

**Dr. Christopher Ilacqua, Dr. Henric Cronström,
James Richardson, Philip Hand,
Neeraj Kharpatte, Ferran Garcia Pagans**

Qlik Sense: Advanced Data Visualization for Your Organization

Learning Path

Create smart data visualizations and predictive analytics solutions



Packt

Qlik Sense: Advanced Data Visualization for Your Organization

**Create smart data visualizations and predictive
analytics solutions**

A course in three modules

Packt

BIRMINGHAM - MUMBAI

Qlik Sense: Advanced Data Visualization for Your Organization

Copyright © 2017 Packt Publishing

All rights reserved. No part of this course may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this course to ensure the accuracy of the information presented. However, the information contained in this course is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this course.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this course by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Published on: December 2017

Production reference: 1121217

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78899-492-7

www.packtpub.com

Credits

Authors

Dr. Christopher Ilacqua
Dr. Henric Cronström
James Richardson
Philip Hand
Neeraj Kharpatte
Ferran Garcia Pagans

Content Development Editor

Snehal Kolte

Production Coordinator

Aparna Bhagat

Reviewers

Arthur Lee
Steve Dark
Holly A. Kraig-Helton
Pablo Labbe Ibaceta
Stefan Stoichev
Gert Jan Feick
Miguel Ángel García

Preface

Qlik Sense is powerful and creative visual analytics software which allows users in discovering data, exploring the data, and digging out the meaningful insights in order to take profit making decisions for your business. This course begins with introducing you to features and functions of the most modern edition of Qlik Sense to build grip with the Qlik Sense application. The course will teach how you can administer the data architecture in Qlik Sense, thereby enabling you to customize your own Qlik Sense application for your business intelligence needs. The course also contains numerous recipes in order to overcome challenging situations while creating fully featured desktop applications in Qlik Sense. The course also explains how to combine Rattle and Qlik Sense Desktop for applying predictive analytics to their data for developing real-world interactive data applications.

On completion of this course, you would be self-sufficient to improve your data analysis and how you can apply predictive analytics to your datasets. Through this course, you would be able to create predictive models and data application allowing you to explore your data insights much deeper.

What this learning path covers

Module 1, Learning Qlik Sense: The Official Guide Second Edition, in this module, you will learn about Qlik Sense, Qlik's self-service visualization platform. Our aim is to help you get more from your data by applying Qlik Sense and its unique capabilities to your analytic needs. At the beginning of this module, we'll cover why Qlik chose to develop Qlik Sense, what data discovery is and can do, and the strategy and vision behind the product. Later, we'll address practical considerations, including the Qlik Sense application's life cycle, how to meet the needs of different types of users, how to develop and administer engaging Qlik Sense applications, data modeling and getting the most out of the QIX engine. The module concludes by outlining a series of example applications built using Qlik Sense, to address analysis needs in sales management, HR, T&E management, and demographics.

Module 2, Qlik Sense Cookbook, this module uncovers all the wonderful features of Qlik Sense product. It will help you to overcome the challenges faced in day to day Qlik Sense implementations. The solutions are discussed through simple and easy to understand recipes.

Module 3, Predictive Analytics using Rattle and Qlik Sense, the objective of this module is to introduce you to predictive analytics and data visualization by developing some example applications. We'll use R and Rattle to create the predictive model and Qlik Sense to create a data application that allows business users to explore their data. We use Rattle and Qlik Sense to avoid learning programming and focus on predictive analytics and data visualizations concepts.

What you need for this learning path

Module 1:

You will need a copy of Qlik Sense Desktop, which is available for free at <http://www.qlik.com/us/explore/products/sense/desktop>. After that, you just need some time and a good comfortable chair. Additionally, the sample application's examples and many others are available for you to explore live on <http://sense-demo.qlik.com/>. Please bookmark this link as additional demonstrations and examples are constantly being added and updated.

Module 2:

The user needs to install Qlik Sense Desktop version 2.1.1, which can be downloaded for free from: <http://www.qlik.com/try-or-buy/download-qlik-sense>. The user also needs to install Qlik Sense Server version 2.1.1 for the recipe titled Publishing a Qlik Sense application on Qlik Sense Server, given in Chapter 4, Managing Apps and User Interface. The Qlik Sense Server installer file can be obtained from: <http://www.qlik.com>.

One needs to login using the customer account credentials to get access to the files under Support | Customer Downloads. You also need to install the SAP connector for the recipe titled Extracting Data from custom Databases from Chapter 1, Getting Started with the Data. In order to work with the SAP connector, you will need to obtain a license from Qlik. A part of this recipe also makes use of QlikView which can be downloaded for free from: <http://www.qlik.com/try-or-buy/download-qlikview>.

Module 3:

To install our learning environment and complete the examples, you need a 64-bit computer:

- OS: Windows 7, Windows 8, or 8.1
- Processor: Intel Core2 Duo or higher
- Memory: 4 GB or more
- .NET Framework: 4.0
- Security: Local admin privileges to install R, Rattle, and Qlik Sense.

Who this learning path is for

This course is for anyone who wishes to understand and utilize the various new approaches to business intelligence actively in their business practice. Basics of business intelligence concepts will be helpful when picking up this course, but not mandatory.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course – what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at <http://www.packtpub.com>. If you purchased this course elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at <https://github.com/PacktPublishing/Qlik-Sense-Advanced-Data-Visualization-for-your-Organization>. We also have other code bundles from our rich catalog of books, videos, and courses available at <https://github.com/PacktPublishing/>. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

Module 1: Learning Qlik Sense: The Official Guide

Second Edition

| | |
|---|-----------|
| Chapter 1: Qlik Sense® and Data Discovery | 3 |
| Continuing disruption | 3 |
| Qlik Sense® and the QlikView.Next project | 5 |
| What is data discovery? | 6 |
| The Qlik® way | 11 |
| Data discovery—the evolution of BI | 13 |
| Summary | 14 |
| Chapter 2: Overview of a Qlik Sense® Application's Life Cycle | 15 |
| Overview of an application's life cycle | 15 |
| Summary | 22 |
| Chapter 3: Empowering Next Generation Data Discovery Consumers | 23 |
| Data discovery consumption requirements | 23 |
| Summary | 38 |
| Chapter 4: Contributing to Data Discovery | 39 |
| Realities of the data discovery contributor | 39 |
| Creating and sharing stories | 55 |
| Summary | 65 |
| Chapter 5: Authoring Engaging Applications | 67 |
| Preparations and requirements | 67 |
| Getting started with the app creation | 70 |
| Loading your data | 72 |

Table of Contents

| | |
|---|------------|
| The analysis interface—sheets and visualizations | 81 |
| The application library | 86 |
| Best practices in data visualization | 89 |
| Migrating applications from QlikView® to Qlik Sense® | 102 |
| Publishing your apps | 104 |
| Summary | 105 |
| Chapter 6: Building Qlik Sense® Data Models | 107 |
| The QIX engine | 107 |
| The Qlik Sense® data model | 108 |
| Structuring your data | 111 |
| The data model viewer | 117 |
| Summary | 119 |
| Chapter 7: Qlik Sense® Apps in the Cloud | 121 |
| Why use the cloud? | 121 |
| Using Qlik Sense® apps in the cloud | 122 |
| Using the Qlik DataMarket® content | 129 |
| Adding the QlikMarket® data | 129 |
| Summary | 133 |
| Chapter 8: Extending the Qlik® Analytic Platform | 135 |
| Qlik® Dev Hub | 137 |
| Developer community – Qlik Branch | 160 |
| Summary | 162 |
| Chapter 9: Administering Qlik Sense® | 163 |
| The Qlik Sense® architecture | 163 |
| Deployment and licensing | 167 |
| Management and monitoring | 171 |
| Security | 182 |
| Summary | 183 |
| Chapter 10: Sales Discovery | 185 |
| The business problem | 185 |
| Application features | 186 |
| Summary | 206 |
| Chapter 11: Human Resource Discovery | 207 |
| The business problem | 207 |
| Application features | 208 |
| How the application was developed | 215 |
| Summary | 219 |

Table of Contents

| | |
|---|------------|
| Chapter 12: Travel Expense Discovery | 221 |
| The business problem | 221 |
| Application features | 222 |
| Developing the application | 230 |
| Summary | 239 |
| Chapter 13: Demographic Data Discovery | 241 |
| Problem analysis | 241 |
| Application features | 243 |
| How the application was developed | 249 |
| Summary | 252 |

Module 2: Qlik Sense Cookbook

| | |
|--|------------|
| Chapter 1: Getting Started with the Data | 255 |
| Introduction | 255 |
| Extracting data from databases and data files | 256 |
| Extracting data from Web Files | 263 |
| Activating the Legacy Mode in Qlik Sense® desktop | 266 |
| Extracting data from custom databases | 267 |
| Invoking help while in the data load editor or the expression editor | 269 |
| Previewing data in the Data model viewer | 271 |
| Creating a Master Library from the Data model viewer | 274 |
| Using a Master Library in the Edit mode | 281 |
| Chapter 2: Visualizations | 285 |
| Introduction | 285 |
| Creating Snapshots | 286 |
| Creating and adding content to a story | 288 |
| Adding embedded sheets to the story | 293 |
| Highlighting the performance measure in a bar chart | 295 |
| Associating persistent colors to field values | 298 |
| Using the ColorMix1 function | 300 |
| Composition | 302 |
| Relationships | 305 |
| Comparison | 306 |
| Distribution | 308 |
| Structuring visualizations | 309 |

| | |
|--|------------|
| Chapter 3: Scripting | 313 |
| Introduction | 313 |
| Structuring the script | 314 |
| Efficiently debugging the script | 315 |
| Packaging the code in script files | 320 |
| How to use sub routines in Qlik Sense® | 322 |
| Optimizing the UI calculation speed | 326 |
| Optimizing the reload time of the application | 328 |
| Using a For Each loop to load data from multiple files | 330 |
| Using the Concat function to store multiple field values in a single cell | 332 |
| Chapter 4: Managing Apps and User Interface | 335 |
| Introduction | 336 |
| Publishing a Qlik Sense® application created in Qlik Sense® desktop | 336 |
| Creating private, approved, and community sheets | 337 |
| Publishing a Qlik Sense® application to Qlik Sense® cloud | 338 |
| Creating geo maps in Qlik Sense® | 342 |
| Reference lines in Sales versus Target gauge chart | 347 |
| Effectively using the KPI object in Qlik Sense® | 350 |
| Creating Tree Maps | 353 |
| Creating dimensionless bar charts in Qlik Sense® | 356 |
| Adding Reference Lines to trendline charts | 359 |
| Creating text and images | 361 |
| Applying limitations to charts | 364 |
| Adding thumbnails – a clear environment | 367 |
| Navigating many data points in a scatter chart | 369 |
| Chapter 5: Useful Functions | 373 |
| Introduction | 373 |
| Using an extended interval match to handle Slowly Changing Dimensions | 374 |
| Using the Previous() function to identify the latest record for a dimensional value | 380 |
| Using the NetworkDays() function to calculate the working days in a calendar month | 382 |
| Using the Concat() function to display a string of field values as a dimension | 386 |
| Using the Minstring() function to calculate the age of the oldest case in a queue | 388 |
| Using the Rangesum() function to plot cumulative figures in trendline charts | 390 |
| Using the Fractile() function to generate quartiles | 392 |

Table of Contents

| | |
|---|------------|
| Using the FirstSortedValue() function to identify the median in a quartile range | 394 |
| Using the Declare and Derive functions to generate Calendar fields | 396 |
| Setting up a moving annual total figure | 398 |
| Using the For Each loop to extract files from a folder | 402 |
| Using the Peek() function to create a currency Exchange Rate Calendar | 404 |
| Using the Peek() function to create a Trial Balance sheet | 408 |
| Chapter 6: Set Analysis | 413 |
| Introduction | 413 |
| Cracking the syntax for Set Analysis | 414 |
| Using flags in Set Analysis | 416 |
| Using the = sign with variables in Set Analysis | 419 |
| Point in time using Set Analysis | 421 |
| Using comparison sets in Set Analysis | 424 |
| Using embedded functions in Set Analysis | 427 |
| Creating a multi-measure expression in Set Analysis | 429 |
| Using search strings inside a set modifier | 431 |
| Capturing a list of field values using a concat() function in Set Analysis | 434 |
| Using the element functions P() and E() in Set Analysis | 435 |
| Chapter 7: Extensions in Qlik Sense® | 441 |
| Introduction | 441 |
| Creating an HTML visualization extension for Qlik Sense® | 442 |
| Defining a Properties panel in Qlik Sense® visualizations | 448 |
| Creating custom components within Qlik Sense® visualizations | 451 |
| Using data with extensions | 454 |
| Chapter 8: What's New in Version 2.1.1? | 463 |
| Introduction | 463 |
| Using the visual exploration capability in Qlik Sense® 2.1.1 | 464 |
| Defining variables in Qlik Sense® | 466 |
| Exporting stories to MS PowerPoint | 470 |
| Using the Qlik Dev Hub in Qlik Sense® 2.1.1 | 474 |
| Using Extension editor in Qlik Dev Hub | 481 |
| Using Qlik Dev Hub to generate mashups | 485 |
| Embedding Qlik Sense® application on a website using a single configurator | 490 |
| Using the Qlik DataMarket | 495 |
| Creating dynamic charts in Qlik Sense® | 498 |
| Using Smart Search | 501 |
| Using smart data load profiling | 503 |
| Conclusion | 507 |

| | |
|---|------------|
| Appendix | 509 |
| Keyboard shortcuts in Qlik Sense® Desktop | 509 |

Module 3: Predictive Analytics Using Rattle and Qlik Sense

| | |
|--|------------|
| Chapter 1: Getting Ready with Predictive Analytics | 513 |
| Analytics, predictive analytics, and data visualization | 514 |
| Purpose of the book | 516 |
| Introducing R, Rattle, and Qlik Sense Desktop | 517 |
| Installing the environment | 519 |
| Downloading and installing Rattle | 521 |
| Installing Qlik Sense Desktop | 523 |
| Exploring Qlik Sense Desktop | 526 |
| Further learning | 530 |
| Summary | 530 |
| Chapter 2: Preparing Your Data | 531 |
| Datasets, observations, and variables | 532 |
| Loading data | 534 |
| Transforming data | 538 |
| Cleaning up | 548 |
| Exporting data | 550 |
| Further learning | 550 |
| Summary | 550 |
| Chapter 3: Exploring and Understanding Your Data | 551 |
| Text summaries | 552 |
| Visualizing distributions | 558 |
| Correlations among input variables | 567 |
| Further learning | 569 |
| Summary | 569 |
| Chapter 4: Creating Your First Qlik Sense Application | 571 |
| Customer segmentation and customer buying behavior | 571 |
| Loading data and creating a data model | 572 |
| Creating a simple data app | 580 |
| Associative logic | 581 |
| Creating charts | 584 |
| Analyzing your data | 598 |
| Further learning | 600 |
| Summary | 601 |

Table of Contents

| | |
|--|------------|
| Chapter 5: Clustering and Other Unsupervised Learning Methods | 603 |
| Machine learning – unsupervised and supervised learning | 603 |
| Further learning | 621 |
| Summary | 622 |
| Chapter 6: Decision Trees and Other Supervised Learning Methods | 623 |
| Partitioning datasets and model optimization | 624 |
| Decision Tree Learning | 625 |
| Entropy and information gain | 626 |
| Underfitting and overfitting | 632 |
| Using a Decision Tree to classify credit risks | 633 |
| Ensemble classifiers | 646 |
| Other models | 653 |
| Further learning | 655 |
| Summary | 656 |
| Chapter 7: Model Evaluation | 657 |
| Cross-validation | 657 |
| Regression performance | 659 |
| Measuring the performance of classifiers | 661 |
| Further learning | 670 |
| Summary | 670 |
| Chapter 8: Visualizations, Data Applications, Dashboards, and Data Storytelling | 671 |
| Data visualization in Qlik Sense | 672 |
| Data analysis, data applications, and dashboards | 685 |
| Data storytelling with Qlik Sense | 692 |
| Further learning | 700 |
| Summary | 700 |
| Chapter 9: Developing a Complete Application | 703 |
| Understanding the bike rental problem | 704 |
| Exploring the data with Qlik Sense | 706 |
| Creating a Qlik Sense App to control the activity | 712 |
| Using Rattle to forecast the demand | 714 |
| Model evaluation | 721 |
| Further learning | 726 |
| Summary | 727 |
| Bibliography | 729 |
| Index | 731 |

Module 1

Learning Qlik Sense: The Official Guide Second Edition

*Get the most out of your Qlik Sense investment with the latest insight and
guidance direct from the Qlik Sense team*

1

Qlik Sense® and Data Discovery

In this chapter, we'll start getting to grips with what Qlik Sense offers by getting a better understanding of Qlik's background and how Qlik Sense was developed. In addition, we will examine the discovery-based approach to business intelligence that Qlik invented.

We will cover the following topics:

- Qlik's history in business intelligence and the evolution of data discovery
- The QlikView.Next project
- The Qlik philosophy and approach to data discovery
- The importance of the empowered user
- How a user really interacts with data
- The difference between traditional BI and data discovery

Continuing disruption

In the world of technology, there's a lot of talk about creating new products that disrupt existing markets, but very few organizations can say they've done it for real. Qlik is one of them.

In 2007, the **business intelligence (BI)** software market changed forever. Oracle bought Hyperion, SAP bought Business Objects, and IBM bought Cognos. The conventional wisdom was that BI would effectively cease to exist as a standalone market, subsumed into larger stacks of technology.

However, this wasn't the case. In fact, by 2007, a revolution was already well underway. The BI world was being fundamentally disrupted, challenged by the new approach pioneered by Qlik (then called QlikTech). The disruptive technology Qlik developed was called QlikView. To differentiate QlikView from the established BI products, Qlik began to call the new disruptive approach **Business Discovery**, later adopting **data discovery** as this term gained industry-wide adoption.

Surprisingly though, when it was launched in 1994, what became QlikView was not consciously targeted at the BI software market. Rather, its initial task was to help its customer understand which of a number of individual parts and manufacturing materials were used across the range of the complex machines it manufactured, and which parts were *not* associated with particular items (a critical point we'll explore later in this chapter and revisit throughout this book). The goal was to visualize the logical relations between the parts, materials, machines, and products. This origin led to an approach completely different from BI at the time, one in which all the associated data points are linked automatically, enabling discoveries to be made through free exploration of data.

As it became more widely used and deployed, it was evident that what QlikView was being used for was a new type of BI. QlikView's speed, usability, and relevance challenged the standard approach that was dominated by IT-deployed data reporting products, which are slow performing, hard to use, and built around models that struggle to keep up with the pace of modern business needs.

QlikView's intuitive visual user interface, patented associative data handling – running entirely in memory – and its capability to draw data together from disparate sources changed the landscape. Discovery-led BI is about giving people the power to interact with and explore data in a much more valuable way than the older, reporting-led BI incumbency could. This is massively compelling to people who need to quickly ask and answer questions based on data in order to learn and make decisions, and proved very compelling to people jaded with the way things had been done before. QlikView became very successful, dominating the market it pioneered.

So what did Qlik do then? Sit back and relax, proud of its disruptive chops, safe in the knowledge that it had recast an established market in its image? No. Far from it. Instead, Qlik took the decision to transform the BI market again with a new product.

Qlik Sense® and the QlikView.Next project

Qlik decided to design and develop a next generation data discovery platform. Developed under the project name QlikView.Next and launched as Qlik Sense, the product was anchored to five themes:

- **Gorgeous and genius:** Within this theme, Qlik focused on three product scenarios, with an overall emphasis on ad hoc analysis. The scenarios were that the product should be visually beautiful, support associative, comparative, and anticipatory analysis, and a seamless experience across all devices.
- **Mobility with agility:** This theme was about all users having access and the ability to answer new analytical questions as they arise in new situations and contexts when using a mobile device, with no difference between static and mobile experiences.
- **Compulsive collaboration:** Business intelligence and collaboration are inseparable; decision-making is, by nature, a collaborative activity. The intent was to build a product that could reside at the forefront of users' shared decision-making and give them the chance to communicate their insights through collaboration and storytelling.
- **The premier platform:** This theme was about enabling Qlik customers and partners to quickly and easily deliver apps and solutions that are perfectly relevant to their constituents. Within this theme, Qlik focused on four scenarios: data access, the development experience, expanding its ecosystem through broadened APIs, and offering a unified platform interface.
- **Enabling new enterprise:** With this theme, Qlik was focused on making capabilities such as security, reliability, and scalability available to all customers, not just the largest ones, and giving administrators and authors the same kind of gorgeous and genius experience other users get.

Making sense of modern business

You may say, "Well, that's all good, but it doesn't really tell me why this matters or why Qlik Sense is important."

To answer this, we have to think about what the focus of technology in our organizations has been in the recent past. For 25 years, most of our investment in IT has been on effectively improving reliability, using ERP or transactional applications to streamline processes, drive out inefficiencies, and deliver our products or services effectively. However, if most organizations, and particularly groups of competitors, are now operating at similar levels of procedural effectiveness, a key question arises, "What do we do differently to win?"

The answer lies in out-thinking our competitors through the use of data and analysis. This requires a shift of focus in both how we run our businesses and the IT world needed to do so. So far, analytics has too often been a poor cousin, something that happens afterwards on the edges, a tactical rather than strategic activity. That's no longer good enough. Businesses using data-driven decision-making perform measurably better than those that don't. When we can see (and measure) new things, we are driven to seek answers and thus, new ways of thinking and operating. Organizations that do not have analytics as a central part of their business activities will not thrive, or even survive, in the new reality.

Qlik Sense is about doing exactly that—freeing up the analytical skills of individuals in organizations, whatever their role. This book shows you how to make the most of that and alter how your organization uses information.

What is data discovery?

So we've already mentioned the new style of BI that Qlik pioneered, data discovery, a few times. In this section, we'll look at that in more detail.

Over the years, there have been many names of the different business intelligence methods and tools, such as:

- Executive Information Systems (EIS)
- Management Information Systems (MIS)
- Online Analytical Processing (OLAP)
- Decision Support Systems (DSS)
- Management reporting
- Ad hoc query and reporting

Do we really need an additional label for something that in principle is the same thing? The answer is yes.

There is a fundamental difference between older technologies and data discovery, and it is in the approach. Most of the preceding tools are oriented towards technology, but data discovery is not. Instead, data discovery is oriented towards people—towards the users who need the information in their daily work.

Most of the preceding tools were developed for a small, select number of decision-makers, but again, data discovery is not. Data discovery is for everyone.

Decisions are made at all levels in a company. Obviously, managers are decision-makers, but we sometimes forget that machine operators and receptionists are also decision-makers, albeit at a more local level. They also need information to make better decisions.

We, at Qlik, believe that information can change the world and that every user can contribute to this transformation. Everyone should easily be able to view data, navigate in data, and analyze data. Everyone should be able to experience that "a-ha" moment of discovery.

Data discovery is not just business intelligence. It is user-centric, dynamic, and empowering. And it is fun!

The empowered user

Since its founding in 1994, Qlik has believed that a business could improve its processes and product quality by empowering employees and encouraging them to engage in lifelong learning. And Qlik meant *all* employees — we saw everyone as a decision-maker, not just managers.

Although some things have changed since then, much remains the same. Users are still often in a situation where they are unable to analyze their data — data that they have the right to see, or should have the right to see, in order to do a good job. Rigid systems, technical limitations, and poor user interfaces are usually the culprits.

However, people's expectations of software have changed dramatically during the last decade. Applications from Google and Apple invite users to interact with simple, friendly interfaces. Search bars, Like buttons, and touchscreens have transformed the way people explore, consume, and share information. Today, people want the same ease of use from their business tools as they get from their consumer tools at home.

The current trends such as the consumerization of software, performance improvements of hardware, usability improvements of software, mobile devices, social networks, and so on just accelerate this change. All these trends are reshaping user behavior. Yesterday, a user was a passive end user, but the user of tomorrow will be both able and demanding. They will demand tools that are fast, flexible, and dynamic. They will demand tools that they can use themselves. The empowered user is here to stay.

Interaction with data

The classic picture of business intelligence is that the user has one or several questions, and that the data holds the answers. So the problem boils down to creating a tool where the user can enter their questions, and the tool can return the answers.

However, this picture is incorrect. The truth is that the user does not always know the question initially. Or rather, if the user knows the question, they often already know the answer. So, the first thing the tool should do is help the user find the questions.

