public sheet, it is moved to the **Published by me** section of the app if the sheet was published by you or the **Community** section if it was published by another user. If the sheet belonged to another user, they will become the owner again. A sheet removed from the public sheets is not included if the app is duplicated or exported.

Do the following:

- In your app, right-click a base sheet and select **Unapprove**.

The sheet is moved to the **Published by me** section or the **Community** section.



*You can also remove a sheet in the sheet view by clicking and selecting **Unapprove**, or by right-clicking a sheet in the sheet navigator.*

Publishing a story

You can publish stories that you have created, so that other users can view them.



You can only publish stories as a part of an app that is already published.

Do the following:

1. From the app overview, click to view the stories.
2. Right-click the story you want to publish and select **Publish**.
The **Publish story** dialog appears.
3. Click **Publish**.

The story is published and moved from **My stories** to **Published by me**.

Other users, with access to the same app, will find the published story in their **Community** section.



*You can also publish a story in the storytelling view by clicking and selecting **Publish story**, or by right-clicking a story in the story navigator.*

Unpublishing a story

You can revert a story that you have published to being unavailable to others by unpublishing it.

Do the following:

1. From the app overview, click to view the stories.
2. Find the published story under **Published by me**.
3. Right-click/long-touch the story and select **Unpublish**.
The **Unpublish story** dialog appears.
4. Click **Unpublish**.
The story is unpublished and moved from **Published by me** to **My stories**.

The story is no longer available to other users of the app.



*You can also unpublish a story in the storytelling view by clicking and selecting **Unpublish story**, or by right-clicking a story in the story navigator.*

Adding stories to the public stories of an app

You can approve published stories, both published stories owned by you or community stories, to add them to the public stories of the app.

Stories added to the public stories , unlike published stories, are included when an app is duplicated or exported. Adding stories to your public stories enables you to collaboratively develop the public stories of your app.

Do the following:

- In your app, right-click on a community story or one of your published story and select **Approve**.

The story is now included in the public stories of the app.



*You can also add a story in the storytelling view by clicking and selecting **Approve**, or by right-clicking a story in the story navigator.*

Removing stories from the public stories of an app

You can remove public stories from your app, both those originally in the app and those approved for inclusion in the public stories.

When you remove a public story, it is moved to the **Published by me** section of the app if the stories was published by you or the **Community** section if it was published by another user. If the story belonged to another user, they will become the owner again. A story removed from the public stories is not included if the app is duplicated or exported.

Do the following:

- In your app, right-click a public story and select **Unapprove**.

The story is moved to the **Published by me** section or the **Community** section.



*You can also remove a story in the story view by clicking and selecting **Unapprove**, or by right-clicking a sheet in the story navigator.*

Publishing bookmarks

You can publish bookmarks that you have created to **Community** in the app overview. Everyone with access to this app can apply these bookmarks.



You can only publish bookmarks as a part of an app that is already published.

Do the following:

1. From the app overview, click to view the bookmarks.
2. Right-click the bookmark you want to publish and select **Publish**.
The **Publish bookmark** dialog appears.
3. Click **Publish**.

The bookmark is published and moved from **My sheets** to **Published by me**.

Other users, with access to the same app, will find the published bookmark in their **Community** section.



You can also publish a bookmark in the sheet view by right-clicking a bookmark in **Bookmarks**.

Unpublishing bookmarks

You can revert a bookmark that you have published to being unavailable to others by unpublishing it.

Do the following:

1. From the app overview, click to view the bookmarks.
2. Find the published bookmark under **Published by me**.
3. Right-click the bookmark and select **Unpublish**.
The **Unpublish bookmark** dialog appears.
4. Click **Unpublish**.
The bookmark is unpublished and moved from **Published by me** to **My bookmarks**.

The bookmark is no longer available to other users of the app.



You can also unpublish a bookmark in the sheet view by right-clicking a bookmark in **Bookmarks**.

Copying links to bookmarks in published apps

You can create and share links to bookmarks in **Public** or **Community**. To copy a link to a bookmark, right-click a bookmark and select **Copy link**. Using the link takes users to the app with the bookmark applied. The link only works for users with permission to access the app.

3.3 Publishing to Qlik Sense Enterprise streams for Qlik Sense Mobile Client Managed

Qlik Sense apps and mashups that are published to a stream in Qlik Sense Enterprise are available for viewing in the Qlik Sense Mobile Client Managed app.

Users that have installed the Qlik Sense Mobile Client Managed app can interact with the Qlik Sense app in the stream, provided they have the appropriate access privileges.

Typically, Qlik Sense apps are developed in a web browser connected to a Qlik Sense Enterprise server. When your Qlik Sense administrator publishes an app or a mashup, it will also be available in the Qlik Sense Mobile Client Managed app. Qlik Sense November 2018 is required to access mashups from the Qlik Sense Mobile Client Managed app.

Your administrator configures access privileges in the QMC. Additionally, your administrator can configure whether or not a Qlik Sense app can be downloaded from a stream in the Qlik Sense Mobile Client Managed app. When you download Qlik Sense, it is also available for viewing when you are offline.

You do not need to install the Qlik Sense Mobile Client Managed app to be able to publish for the mobile app. However, we recommend testing all Qlik Sense apps in the mobile app, if that is where the app will be used.

3.4 Publishing from Qlik Sense Enterprise on Windows to other hubs

If you want to make your apps available to users who do not have access to the Qlik Sense Enterprise on Windows hub, you can publish them to a cloud hub.

In a multi-cloud enabled Qlik Sense Enterprise deployment, your Qlik Sense administrator creates distribution policies in QMC. The Qlik Sense administrator then creates streams for your apps. When you publish or move an app to a stream with distribution policy app properties applied, Qlik Sense copies the app to the cloud hub. In the cloud hub, the app is staged. The tenant or analytics administrator then makes the staged app available in a managed space in the cloud hub.

Staged apps

In Qlik Sense SaaS, apps without owners that are not in spaces are called staged apps. When an app is published to a cloud hub from Qlik Sense Enterprise on Windows, it will not have an owner or space. A tenant or analytics administrator can assign it to a managed space from the Management Console.

For more information on managing staged apps in Management Console, see [Managing apps](#).

Tags

If you want to use tags in apps that are distributed from Client-Managed Qlik Sense to Qlik Cloud, you need to create a custom property called *Tags* and then add appropriate values that can be used when distributing the app. If a distributed app has a tag value *Sales*, that value is synced with the cloud app and visible in the app details. If a distributed app doesn't have any tag value, the displayed value in the app details is empty.

Streams and tags

The Qlik Sense administrator can have a distribution policy that distributes published apps with added app properties with one or more tags to a cloud hub.

When an app is published to a stream in Qlik Sense Enterprise on Windows, distribution policies determine the target cloud hubs that will receive the app with the tags. You can set app properties that control the destination cloud hub and tags when publishing in the Qlik Sense Enterprise on Windows hub. Qlik Sense Enterprise on Windows copies the app from the stream to the cloud hub with the tags whenever you publish. The destination for each distribution is controlled by distribution policies managed by your Qlik Sense administrator.

Apps can be published to only one stream, but they can be published with multiple tags to multiple cloud hubs.



Your administrator can see the time of the last sync in the stream properties.

Depending on your organization you may prefer to move apps to an archive stream with no distribution policy instead of just removing the app.

Tagged apps can be modified in Qlik Sense Enterprise on Windows hub, like an app in any other Qlik Sense Enterprise on Windows stream. In Qlik Sense Enterprise on Windows hub. You can:

- Duplicate apps in the stream.
- Republish the app in the stream.
- Manage app custom properties.
- Open the app in **Published**.

In an open app in Qlik Sense Enterprise on Windows, you can:

- Publish and unpublish personal sheets and stories.
- Add public sheets and stories to the app or remove them from the app.

Publishing apps to cloud hubs with tags

You can publish apps to cloud hubs with tags to which you have write access. You do this at the same time as publishing to a stream. Apps that are already in a stream can also be published to cloud hubs with tags.

When an app is published to a stream in Qlik Sense Enterprise on Windows, distribution policies created by your Qlik Sense administrator determine the target cloud hubs that will receive the app with tags using custom app properties. When publishing an app to a stream, you can set app properties that control the destination cloud hub and tags. You can also apply the app properties to a published app already in a stream. When you set these properties, Qlik Sense copies the app in the stream to the selected cloud hubs with the selected tags.

Once an app is published to a cloud hub, a tenant or analytics administrator can make the staged app available in a space in the cloud hub.

If your deployment has multiple cloud hubs, you will need to choose one.