Finding the questions is a process that involves exploring the data. It involves testing what you suspect but don't know for sure. It also involves discovering new facts. Further, it involves playing with data, turning it around, scrutinizing the facts, and formulating a possible question. You use your gut feeling as a source of ideas and use the data to refine the ideas into knowledge; or to discard the ideas, if facts show that the ideas are wrong. You need to be able to play with the data, to turn facts around and look at them from different angles before you can say that you understand the data, and you need to understand the data before you can talk smartly about it.

When you have found a relevant question, you also need to be able to conduct an analysis to get a well-founded answer to the question.

Finally, the process involves presenting the answer to the question to other people as a basis for a decision or an action. The tool must support the entire process of going from ignorance to insight.

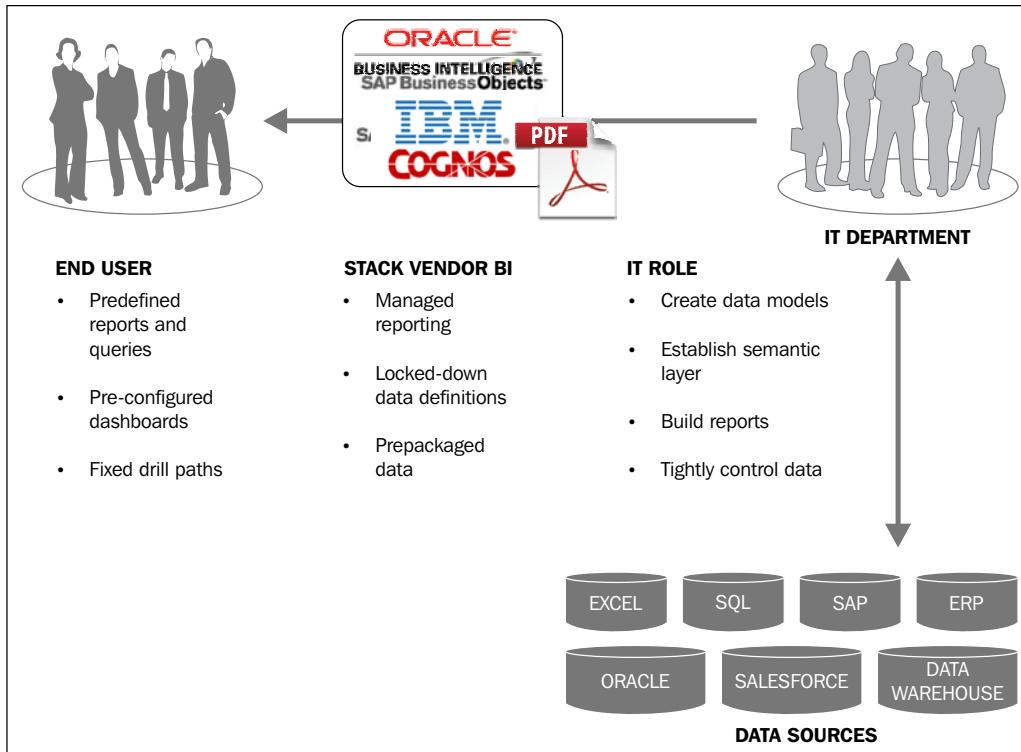
Hence, one major difference between data discovery and the more old-fashioned tools is that data discovery software supports the entire process – the process of coming from a blank mind, not knowing what you are looking for, all the way to attaining knowledge and taking action.

This is what data discovery is all about – helping you to prepare before you speak, act, or make a decision. It is the process of going from the darkness to the light, from the unknown to the known, from ignorance to insight. It is the process of going all the way from a blank mind to a substantiated argument.

Traditional business intelligence architecture

It is quite clear that users representing the business want the ability to ask and answer questions on their own so that they can make better decisions, but traditional business intelligence solutions aren't well suited for user demands. Instead, it is common that the systems are created in a report-centric manner, where governance and system demands set the goals, rather than user demands. The solutions often have preconfigured dashboards, fixed drill-down paths, predefined queries, predefined views, and very little flexibility.

With traditional BI, the creation of the business intelligence solution often belongs to the IT organization, which has to do the following: create data models, establish a semantic layer, build reports and dashboards, and protect and control the data. Often, the creation of business intelligence solutions is not driven by user demands. The following figure depicts the traditional BI architecture:



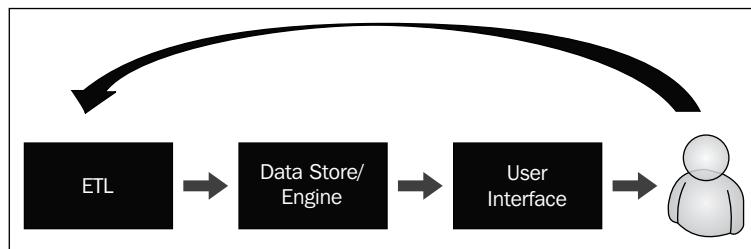
When analyzing data, you might want to set filters so that you can make selections, but with traditional tools, you often need to start at the top of predefined hierarchies. So, instead of selecting a customer directly, you may need to answer this question, "Which market does this customer belong to?", then the country the customer belongs to, and only then can you specify the customer.

Further, in the drill-down hierarchy, you are often limited to the choice of one or all. For example, you can look at either a single customer or all of them. The possibility of choosing two or three specific customers doesn't exist, unless this has been specifically predefined by the data model developer.

Numbers are often precalculated to ensure short response times, but this has a drawback that if the developer hasn't anticipated a specific calculation, the tool will not be able to show it.

Further, the architecture of the tool is often made in three layers—referred as the **stack**. The first layer is the **Extract, Transform, Load (ETL)** layer, or the data load layer. The second is the **Data Store / Engine** layer, and the third is the **User Interface (UI)** layer. The three layers are different pieces of software, sometimes delivered by different software vendors.

These three layers also demand different skill sets. Often, the ETL expert knows little or nothing about the UI software, and the UI expert knows little or nothing about the ETL.



The product stack in traditional BI

This architecture also leads to problems. When an application is built, the feedback comes from users trying to use the application. It could be that KPIs are incorrectly calculated or that dimensions or measures are missing. It could also mean that the user realizes that the initial requirements were incorrect or insufficient. The feedback could imply changes in the UI, or in the data model, or even in the ETL component.

This type of feedback is normal—it happens with all business intelligence tools. It only means that the development of applications is a process where you need to be agile and prepared. The expectation that you should be able to define an application completely and correctly prior to a prototype or an intermediate version is just unrealistic.

This is where the architecture leads to problems. In order for a project to be successful, you need to be able to implement change requests and new user demands with short notice, and this is extremely difficult since three different pieces of software and three different groups of people are involved. The distance between the user and the ETL component is just too great for efficient communication. Hence, the traditional architecture leads to a broken process.

The Qlik® way

Qlik has tried to solve all the drawbacks discussed in the preceding section by doing things differently.

First of all, you click and view. You don't need to formulate your question or tell the system more specifically what you want to look at. You just click, and by that, you say, "Tell me more about that...". Then you look at the calculation, KPI, or the field that might be interesting.

Color coding

The color coding defines the answer. Some things are associated with what you clicked on, and they remain white. Others that are not associated become gray. The color coding is for simplicity. The user quickly gets an overview and understands how things work.

Showing the excluded reveals the unexpected, creates insight, and creates new questions. Hence, the gray color is an important part of making the Qlik experience an associative one—a data dialog and an information interaction—rather than just a database query. Showing you that something is excluded when you didn't expect it means answering questions you didn't ask. This surprise creates new knowledge in a way that only a true data discovery platform can.

Freedom of data navigation

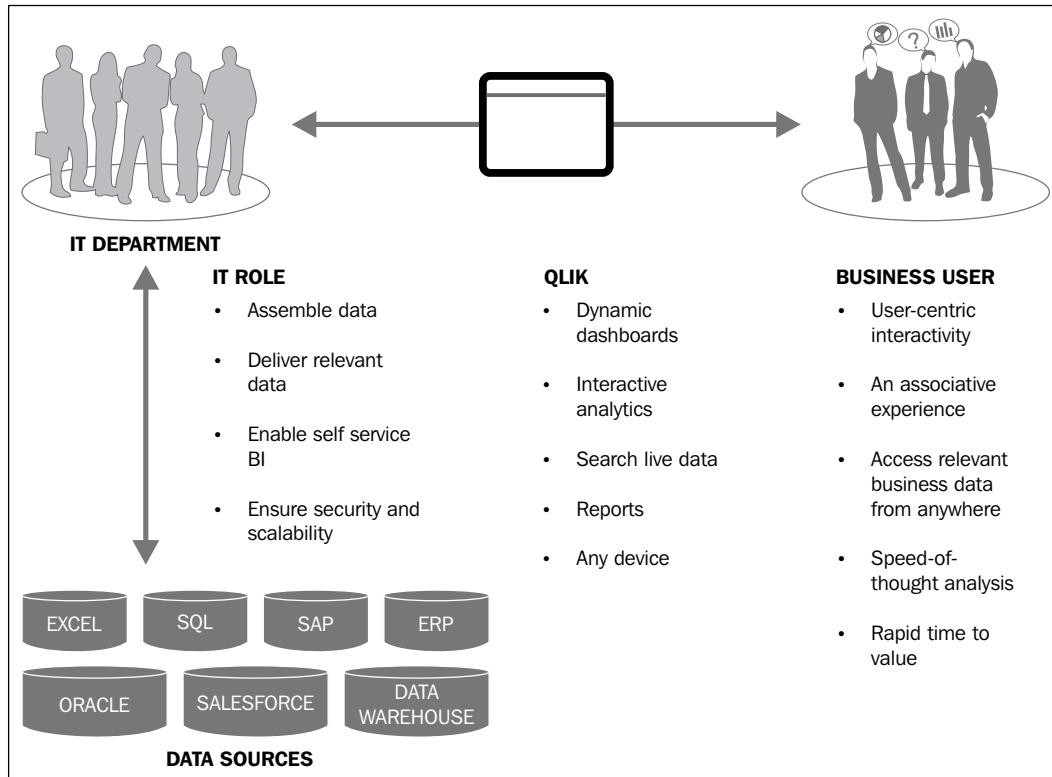
Via the associative experience, a user has total freedom to navigate through data and make any combination of selections. Any number of values can be selected. No drill-down paths need to be predefined. This allows the user to follow their own train of thought instead of someone else's. Start anywhere and just follow your intuition.

This total freedom when exploring data is really the core attribute of data discovery.

Calculation on demand

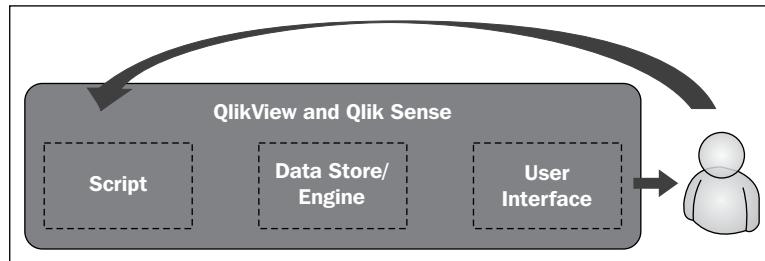
Further, no numbers need to be precalculated. Via the QIX engine, QlikView and Qlik Sense calculate everything on demand, usually in a fraction of a second. The short response time allows the user to *have a conversation* with the data, where one answer leads to the next question, which in turn leads to next, and so on. Only this way can you interact with data so that you learn from it.

The developer does not need to anticipate all questions that the user will pose. All they need to do is to create a logical, coherent data model, and Qlik Sense will be able to answer the question correctly:



The stack (ETL-Data Store / Engine-UI) is replaced by a single integrated environment. This makes it possible to develop applications in close cooperation with the users, and it can often be done by the users themselves. Feedback is implemented instantaneously and the changes can be evaluated just seconds later. This shortens the development cycle and ensures that the application meets the user demands much sooner than it would otherwise.

This stepwise implementation is crucial for the success of a business intelligence project. It is also the core of modern agile methodologies that are used in all types of software development.



With Qlik Sense, all BI stack functions are integrated into one tool

The development of business intelligence applications must be done as close to the user as possible to enable user feedback and short development cycles. It does not necessarily imply self-service capability, although it is good if this capability exists.

With the introduction of Qlik Sense, the groundbreaking work continues by enabling a new class of users who are highly mobile and require greater self-service capabilities. In Qlik Sense, the self-service capability has become a core feature. Users can define new graphs and visualizations that the app developer didn't think of. This functionality empowers the users even further.

With Qlik Sense, it has also become easier to share your findings and communicate them. This is something that is necessary in all environments where human interaction is important, which is pretty much everywhere.

Data discovery—the evolution of BI

Data discovery is the future of business intelligence. With data discovery, users pursue their own path to insight, make discoveries collaboratively, and can arrive at a whole new level of decision-making. Users are not limited to predefined paths or precalculated numbers. They do not need to formulate questions ahead of time. They can interact with data, find the questions, ask what they need to ask, and explore up, down, and sideways, rather than only drilling down in a predefined hierarchy.

Organizations might still need standardized reporting for many cases, but data discovery is the approach that ultimately fulfills the promise of business intelligence for everyone.

Data discovery is the inevitable consequence of demands from active users who want information from the ever-increasing amount of data. From the very beginning, the core of the Qlik philosophy was the empowered user. It affects both the view of how BI solutions should be developed and how the user interface of the tool should be designed.

In summary, data discovery is user-centric; it is BI for the empowered user. It means total freedom in how data is explored. It should be simple and have as few limitations as possible. Data discovery means a user-centric development process so that user feedback can be implemented instantaneously.

Summary

In this chapter, we looked at why Qlik developed Qlik Sense and at the ethos and value of data discovery in contrast to older forms of BI.

In the next chapter, we will look in detail at Qlik Sense itself and how its features help in meeting these requirements, beginning with the application life cycle.

2

Overview of a Qlik Sense® Application's Life Cycle

In the previous chapter, we outlined the evolving requirements driven by the market, and more importantly by business users seeking to help make better decisions within their organization. This chapter's goal is to highlight key features and benefits of Qlik Sense in meeting these requirements. There are thousands of features in the initial release of the software, and this chapter will serve as a guide to the major components, features, and benefits of Qlik Sense as you start exploring it.

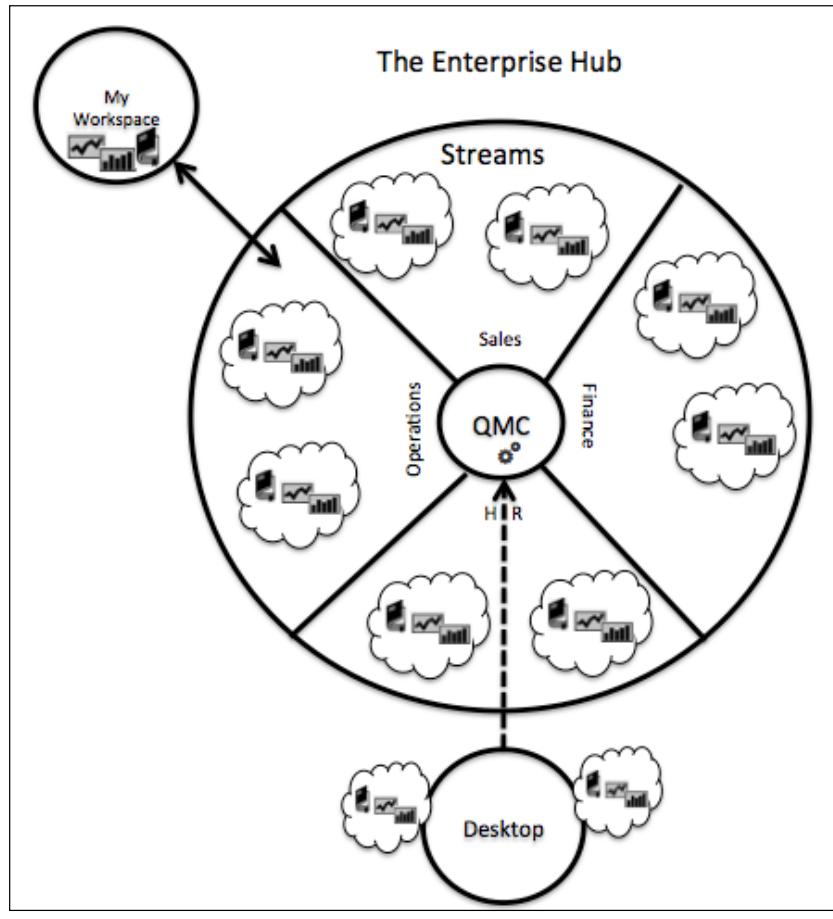
In this chapter, we'll cover the following topics:

- Overview of the hub
- Starting application authoring
- Components of a Qlik Sense application
- Sharing an application

Overview of an application's life cycle

As we begin our overview of a Qlik Sense application life cycle, it is best to start at the center of a Qlik Sense community collaboration, which is called the hub. The hub is made up of a number of streams that contain applications that are published by authors as well as users who can extend these applications by adding personal sheets and data stories. The **Qlik Sense Management Console (QMC)** governs this publishing through streams that have security rules. This approach provides the highly governed system that IT needs, while granting users the ability to explore information and share and collaborate on their findings.

Let's dig a bit deeper in each of these areas:



Overview of the Qlik Sense hub

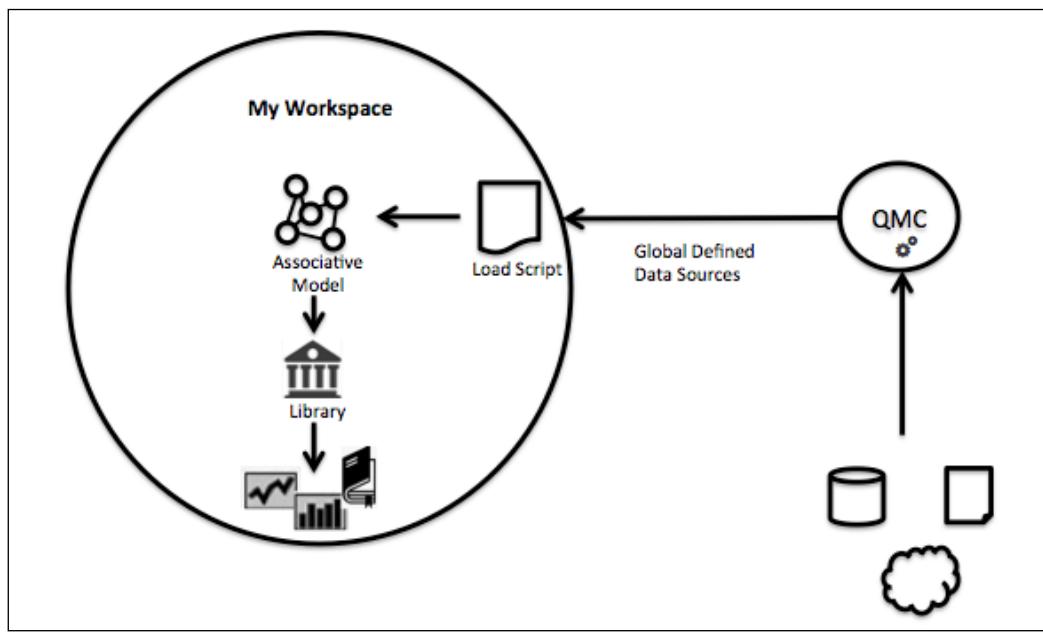
Starting application authoring

The need for a Qlik Sense application often starts with simple questions, such as these: Why are sales down in my region? What products are not selling well? Are there opportunities to sell additional products to existing customers? When a customer purchases a product, do they also purchase a companion product? These types of questions lead to the identification of the place to find this data. Qlik Sense provides two starting points that can be from either the Personal (Desktop) or Enterprise Edition. This chapter will focus primarily on the Qlik Sense Enterprise Edition and mention the differences in the Desktop Edition.

The hub is made up of two main parts. The first is **My Workspace**, which enables users to create new Qlik Sense applications. The second part comprises defined **Streams**, which contain published applications to be used and extended by users. Streams are defined in the QMC, which provides a broad range of security rules to meet organizational requirements. Once an application is completed, it can be published to an authorized stream by the author. When published, the application cannot be altered without republishing by the author. The Desktop Edition contains only a hub for the creation of Qlik Sense applications and the application author must send all artifacts of the application, which must include at least the Qlik Sense document (QVF) and extensions used in the development of the application. Once received by the administrator, these artifacts are imported through the QMC and then published.

What makes up a Qlik Sense® application?

Now, let's turn our attention to what components make up a Qlik Sense application. They are shown in the following diagram:



A Qlik Sense application component

Qlik Sense applications are made up of a number of components. Starting from the data source, these components include the following:

- **Global Defined Data Sources** are defined outside of Qlik Sense and managed by QMC.
- Based on these governed data sources, a **Load Script** is generated through or written, which transforms this data into Qlik's in-memory data model.
- Once the Qlik Sense data model is defined, the author can determine which fields will have the most value for users in the creation of private sheets for personal analysis. These fields will be used to create dimensions and real-time calculation expressions for measures.
- Additionally, fully defined charts for the most common views of information can be stored in the **Library**.
- Once the Library is defined, sheets (collections of objects), data stories, and bookmarks can be created.

All these components combine to create a dynamic baseline application to be explored by users.

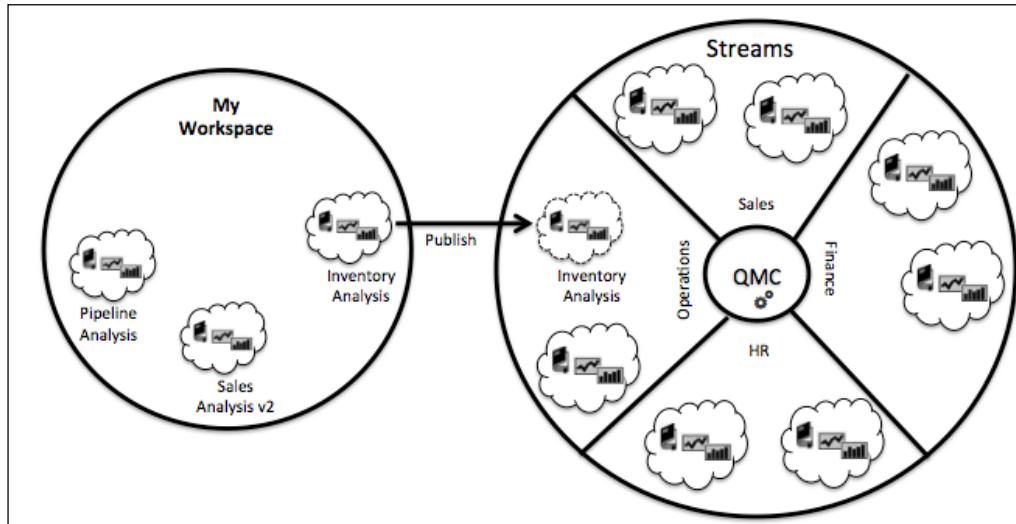
Sharing an application

Let's turn our attention to how an application is shared with the Qlik Sense community. There are two methods:

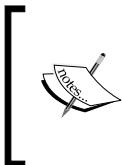
- Qlik Sense Enterprise application publication
- Qlik Cloud



Once a Qlik Sense application is complete, the author can share it by publishing it to a stream in the Qlik Sense hub. The publishing process can be accomplished by an administrator who is responsible for a stream and has publishing rights in the QMC. A Qlik Sense author notifies a stream administrator that a Qlik Sense application is ready for publishing. The stream administrator logs into the QMC, identifies the Qlik Sense application by name and author, and publishes the application in a stream.