To avoid exposing restricted data, remove all attached files with section access settings before publishing the app. Attached files are included when the app is published. If the published app is copied, the attached files are included in the copy. However, if section access restrictions have been applied to the attached data files, the section access settings are not retained when the files are copied, so users of the copied app will be able to see all the data in the attached files.

Publishing an app to cloud hubs with tags



The tags and destination cloud hub custom properties are determined by your Qlik Sense administrator. The names may vary from the names used in this procedure.

Do the following:

1. In **Work** in the Qlik Sense Enterprise on Windows hub, right-click the app and then select **Publish**.
2. In the **Stream name** drop-down list, select a stream.



There is no drop-down list if you only have access to one stream.

3.



It is possible to publish many apps with identical names to a stream. Qlik Sense will indicate when there are published apps with the same name in a stream.

4. Optionally, you can rename the app in the **App name** field. This field displays the name of the app that you selected from the hub.
5. Click **Manage**.
6. For **Tags**, select one or more tags from the list of values.
7. Click **Apply**.
8. Click **Publish**.

The published app is now staged in the cloud hub with the tags selected in the **Manage properties** dialog. The tenant administrator can add the staged app to a managed space.

Distributing a published app in a stream to cloud hubs with tags

Do the following:

1. In a stream, right-click the app and select **Manage properties**.
2. For **Tags**, select one or more tags from the list of values.
3. Click **Apply**.

The published app is now staged in the cloud hub with the tags selected in the **Manage properties** dialog. The tenant administrator can add the staged app to a managed space.

Editing the tags of a published app

Do the following:

1. In the stream, right-click the app and select **Manage properties**.
2. Edit the tags as needed.
3. Click **Apply**.

If a new tag was selected, the app will be distributed to the corresponding tenant.

3.5 Publishing between Qlik Sense platforms

You cannot publish apps directly between some Qlik Sense platforms.

For example, you cannot publish an app from Qlik Sense Desktop to Qlik Sense Enterprise. Instead, you have to download a copy of the app from one platform, and then upload the copy to the other platform. You can then publish the app in the platform to which it is copied.

For information about how Qlik manages privacy in its products, see [Qlik Product Privacy Notice](#).



You cannot publish apps in Qlik Sense Desktop.

When copying an app to another platform, consider the following:

- You will have to recreate data connections on the destination platform.
- Some functionality may not be supported on the destination platform.
- You need the appropriate privileges to perform certain actions. For example, only the Qlik Sense Enterprise administrator can import or export apps from the Qlik Management Console.

In Qlik Sense Enterprise, apps can be imported and exported in the QMC.

In Qlik Sense SaaS, apps can be imported and exported in the hub. To export, click *** on an app. To import, click **Add new** and then click **Upload app**.

In Qlik Sense Desktop, apps can be copied to or from this location: C:\Users\username\Documents\Qlik\Sense\Apps

3.6 Qlik NPrinting reports in Qlik Sense

Reports can be distributed from Qlik NPrinting to the Qlik Sense hub.

Recipients must be named users who are part of the Active Directory user group in both Qlik Sense and Qlik NPrinting. The reports are private and can only be viewed by the user to whom they are distributed.

Reports can be viewed from the hub by clicking **Reports**. Report formats include PDF, Excel, Word, PowerPoint, PixelPerfect, and HTML.

Limitations

- You cannot delete Qlik NPrinting reports from the Qlik Sense hub.
- Reports cannot be found with Qlik Sense search.

Distributing Qlik NPrinting reports to Qlik Sense

You can schedule the distribution of Qlik NPrinting reports to Qlik Sense.

Requirements

To enable and configure the distribution of Qlik NPrinting reports to Qlik Sense, the following is required:

- Your Qlik NPrinting version is 17.1 or newer.
- Certificates (client.pfx, server.pfx, and root.cer) have been exported, by your Qlik Sense system administrator, from the Qlik Sense QMC. You have to include a secret key when exporting the certificates.
- The certificates have been installed, by the Qlik NPrinting administrator, on all computers where the Qlik NPrinting scheduler service is running.
- A firewall rule has been created allowing inbound connections on port 4243 of the Qlik Sense QRS.
- To configure report distribution in Qlik NPrinting, you need to be a Qlik NPrinting scheduler service user with **Publish** tasks authorization. The following information is also required:
 - the central node address for the Qlik Sense server. This is available from the Qlik Sense QMC under **Nodes**.



See the Qlik NPrinting help for more detailed procedures about installing certificates, and configuring report distribution from Qlik NPrinting to Qlik Sense.

Do the following:

1. Log into Qlik NPrinting as a user with **Publish** tasks authorization.
2. Select **Destinations** and then select **Hub**. Add the Qlik Sense hub as destination. You need the central node address for the Qlik Sense server.
3. Create a **Publish** task and then specify a Qlik NPrinting app, reports, users, schedule, and the hub as a destination.

Connecting to Qlik Sense apps in Qlik NPrinting

You can create connections from Qlik NPrinting to Qlik Sense apps. You can then use the data to create and design reports in Qlik NPrinting.

Requirements

To enable and create connections from Qlik NPrinting to Qlik Sense apps the following is required:

- The same certificate requirements apply as those for distributing reports to from Qlik NPrinting to Qlik Sense. However, the certificates have to be installed on all Qlik NPrinting Engine computers.
- A firewall rule has been created allowing inbound connections on port 4243 of the Qlik Sense QRS.
- To create a connection to a Qlik Sense app in Qlik NPrinting, you need to be an Qlik NPrinting user with **Connections** tasks authorization. The following information is also required:
 - the proxy address (full computer name) for the Qlik Sense server.
 - the Qlik Sense app id. This is available from the Qlik Sense QMC under **Apps**.
 - the Windows domain and user name of the Qlik Sense administrator. This is available from the Qlik Sense QMC under **Users**.



See the Qlik NPrinting help for more detailed procedures about installing certificates, and configuring connections from Qlik NPrinting to Qlik Sense apps.

Do the following tasks in Qlik NPrinting to create a connection to a Qlik Sense report:

1. Log into Qlik NPrinting as a user with **Connections** tasks authorization.
2. Select the **Apps** drop-down list, and then select **Apps** from the list. Click **Create App** to create a new Qlik NPrinting app.
3. Select **Apps** and then select **Connections**. Click **Create Connection**, select the app that you just created, and then click Qlik Sense as your **Source**. You will require the following information to finish creating the connection:
 - the proxy address (full computer name) for the Qlik Sense server.
 - the Qlik Sense app id.
 - the Windows domain and user name of the Qlik Sense administrator.

3.7 QlikView documents in Qlik Sense

You can publish a link to a QlikView document in the Qlik Sense hub instead of distributing files by email or to a specific folder location.

You can use this method when you want to share a QlikView document with a named user that is part of the **Active Directory** user group in both QlikView and Qlik Sense. In QlikView, the administrator must set up distribution to Qlik Sense.

Requirements

QlikView and Qlik Sense requirements

To enable the publishing of QlikView documents links, the following is required:

- Your QlikView version is 12.00 SR3 or newer.
- Your QlikView installation has a Publisher license.
- A different set of certificates (*client.pfx*, *server.pfx*, and *root.cer*) has been exported, by your system administrator, from the Qlik Sense Management Console to each of the QlikView Distribution Service (QDS) machines.
- Qlik Sense has been configured to allow shared content.
- The server connection to the QlikView AccessPoint has been configured to use the machine name.

QlikView certificate requirements

When creating a set of certificate for QlikView, the following is required:

- The certificates must be created using the full name, including the domain of the QDS machine.
- The certificates must be password protected.
- A secret key pair must be created.

Configuration requirements

The following topics provide details about the configurations needed when publishing QlikView document links in the Qlik Sense hub:

- [Configuring Qlik Sense to allow users to publish a link to shared content](#)
- [Configuring the QlikView Distribution Service with the Qlik Sense certificates](#)
- [Creating a task to publish a link to a QlikView document in the Qlik Sense hub](#)

Publishing links to QlikView documents in the Qlik Sense hub

You share links to QlikView documents in the Qlik Sense hub by using the QlikView Management Console. To view QlikView documents, log in to the Qlik Sense hub using the same credentials as a the named user that has access to the document was shared with. Select a link to a document in **QlikView documents** to open the QlikView AccessPoint in a new window. When viewing QlikView documents in Qlik Sense, no changes can be saved.



QlikView documents cannot be displayed using the mobile view for small devices.

Do the following:

1. Click the **Documents** tab.

The **Source Document** page opens.



Only source documents can be published.

2. Expand a QDS machine instance and locate the document you want to share.
3. Click to create a new task.
4. On the **Distribute** tab, click to add a recipient.
5. Select the **Named User** user type.
6. Click to add a user.