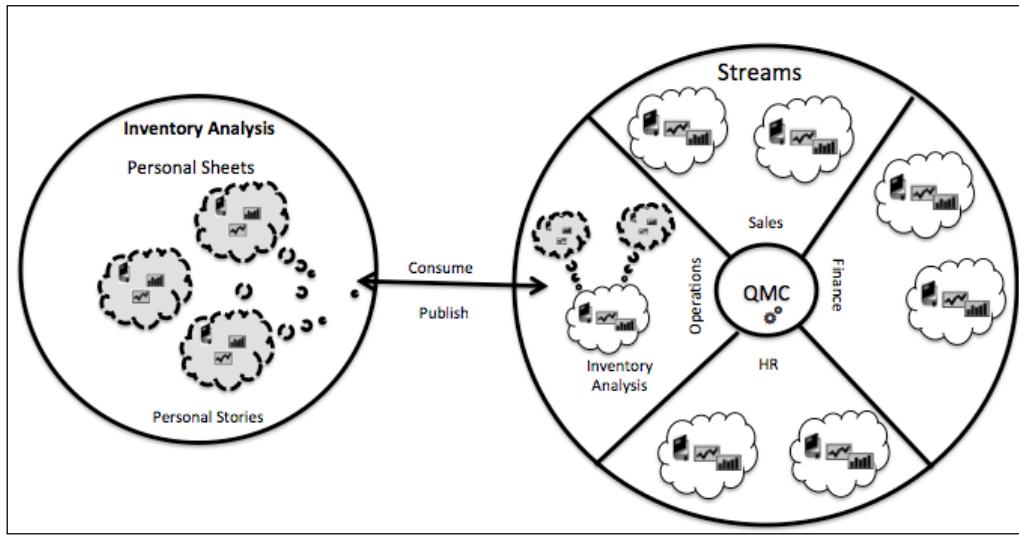
Qlik Sense Enterprise application publication



This method moves the application from the personal workspace, so a copy of the application should be made prior to publishing. Publishing and best practices for delegating publishing rights to an author in the QMC will be discussed in more detail in *Chapter 9, Administering Qlik Sense®*.

Overview of a Qlik Sense® Application's Life Cycle

Once an application is published to a stream, it is ready to be explored by users:



Qlik Sense Enterprise application consumption



Qlik Sense Cloud

Qlik provides a free and easy way for up to five people to create and share Qlik Sense visualizations in the cloud. There are two ways to take advantage of Qlik Cloud:

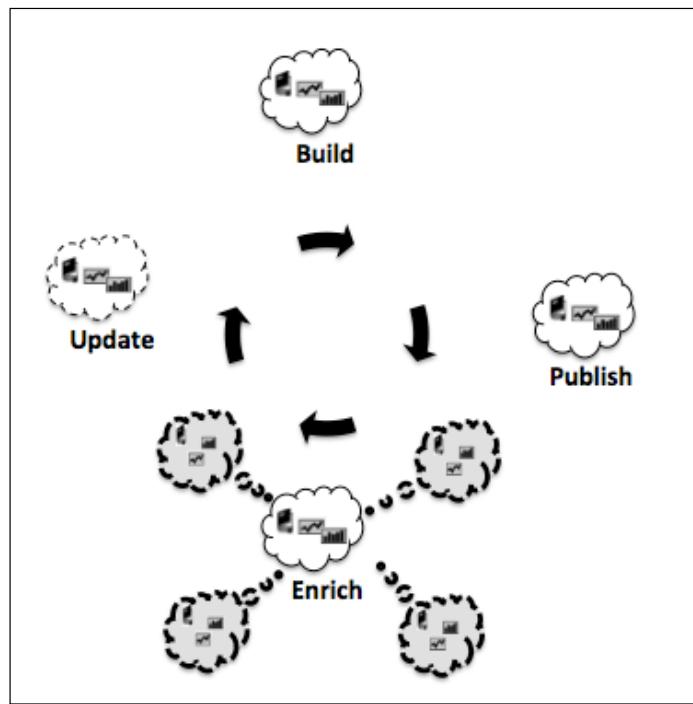
- Download Qlik Sense Desktop and create an application. Once created, register via the Qlik Sense client and upload your application.
- With the release of Qlik Sense Cloud, anyone can start their data exploration immediately by registering at <http://www.qlik.com/us/explore/products/qlikSenseCloud> and create your application and share it directly on the cloud.

Qlik Sense Cloud will be explored in more detail in *Chapter 7, Qlik Sense® Apps in the Cloud*.

Continuing the application's life cycle

One of the key features of a Qlik Sense application is its dynamic nature, which helps meet the broad requirements of data discovery. Users can explore the published sheets and data stories as well as create and share private sheets and stories based on the application library. The library allows for the creation of personal sheets and data stories in a controlled manner. As mentioned earlier, the library is a collection of dimensions, measures, and charts that are defined by the application author and cannot be modified once they are published to a stream but can be republished from the author's workspace. It allows a user to extend an application and share findings through personal sheets and data stories, while keeping consistent definitions across an organization.

Taking a step back, let's look at this new application model. A published Qlik Sense application is just at the beginning of its life cycle. Once published, the application can be expanded by the contributor within the stream using additional published sheets and stories based on the original application.



Qlik Sense application life cycle

Summary

Enterprise Qlik Sense applications are built based on governed data sources defined in the QMC. These data sources are transformed into a QIX Engine. Once the model is defined, key dimensions and measures are created within the library. This library will be used to create sheets. Next, the application is published to a stream within the hub for consumption. The application is then explored by users, and key findings can then be shared through bookmarks, private sheets, and data stories. These artifacts enrich the application and can be published back in the stream for collaboration between other members of the stream.

The next chapter will explore each of these capabilities in more detail and how they meet the needs of key stakeholders within your organization.

3

Empowering Next Generation Data Discovery Consumers

In the previous chapter, we outlined the Qlik Sense application life cycle, which provided an overview of the key Qlik Sense application components. This chapter's goal is to highlight key features in the context of the specific user requirements that Qlik has identified as defining a data discovery consumer.

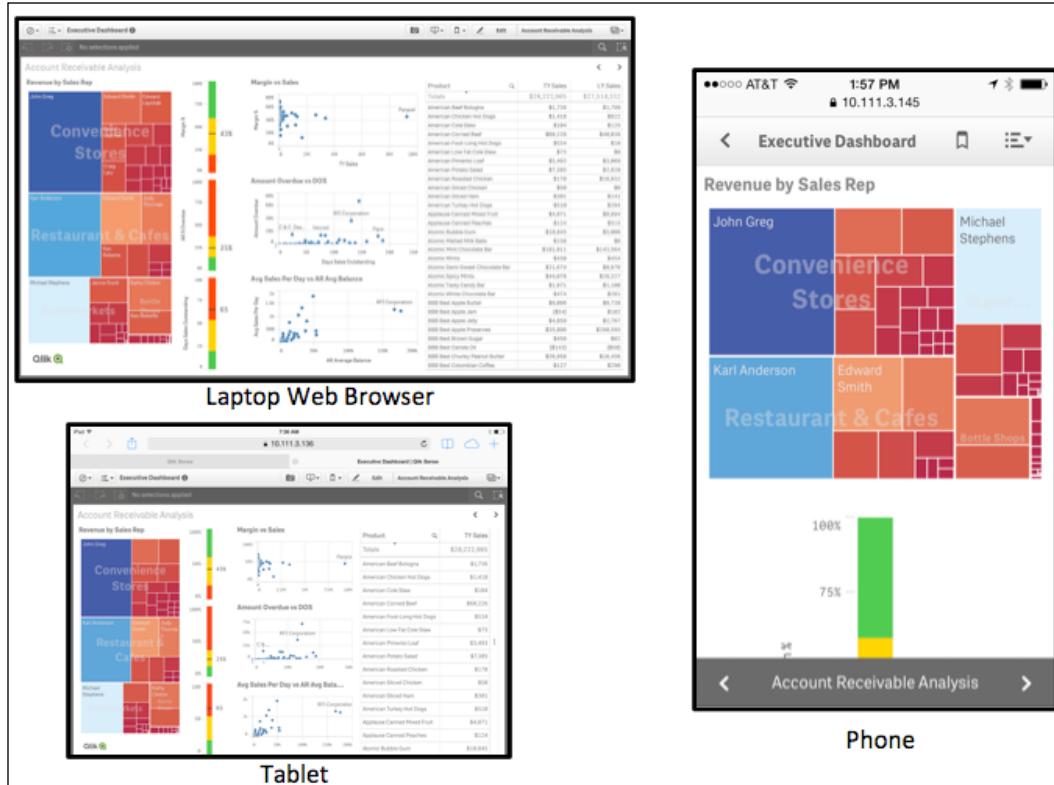
In this chapter, we'll cover the following topics:

- Data discovery consumption requirements
- The hub
- Navigating and leveraging the associative experience

Data discovery consumption requirements

People's expectations of what technology should be and how it should work have been set high with the rise of mobile and touch devices. The notion of a fixed, predictable desktop has changed to a dynamic, unpredictable virtual desktop that exists on whatever device you have access to at the moment. This can include traditional desktop PCs running Windows, laptops, ultrabooks powered by Microsoft Windows, Apple Mac OS, hybrid devices running Windows 8.x, tablets, Chromebooks, smartphones... the list goes on. This new environment requires new approaches in both architecture and application design that create smarter applications to meet the demands of a broader access from varying devices. Qlik Sense was designed from the ground up to meet the diversity of requirements that now exist in your enterprise when it comes to delivering data to support decision-making.

Qlik Sense adapts to very different devices, including a laptop via Microsoft Windows, Apple iPad Air, and finally, an iPhone 5s, to name a few. The following screenshot shows the diversity of consumption by users today:



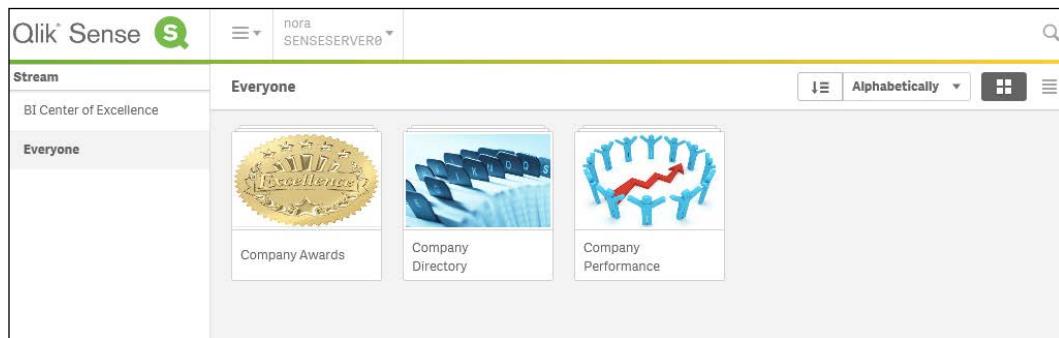
Diversity of consumption

The key thing is that these Qlik Sense screenshots could have been taken using any device on the market. Critically, and uniquely, Qlik Sense uses **Responsive Web Design (RWD)**, along with progressive disclosure to provide an optimal data discovery experience for users, whatever the form factor of the device. This is at the heart of the Qlik Sense architecture, the aim being to develop an app once and for it to be consumed/extended across any HTML5-compatible browser. For consistency and ease of illustration, the following key components of a Qlik Sense application will be illustrated from a laptop browser, but all these capabilities are available across tablets and smartphones as well. The following key Qlik Sense application components will be reviewed from a consumer perspective where the user has read-only access.

Introducing the hub

As noted in the application life cycle in the previous chapter, Qlik Sense provides a rich collaborative environment that is governed by the QMC through streams. Let's begin our review with the hub, which is the center of a data discovery community. The hub is a collection of streams, which contain Qlik Sense applications. Through the QMC, an administrator defines the streams, and Qlik Sense inherits security access to these streams and applications through security rules. Security rules are covered later in *Chapter 9, Administering Qlik Sense®*, and additional detailed examples are available in the Qlik Sense server user guide.

In this case, the consumer, let's call her Nora, has access to a default stream called **Everyone** as well as an administer-defined stream called **BI Center of Excellence**. The hub is designed for touch-friendly navigation (that is, it's designed to support selection and navigation using fingers!) between streams on the left-hand side of the display, searching and organizing the view in a number of sorted ways. Let's take a look at the hub:



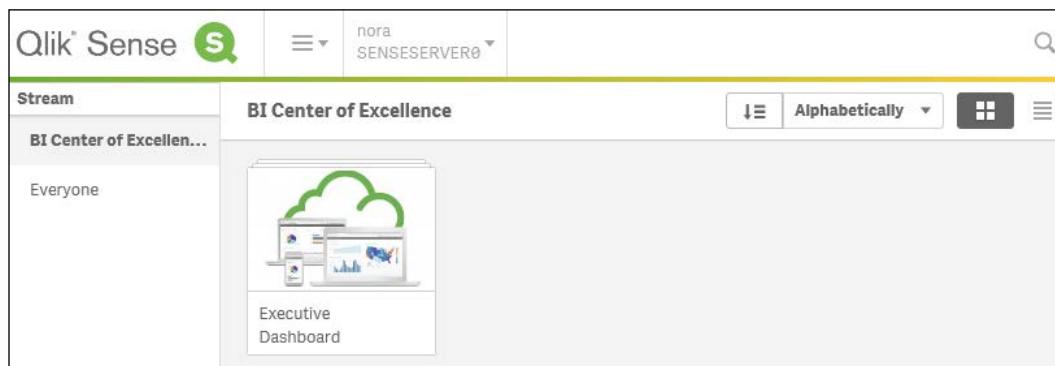
The screenshot shows the Qlik Sense hub interface. At the top, there is a header with the Qlik Sense logo, a search icon, and a dropdown menu set to 'nora' and 'SENSESERVER0'. Below the header, the left sidebar lists 'Stream' and two entries: 'BI Center of Excellence' and 'Everyone'. The main content area is titled 'Everyone' and displays three applications: 'Company Awards' (represented by a gold seal icon), 'Company Directory' (represented by a blue folder icon), and 'Company Performance' (represented by a brain with a red arrow icon). To the right of the applications are sorting and filtering controls: a dropdown set to 'Alphabetically' and a grid icon.

The hub

Now, let's turn our attention to streams.

Introducing streams

Streams are an organizing principle for applications as well as security. Qlik thinks of streams as work streams for information that can be categorized based on maturity with gradual expansion of access by audience, subject matter, or any other organizing principle. Nora has access to two streams, the **Everyone** stream, which is a public stream created during the server installation, and the **BI Center of Excellence** stream. The **BI Center of Excellence** stream contains a single application called **Executive Dashboard**. **Executive Dashboard** will be used to illustrate how Qlik Sense provides insights to business decision-makers.

A screenshot of the Qlik Sense web interface. At the top, there's a header with the Qlik Sense logo, a user dropdown for 'nora' on 'SENSESERVER0', and a search bar. Below the header, the main area shows a sidebar titled 'Stream' with 'BI Center of Excellence' selected. The main content area displays a card for the 'Executive Dashboard' application, which features a small icon of a dashboard with charts and a cloud, and the text 'Executive Dashboard'. There are also some filter and sorting options at the top of the content area.

The BI Center of Excellence stream

Let's start with the components of a Qlik Sense application.

Exploring the components of the application

Qlik Sense applications are made up of three main components, which include **sheets**, **bookmarks**, and **stories**. In the case of Nora, who has consumer access, each of these components have been defined by the application author and are identified by the label **Approved**. This label identifies these items as part of the core components created by the author and cannot be modified once published.

Sheets

Sheets are a core building block of Qlik Sense. Each sheet contains a collection of objects that are arranged to provide context for analysis on a particular subject. In this case, the sheets are contained in the application called **Executive Dashboard**. Note that sheets fall into the following three categories:

- **Base sheets:** These sheets are defined by the author of the application and become read only after publishing. They cannot be modified but can be duplicated as a private sheet for modification.

- **My sheets:** These sheets are similar to community sheets but are unpublished, so they can only be viewed by the author.
- **Community sheets:** These are private sheets that have been defined by a user and published to the hub. These can be defined based on duplicated approved sheets and/or new sheets that are assembled through the use of the application library. This will be discussed in detail in the next section, *Realties of data discovery power user*.

In the case of the **Executive Dashboard** application, there are five approved sheets that cover key application areas: **KPI Dashboard**, **Sales Analysis**, **Account Receivables Analysis**, **Inventory Analysis**, and **Product Analysis**. Each of these sheets provide a baseline for the consumer's analysis and exploration.

The screenshot shows the Qlik Sense application interface for the "Executive Dashboard". At the top, there is a header bar with the application name and some status information. Below the header, there is a main content area divided into several sections:

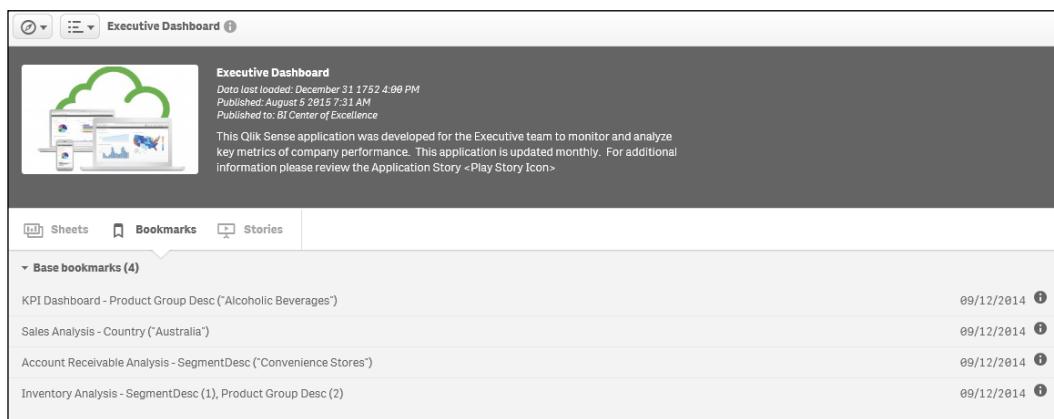
- Base sheets (5):** This section contains five thumbnails representing different analytical reports:
 - KPI Dashboard: A circular gauge chart.
 - Sales Analysis: A line chart showing sales trends over time.
 - Account Receivable: A bar chart with a color-coded legend.
 - Inventory Analysis: A bar chart showing inventory levels.
 - Product Analysis: A pie chart.
- My sheets (0):** This section shows a placeholder for creating new sheets, with a button labeled "Create new sheet".
- Community (2):** This section displays two community sheets:
 - Inventory Variance Analysis: A bar chart.
 - Pipeline Analysis: A chart showing pipeline stages.

The application overview

Additionally, there are two community sheets, **Pipeline Analysis** and **Inventory Variance Analysis**, which were created by users who have *contributor* access rights. This is a power capability of Qlik Sense that allows users to share key findings across applications. Like base sheets and my sheets, approved sheets are stored with the Qlik Sense application.

Bookmarks

Qlik Sense continues this popular feature, which was established in QlikView. Bookmarks allow a user to save the state of a sheet (their selections) so that they can be revisited at a future time, shared, and can be used to create data stories that allow users to combine key discoveries across many Qlik Sense sheets and add additional context through annotations. This example application contains four bookmarks as part of the published application.



The screenshot shows the 'Executive Dashboard' application interface. At the top, there's a header with a refresh icon, a dropdown menu, and the title 'Executive Dashboard'. Below the header is a banner featuring a cloud icon and text about the application's purpose: 'This Qlik Sense application was developed for the Executive team to monitor and analyze key metrics of company performance. This application is updated monthly. For additional information please review the Application Story <Play Story Icon>'.

Below the banner is a navigation bar with tabs: 'Sheets' (selected), 'Bookmarks' (highlighted with a blue border), and 'Stories'. Under the 'Bookmarks' tab, there's a section titled 'Base bookmarks (4)' containing four entries:

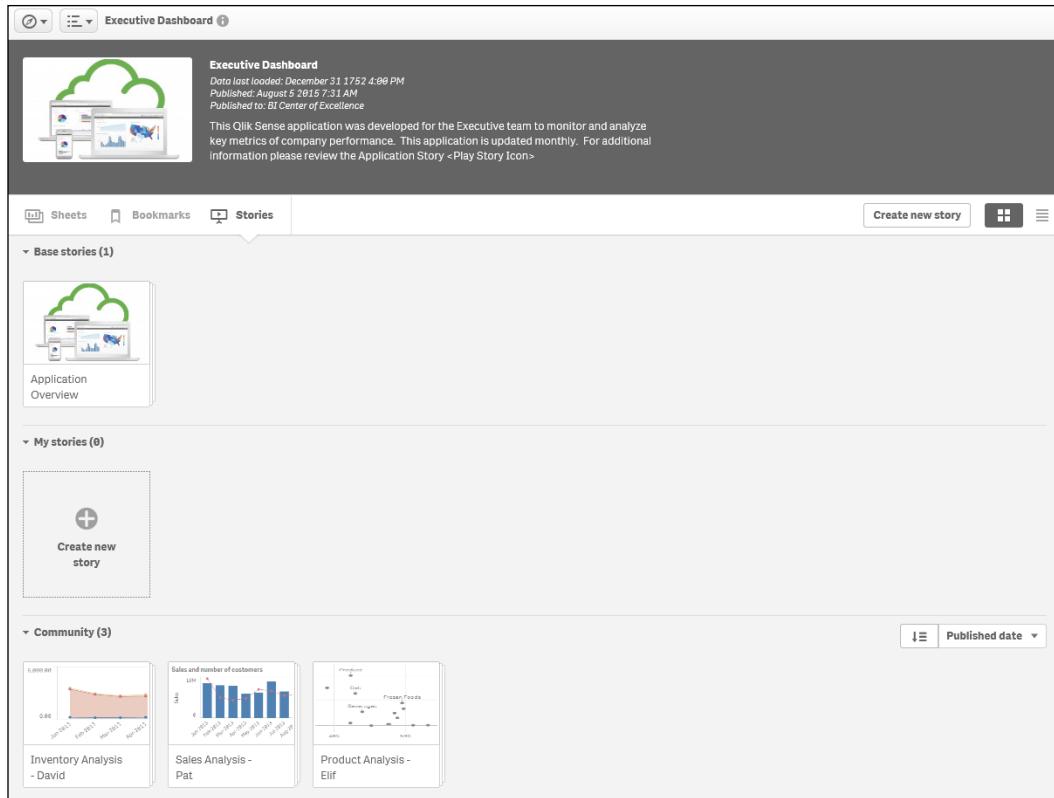
| Bookmark Name | Date Published |
|--|----------------|
| KPI Dashboard - Product Group Desc ("Alcoholic Beverages") | 09/12/2014 |
| Sales Analysis - Country ("Australia") | 09/12/2014 |
| Account Receivable Analysis - SegmentDesc ("Convenience Stores") | 09/12/2014 |
| Inventory Analysis - SegmentDesc (1), Product Group Desc (2) | 09/12/2014 |

Application bookmarks

The **Approved bookmarks** section includes the KPI dashboard for alcoholic beverages, Australia's sales analysis, convenience store account receivables, and convenience stores' inventory analysis for deli and alcoholic beverages. Qlik Sense consumers can create bookmarks to save key discoveries to view at a later date. Once interesting information is found, a user may wish to combine visualizations and add annotations that highlight any key discoveries. This leads us to our next topic, *Data storytelling*.

Data storytelling

Qlik Sense Stories are a collection of snapshots of key findings (visualization objects) that are assembled to share insights with others in an organization. Snapshots are a graphical representation of the state of visualizations at a certain point in time and are stored in the story media library. Although snapshots are static, they contain embedded bookmarks back in the source sheet, which enables users (or people who want to debate the detail of a narrative) to continue the exploration with live data from the point at which the snapshot was taken. Like sheets and bookmarks, base stories (published with the application by the author) and community stories (published by users who have contributor rights) can be seen.

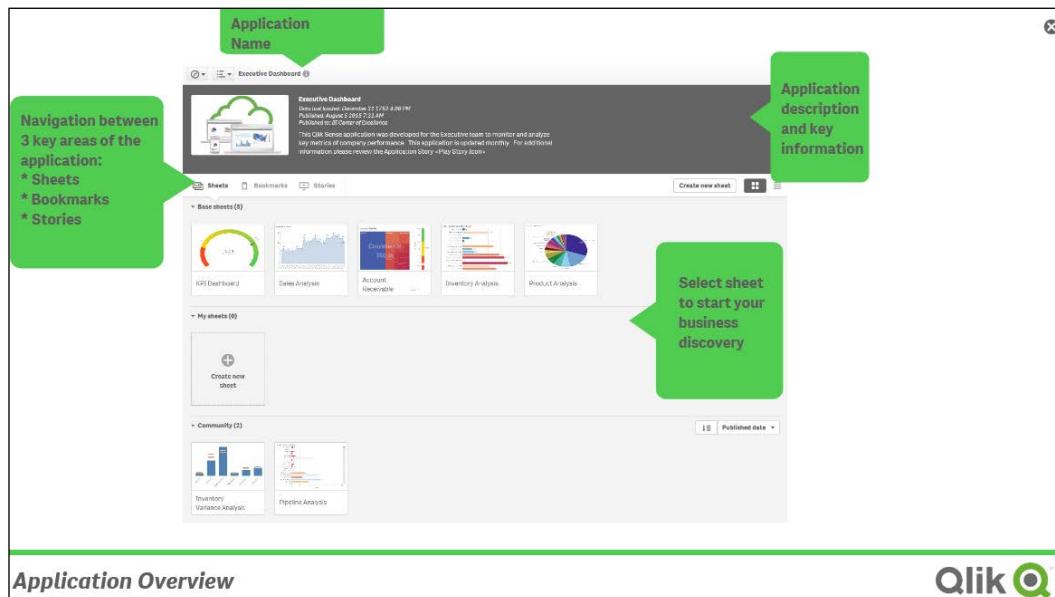


Application stories

The **Executive Dashboard** application, shown in the preceding screenshot, contains four stories available for Nora to review. Community stories were published by Elif, David, and Pat highlighting product analysis, inventory analysis, and sales analysis, respectively. Additionally, there is an approved story named **Application Overview**, which was published as part of the application by the author to outline the goals and use of the application. It is a recommended best practice for application authors to include a story to spur the adoption of an application within the user community. This topic leads us to our next topic, *Navigating and leveraging the associative experience*, in which we will use the **Application Overview** story to provide an overview of the application.

Navigating and leveraging the associative experience

As mentioned earlier, Qlik's intent in building Qlik Sense was to create a user experience that provides a natural and intuitive way to explore data and share key findings. To facilitate our discussion, we will refer to the **Application Overview** story. When selecting an application from the hub, Nora is provided with an application overview. This displays the application name, a short description, and a published date and time that provides key context for the timelines of the information.

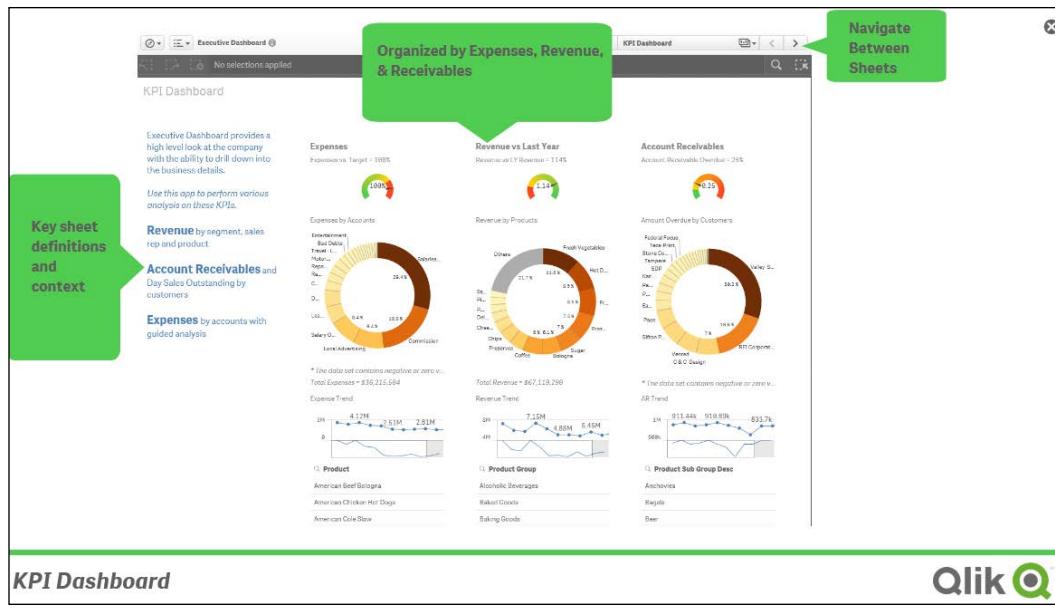


The Executive Dashboard overview

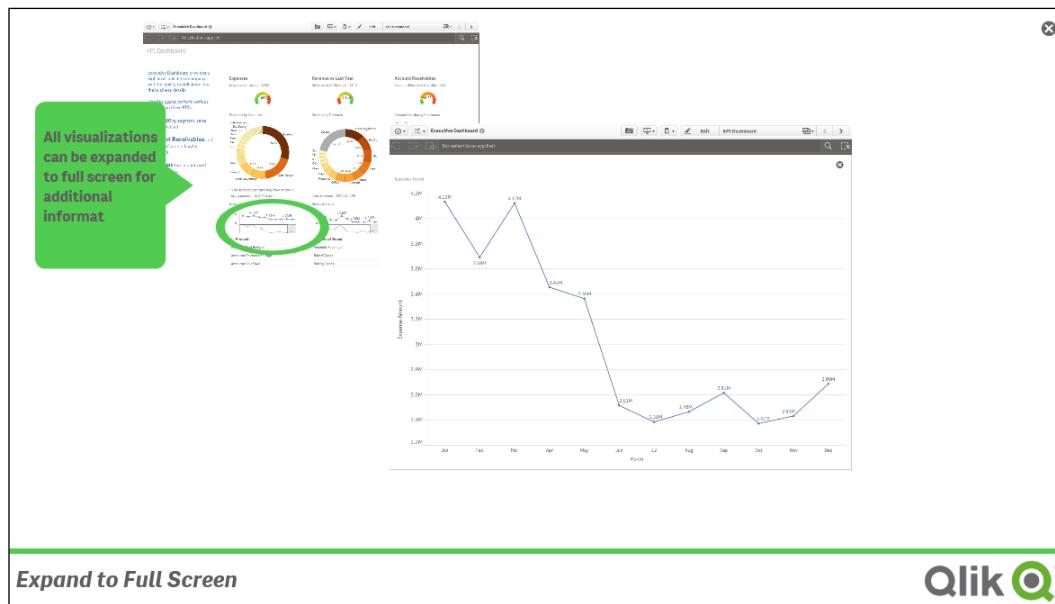
Navigation

Additionally, there are three key areas to explore in a Qlik Sense application; they include sheets (highlighted), bookmarks, and finally, stories, which were discussed earlier. This application contains both approved sheets (developed by the application's author) and community sheets that are the results of contributors who have published private sheets they wish to share with the community. This process will be discussed in detail in the next section.

Now, let's open the first sheet named **KPI Dashboard**. As discussed earlier, sheets are an amalgamation of smart objects that display information based on the amount of space available. In **KPI Dashboard**, we can see that the sheet is divided into three key areas: **Expenses**, **Revenue vs Last Year**, and **Accounts Receivables**:



Each of these objects can be used as a filter to see data association and just as importantly, to see nonassociated data (informally known as "The Power of Gray" based on its default coloring that users of QlikView have enjoyed for years). Additionally, each of these objects can be expanded to fullscreen, as shown in the next screenshot. The expense sparkline chart can be expanded to fullscreen to reveal additional data points and trends. This also facilitates viewing and selections on mobile devices, where screen real estate is limited.



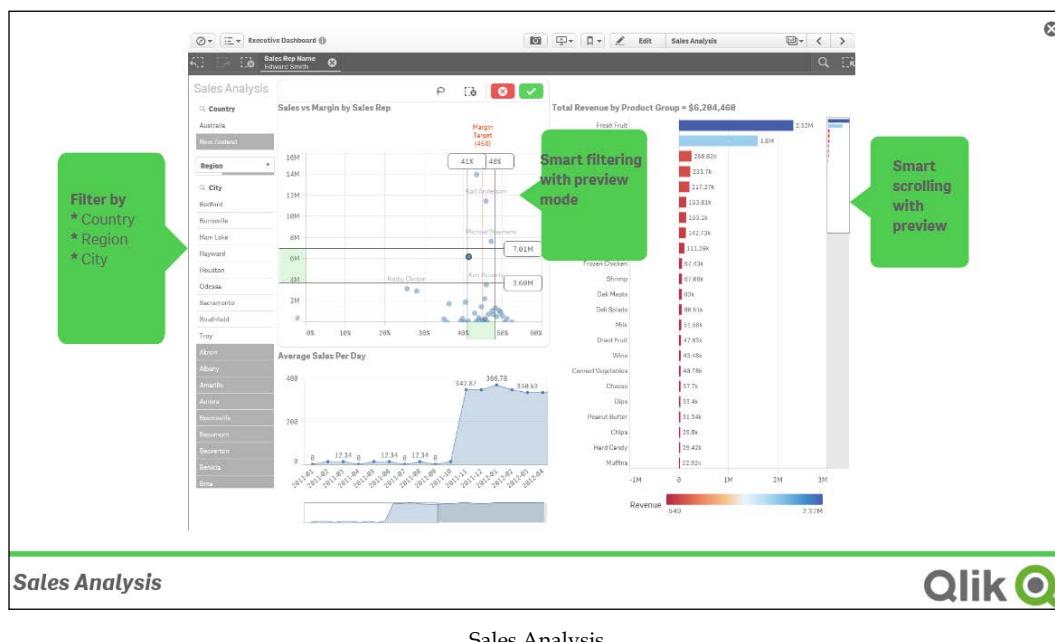
Expand to Full Screen

Smart visualizations

As we review the **Sales Analysis** sheet, there are a number of innovative features that highlight the capabilities of Qlik Sense. First, let's review the sales margin versus sales revenue scatter chart. What makes this chart smart is how Nora interacts with it.

As mentioned earlier, Qlik Sense was developed for mobile devices, which implies touch interaction. In this case, the scatter chart supports multitouch selections on both the axes. In this example, Nora has selected to highlight the performance of sales representatives who have margins between 41 to 48 percent and sales between 3.69 million to 7.01 million. Additionally, these selections are in preview mode, which allows Nora to see the impact of these selections before confirming and moving on to the next phase of her discovery.

A second area to highlight is the use of smart scrolling noted in both the **Average Sales Per Day** area and the **Total Revenue by Product Group** horizontal chart. The scroll bars use thumbnails of the chart so that Nora can easily navigate to the key area for review. Additionally, scroll bars appear after the chart has reduced its size to a point where the entire dataset can no longer be shown in the allocated space within the sheet. This allows Nora to enter the numbers in a range selection within any chart, for example, the scatter – you can type in the exact number for the range filter. Also, you can move the filter range keeping the range as you scroll along the *x* or *y* axis.

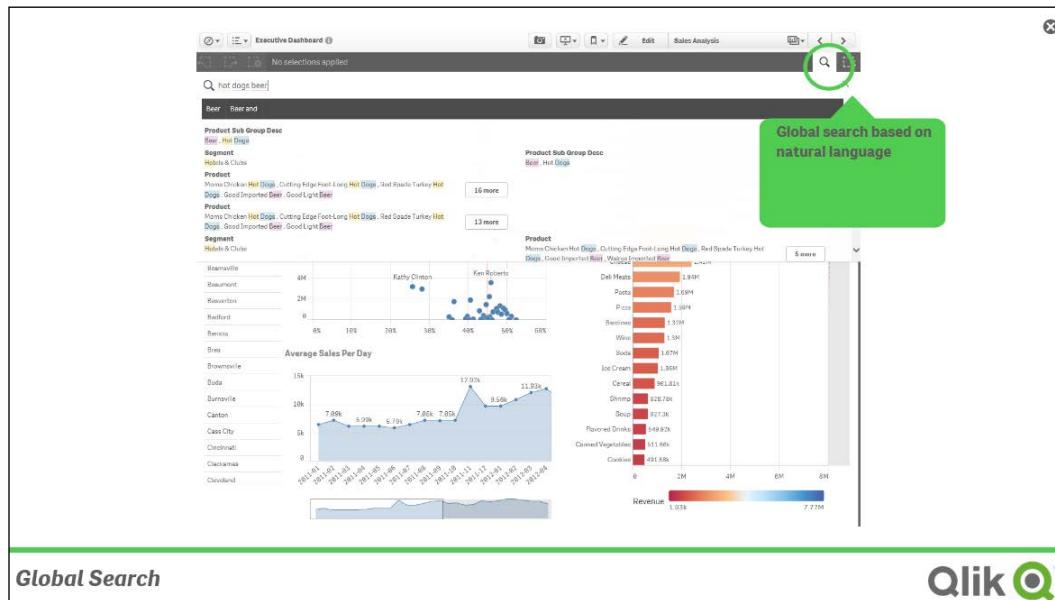


Sales Analysis



Global search

Selections and filtering can also be accomplished through the Qlik Sense global search capabilities. This allows for contextual search to narrow down the search criteria without restarting the search, like other search engines. Using the power of the associative engine, Nora can type various products to preview their impact on revenue and any association between these products. In this case, the search was conducted on hot dogs and beer. Note that there is no specific query language needed or requirements to be formed in a specific syntax. Additionally, the result set is shown in preview mode, where the search can be appended and/or modified before commitment to these filters. This facilitates quick interrogation of the data and helps users make more insights.



Global Search

Global filtering

To accompany global search, a fully structured approach to filtering is available on every sheet in the top right-hand corner called the global filter. In the global filter, we can see current selections in the top half of the sheet highlighted in green. The bottom half of the sheet is reserved for dimensions that have not been included in the filtering. Note the associated colors of green for selected elements, white for none selected, and gray for nonassociated elements. Light gray indicates excluded only by selection in the same field, whereas dark gray means excluded by selection in other fields. We can see that the current selections of **ARAge** as **31-60 Days**, **Customers** as **A&R Partners**, and **A2Z Partners** and **AccountDesc** as **Communications** are selected and highlighted in green. If we look at the **Customer** dimension, we see that all other customer names are dark gray because a customer can only have one name in this model. We also see that the other **ARAge** and **AccountDesc** dimension elements are light gray because they are excluded based on selections in other fields. This could change with a change in the selection criteria. Based on this example, global filtering provides a very powerful view of the relationships in the application's associative data model. It also centralizes filtering, leaving valuable screen real estate for visualization based on filtering and the exploration of information, and once selected, it appears in the **SELECTIONS** pane.

Associated Color Key:

- * Green: Associated
- * White: Not Associate for selection
- * Grey: No association

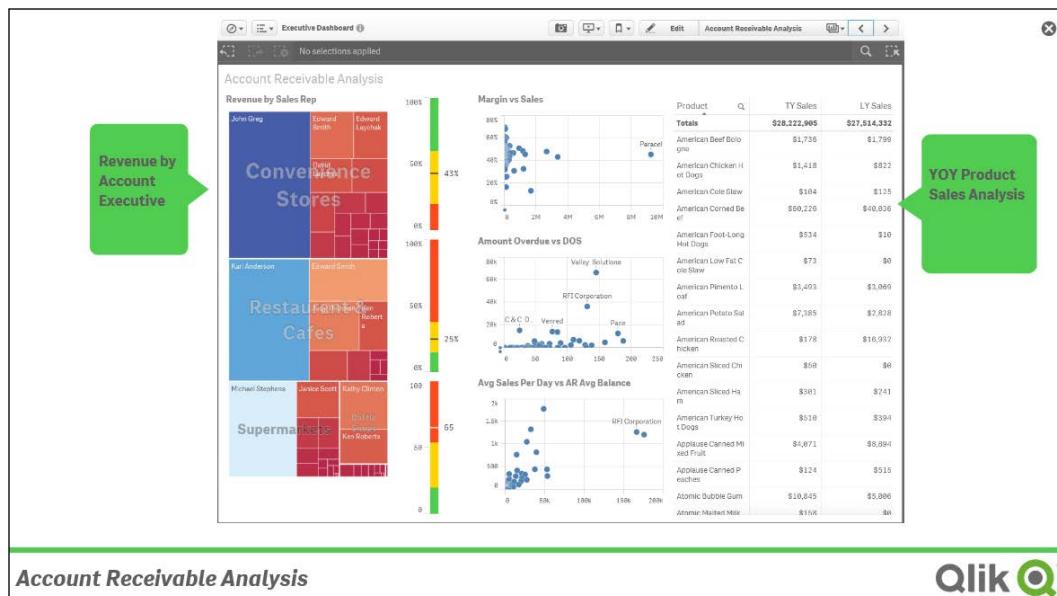
Global filtering with associative relationships

Global Filtering

Qlik

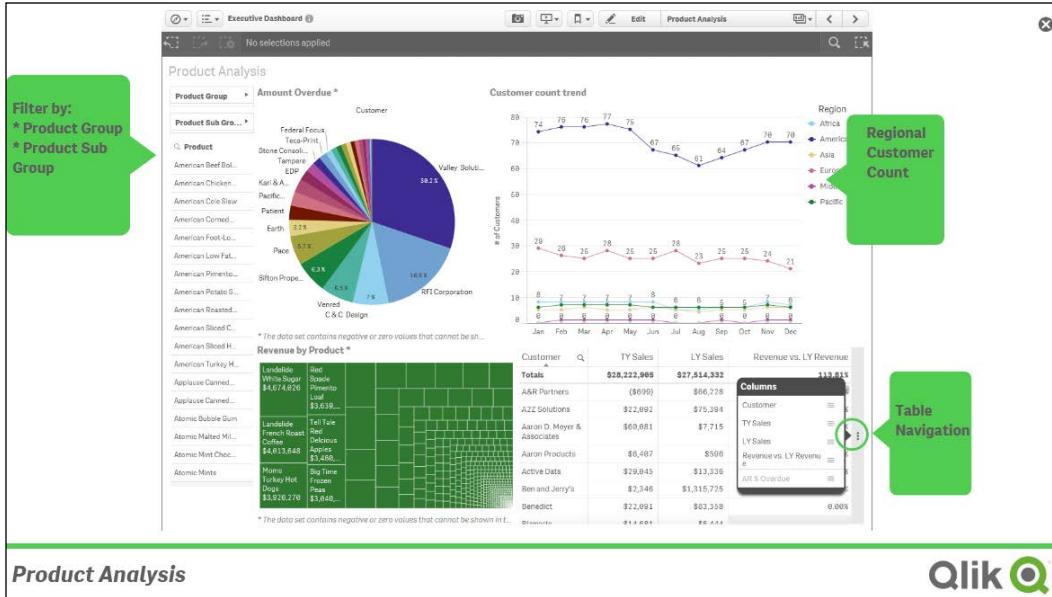
Global Filtering

Now, let's turn our attention to the **Account Receivables Analysis** sheet. This sheet is an interesting example of where there are no formal filter panes or listboxes (as there would commonly be in a QlikView app). Instead, each of the objects can be used to select areas to explore, and global filtering and global search can be used to augment or refine the selections at a finer detail level. In this case, revenue contribution for sales representatives by channel is displayed. Qlik Sense also supports a full range of objects, such as the table object to the right, which can be used to filter columns and supports exception formatting for variance reporting.



The Account Receivable Analysis sheet

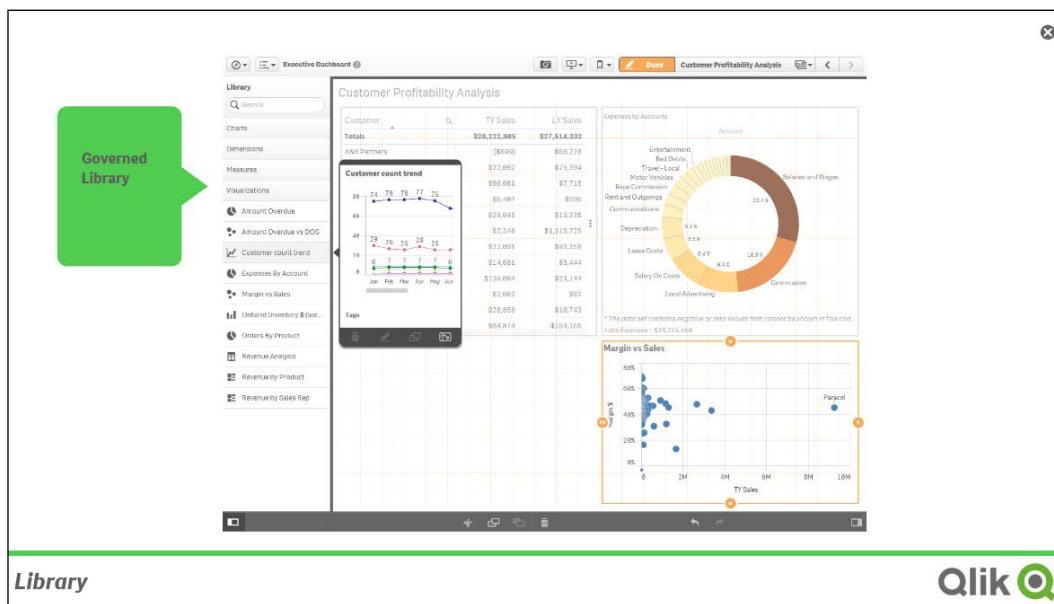
Finally, the table object has the ability to hide and show columns based on the allocated space for the table. The column selection menu within the table allows Nora to orient columns based on the viewable space available to the table.



Product Analysis

Extending with Library

As noted in *Chapter 1, Qlik Sense® and Data Discovery*, the rise of BI consumerism and self-service is becoming an increasingly important attribute to meet the needs of the next generation consumers. Qlik Sense embraces this important requirement through **Library**. The Qlik Sense Library is a governed area where an application's author can store dimensions, measures, and preconfigured charts that can be used to create compelling analysis that can be shared across an organization. In this case, Nora is taking advantage of the **Customer count trend** line chart to extend an application.



Qlik Sense Library

The Qlik Sense Library is at the center of a broad range of governed self-service capabilities that drives insight within an organization.

Summary

This chapter covered how Qlik Sense meets the new requirements of consuming and extending discovery-based applications, meeting these requirements across a myriad of platforms spanning PC, Mac, and the never-ending flow of new mobile devices. This required Qlik Sense to be built with a new approach that is responsive to these new realities of self-service and mobile use.

Now, let's turn our attention to the contributor who seeks to not only consume but also extend and share their data discovery insights.

4

Contributing to Data Discovery

In the previous chapter, we outlined data discovery consumption requirements, which provided an overview of key Qlik Sense capabilities for users who wish to consume an application that is prebuilt. This chapter's goal is to highlight key features in the context of the specific user requirements that Qlik has identified as being needed by a data discovery contributor, or someone who seeks to share key findings from their analysis in a governed manner.

In this chapter, we will cover the following topics of Qlik Sense:

- Data discovery contributor requirements
- Bookmarks
- Private sheets
- Private stories
- Publishing to an existing application

Realities of the data discovery contributor

One of the strengths of Qlik Sense applications is the ability to share and extend the value of applications with other members of the stream. As noted in *Chapter 2, Overview of a Qlik Sense® Application's Life Cycle*, there are a number of useful ways to share key business discoveries. These include the following:

- Bookmarks
- Private sheets
- Stories

Each of these capabilities helps analysts not only to consume Qlik Sense applications but also to share and spur additional conversation and insights. The stream administrator covered in *Chapter 9, Administering Qlik Sense®*, enables these contributor capabilities. Let's take a closer look at each of these capabilities through the role of an analyst named Pat.

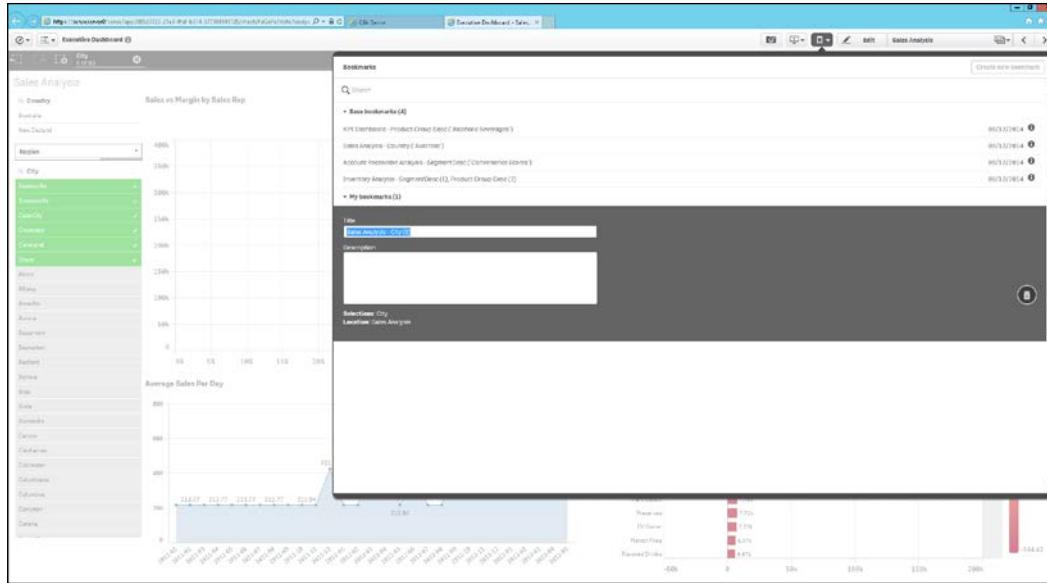
Creating private bookmarks

A private bookmark is the beginning of an analysis that drives collaboration across an organization. Bookmarks allow an author and a contributor to save the state of a sheet within a Qlik Sense application. In the previous example, the **Executive Dashboard** application, the author defined approved bookmarks. These public bookmarks are part of the published application to help users start their data discovery process. This capability is also available to contributors to save key business discoveries for a later time.