*The named user must be part of the **Active Directory** user group in both QlikView and Qlik Sense.*

7. On the **Document Information** tab click to add an attribute.
8. Type *ShowInSenseHub* in the **Name** field and *true* in the **Value** field.
9. Click **Apply**.

The task may be run and will now add a link to the QlikView document in the Qlik Sense hub.

4 Downloading and printing

You can share insights with people outside the boundaries of a Qlik Sense system by downloading and printing sheets and visualizations. For example, you can coordinate with suppliers, making shipping decisions based on your data.

You can download sheets and visualizations as PDFs and images.

Qlik NPrinting can connect to Qlik Sense and generate reports in different output formats. For example, you can create a daily Excel report listing medical supplies that need to be re-ordered. For more information, see: [About Qlik NPrinting](#).

4.1 Downloading data from a visualization

You can download data from a visualization and save it as an .xlsx file.

For the number of row and column limitations of an Excel export file, see [ExportData method](#).

Do the following:

1. Right-click the visualization that you want to download data from.
2. Select  **Download as...** and **Data**.
3. Click the link to download the data file.
4. Open or save the file.



In filter panes with more than one dimension, you can either select all dimensions or a single dimension.



When exporting data, empty column names in a table will cause a corrupted data error. To avoid this error, always add a name to a column.

Downloading data from a table

In the **Data export settings** dialog you can select to have enhanced table formatting applied to your export, including styling and row totals. Enhanced table formatting cannot be applied to pivot table exports.

This export is performed by the browser, and therefore may require more time to complete, especially for large data sets.

Styling is applied to the data with a few exceptions:

- Download to Excel with enhanced table formatting works for tables with maximum 2,097,152 cells.
- Download to Excel with enhanced table formatting is not supported via the API.
- Images in table cells are not included.

- Mini charts in table cells are not included.
- Indicators in table cells are not included.
- The following number formatting modes are not supported:
 - Auto
 - Duration
 - Custom
- Locale is not supported in some cases. For example, Excel uses delimiters defined on the local PC, and these cannot be overridden.
- If specific alignment is set for a column, the title will inherit this in the downloaded Excel export file.
- Dual fields that have both `qText` and `qNum` with no formatting are exported as number.
- Starting date in Qlik Sense and Excel differs.
- Right-to-left (RTL) text is not included in the download.
- Opacity for cell fill color is not included in the download.
- Color defined through CSS classes in custom themes is not included in the download.

4.2 Downloading a sheet

In Qlik Sense you can download an entire sheet as a PDF file.

Sheets downloaded as a PDF will never exceed the selected paper size and orientation. If a sheet cannot fit into a PDF page, it is resized. Extended sheets and custom-sized sheets may have a lower quality PDF output if the sheet is too large to clearly display on a single PDF page.

The download process will look different if you are using a touch screen device.

About aspect ratio

There are two aspect ratio options:

Keep current size

- The whole sheet is scaled to fit the chosen PDF page format and orientation.
- Chart proportions stay the same as seen in the browser window. This means visualizations may be cropped.
- Sheet will be right-aligned on the page.
- Resolution is affected by browser window size. A PDF printed from a browser window larger than the chosen page format may appear grainy, because the number of printed pixels is less than the original.

Fit to page

- The whole sheet is scaled to fit the chosen PDF page format and orientation.
- Chart proportions are changed to fill the page. Aspect ratio will change accordingly. There may be more space around charts in order to make the page appear full.
- Sheet will be right-aligned on the page.

Downloading sheets

Do the following:

1. Open the sheet you want to download.
2. Click the navigation button in the toolbar (•••) and select **Download sheet as PDF**.
The **PDF settings** dialog appears.
3. Use the **Paper size** drop down menu to select page dimensions.
4. Under **Resolution (dots per inch)** use + or - to increase or decrease the resolution accordingly.
You can also type an exact value. The minimum value is 72 DPI, the maximum is 300 DPI.
5. Choose the **Orientation** by selecting **Portrait** or **Landscape**.
6. Under **Aspect ratio options** you can select:
 - **Keep current size**
 - **Fit to page**
7. Click **Export** to start image creation.

Limitations

- Pivot tables are downloaded as non-expanded. Partial expansions or full view are not kept.
- Third-party extensions, filter panes, and action buttons are downloaded as blank images.
- Visualization extensions (custom objects) cannot be downloaded as PDFs. They can be downloaded as images.
- Dynamic view objects are not supported
- There is no preview in the **Download** dialog if the browser does not have a PDF viewer plugin installed.
- When you download a sheet from an app with a custom theme applied, the theme is only applied to the visualizations. Styling for the sheet, such as background color, will not be applied in the downloaded PDF.
- Extended sheets and custom-sized sheets may have a lower quality PDF output if the sheet is too large to clearly display on a single PDF page.

4.3 Downloading a visualization

You can download visualizations as images or PDF files.

Downloading visualizations from desktop devices

Download as an image

Do the following:

1. Click ••• at the top right of the visualization or right-click on the visualization.
2. In the options menu, select **Download as... > Image**.
The dialog **Image settings** appears.
3. Select to keep **Current** options or to change them by using **Custom** options.

- Selecting **Current** shows the width and height of the original chart and the screen resolution in dpi. You can select the output format between .png and .jpeg by using the **Type of file** drop-down menu.
 - You can customize the dimensions and the resolution of the exported image by clicking **Custom:**
To set a new image width or height, click + or - to increase or decrease the width or height accordingly. You can also type an exact value. The minimum value is 8 pixels, the maximum is 2,000 pixels.
To set a new image resolution, click + or - on the sides of **Resolution (dots per inch)** to increase or decrease the resolution accordingly. You can also type an exact value. The minimum value is 72 dpi, the maximum is 300 dpi.
4. Click **Export** to start image creation.

Best practice

Here are a few tips to help you download a visualization as an image.

- The maximum size of an image that can be exported is 2,000 by 2,000 pixels. If the export would result in a larger image, you must reduce its size under **Custom**.
- If you want to keep the aspect ratio you have to change **Width (pixels)** and **Height (pixels)** accordingly.
- If you increase the resolution of the image, you need to increase the width and height by the same scale to maintain the size of the image.

Download as PDF

Do the following:

1. In the menu select **Download as...** and **PDF**.
The dialog **PDF settings** appears.
2. Select the **Paper size** by scrolling the related drop down menu and clicking on the selected one.
3. You can increase or decrease the **Resolution (dots per inch)** by clicking on + or -. You can also type an exact value. The minimum value is 72 dpi, the maximum is 300 dpi.
4. Choose the **Orientation** by selecting the button **Portrait** or **Landscape**.
5. In the **Aspect ratio options** you can select:
 - **Keep current size** to insert the visualization into the PDF without changing its size. If the resulting PDF is smaller than the visualization it will be cropped.
 - **Fit to page, without keeping aspect ratio** will change height and width of the visualization to fill the entire page. Aspect ratio will change accordingly.
6. Click **Export** to start image creation.



To create a paper copy of the visualization you can print the PDF file.

Downloading visualizations on mobile devices

You can also download visualizations from your mobile device.



On Android devices, the download of a visualization is started in a new tab. If the server does not have a trusted certificate, you will see a security warning instead of a download prompt. When exporting from mobile devices, exporting to PDF is the default option.

Download as PDF

When downloading from mobile devices, PDF is the default option.

Do the following:

1. Tap on the visualization you want to download to zoom it.
2. Click and select **Export**.
Keep the default option **PDF** on the **Select type of file** drop-down menu.
3. Select the **Paper size** by scrolling the related drop down menu and clicking on the selected type.
4. Choose the **Orientation** by selecting **Portrait** or **Landscape**.
5. Click **Export** to start the creation of the PDF.
6. To download the PDF, click on the link **Click here to download your PDF file**.

Download as an image

Do the following:

1. Tap on the visualization you want to download to zoom it.
2. Click and select **Export**.
3. Open the drop-down **Select type of file** and click **PNG** or **JPEG** to select the image format you want.
4. Click **Export** to start image creation.
5. To download the image, click on the link **Click here to download your image file**.

Limitations

- Only the visible part of the visualization will be downloaded. For example, if you download a table that has scroll bars, you will not get the entire table. You will get an image that shows the scroll bars and the window area that they specify.
- The following type of objects cannot be downloaded:
 - Filter panes
 - Sheet titles
- You can download a visualization extension (custom object), if the visualization extension and the security rules for your installation are set up to allow it.

4.4 Downloading a story

You can download a story as a PowerPoint presentation or a PDF file. The story will be exported with the selection states used to create the story. Any selections applied while playing the story are ignored.



This feature is not available on mobile devices.

Downloading as a PowerPoint presentation

You can create a PowerPoint presentation from a story, using the data storytelling feature.

Do the following:

1. Open the story that you want to download.
2. Click *** and select **Download story as PowerPoint**.
The **PowerPoint settings** dialog appears.
3. If you want to change the default **Slide size** and **Resolution (dots per inch)**, make selections from the two drop-down lists.
4. The bottom line of the dialog shows the result of your settings.
5. Click **Export**.
When the PowerPoint presentation is ready, a link will appear in the dialog.
6. Click the link.
The presentation will be downloaded to the default download location for your browser.
7. Click **Cancel** to close the dialog.

Pre-configured PowerPoint slide sizes

By default, the exported presentation will have a slide size of 960 by 540 pixels with a widescreen aspect ratio of 16:9, and a resolution of 220 dpi (dots per inch).

You can choose from three pre-configured slide sizes and aspect ratios for your PowerPoint slides. A custom option is also available. These are the pre-configured slide sizes:

Pre-configured PowerPoint slide sizes

Pre-configured slide sizes	Aspect ratio	Width (pixels)	Height (pixels)
Standard (4:3)	4:3	960	720
Widescreen (16:9) (default)	16:9	960	540
Widescreen (16:10)	16:10	960	600

Three resolutions are available: 220, 150, and 96 dpi.

Setting custom slide height and width

You can set the dimensions of the exported slides to values other than the pre-configured values by selecting **Custom** from the **Slide size** drop-down list in the **PowerPoint settings** dialog. If you do this, the dialog changes to allow you to set slide width and height in pixels.

Do the following:

- Click + or - to increase or decrease the slide height or width. You can also type an exact value directly into the field.



To preserve the aspect ratio, make sure you adjust **Width (pixels)** and **Height (pixels)** accordingly.

Downloading as a PDF

You can download a story as a PDF using the data storytelling feature.

Do the following:

1. Open the story that you want to download.
2. Click **...** and select **Download story as PDF**.
The **PDF settings** dialog appears.
3. If you want to change the default **Paper size**, select it from the two drop-down lists.
4. To set a new image resolution, click + or - for **Resolution (dots per inch)** to increase or decrease the resolution accordingly. You can also type an exact value. The minimum value is 72 dpi, the maximum is 300 dpi.
5. Choose the **Orientation** by clicking **Portrait** or **Landscape**.
6. In the **Aspect ratio options**, you can select:
 - **Keep current size** to insert the visualization into the PDF without changing its size. If the resulting PDF is smaller than the visualization, it will be cropped.
 - **Fit to page** to change the height and width of the visualization to fill the entire page. Aspect ratio will change accordingly.
7. The bottom line of the dialog shows the result of your settings.
8. Click **Export**.
When the PDF presentation is ready, a link will appear in the dialog.
9. To download the PDF, click **Click here to download your PDF file**.
The presentation will be downloaded to the default download location for your browser.
10. Click **Cancel** to close the dialog.

4.5 Troubleshooting - Downloading

This section describes the problems that can occur when exporting data or stories in Qlik Sense.

Anonymous users cannot download app data

I want to let anonymous users download data, for example, download as image/pdf/data for visualizations.

Possible cause

There is no security rule granting anonymous users the right to download data.

Proposed action

You can enable download of data for anonymous users by creating a copy of the security rule ExportAppData and modifying the copy to only have resource.HasPrivilege("read") in **Conditions**. For more information, see [Security rules installed in Qlik Sense](#).

I cannot download a visualization as an image

I tried to download a visualization as an image, but the download failed.

Possible cause

The visualization you want to download is too large. The maximum size a download image can be is 2,000 by 2,000 pixels.

Proposed action

In the **Image settings** dialog, as you download, choose the **Custom** button and set the image size to 2,000 by 2,000 pixels.

I have blank characters in PDF files

I have blank characters in PDF files that were created when downloading stories with labels in Japanese, Korean, Simplified Chinese, or Traditional Chinese.

Possible cause

Downloading a story with these languages requires the appropriate font, or else the default Times New Roman font will be used. Times New Roman does not support Japanese, Korean, Simplified Chinese, or Traditional Chinese.

This only applies to text in title or paragraph objects in the story. It does not apply to embedded visualizations.

If ja, ja-JP, ko, ko-KR, zh-CN, zh-TW is declared in the CollationLocale setting for the application in the Data Load Editor, the downloading feature will use the first available font from the following ordered lists to generate the PDF:

Available fonts	
CollationLocale language	Font stack
Chinese Simplified	SimSun, SimHei, FangSong
Chinese Traditional	PMingLiU, MingLiU, Microsoft JhengHei, Microsoft JhengHei UI
Japanese	Meiryo, Meiryo UI, Yu Gothic, Yu Gothic UI, MS UI Gothic
Korean	Malgun Gothic, BatangChe

If a font from the above list is not found, characters in your story requiring a Chinese, Korean or Japanese font will not be printed. There will just be a blank space.

Proposed action

Install one of the fonts used to download PDF for the language that you are using. You cannot choose the font because the system will only use those in the table in the specified order.

Make sure that you use labels that are the same language that you set in the CollationLocale. If they differ, only common characters will be correctly rendered in the created report.

Right to left languages, like Arabic or Hebrew, are not currently supported on labels.

I downloaded a sheet but the data view tables changed back to visualizations

I downloaded an entire sheet, but all the visualizations I had changed to tables of data have changed back to the original visualizations.

Possible cause

The data view of a visualization cannot be downloaded..

Proposed action

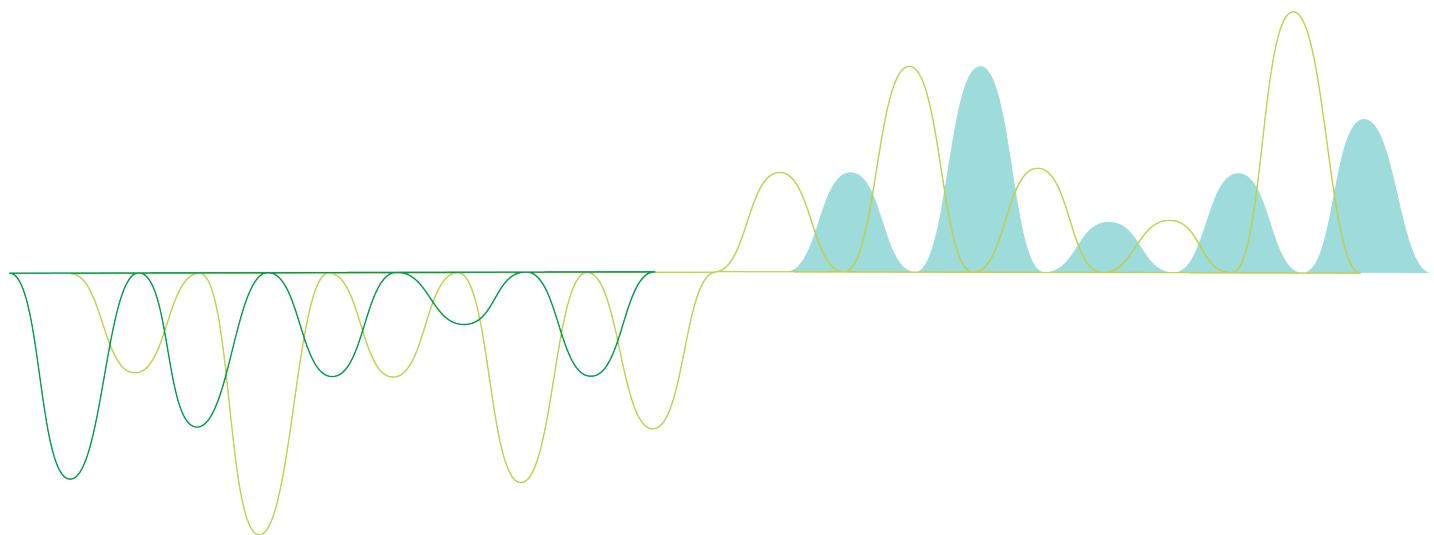
Download the data for each visualization using the download as data option.

Qlik Sense Desktop

Qlik Sense®

May 2023

Copyright © 1993-2023 QlikTech International AB. All rights reserved.