For example, say Pat conducts a sales analysis on products sold in key cities, as shown in the city's sales analysis in the following screenshot:

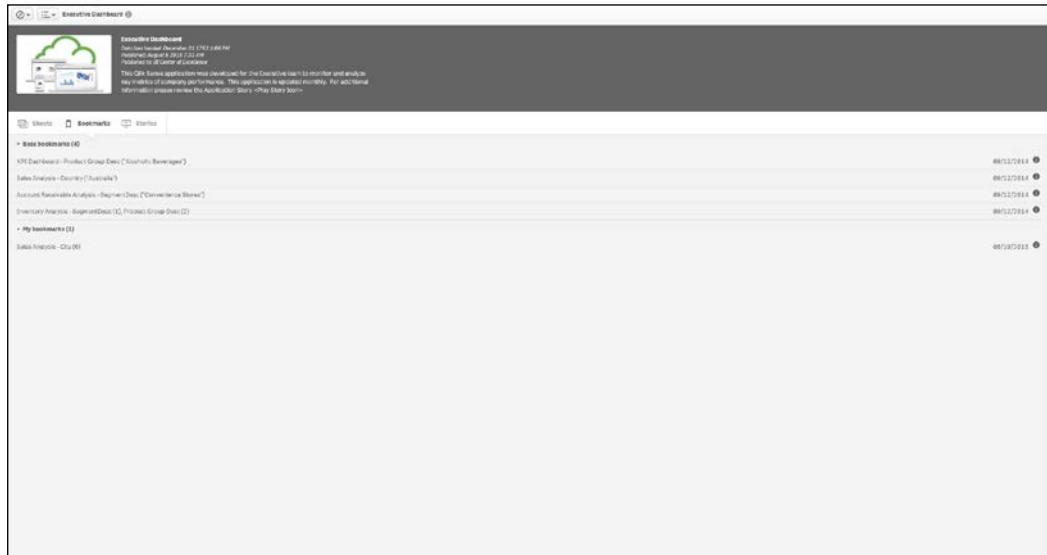


Pat has selected six key cities for analysis of sales representative performance and products sold. This view is interesting, so Pat decides to bookmark this sheet with these selections. Note that when selecting the bookmark icon, all approved and saved private bookmarks are available for navigation. Additionally, the **Create new bookmark** button is available and will automatically create a default title based on the sheet name and selections:



City sales analysis bookmark

Once saved, the bookmark becomes a part of the application under **My bookmarks** and can only be accessed by the creator, which in this case is Pat:

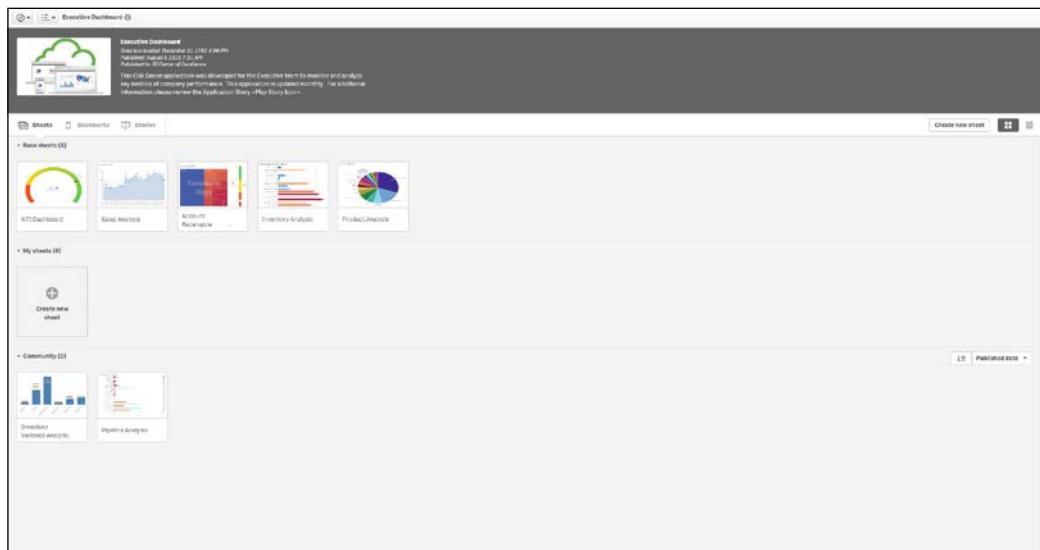


The Executive Dashboard bookmark

To summarize, simple Qlik Sense bookmarks can play an important part in bringing context to the beginning of an analysis as well as saving key insights gained from an analysis. Although separate features, Qlik is seeing early adopters use bookmarks as the start of building critical mass with insight that can be shared through published sheets and stories, which are the topics of our next sections.

Creating and sharing private sheets

As discussed in *Chapter 2, Overview of a Qlik Sense® Application's Life Cycle*, the building block of a Qlik Sense application is a sheet. In the **Executive Dashboard** community, we can see the sheets associated with the **Executive Dashboard** application. These include **Approved sheets** (published by the application author), **My sheets**, which are private sheets defined by the contributor (Pat), and finally, **Community**, which are private sheets published by other contributors:

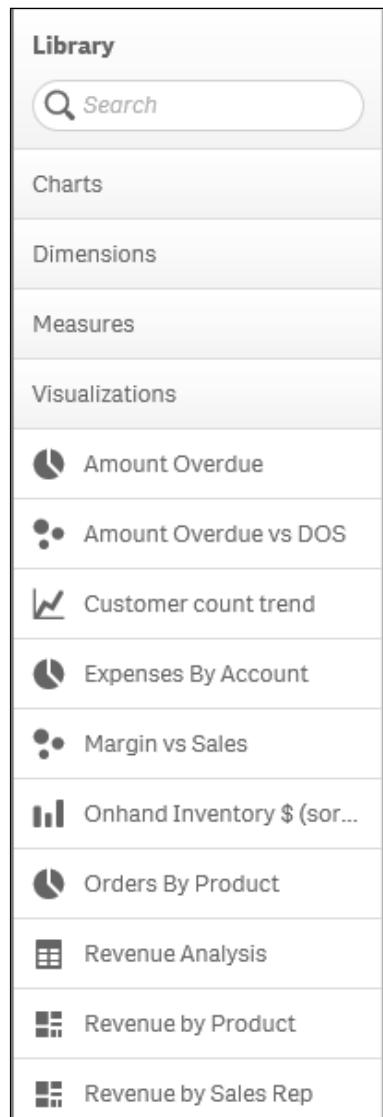


The Executive Dashboard community

Now, let's dig a bit deeper into how these sheets are built. There are two main ways in which private sheets are built, as follows:

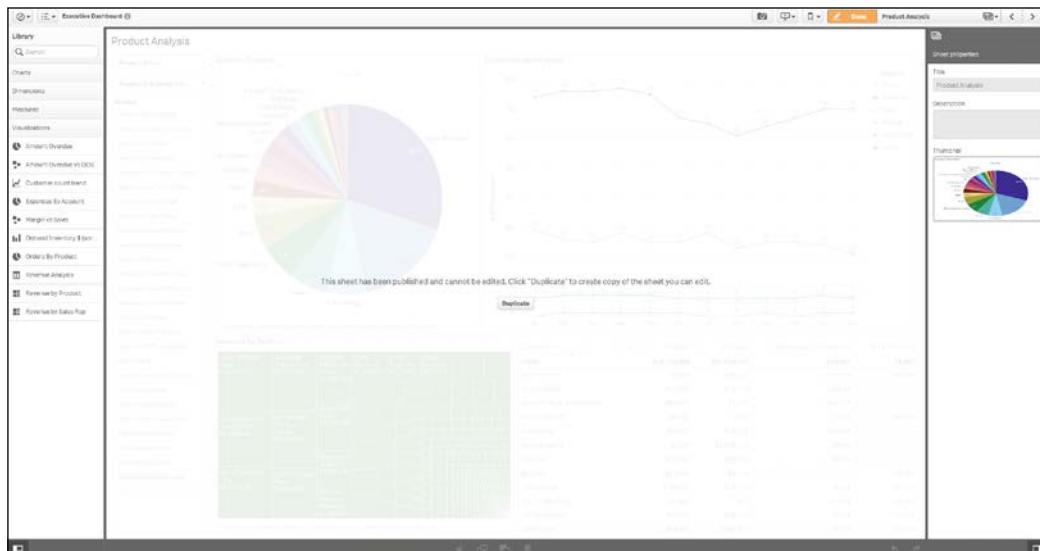
- Duplicate an approved sheet
- Create a new private sheet

In both cases, a key feature that allows a contributor to build strongly governed private sheets is the Qlik Sense Library. The Qlik Sense Library is a key component of an application that allows the author to expose key portions of the associative model in the form of **Dimensions**, **Measures**, **Charts**, and predefined **Visualizations**. How the Qlik Sense Library is created will be covered in more detail in the next chapter. The following screenshot shows **Library**, which can be searched for dimensions, measures, and prebuilt visualizations:



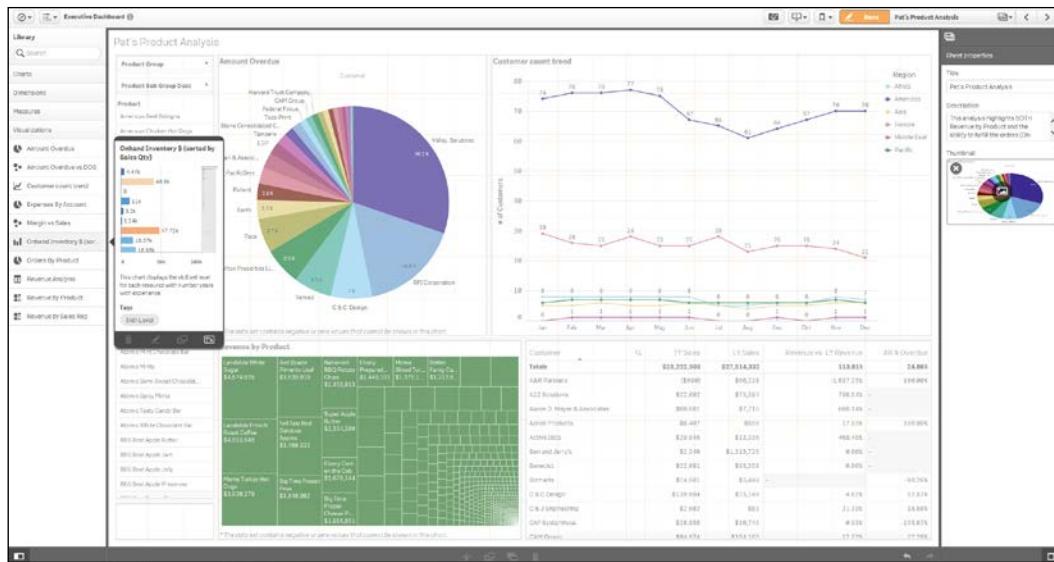
Creating a private sheet

Now, let's turn our attention to creating a private sheet by the first method, duplicating an existing sheet, and then editing it to meet your requirements. The advantage of this method is that Pat can start the creation of her product analysis based on the approved **Product Analysis** sheet. The process begins with selecting the sheet that best aligns with the content you wish to analyze. In this case, Pat wishes to create a product analysis that integrates the inventory on hand with the approved **Product Analysis** sheet. As the **Product Analysis** sheet is an approved sheet, it cannot be edited and must first be duplicated before changes can be made.



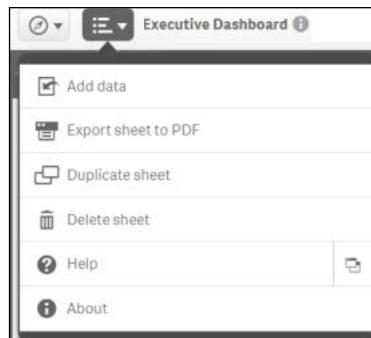
Duplicating the Product Analysis sheet

Once the sheet is duplicated, it is automatically converted into a private sheet, where Pat can rename and alter the content and layout of the sheet through the use of **Library**. Note that Pat has renamed the sheet to **Pat - Product Analysis** as well as added a helpful description, which highlights the goals of this sheet—This analysis highlights both Revenue by Product and the ability to fulfill the orders (On hand Inventory\$) to recognize revenue. Additionally, there is a wide selection of preconfigured charts as well as dimensions and measures she can take advantage of in **Library**. In this example, Pat will replace the customer count line chart with the **Onhand Inventory \$ (sorted by Sales Qty)** horizontal bar chart from **Library**. This is one example of a variety of governed changes available to Pat in designing a new sheet. We will explore the breadth of changes to develop private sheets in the next section.



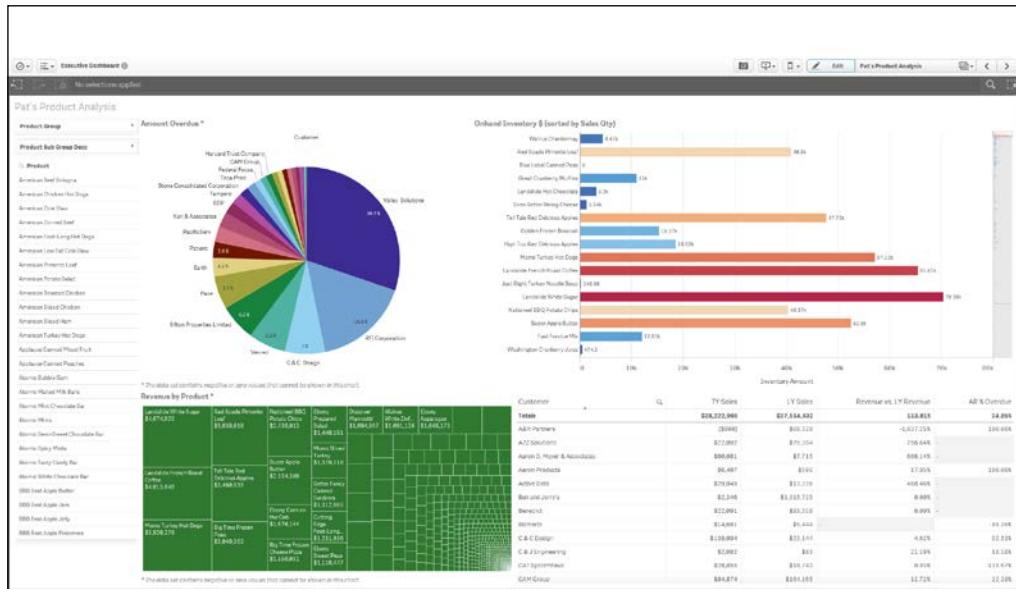
Creating the Pat-Product Analysis sheet

With the **Onhand Inventory \$ (sorted by Sales Qty)** chart from **Library** dragged and dropped onto the sheet, Pat is ready to end the editing process. Since this process is all server-based, there is no need to save the sheet but rather just click on the **Done** button. Additionally, this sheet can be exported as a PDF and then either distributed via e-mail or printed:



Contributing to Data Discovery

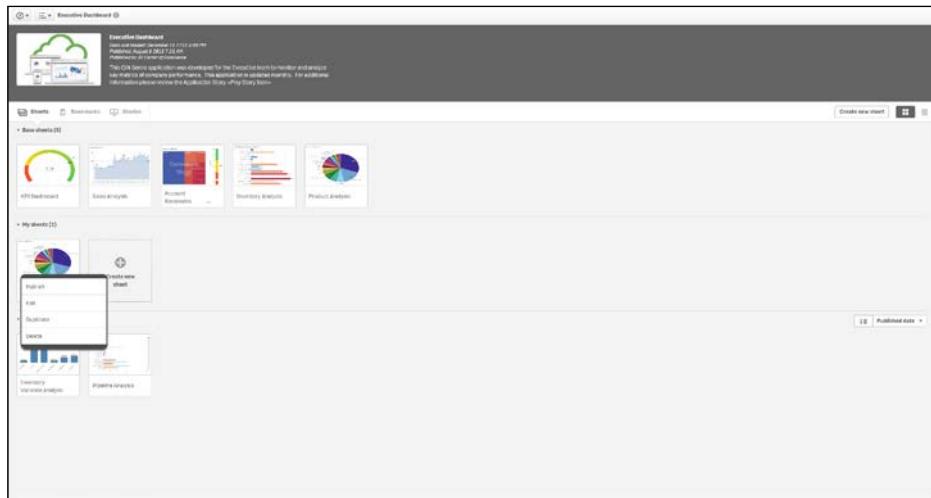
Export the sheet to PDF. This is how it will look:



The Pat-Product Analysis sheet

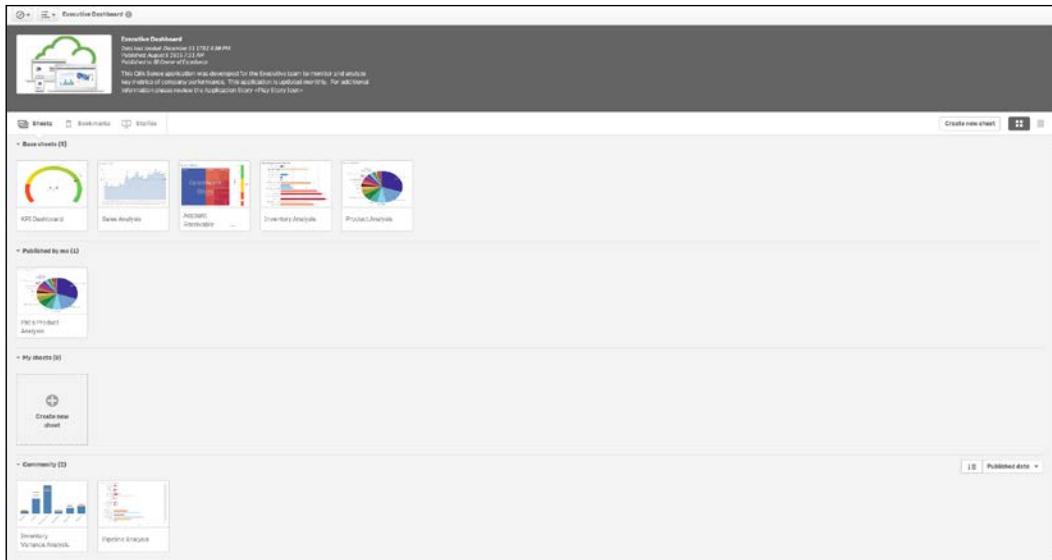
Publishing a private sheet

Now that the sheet is complete, let's return to the application overview. As you can see in the following screenshot, **My sheets** now contains a new sheet called **Pat-Product Analysis**, and with a right-click, it is ready to be published to the community:



Publishing the Pat-Product Analysis sheet

When the sheet is published, a new section will appear called **Published by me** that contains all published sheets by Pat. Also, note that Pat has a number of sheets that are in progress in the **My sheets** section. These published sheets can be duplicated by others who have access to this stream and are extended and shared as well.

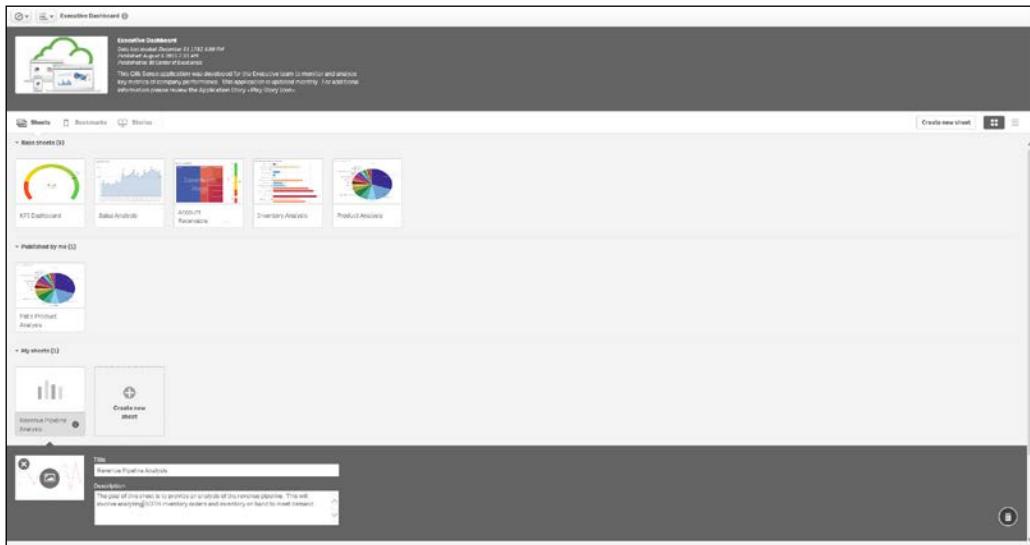


Pat's sheets

In summary, a duplicated approved sheet is an excellent way to start the creation of a private sheet as it has the advantage of leveraging the existing defined sheets from the published application or the work of other analysts in the community. Now, let's turn our attention to creating a new sheet.

Creating a new sheet

A second approach to sharing key business insights is to create a new sheet. As shown in **Creating a New Sheet** in the following screenshot, Pat creates a new sheet called **Revenue Pipeline Analysis**, which contains both order information and inventory on-hand information to meet customer demand. This allows Pat to create and share new information across the organization:

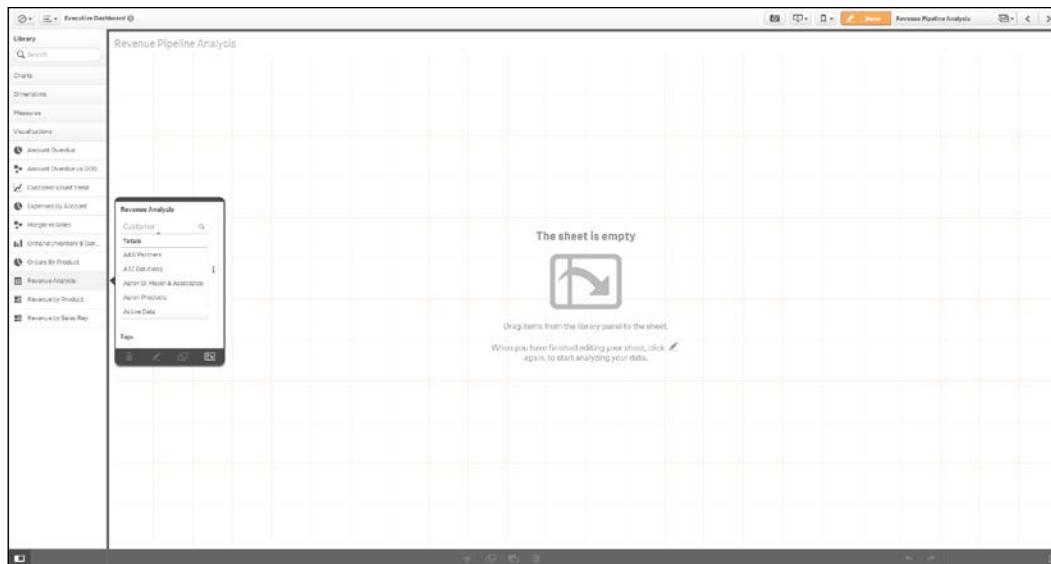


Creating a new sheet

Once the sheet is created, **Library** and sheet properties are exposed and the sheet appears with a faint grid. This grid is a part of the responsive web design experience and facilitates the orientation and placement of objects from **Library**. This not only helps in the creation of the sheet but also plays a key role in how the objects will be viewed and consumed across multiple devices. Also, note that the creation and assembly of new objects is easily done by users due to the associative engine. Because of the associative model, every object is connected and no author prewiring is required. The associative engine permeates the use of Qlik Sense, not only its use, but also the creation of compelling solutions.