© 2023 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 About this document	5
2 Comparing versions of Qlik Sense	6
3 Installing Qlik Sense Desktop	7
3.1 System requirements	7
3.2 What is included in Qlik Sense Desktop	8
Desktop items and start menu items	8
Qlik Sense Desktop installation location	8
Examples, apps and logs	8
3.3 Obtaining the setup file	8
3.4 Installing Qlik Sense Desktop	8
Qlik Sense Desktop standard installation	8
Qlik Sense Desktop custom installation	9
3.5 Upgrading Qlik Sense Desktop	10
App migration	10
3.6 Qlik Sense Desktop ports	11
3.7 Qlik Sense Desktop storage	11
Default storage	11
Portable format	11
3.8 Modifying extension bundles installation	11
3.9 Repairing Qlik Sense Desktop	11
3.10 Uninstalling Qlik Sense Desktop	12
3.11 Silent installation of Qlik Sense Desktop	13
Syntax	13
Commands	13
Arguments	14
Examples	14
3 Downloading installation files	16
4 Starting Qlik Sense Desktop	17
4.1 Authenticating against Qlik Sense SaaS	17
Download and authenticate Qlik Sense Desktop	17
4.2 Authenticating against Qlik Sense Enterprise server	17
Retrieving a client authentication link from the Qlik Sense Enterprise hub	18
Receiving the client authentication link from your administrator	18
Receiving a hubs.ini file from your administrator	19
4.3 Logging out	19
4.4 Opening Qlik Sense Desktop in a web browser	19
5 The hub in Qlik Sense Desktop	20
5.1 A: Toolbar	20
5.2 B: Login information	21
5.3 C: Create new app	21
5.4 D: Work	21
5.5 E: Main area	21
5.6 F: Getting started	21
6 Opening an app in Qlik Sense Desktop	22
6.1 Opening an app from the hub	22

Contents

6.2 Opening an app from outside the hub	22
7 Moving an app from a Qlik Sense Desktop installation	23
7.1 Moving an app created in your current installation of Qlik Sense Desktop	23
7.2 Moving an app created in an older installation of Qlik Sense Desktop	23
7.3 Moving an app to another Qlik Sense platform	24
8 Troubleshooting - Qlik Sense Desktop	25
8.1 I cannot install Qlik Sense Desktop	25
8.2 I cannot find the log files for Qlik Sense Desktop	25
8.3 I cannot find the installation log files for Qlik Sense Desktop	25
8.4 My session expired	25
8.5 App thumbnails are missing in the hub	25
8.6 Images are missing when moving an app	26
8.7 I dropped my qvw file onto the hub, and now I cannot find the qvw file, and my app is not available in the hub	26
8.8 I cannot open an app	26
8.9 I cannot download data from an object	27

1 About this document

Read and learn about the areas where Qlik Sense Desktop differs from Qlik Sense.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

You find the online help, additional guides and much more at help.qlik.com/sense.

2 Comparing versions of Qlik Sense

There are some differences between Qlik Sense Desktop and Qlik Sense Enterprise:

- You can only run Qlik Sense Desktop on your local Windows computer - use of multiple screens or tablets, for example, is not supported.
- Apps cannot be published in Qlik Sense Desktop, and because of this, there is no support for streams.
- Security functionality is not supported in Qlik Sense Desktop.
- There is no autosave function in Qlik Sense Desktop – you have to save your work manually by clicking **Save** in the toolbar. The app is automatically saved when reloading the script.
- Duplicating apps is not supported in Qlik Sense Desktop.
- Qlik Sense Desktop will run in the language of your operating system and the language cannot be altered.
- Dynamic views are not supported in Qlik Sense Desktop.

3 Installing Qlik Sense Desktop

This section describes how to install Qlik Sense Desktop on your computer.

3.1 System requirements

To successfully install and run Qlik Sense Desktop, the requirements listed in this section must be fulfilled.

Qlik Sense Desktop requirements	
Operating system	Microsoft Windows 10 (64-bit version only) Microsoft Windows 11
Processors (CPUs)	Intel Core 2 Duo or higher recommended. Advanced Vector Extensions (AVX) support.
Memory	4 GB minimum (depending on data volumes, more may be required).  <i>Qlik Sense uses an in-memory analysis technology. The memory requirements are directly related to the amount of data being analyzed.</i>
Disk space	5.0 GB total required to install
.NET framework	4.8 or higher
Security	Local admin privileges needed to install.
Minimum screen resolution	<ul style="list-style-type: none">Desktops, laptops and tablets: 1024x768Small screens: 320x568
Browser support	<ul style="list-style-type: none">Microsoft EdgeGoogle ChromeMozilla Firefox  <i>By default, Qlik Sense Desktop runs in a window of its own. But you can also open it in a web browser.</i>  <i>Mozilla Firefox requires hardware acceleration, not supported in virtual environments.</i>

3.2 What is included in Qlik Sense Desktop

Desktop items and start menu items

After having completed your installation of Qlik Sense Desktop, a shortcut will be available from the desktop as well as from the **Start** menu (**Start > All Programs**):

- Shortcut to Qlik Sense Desktop
The hub is the starting point when you run Qlik Sense. That is where you find all the Qlik Sense apps that you have created.

Qlik Sense Desktop installation location

After having completed your installation of Qlik Sense Desktop, Qlik Sense Desktop is installed to *Users\{user}\AppData\Local\Programs\Qlik*.

You can also specify the installation location with the **Custom Installation** option.

Examples, apps and logs

There are some example files included in the installation of Qlik Sense Desktop. These are installed to *Users\{user}\Documents\Qlik\Examples*. For example, a number of extension code examples are installed to the *Extensions* sub-folder.

The installation of Qlik Sense Desktop also comes with example apps. These are installed to *Users\{user}\Documents\Qlik\Sense\Apps*.

In your installation of Qlik Sense Desktop, logs are found in *Users\{user}\Documents\Qlik\Sense\Log*.

3.3 Obtaining the setup file

The *Qlik_Sense/Desktop_setup.exe* file can be obtained from  [Product Downloads](#). Save it to a folder on your computer.

3.4 Installing Qlik Sense Desktop

You can either perform a standard installation, where installation and storage locations are set to default options, or specify the locations in a custom installation.

Qlik Sense Desktop standard installation

Do the following:

1. Double-click *Qlik_Sense/Desktop_setup.exe* to start the installation.
The welcome dialog is displayed.
2. Click **Install** if you want to perform a standard installation.
The **License agreement** dialog is displayed.

3. Read the license agreement, select **I accept the license agreement**, and click **Next**.



You also have the option to print the license agreement to a local printer.

4. On the **Ready to install** screen, optionally select to create a desktop shortcut. Click **Install**.
5. In the **Extension bundles** section, optionally select to install the extension bundles. Then, select which extension bundles you want to install from the list of those available for your Qlik Sense installation. You can always add or remove extension bundles from your Qlik Sense installation at a later moment. See: *Modifying extension bundles installation (page 11)*.
6. If you have chosen not to install the extension bundles, click **Install**. Otherwise, click **Next**.
7. If you are installing any of the extension bundles, accept the extension bundle license agreement. Then, click **Install**.
8. When the installation has completed, the **Installation summary** is displayed. Click **Finish** to close the **Installation summary**.

You have now successfully installed Qlik Sense Desktop on your computer.

Qlik Sense Desktop custom installation

Do the following:

1. Double-click on *Qlik_Sense/Desktop_setup.exe* to start the installation.
The welcome dialog is displayed.
2. Click **Custom Installation**.
The **License agreement** dialog is displayed.
3. Read the license agreement, select the **I accept the license agreement** check box, and click **Next**.



You also have the option to print the license agreement to a local printer.

4. Type or browse to the location where you want to install Qlik Sense Desktop and click **Next**.
5. Type or browse to the location where you want Qlik Sense Desktop to store app content, and click **Next**.
6. On the **Ready to install** screen, optionally select to create a desktop shortcut. Click **Install**.
7. In the **Extension bundles** section of the **Ready to install** screen, optionally select to create a desktop shortcut. Then, select which extension bundles you want to install from the list of those available for your Qlik Sense installation. You can always add or remove extension bundles from your Qlik Sense installation at a later moment. See: *Modifying extension bundles installation (page 11)*.

8. If you have chosen not to install the extension bundles, click **Install**. Otherwise, click **Next**.
9. If you are installing any of the extension bundles, accept the extension bundle license agreement. Then, click **Install**.
10. When the installation has completed, the **Installation summary** is displayed. Click **Finish** to close the **Installation summary**.
You have now successfully installed Qlik Sense Desktop on your computer, using customized installation and storage locations.



To silently install Qlik Sense Desktop, see Silent installation of Qlik Sense Desktop (page 13).

3.5 Upgrading Qlik Sense Desktop

The upgrade option is available when Qlik Sense has been previously installed and a newer version of the setup file *Qlik_Sense/Desktop_setup.exe* is executed.

Do the following:

1. Double-click on *Qlik_Sense/Desktop_setup.exe* to start the installation.
The welcome dialog is displayed.
2. Click **UPGRADE**.
The **License agreement** dialog is displayed.
3. Read the license agreement and then tick the **I accept the license agreement** check box (if this is the case) and click **Next**.



You also have the option to print the license agreement to a local printer.

The **Ready to install** dialog is displayed.

4. Click **Upgrade** to start the installation.
When the installation has completed, the **Installation summary** is displayed.
5. Click **Finish** to close the **Installation summary**

You have now successfully upgraded to a newer version of Qlik Sense Desktop.



To silently upgrade Qlik Sense Desktop, see Silent installation of Qlik Sense Desktop (page 13).

App migration

After upgrading Qlik Sense Desktop, apps need to be migrated to ensure compatibility. In the hub, app thumbnails are not displayed before the app is migrated.

Migration is performed automatically when you open an app for the first time after an upgrade. Before an app is migrated, a backup copy is created in *Users\{user}\Documents\Qlik\Sense\AppsBackup*. You can use the backup copy if you want to open the app in a previous version of Qlik Sense Desktop.

3.6 Qlik Sense Desktop ports

Qlik Sense Desktop uses port 4848 by default.

3.7 Qlik Sense Desktop storage

This section describes where the Qlik Sense apps are stored when running Qlik Sense Desktop.

Default storage

By default, Qlik Sense stores apps in the local file system under *C:\Users\{user}\Documents\Qlik\Sense*.

Portable format

A Qlik Sense app can be stored in the local file system in the proprietary *.qvf* format, which is a portable format.

A single app is stored as *<AppName>.qvf*, where *<AppName>* is the title of the app.

3.8 Modifying extension bundles installation

You can add or remove extension bundles from your Qlik Sense Desktop installation at any moment.

Do the following:

1. In **Control Panel**, open **Programs and Features**.
2. In the list of programs, double-click the extension bundle that you want to modify.
3. The Extension Bundle Setup Wizard opens. Click **Next**.
4. Select **Change**.
5. On the **Custom setup** screen, click on the bundle icon to select how to modify the bundle installation:
 - If the bundle is installed, select **Entire feature will be unavailable** to uninstall it.
 - If the bundle is not installed, select **Entire feature will be installed on local hard drive** to install it.
- Then, click **Next**.
6. Click **Change**.
7. Click **Finish** to close the Extension Bundle Setup Wizard.

3.9 Repairing Qlik Sense Desktop

The **Repair** option restores all missing files, shortcuts and registry values.

Do the following:

1. To start repairing the installation, open the **Control Panel** and select **Uninstall a program**. Then select Qlik Sense Desktop from the list of programs and click **Change**.

The Qlik Sense Desktop **Setup maintenance** dialog is displayed.



You can also perform this action by double-clicking the Qlik_Sense/Desktop_setup.exe file.

2. Click **REPAIR**.

The **Ready to repair** dialog is displayed.

3. Click **Repair**.

The repair starts and the progress is displayed.

4. When the repair process is finished, the **Repair summary** dialog is displayed to confirm that Qlik Sense Desktop has been repaired successfully.

5. Click **Finish**.

You have now successfully repaired your Qlik Sense Desktop installation.



To silently repair Qlik Sense Desktop, see Silent installation of Qlik Sense Desktop (page 13).

3.10 Uninstalling Qlik Sense Desktop

Do the following:

1. To start uninstalling Qlik Sense Desktop, open the **Control Panel** and select **Uninstall a program**. Then select Qlik Sense Desktop from the list of programs and click **Uninstall**.

A confirmation dialog is displayed asking if you are sure that you want to uninstall Qlik Sense Desktop from your computer.



*You can also uninstall Qlik Sense Desktop by double-clicking the Qlik_Sense/Desktop_setup.exe file and then selecting **Uninstall** from the maintenance dialog. In that case, you must use the correct version of the setup file when modifying your Qlik Sense Desktop installation, that is the same version used when installing Qlik Sense Desktop.*

2. Click **Uninstall**.

The uninstall process starts and the progress is displayed.

3. When the uninstall process is finished, the **Uninstall summary** dialog is displayed to confirm that Qlik Sense Desktop has been uninstalled successfully.

4. Click **Finish**.

You have now uninstalled Qlik Sense Desktop.



To silently uninstall Qlik Sense Desktop, see Silent installation of Qlik Sense Desktop (page 13).

3.11 Silent installation of Qlik Sense Desktop

When running a silent installation, Qlik Sense Desktop is installed with no dialogs at all. This means all features, properties and user selections have to be known before performing a silent installation. All setup options that are available in the user interface of the installer can be performed with silent operations.

Do the following:

1. Select **Start > All Programs > Accessories > Command Prompt**.
The **Command Prompt** window is displayed.
2. In the **Command Prompt** window, navigate to the folder containing the *Qlik_Sense/Desktop_setup.exe* file.
3. Enter *Qlik_Sense/Desktop_setup.exe* followed by the silent installation syntax preferred.

Syntax

```
Qlik_Sense/Desktop_setup.exe [-silent] [-uninstall] [-repair] {-log  
path\filename} {layout=path} {accepteula=1|0} {desktopshortcut=1|0}  
{installdir=path} {storagepath=path}  
{bundleinstall=dashboard|visualization}
```

`Qlik_Sense/Desktop_setup.exe -? or -h` Brings up the on-screen silent setup help.

Commands

-silent (or -s)		Command line-driven setup without UI (mandatory).
-uninstall		Uninstall the product silently. It must be used with -silent command.
-repair		Repair the product silently. It must be used with -silent command.
-log (or -l)	[log file name with path]	Log file directory and log file name.  <i>The user must have access to this directory.</i>
-layout	[destination directory]	Extracts files (including .msi files) to the destination directory.  <i>This argument should not be combined with other command line arguments.</i>

Arguments

Arguments are separated by space and presented in the form [Argument]=[Value]". The double quotes can normally be omitted but may be needed, for example, when a path contains spaces.

The default values are the same as those used in the setup user interface.

accepteula	1 0	Accepts the Qlik User License Agreement.  <i>This argument is mandatory when installing or upgrading, and you must accept the QULA to install successfully.</i>
desktopshortcut	1 0 (defaults to 1 on clean installs)	Installs desktop shortcuts.
installldir	[path to custom install directory]	Defines the directory if the default install directory will not be used. Default install directory: %LocalAppData% Programs.
storagepath	[path to custom directory used for apps, logs, extensions]	Defines the path to custom directory used for apps, logs, extensions. Must be supplied with the installldir parameter. Default storage path: C:\Users\{user}\Documents\Qlik\Sense.
bundleinstall	dashboard, visualization	Includes the dashboard and visualization bundles.

Examples

Install or upgrade Qlik Sense Desktop

The following example installs Qlik Sense Desktop or upgrades the current setup.

```
Qlik_Sense_Desktop_setup.exe -s accepteula=1
```

Install Qlik Sense Desktop with object bundles

The following example installs Qlik Sense Desktop, including the Dashboard and Visualization object bundles.

```
Qlik_Sense_Desktop_setup.exe -s accepteula=1  
bundleinstall=dashboard,visualization
```

Install Qlik Sense Desktop without desktop shortcuts

The following example installs Qlik Sense Desktop without desktop shortcuts. Additionally, the installation logs are created in a custom folder.

3 Installing Qlik Sense Desktop

```
Qlik_Sense/Desktop_setup.exe -s -l c:\mylogpath desktopshortcut=0  
accepteula=1
```

Repair Qlik Sense Desktop

The following example repairs an existing installation of Qlik Sense Desktop.

```
Qlik_Sense/Desktop_setup.exe -s -repair
```

Uninstall Qlik Sense Desktop

The following example uninstalls Qlik Sense Desktop.

```
Qlik_Sense/Desktop_setup.exe -s -uninstall
```

3 Downloading installation files

Qlik Cloud makes use of utilities and connectors that are installed on your own systems. These tools are available for administrators to download from the **Tools** page in Qlik Cloud or from the Qlik Download Site. You can find the site in Qlik Community under Support > Product News > Downloads.