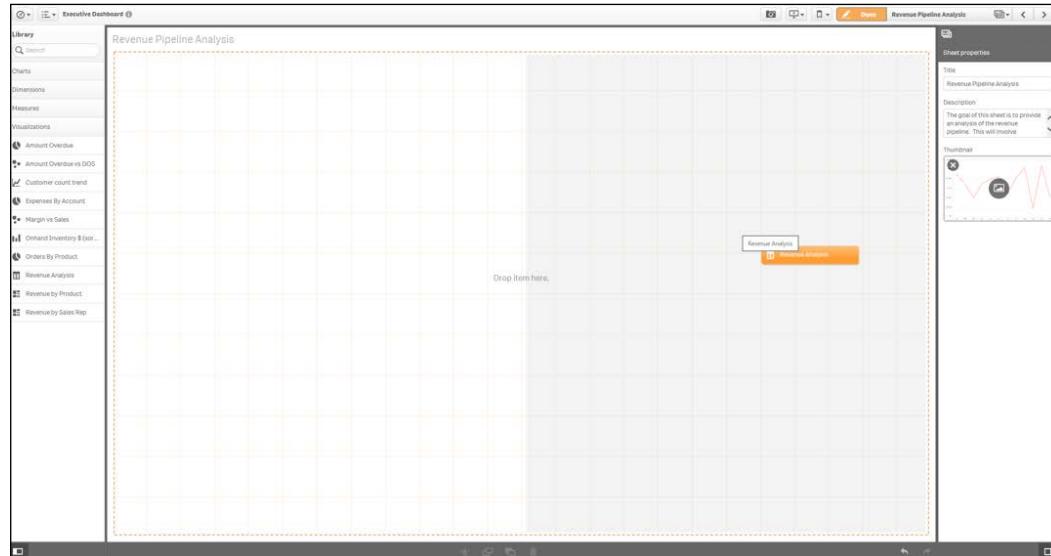
Adding a predefined visualization to a new sheet

One of the key areas Pat is interested in is the customer revenue this year and in the previous year to help her better anticipate customer demand. Hopefully, the author of the application anticipated this common request and stored a table chart under **Visualizations in Library**. Specifically, the **Revenue Analysis** table chart is available in **Library** with a thumbnail shown to help Pat evaluate its applicability to the sheet content:



Adding a predefined visualization to a new sheet

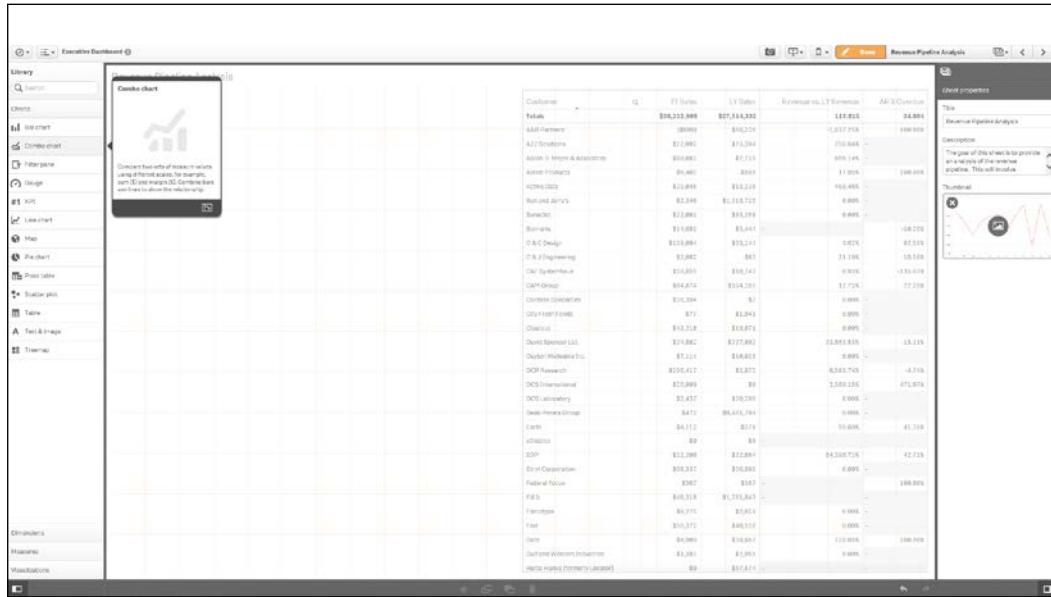
Adding the **Revenue Analysis** object is a simple drag and drop movement. Note that the sheet grid will automatically make recommendations on the placement of the object:



Drag and drop the object

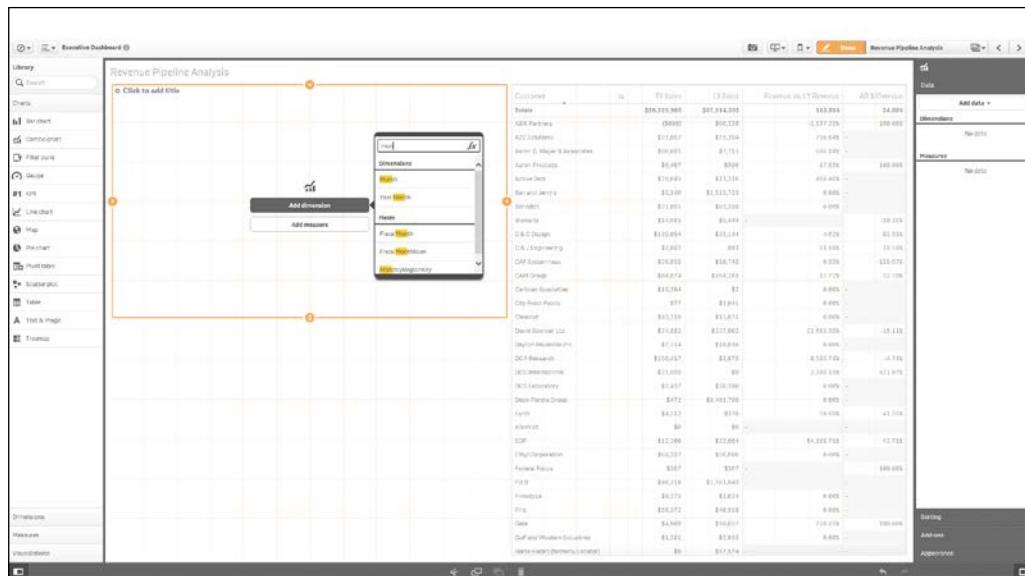
Creating a Combo chart object

Once the object is placed, Pat notices that there are no visualizations available that allow her to see the trend of inventory on hand and sales orders. This requires her to create a new chart based on dimensions and measures defined in **Library**. So, to begin this process, Pat selects and drags **Combo chart** to the sheet noted in the following screenshot:



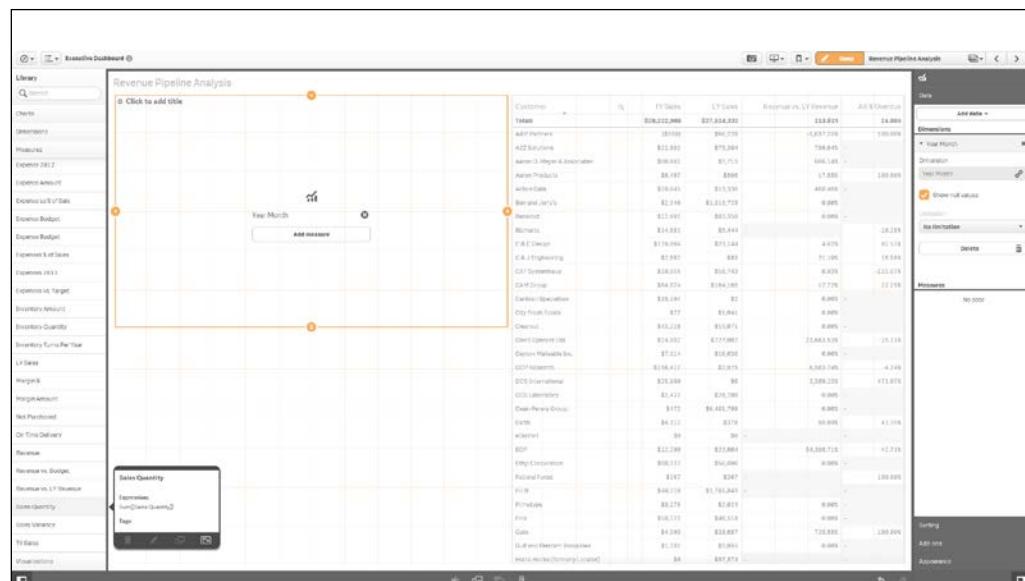
Creating a Combo chart object

Once the Combo chart is in position, the object guides Pat on the requirements for visualization. The object highlights the requirement of at least one dimension and a measure. To speed up the task, the Qlik Sense search capability can be used to find the dimension; in this case, **Year Month**:



Adding a dimension

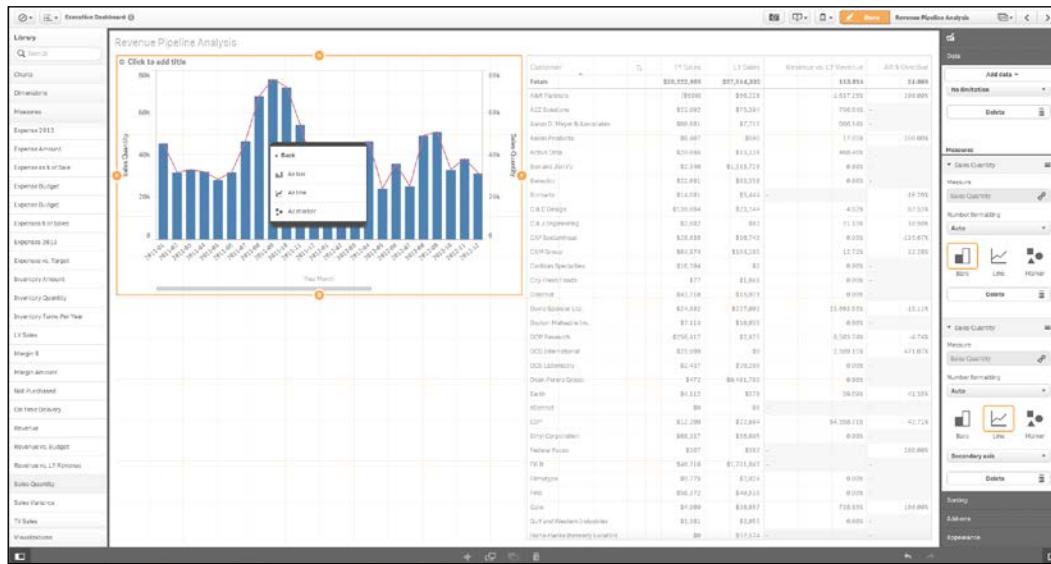
The next step is to add the measures; as this is **Combo chart**, there will be two measures. The first measure added will be **Sales Quantity**. The **Sales Quantity** measure is available in **Measures**, and a tooltip reveals the expression that shows how it is defined:



Adding a Sales Quantity measure

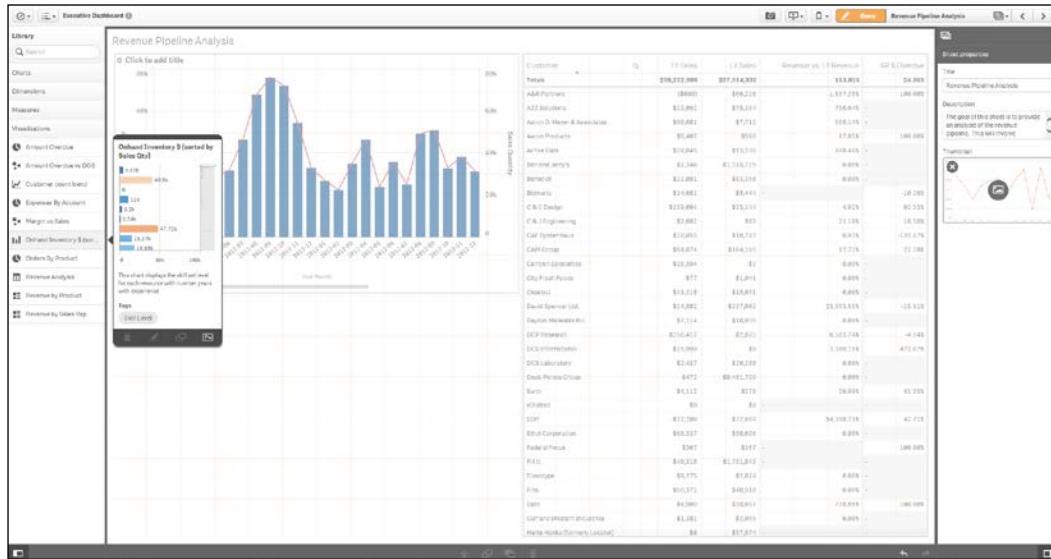
Contributing to Data Discovery

Additionally, as the measures are dragged and dropped on **Combo chart**, the object continues to guide Pat on how to visualize the data. Qlik Sense provides guidance to add the inventory quantity and options for display. **Combo chart** supports bar, line, and marker chart types. In this case, Pat selects a line to compliment the **Sales Quantity** measure that is already displayed as a bar. The selection of charts can also be changed quickly while keeping the defined dimensionality of the previous chart:



Adding the Inventory Quantity measure

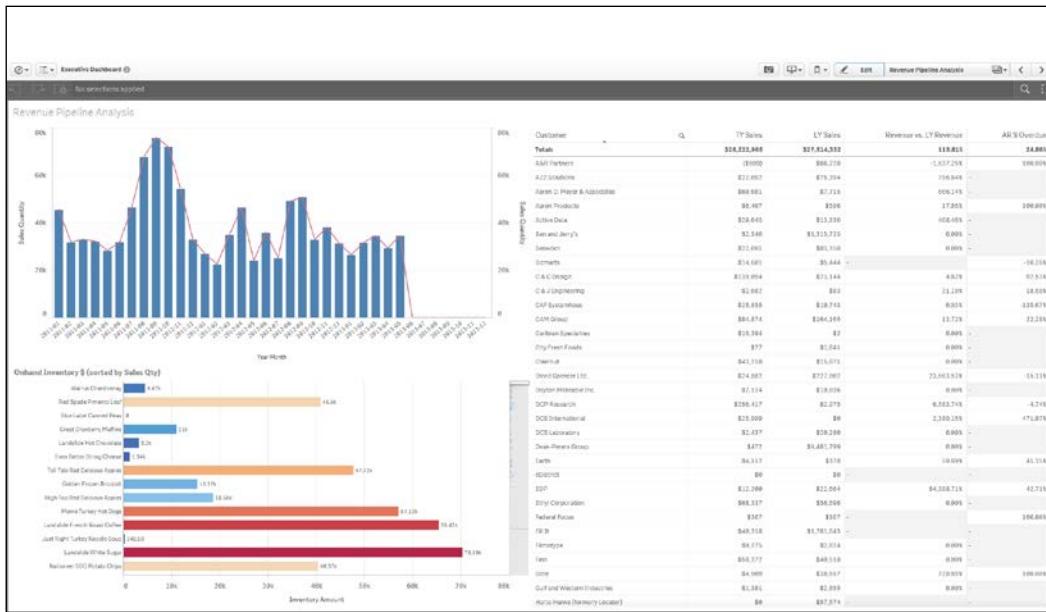
Finally, Pat completes the sheet layout by adding the **Onhand Inventory \$ (sorted by Sales Qty)** chart available in the **Visualizations** portion of **Library**, which is shown as follows:



Adding the Onhand Inventory \$ (sorted by Sales Qty) chart

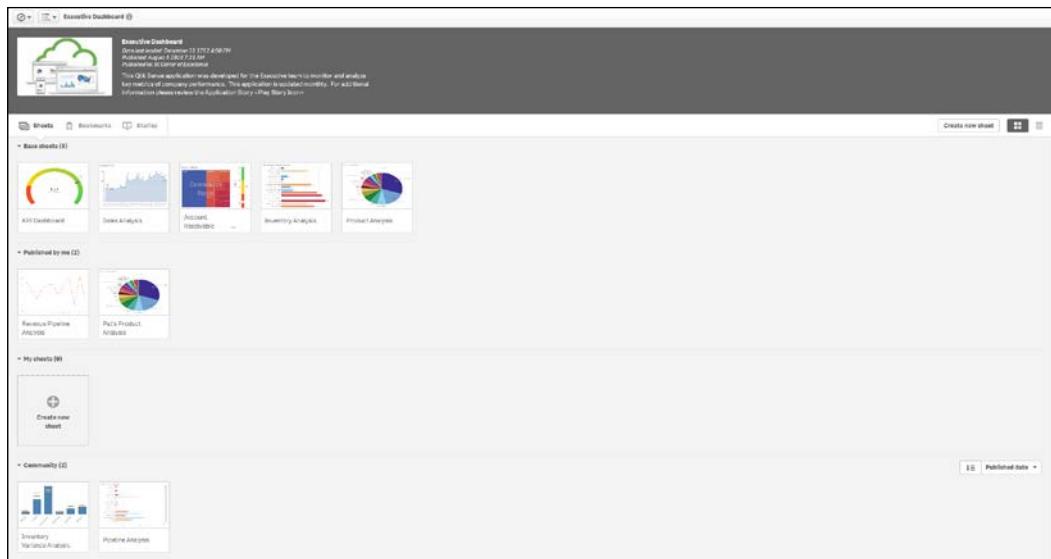
Publishing a private sheet

With the new **Revenue Pipeline Analysis** sheet completed, Pat is ready to publish with a right-click, as described earlier. Also, it is worth mentioning that this assembled sheet is fully selectable during the process of assembly, and no wiring (connecting) of these objects is needed to allow them to communicate with each other across all sheets:



Completed Revenue Pipeline Analysis sheet

Once the sheet is published, it is available to the **Executive Dashboard** application, where it can be consumed, duplicated, and expanded by other members of the community:



Published Revenue Pipeline Analysis sheet

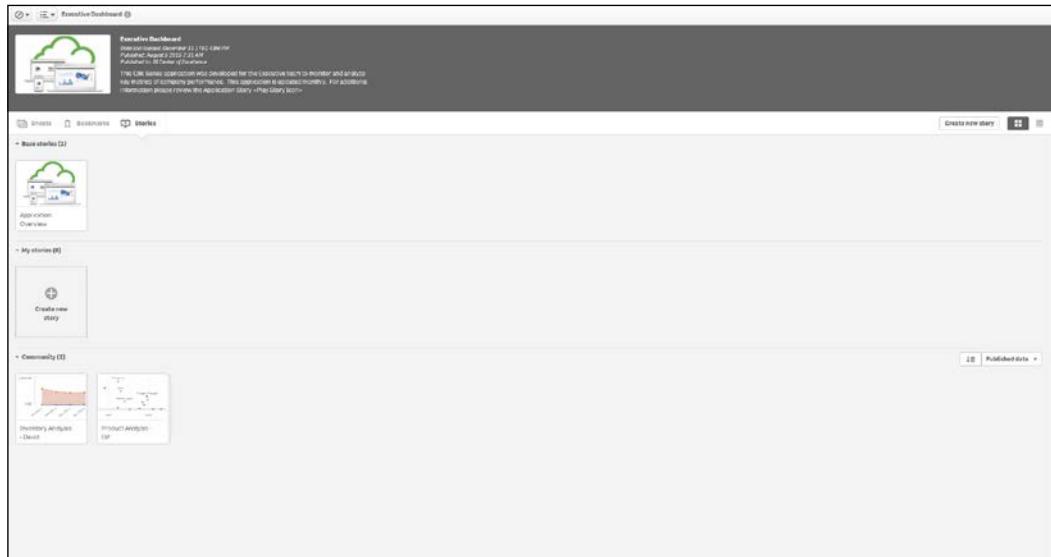
In summary, creating new sheets provides an alternative way to collaborate with members of the application's community. It allows contributors in a governed environment to start with a blank sheet to organize and share their thoughts and insights, and is managed centrally in the Qlik Management Console. Now, let's turn our attention to creating a Qlik Sense story, which adds additional capabilities for collaboration.

Creating and sharing stories

Qlik Sense Stories provide an additional capability to collaborate and share business discoveries within the Executive Dashboard community. In the story overview, we can see that similar to bookmarks and sheets, stories have **Approved stories** (defined by the author of the application), **My stories** (private and only viewable by the author), and **Community** (published) sections. We covered the role of approved stories as a way for application authors to provide an overview about the application and intended use.

Contributing to Data Discovery

In this section, we will focus on the creation of a story by a contributor (Pat) who will use this capability to present a sales analysis to the community:



Story overview

Defining a story

To begin with, Pat creates a new story by selecting the **Create new story** option under **My stories**. The default name is **My new story**, which Pat changes to **Sales Analysis - Pat** to reflect the goal of the story. Additionally, a description can be added to provide information on the goals of the story:



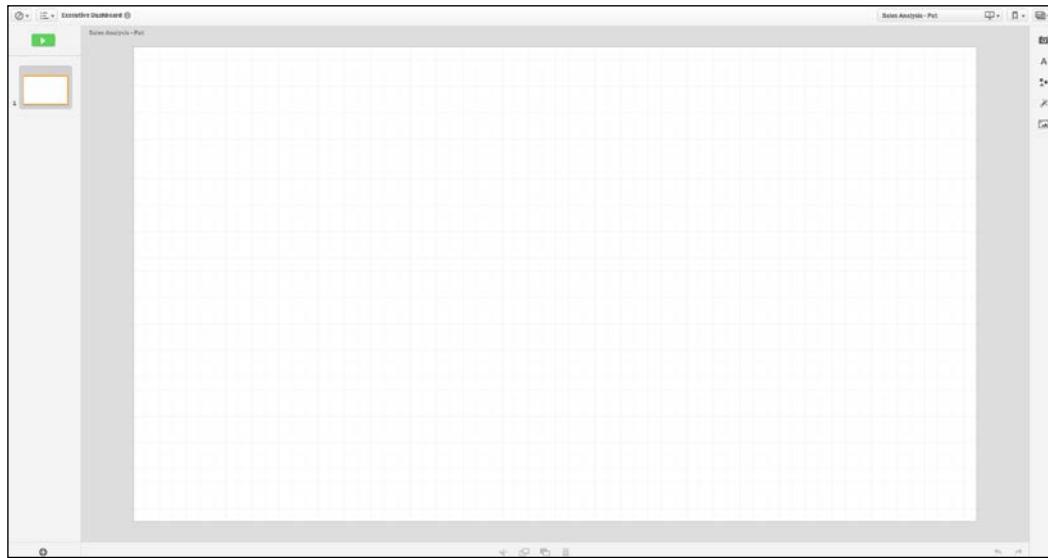
Defining a story

Once the story is defined, Pat enters the story workspace, which provides a broad set of tools to create rich presentations that are dynamically linked to the Qlik Sense application. The story workspace contains the ability to create sheets as well as access the following libraries:

- Snapshot
- Text
- Shape
- Effect
- Media

Contributing to Data Discovery

We will explore each of these areas as Pat defines her presentation:



Story workspace

Creating snapshots

Let's start with creating snapshots. The ability to create a snapshot is a general capability found on all sheets within an application. Snapshots provide analysts like Pat with the ability to capture insights across a Qlik Sense application and organize them with additional context through stories. By selecting the camera, all objects for which snapshots can be created are highlighted with an orange outline. Additionally, each object also contains an indicator that highlights the number of times snapshots have been created for the object. As you can see, Pat has been quite busy in selecting key objects for her story:



Creating snapshots

Now that Pat has selected her snapshots, she prepares to organize them in a story. Note that all snapshots are stored in **Snapshot Library**:



Story snapshots



These snapshots are organized by the date and time when they were taken. This is an important consideration because it means that snapshots are like photos, storing the visualization and data of the time the snapshot was taken. By design, snapshots are not updated when the application data is changed.