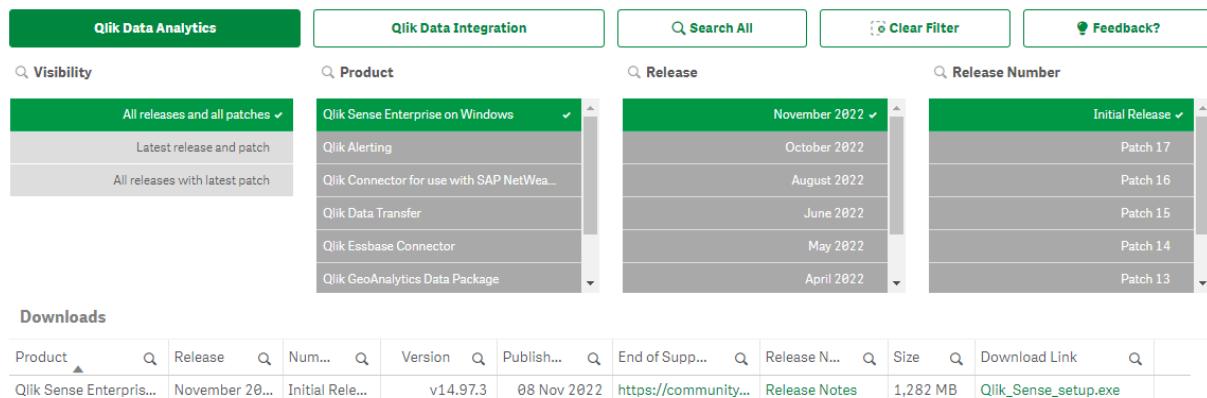
Do the following:

1. In Qlik Cloud, click your profile icon in the top right-hand corner, and then click **Profile settings**.
2. Under **Other**, click **Tools**.
3. Select a file to download.

Alternatively, do the following:

1. Go to  [Product Downloads](#).
2. Select **Qlik Data Analytics** or **Qlik Data Integration**, and then select your product.
3. Use the filters to narrow your list of possible downloads.
4. Click a link in the **Download Link** column in the **Download Assets** table to start the download.

*Example from the **Download Site** where the files have been filtered on product and release.*



The screenshot shows the Qlik Download Site interface. At the top, there are four tabs: "Qlik Data Analytics" (selected), "Qlik Data Integration", "Search All", and "Clear Filter". Below the tabs are four filter dropdowns: "Visibility" (set to "All releases and all patches"), "Product" (set to "Qlik Sense Enterprise on Windows"), "Release" (set to "November 2022"), and "Release Number" (set to "Initial Release"). A "Feedback?" button is also present. At the bottom, there is a table titled "Downloads" with columns: Product, Release, Num..., Version, Publish..., End of Supp..., Release N..., Size, and Download Link. One row is visible, showing "Qlik Sense Enterprise on Windows" with version "v14.97.3" and size "1,282 MB". The "Download Link" column contains a link: "Qlik_Sense_setup.exe".

4 Starting Qlik Sense Desktop

You start Qlik Sense Desktop from the **Start menu** under the Qlik Sense program group.

Before you can start using Qlik Sense Desktop, you need to authenticate yourself against a Qlik Sense Enterprise server. You need to have a working network connection to enable authentication.

After you have been authenticated once, internet access is not required to continue using Qlik Sense Desktop. However, you have to re-authenticate yourself if thirty days have passed since you last authenticated, if you have logged out, or if your administrator has revoked your user access for Qlik Sense Enterprise server. If you are using SAML authentication and close the browser, the session ends and the cookie is deleted so you must re-authenticate yourself to start a new session.



Qlik Sense Desktop runs in the language of your operating system and the language cannot be altered, unless you open and run Qlik Sense Desktop in a web browser.

4.1 Authenticating against Qlik Sense SaaS

You can authenticate your Qlik Sense Desktop client against Qlik Sense SaaS. To do so, you must generate an authentication link.

How to Authenticate Qlik Sense Desktop with Qlik Sense SaaS

Download and authenticate Qlik Sense Desktop

Do the following:

1. Open the hub. For more information about the hub, see *The hub in Qlik Sense Desktop (page 20)*.
2. Click your profile in the top right corner and select **Profile settings**.
3. Under **Other**, select **Tools**.
4. Under **Qlik Sense Desktop**, click **Download** to download Qlik Sense Desktop.
5. Install Qlik Sense Desktop.
6. Click **Authenticate** to add a server authentication link to your Qlik Sense Desktop installation. You can then click on that link in Qlik Sense Desktop to authenticate.

4.2 Authenticating against Qlik Sense Enterprise server

If you have user access for Qlik Sense Enterprise, you can authenticate against the Qlik Sense Enterprise server when you start Qlik Sense Desktop.

Before you can authenticate, the Qlik Sense Enterprise authentication link must be generated by your administrator in the Qlik Management Console.

Your Qlik Sense administrator will provide you with information about how you can receive the link using one of the following methods:

- Retrieving the client authentication link from your Qlik Sense Enterprise hub .
- Receiving the client authentication link from your administrator.
- Receiving a hubs.ini file from your administrator that contains the authentication link.

Qlik Sense Desktop must be installed on your computer before you start any of the following procedures.



Client authentication is not supported on test servers.

Retrieving a client authentication link from the Qlik Sense Enterprise hub

Do the following:

1. Start Qlik Sense Enterprise.
2. Click ******* in the top toolbar of the hub, and then click **Client authentication**.
3. A dialog box opens asking you to confirm that you want to open the authentication link using the Qlik Sense. Confirm the dialog.
Qlik Sense Desktop opens and a new authentication button for the enterprise server is added to the welcome page under **Authenticate against Qlik Sense Enterprise**.
4. Click the authentication button to log in. You may be asked to enter your Qlik Sense Enterprise credentials.
You are now authenticated and Qlik Sense Desktop opens.

After this, when you launch Qlik Sense Desktop, you can click the authentication button and log in using your Qlik Sense Enterprise credentials.

Receiving the client authentication link from your administrator

Do the following:

1. Click the authentication link provided by your Qlik Sense administrator.
If you cannot click the link, copy the link into your browser and press return.
If you are using Google Chrome, you must select the link option from the address bar that does not say **Google Search**.
2. A dialog box opens asking you to confirm that you want to open the authentication link using the Qlik Sense. Confirm the dialog.
Qlik Sense Desktop opens and the authentication link is added to the welcome page under **Authenticate against Qlik Sense Enterprise**.
3. Click the authentication link. You may be asked to enter your Qlik Sense Enterprise credentials.
You are now authenticated and Qlik Sense Desktop opens.

After this, when you launch Qlik Sense Desktop, you can click the authentication button and log in using your Qlik Sense Enterprise credentials.

Receiving a *hubs.ini* file from your administrator

Do the following:

1. Copy the *hubs.ini* file that was provided to you by your administrator to the following location:
`C:\Users\<user name>\Documents\Qlik\Sense\Hubs`.
The next time that you launch Qlik Sense Desktop, an authentication link will be added to the welcome page under **Authenticate against Qlik Sense Enterprise**.
2. Click on the authentication link. You may be asked to enter your Qlik Sense Enterprise credentials.
You are now authenticated and Qlik Sense Desktop opens.

After this, when you launch Qlik Sense Desktop, you can click the authentication button and log in using your Qlik Sense Enterprise credentials.

4.3 Logging out

You can choose to log out from Qlik Sense Desktop.

Do the following:

- Click  and then click  in the pop-up menu.

4.4 Opening Qlik Sense Desktop in a web browser

By default, Qlik Sense Desktop runs in a window of its own. But you can also open it in a web browser.

You must log in using Qlik Sense Desktop before you can open it in a web browser.

Do the following:

1. Start Qlik Sense Desktop from the start menu.
2. Open a (supported) web browser.
3. Type `http://localhost:4848/hub` in the browser address bar.

Qlik Sense Desktop opens in the web browser, showing the hub with all your apps.

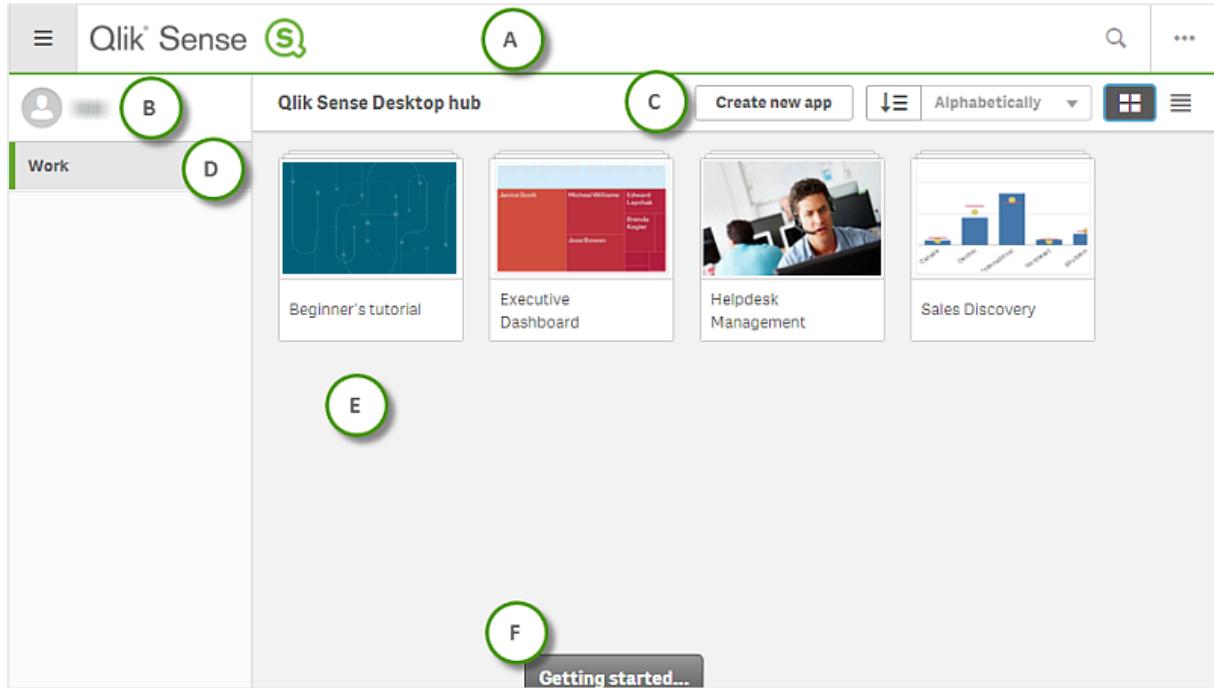


You can now alter the language of Qlik Sense Desktop.

5 The hub in Qlik Sense Desktop

When Qlik Sense Desktop starts up, you arrive at the hub. The hub is where you find all your apps. Click an app in the hub to open it in a separate tab.

The Qlik Sense Desktop hub with the Work view open.



5.1 A: Toolbar

The toolbar contains the global menu and other useful commands.

Toolbar options

UI item	Description
☰	Toggle on and off the navigation menu
🔍	You can search to easily find a specific app.
⋮	Global menu with options Dev Hub , Help and About . You can click the Client authentication link to enable using Qlik Sense Desktop.

5.2 B: Login information

Login information options

UI item	Description
	<p>Your login information is displayed when you are logged in. Depending on how the Qlik Sense system has been set up, you may have to log in by clicking the Log in button.</p> <p>When logged in, you can click  and then click  in the pop-up menu, to open the Profile dialog with the following information: User ID and User directory.</p>
Log out	Click  and then click  in the pop-up menu to log out.

5.3 C: Create new app

App options

UI item	Description
Create new app	Create a new app. The button is only available if you have permission to create apps.
	You can sort the apps alphabetically , by created date, or by published date, descending or ascending.
	Toggle between grid view and list view of the hub, depending on what kind of overview you prefer.

5.4 D: Work

Personal with all your own apps under **Work**.

5.5 E: Main area

All your apps are displayed in the main area.

5.6 F: Getting started

Go to the Qlik Sense Desktop webpage to learn more about how to get started with Qlik Sense Desktop.

6 Opening an app in Qlik Sense Desktop

With Qlik Sense Desktop, you can open apps from the hub, or from a file location.

6.1 Opening an app from the hub

You open an app from the hub by clicking it.

6.2 Opening an app from outside the hub

With Qlik Sense Desktop, you can open an app that is not in the hub. You can do this in two ways.

- Navigate to where you keep your app, and drag it to the hub.
- You can also copy the app using Ctrl+C, and then open it in the hub using Ctrl+O.



If you want the app to be part of your hub content, you can move the app file (with extension .qvf) to the app directory, typically <user>|Documents|Qlik|Sense|Apps.

7 Moving an app from a Qlik Sense Desktop installation

When you save an app that you have created in Qlik Sense Desktop, the images included in the app are bundled together with the rest of the contents of the app. This makes it easier to share an app with another person, or to move the app to another computer.

You can move an app that you have created in Qlik Sense Desktop, including its images, to another computer with Qlik Sense Desktop, for example by e-mailing the app.

7.1 Moving an app created in your current installation of Qlik Sense Desktop

If the app is created in your current installation of Qlik Sense Desktop, the images in the app are automatically bundled together with the rest of the contents of the app.

Do the following:

1. Create an app using Qlik Sense Desktop.
2. Save the app.
3. Locate the app on your hard drive. The default location is `<user>|Documents|Qlik|Sense|Apps`.
4. Copy the app to, for example, a portable device.
5. Paste the app in the *Apps* folder on another computer with Qlik Sense Desktop.

The app is now available from the hub.

7.2 Moving an app created in an older installation of Qlik Sense Desktop

If your app was created in an older version of Qlik Sense Desktop, prior to version 2.0, the images in the app are not bundled together with the rest of the contents of the app. You must open and save the app before the app is moved.

Do the following:

1. Open the app using the new version of Qlik Sense Desktop.
2. Make a change in the app.
3. Save the app.
Now the images included in the app are bundled together with the rest of the contents of the app.
4. Locate the app on your hard drive. The default location is `<user>|Documents|Qlik|Sense|Apps`.
5. Copy the app to, for example, a portable device.
6. Paste the app in the *Apps* folder on another computer with Qlik Sense Desktop.

The app is now available from the hub.

7.3 Moving an app to another Qlik Sense platform

You can move an app created in Qlik Sense Desktop to a different Qlik Sense environment, for example Qlik Sense Enterprise. To move the app you will have to copy it to your computer, and then upload the app to the Qlik Sense environment of your choosing.

The app will be available from the hub.



If you move a Qlik Sense Desktop app to a Qlik Sense environment, you need to handle the images separately. The same applies when moving an app that was created using Qlik Sense.

8 Troubleshooting - Qlik Sense Desktop

This section describes problems that are specific to Qlik Sense Desktop.

8.1 I cannot install Qlik Sense Desktop

Possible cause

The system requirements are not fulfilled, or you do not have local administrator privileges to install.

8.2 I cannot find the log files for Qlik Sense Desktop

The location of the log files in Qlik Sense Desktop depends on where you installed the application.

The default location is <user>|Documents|Qlik|Sense\Log.

8.3 I cannot find the installation log files for Qlik Sense Desktop

If you click **Cancel** during installation, or the installation did not complete successfully, you can find detailed information in the installation log located in your **temp** folder accessed with the environment variable %temp%.

8.4 My session expired

I was using Qlik Sense Desktop, logged in with my Qlik Sense Enterprise server credentials. Then I received the error message **Your session has expired** and now I cannot log in again.

Possible cause

You do not have user access or professional access on the Qlik Sense Enterprise server anymore.

Proposed action

Ask your administrator to give you user access or professional access.

8.5 App thumbnails are missing in the hub

Possible cause

You have upgraded Qlik Sense Desktop to a newer version. In the hub, app thumbnails are not displayed before the app is migrated.

Proposed action

Open the app. Migration is performed automatically when you open an app for the first time after an upgrade.

8.6 Images are missing when moving an app

Possible cause

You have upgraded Qlik Sense Desktop to a newer version, and the app you want to move was created in a Qlik Sense Desktop installation prior to version 2.0.

Proposed action

Before moving the app to another computer, open the app with your new version of Qlik Sense Desktop. Make a change and save the app.

The images included in the app are now bundled together with the rest of the contents of the app.

8.7 I dropped my qvw file onto the hub, and now I cannot find the qvw file, and my app is not available in the hub

Possible cause

You have dragged your QlikView document (qvw file) from a folder and dropped it onto the Qlik Sense Desktop hub, to open it as a Qlik Sense app.

When you make changes to the app and save the app, the following happens:

- The app is saved into the Qlik Sense format (qvf file) in the folder where your QlikView document (qvw file) was stored.
- Also, the QlikView document file (qvw) is removed from the folder and automatically converted into a backup file (qvw.backup) stored here: <user>|Documents|Qlik|Sense|AppsBackup.

Proposed action

If you want to open the backup file, you find it in this folder: <user>|Documents|Qlik|Sense|AppsBackup.

If the Qlik Sense app (qvf file) becomes stored in another folder than <user>|Documents|Qlik|Sense|Apps, move it to the Apps folder to make it available from the hub.

8.8 I cannot open an app

When I try to open an app this error message is displayed: **The object could not be saved.**

Possible cause

The total path length for backing up the app exceeds the maximum of 260 characters.

The total path includes the backup directory, the product version and the timestamp of the backup date and the app name: <user>|Documents|Qlik|Sense|AppsBackup|<app name>

Proposed action

Rename the qvf file to shorten the total path. This will enable opening the app.

8.9 I cannot download data from an object

When I try to download data from an object, using Qlik Sense Desktop, this error message is displayed: **The object could not be saved.**

Possible cause

The object's title length is too long. It exceeds the maximum of 174 characters.

Proposed action

Shorten the title of the object. This will enable download data.