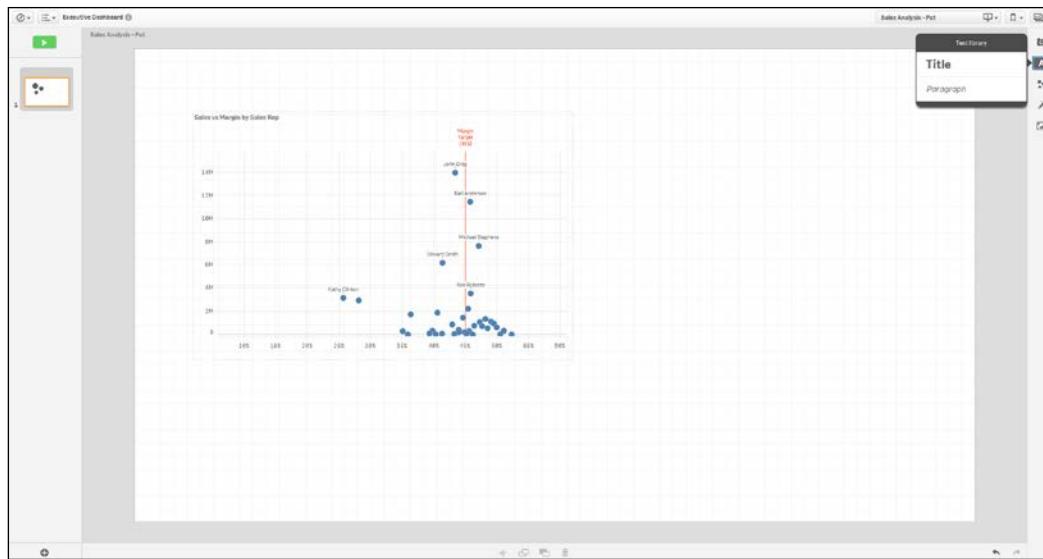
Once the snapshots are taken, Pat locates the shot and then drags and drops it onto the grid. Also, note that each snapshot can be (unlocked) edited with the ability to modify some of the properties, which can include turning on/off titles and labels:



Adding a snapshot

Adding text

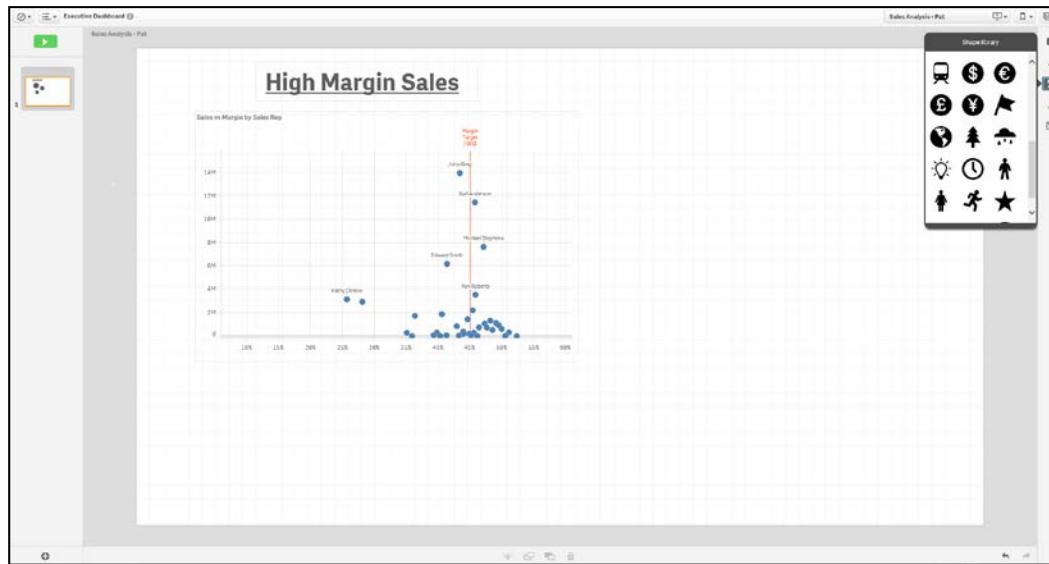
Now, let's add text to this sheet, which is accomplished through **Text library**. The **Text library** facilitates both the creation of titles as well as paragraphs that can be used to add comments to highlight key business discoveries:



Adding text

Adding shapes

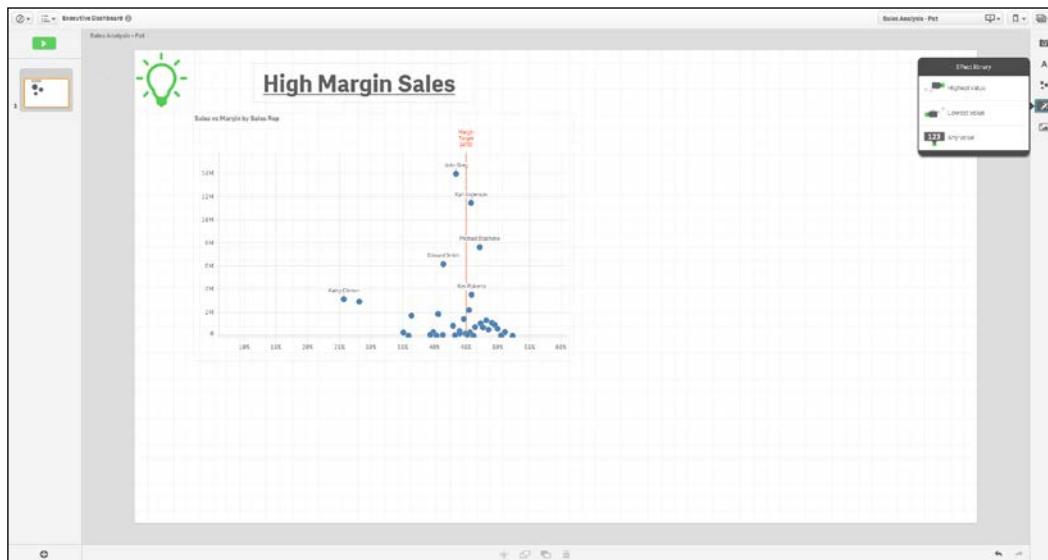
Pat has added the title **High Margin Sales** and emphasized it with bold and underline styles. Additionally, there is **Shape library**, which allows the integration of various shapes to highlight and emphasize the story:



Adding a shape

Contributing to Data Discovery

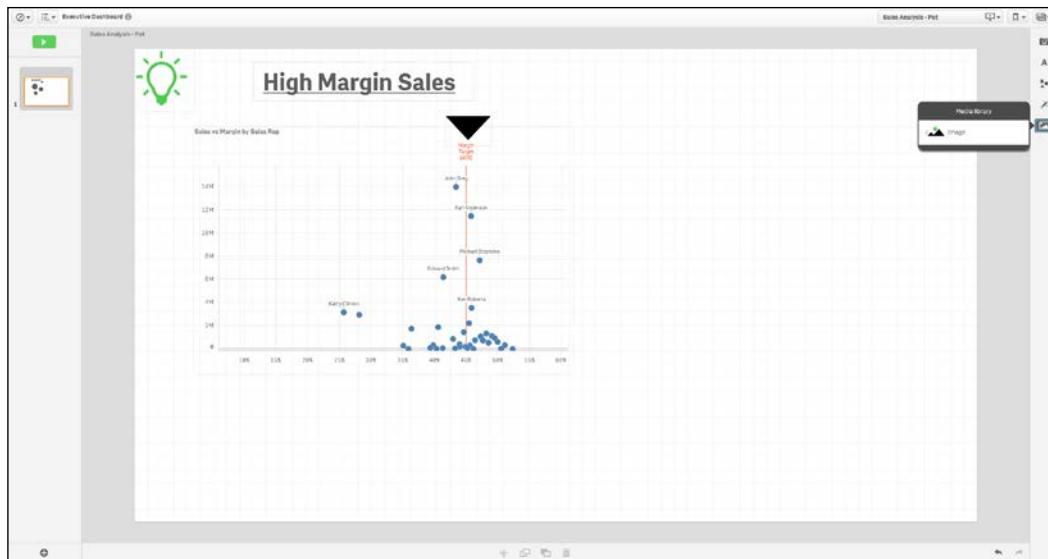
Pat chose the light bulb symbol to identify key ideas in this story. The symbol's default color is black, but can be changed. Additionally, there is **Effect library**, which can be used to highlight the lowest, highest, or a particular value within a chart:



Adding effects

Media library

The final major area is **Media library**, which offers contributors the ability to add images from outside Qlik Sense. Images are made available and managed by the Qlik Management Console through **Content library**. Additional information on this process is available in *Chapter 9, Administering Qlik Sense®*.



Adding media

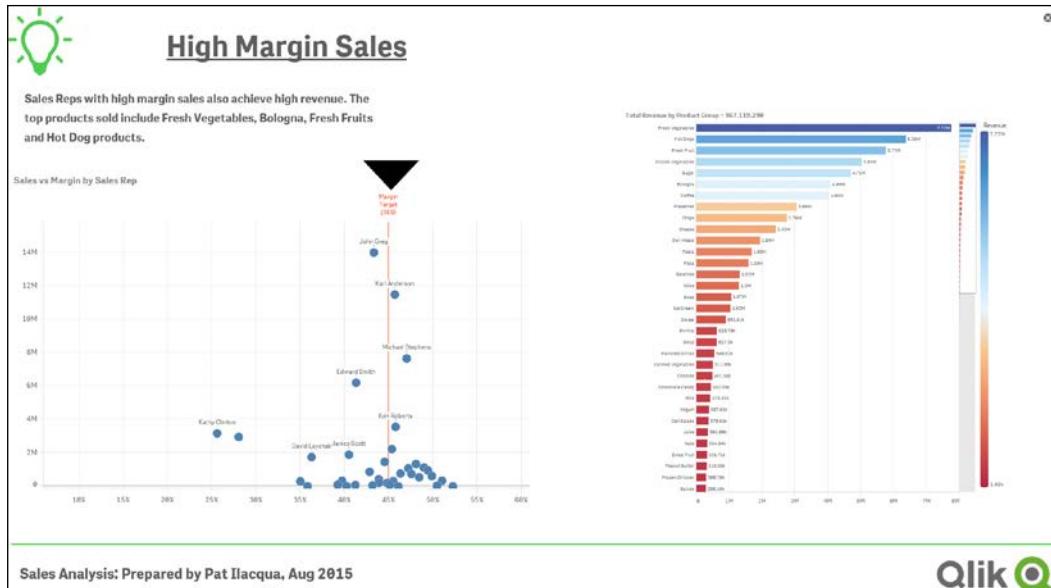
Additionally, Pat would like to add a portion of the **Sales Analysis** application directly in her story. Qlik Sense Stories also allow an approved sheet to be embedded within the **Sales Analysis** story. This enables Pat to share her analysis through a number of slides and also offers the viewer, the ability to continue their exploration through an active sheet.



Adding a dynamic slide

Contributing to Data Discovery

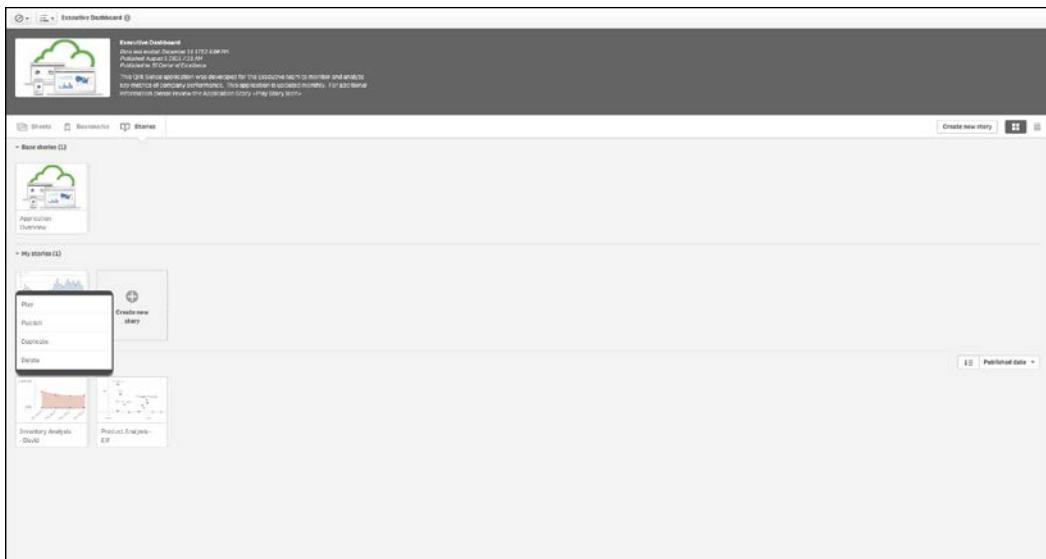
Once the slides are completed, Pat can review how the slides will be viewed by selecting the Play the story button. Also, note that each of the snapshots has an embedded bookmark that can be selected by right-clicking on **View source**, and the viewer will be directed back to the application sheet to continue their exploration:



Final review

Publishing your story

Now that Pat is comfortable with the **Sales Analysis** story, the publishing process is similar to the publishing process of private sheets. To accomplish this, as illustrated, Pat right-clicks and selects **Publish** to move the story to the community and make it read only:



Publishing the story

Summary

One of the strengths of Qlik Sense applications is the ability it offers contributors to actively share, collaborate, and extend the value of the application with members of the stream. Qlik Sense has a number of exciting ways to share key business discoveries. These include bookmarks, published sheets, and stories. Each of these approaches is highly governed and provides a wide range of capabilities to meet the needs of a contributor.

In the next chapter, we will explore using the skills we learned alongside some ideas of best practices in how to create author-engaging applications for Qlik Sense.

5

Authoring Engaging Applications

In the previous chapters, we looked at the application life cycle and the different roles of users: the consumer and the contributor. Having established the basic requirements, in this chapter we will dive into the details of app creation and discuss how it is done. We'll also look at best practices of visualization and how to employ them using Qlik Sense.

In this chapter, we will discuss the following topics:

- The process of building an app
- Data connectors
- The data model viewer
- Sheet objects – visualizations
- Best practices
- Migrating QlikView applications into Qlik Sense

Preparations and requirements

Often the initial step in building an app is that you have some data that you want to analyze, but you *don't necessarily know* exactly what you want to look for in the data. As a business user, you can—and should—just load this data into Qlik Sense and start developing. Our experience is that the best way to develop the app is to start *without* first defining the requirements.

The reason is that when you load data and start to create visualizations, you *learn* from data. This knowledge is very important once you start defining what you want to analyze. Hence, you should first develop a basic app, then take a break and evaluate what you learned. *Now* is the right time to start formulating the requirements.

Another common case is the opposite situation: you know that you want to calculate a specific KPI, for example, supplier efficiency, but you don't necessarily know what data you need to be able to do this. In this case, you need to start with some research about where to find the relevant information, that is, in which database and in which tables.

The requirement specifications

If you define a larger project, you will use what you know as a starting point for the requirement specifications for your app. The following questions might pop up:

- **Data:** Which data sources should be used? Which tables should be used? How should the tables be linked? Are there common keys? Is there more than one source for the transactions? Are there tables missing? How should the customer hierarchy be resolved?
- **KPIs:** Which calculations should be made? You could consider turnover, profit, cost, delivery accuracy, or product quality. Which definition of gross margin should be used? How should the given discount affect the calculation of a salesman's bonus? Which accumulations are needed: year-to-date or month-to-date?
- **Dimensions:** How should the KPIs be displayed? You could consider showing them per year, per customer, per salesman, per region, or per product. Which comparisons should be made: year-over-year or month-over-month? Are there drill-down hierarchies that need to be defined?
- **Security:** Is the data confidential? Who gets to see what? Can we allow offline usage? Is the authorization data driven or static? Do we need to include authorization information in the data model, or can we postpone the decision around security?

You will soon realize that creating the requirement specification is not an easy task.

The communication problem

Discovering exactly what users, stakeholders, and sponsors want you to create is often the most difficult part of a business intelligence project. The communication between IT experts and nontechnical business users is often full of misunderstandings and misinterpretations. Business users often don't know what they want until they see it, and they frequently can't articulate their expectations in languages that IT experts use to design systems.

Few business users will know what a **data model** really means, so expecting them to be able to exactly define the requirements in technical terms is futile. Experienced authors can extract this information through discussions and clever questioning, but the number of people who are able to do this within an organization is limited.

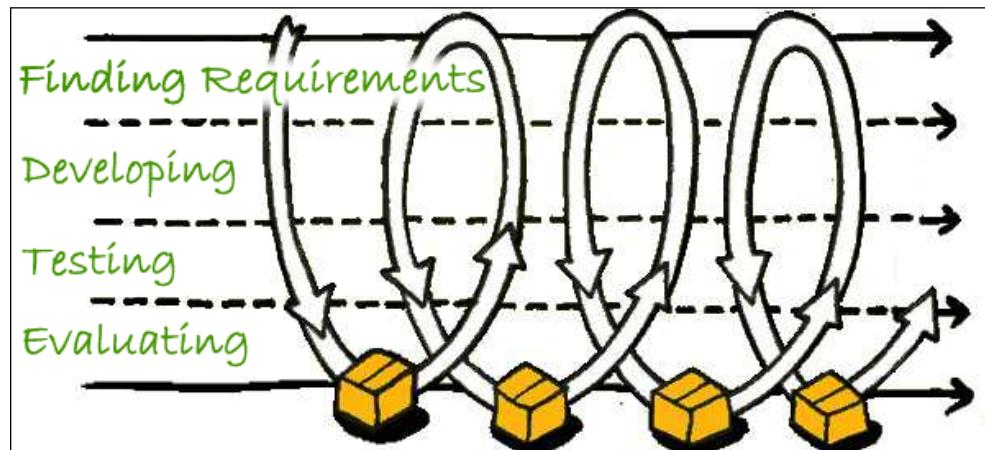
IT professionals often frame their requirement questions in technical language, for example, "Which table in the database should be used?" or "Which fields should be used to calculate the KPI?". However, business users may not have the technical knowledge to respond to these questions. Business users often explain their expectations in a technically vague language, which is not specific enough for designers to develop solutions.

On the other hand, *the business user is the customer*. The very reason why we develop an app in the first place is to supply the business user with a tool to analyze and learn from data. So, the requirement specifications *must* focus on the business user.

A step-wise implementation

The solution to this communication problem is to use a step-wise implementation, where the app developer iteratively finds the requirements, develops the app further, tests what has been done, and finally evaluates the app together with the business user. The evaluation will lead to new requirements and to changes or refinements of the old requirements. The steps must be small and the typical cycle is hours or days.

In other words, you *discover* the requirements together with the business user. As the development proceeds, the app will converge to the needs of the business user.



The iterative development process

This means you *cannot* begin your app development with a detailed requirement specification. Rather, you should start with a very basic specification containing information about some of the needed data sources and ideas of some of the required visualizations.

Hence, irrespective of whether you are a business user or an app developer responsible for data modeling and difficult formulas, you should start by spending an hour or so to load the data and create some graphs with the goal to *learn* from data. Then, you are in a much better position to define or discuss requirements further.

The process

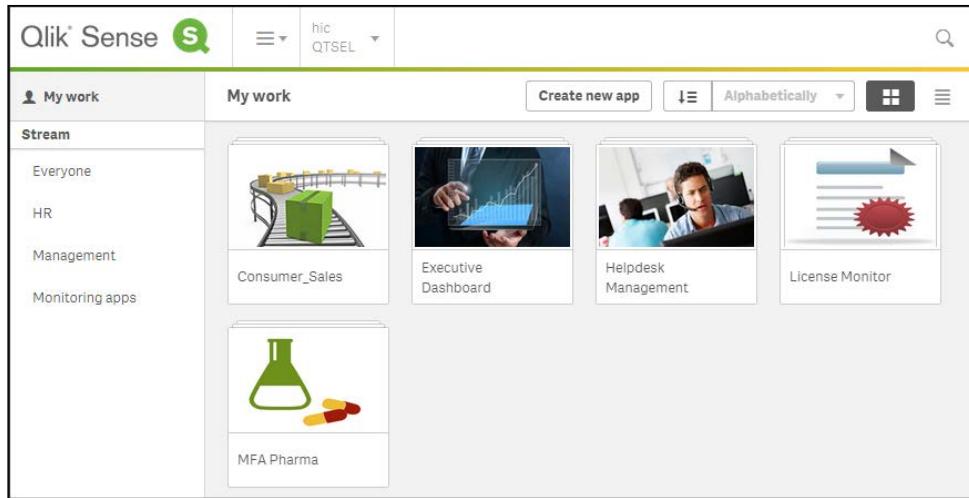
The first step in building the app is to load the data. The data can be one single table or several tables linked logically by **key fields**. Key fields are fields that exist in more than one table and link rows in one table with rows in another. Together, they form a data model. The next chapter will discuss the data model, so, for the moment, we will not get into the details of this.

When you have a data model, you can start building the layout, which consists of different objects, for example, lists, graphs, tables, and filter panes, placed on different worksheets. The objects can contain formulas that define different calculations that will be calculated as the users make their selections.

The previously explained development model assumes that you have both a developer and a business user that participate in the development process. In real life, you will notice that the initial development efforts will usually be like this, but as the app takes shape, the business users will want to do more and more on their own—which is good. After all, the goal is to have business users who are self-sufficient and create apps as much as possible on their own.

Getting started with the app creation

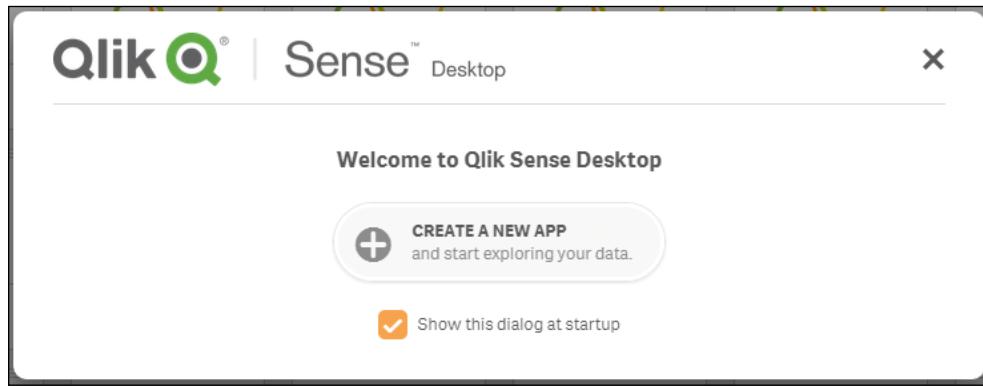
When you first open Qlik Sense, you come to the hub. This is the place where you have an overview of all your apps. The hubs look slightly different in the Desktop and Server versions, but they are essentially the same. The following screenshot shows what a hub looks like:



The Qlik Sense hub

Creating a new app

In Qlik Sense Desktop, you are greeted with a dialog that asks you to create an app as shown in the following screenshot. In the Qlik Sense server, you will find the corresponding functionality on a button labeled **CREATE NEW APP** in the toolbar:

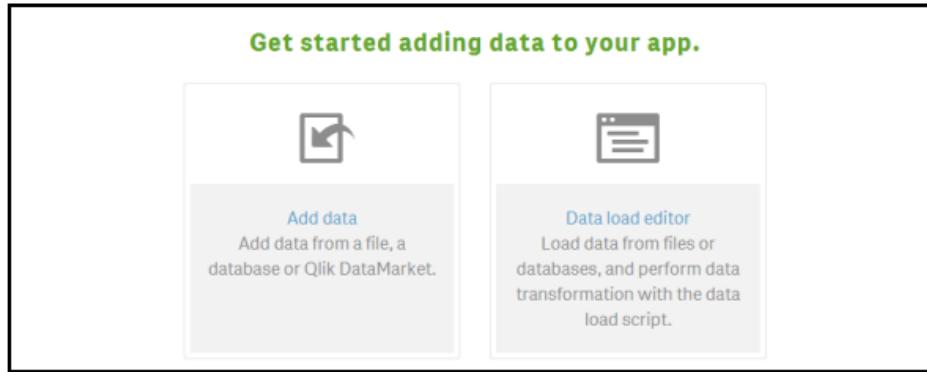


The Qlik Sense Desktop welcome dialog

Creating an app means you will create an entity that will hold both the data and everything else needed to analyze it. In Qlik Sense Desktop, this is a file created in `C:\Users\User\Documents\Qlik\Sense\Apps`.

Loading your data

Once you have named and opened your app, you will get a screen where Qlik Sense asks you to load your data:

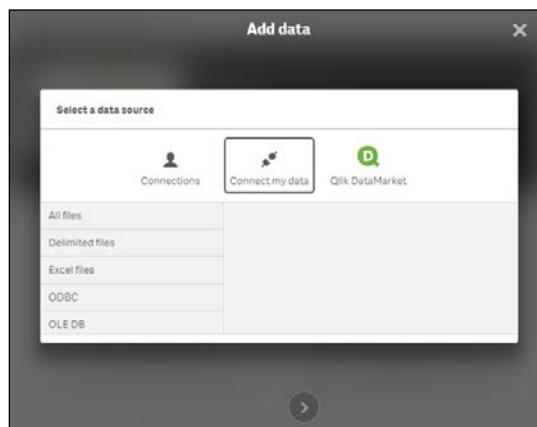


The Get started screen

When you have this option before you, you can load data in several different ways. The **Add data** command to the left will start with a wizard that helps you define what you want to load. In the background, it adds code to a script that defines the load sequence. It will, however, never show the script.

This is different from **Data load editor** that will take you to a script editor, where you can change the script directly.

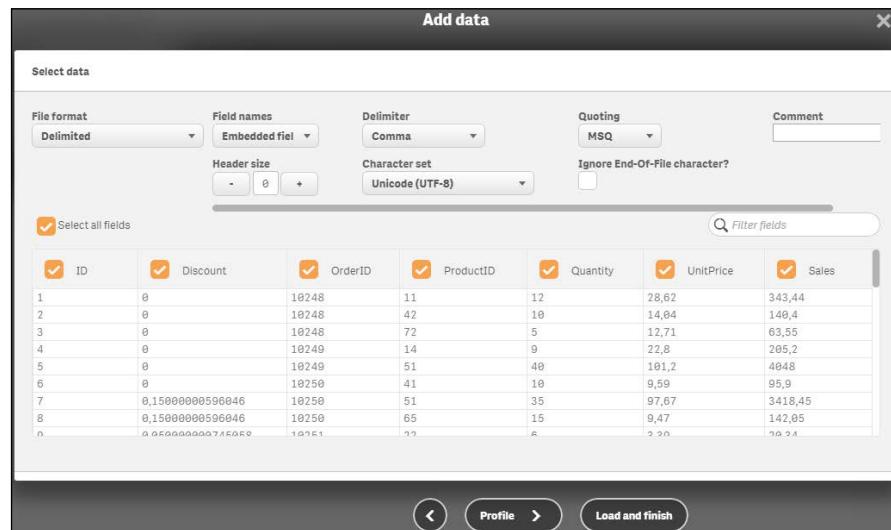
The easiest way is to use **Add data** dialog to the left. This will open the **Add data** dialog where you can define a data source; either a database table or a file, for instance, an Excel spreadsheet.



The Add data dialog

Use **Connections** for folders and connections that you have previously used, and use **Connect my data** if it is a new data source. In this dialog, you can select your database table or browse your way to a file containing a table.

Selecting a file will open a file wizard where you can tweak the details of how the file should be loaded so that you get the data you want, as shown in the following screenshot:



The file wizard

You can specify the file type, whether the file contains empty lines at the top (before the data starts), whether the first line contains the field names or not, and so on. Make sure you get all the settings right before you click on **Load and finish**. If it is the first table you load, you don't need to use the **Profile** button. The next section explains where we load additional tables.

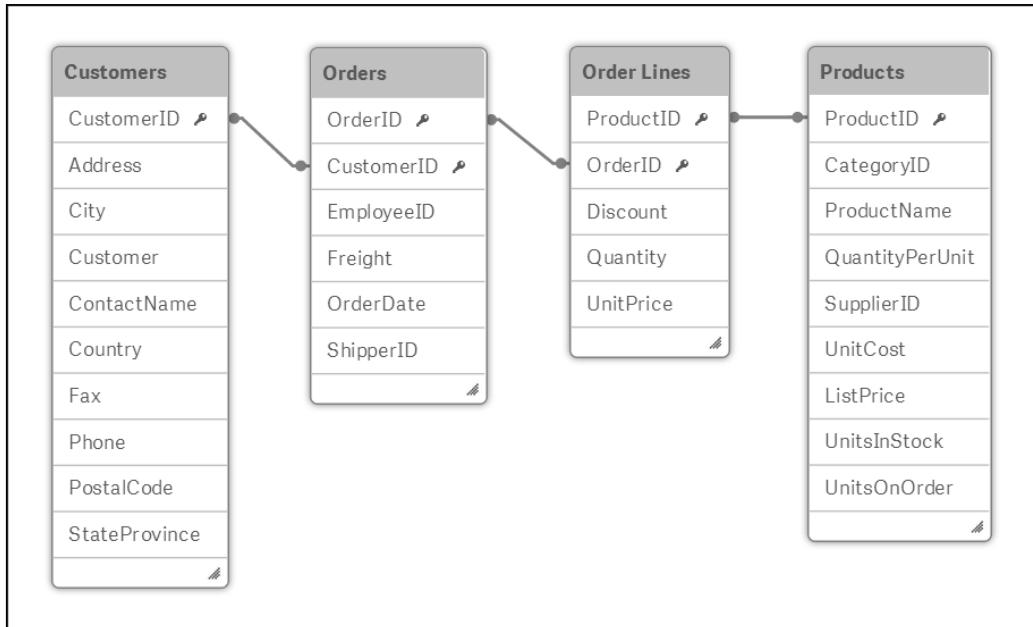
Clicking on **Load and finish** will store your settings and load the data; if everything goes well, you will get a message that the data was loaded successfully. At this stage, you can either start to edit the sheet or close the dialog and perhaps load additional tables.

Loading additional tables

It is very common that you want to analyze data that is stored in several different tables. For example, you could have four tables: one table for the orders (one row per order), one table for the customers who placed the orders (one row per customer), one table for the order lines (one row per order line), and one table for the products (one row per product).

For such a case, the `Orders` table will contain a field that specifies the customer that placed the order—a customer ID. In the same way, the `Order Lines` table will contain a field that points out which order the record belongs to, an order ID, and another field that tells which product the record refers to, a product ID. Such fields are called keys, and Qlik Sense uses these to link the tables and make sense of the data.

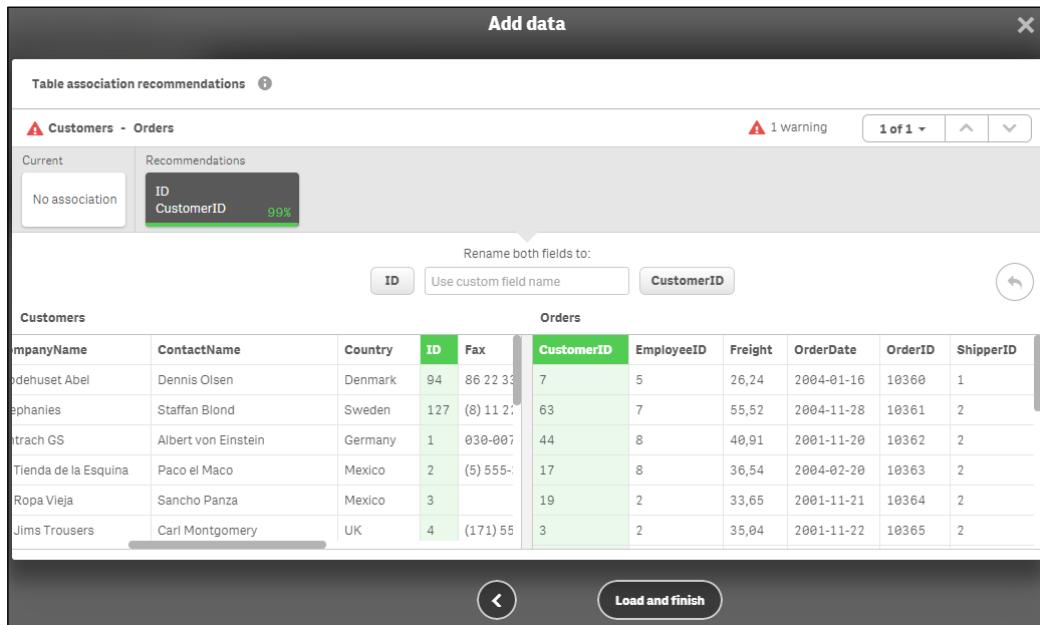
This way, all four tables are linked logically, as shown in the following picture:



A simple data model made from four tables

To obtain this data model in your app, you need to repeat the procedure for loading data that we discussed previously for each additional table. You can then open the **Add data** dialog from one of the menus to the left in the toolbar. This will take you to the familiar file wizard shown in the previous section, where you can define the file properties of the additional table.

Optionally, you can also go through the profiling step, which helps you define the keys. The profiling looks at the field values and compares these with those of the previously loaded fields. When this is done, it suggests which fields to use (or not to use) as links, and renames these fields appropriately so that they become keys in your data model:

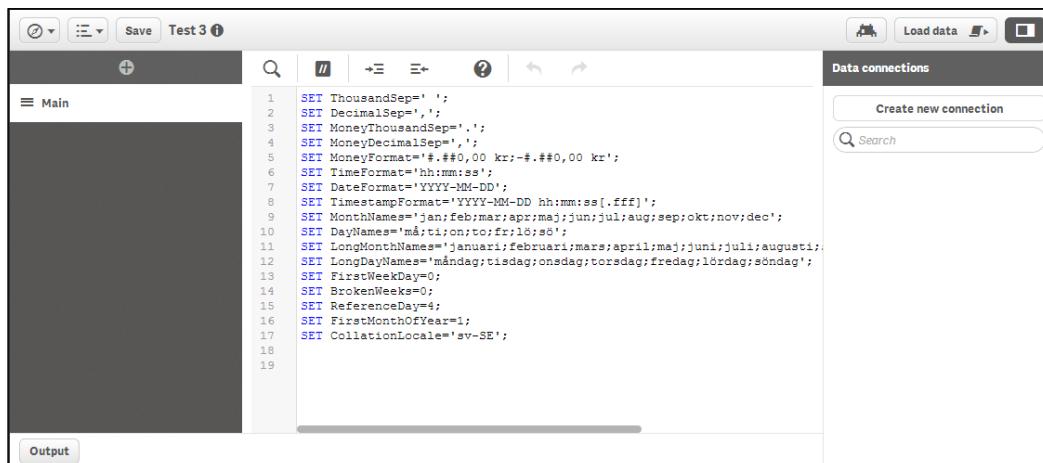


The profiling wizard

The next chapter will explain more in detail about data modeling and what you should think about when loading several tables.

Using the Data load editor

On the screen where Qlik Sense asked you to load your data, there was a second option, **Data load editor**. Clicking on this option will open a new tab with a script editor, as shown in the following screenshot:

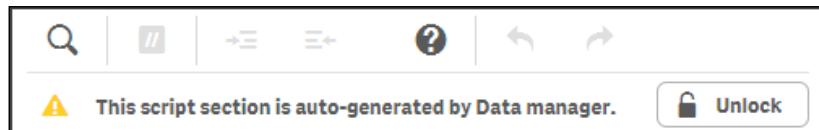


```
1 SET ThousandSep=' ';
2 SET DecimalSep(',');
3 SET MoneyThousandSep(',');
4 SET MoneyDecimalSep(',');
5 SET MoneyFormat="#.##0,00 kr;-.##0,00 kr";
6 SET TimeFormat='hh:mm:ss';
7 SET DateFormat='YYYY-MM-DD';
8 SET TimestampFormat='YYYY-MM-DD hh:mm:ss[.fff]';
9 SET MonthNames='jan;feb;mar;apr;jun;jul;aug;sep;okt;nov;dec';
10 SET DayNames='må:t;on;tir;on;tir;fr;sa';
11 SET LongMonthNames='januari;februari;mars;april;maj;juni;juli;augusti';
12 SET LongDayNames='måndag;tisdag;onsdag;torsdag;fredag;lördag;söndag';
13 SET FirstWeekDay=0;
14 SET BrokenWeeks=0;
15 SET ReferenceDay=4;
16 SET FirstMonthOfYear=1;
17 SET CollationLocale='sv-SE';
18
19
```

The Qlik Sense Data load editor

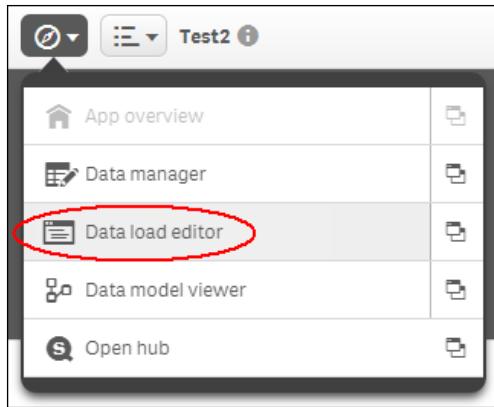
This editor is very similar to the QlikView script editor. It allows you to make very complex data transformations and basically load and transform any table. However, as with all powerful tools, it is also easy to make mistakes. Use it with caution.

When you define your data using the **Add data** command, the data load sequence will be stored in the load script as one or several `Load` statements. This means, these can be edited in **Data load editor** under the **Auto-generated** section, if you want to tweak them after they have been created. The section must, however, first be unlocked:



You can also create your script from scratch using **Data load editor**. If so, you must first create your data connections. These can be file folders, connections to regular databases, or connections to other data sources using other connectors.

This is how you do it: open the **Data load editor** from the initial dialog or the menu in the toolbar:

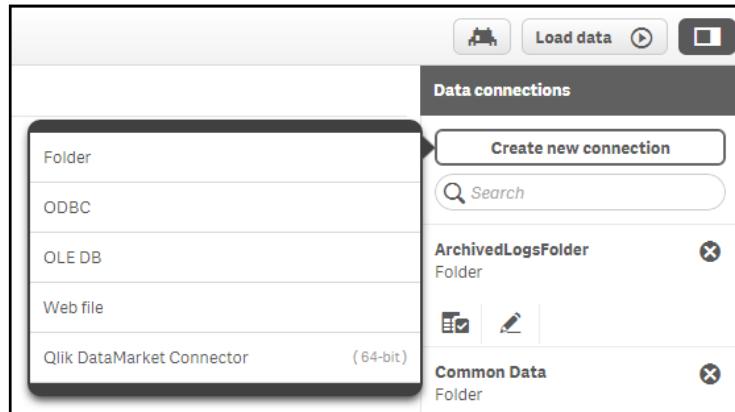


The Data load editor command



Clicking on the icon to the right in the menu will open the dialog in a new tab.

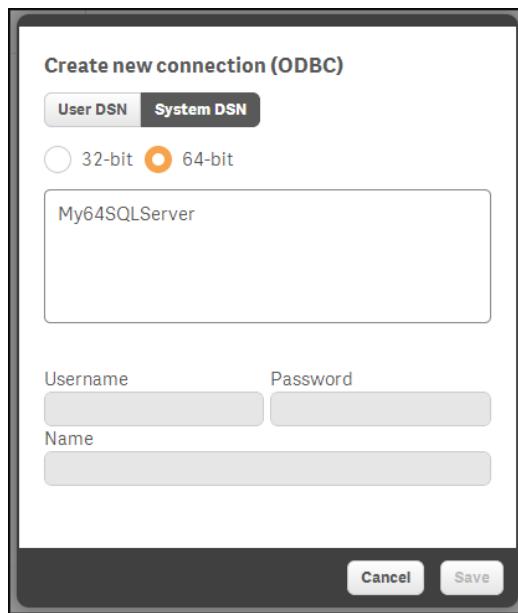
Now, you will have **Data load editor** open. To the right, you have the **Data connections** panel. If you click on the **Create new connection** button, you will open a menu, where you can choose the connection type and then specify the properties of the connection in the subsequent dialog:



Adding a data connection

Creating a database connection

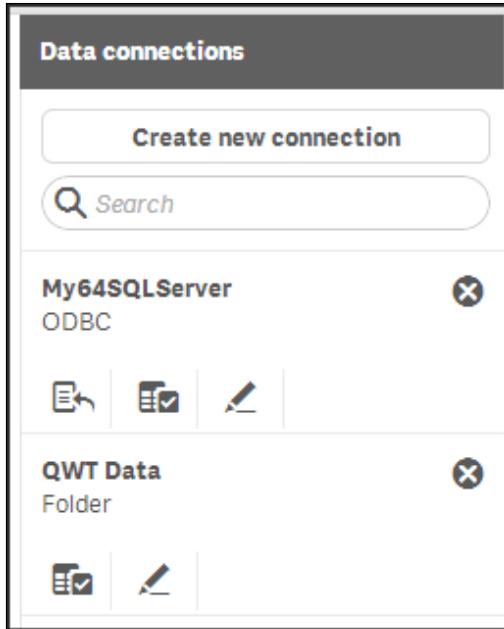
If you want to create a database connection using **Open Database Connectivity (ODBC)**, you should choose **ODBC**. This opens the ODBC connection dialog, where you can choose the data source to be used:



The ODBC connections dialog

The data sources that you see are the ones defined in the Windows operating system. This means if you do the development on a server, the list is limited to those defined by the server administrator.

Once you have created these connections, you will have them displayed in a list of data connections, as shown in the following screenshot:



The list of data connections

Database connections, for example, the ODBC connection in the preceding screenshot, have three icons. The left one creates a `Connect` statement, the middle one creates a `Select` statement, and the right one edits the connection itself. Folder connections only have two icons. The left one creates a `Load` statement and the right one edits the connection itself.

Hence, to create a `Load` statement, you should click on the left icon for a folder connection and find the file that contains the table. This way, you can create a script in very much the same way as you would in QlikView, if you are familiar with it.

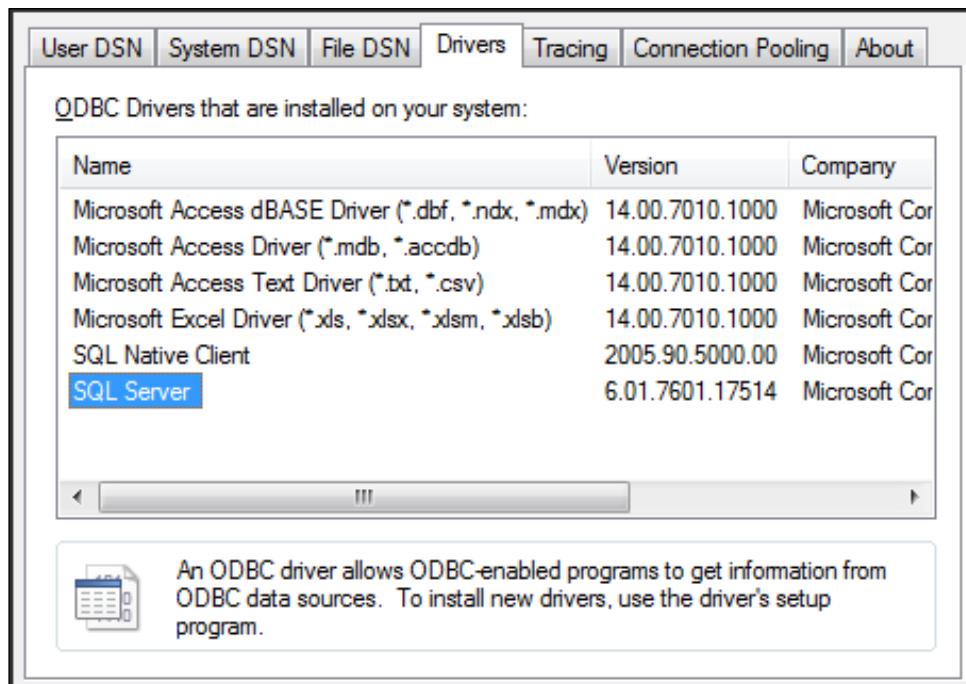
When you have created the script, you need to run it to load the data. This is done by clicking on the **Load data** button in the toolbar of **Data load editor**.

Data connectors

As you have seen, Qlik Sense can, in addition to loading data from files, connect to databases using the ODBC and OLEDB interfaces. To see which databases you can connect to, you need to open **Data load editor** and click on **Create new connection**.

When you select **OLEDB** and then **Select provider**, you will see a list of the installed OLEDB providers. If your database isn't listed, you need to install the appropriate software from your database provider.

If you choose ODBC, you will see the defined data sources. However, you may still have drivers installed for which there are no data sources defined. To find out whether this is the case, you must open **ODBC Administrator** in Windows and look in the **Drivers** tab (as shown in the following picture). If your database isn't listed, you need to install the appropriate software from your database provider:

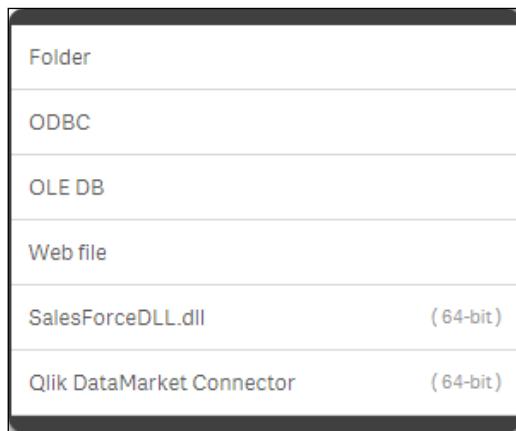


The Windows ODBC Administrator

[ The default ODBC administrator is opened by navigating to **Control Panel** | **Administrative tools** | **Data Sources (ODBC)** in Windows. However, on a 64-bit OS, you may also want to use 32-bit drivers. To manage these, you need to open C:\Windows\SysWOW64\odbcad32.exe.]

Once the ODBC driver is installed, you need to define a data source. We recommend that you do this on the **System DSN** tab in **ODBC Administrator**. When this is done, the data source will appear in the Qlik Sense ODBC dialog.

You can also use custom connectors with Qlik Sense, such as the Salesforce connector (as shown in the next picture) that you can download from the Qlik download page. These should be put in C:\Program Files\Common Files\Qlik\Custom Data. They will then appear in your list of connectors:



The list of connectors, including two custom connectors

The analysis interface—sheets and visualizations

Once you have loaded the data into Qlik Sense, it is time to create the visualizations in the analysis user interface. A basic set of sheets and visualizations should normally be supplied by the application developer, and additional ones can be created by the users themselves.

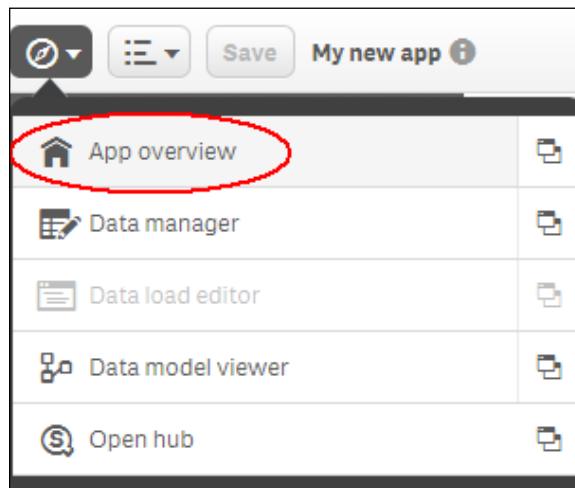
Creating a sheet

When you have loaded the data, Qlik Sense will usually create the sheet for you, and take you there. So, if you see a big blank area with the text **The sheet is empty**, you can skip to the next section:



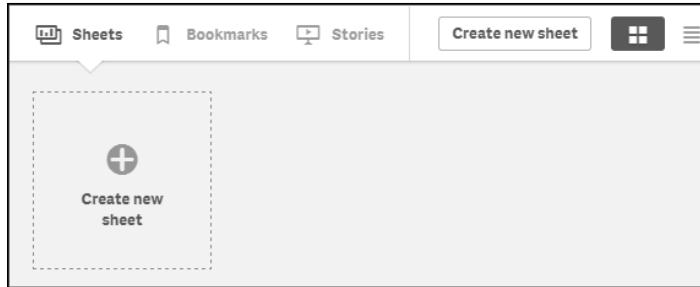
But if you are still in the Load editor, you may need to perform the following steps to create a sheet:

1. Go to **App overview** using the command in the top-left menu as shown in the next picture:



The App overview command

2. In **App overview**, you can create your first sheet by clicking on the sheet placeholder to the left, or on the button to the right:



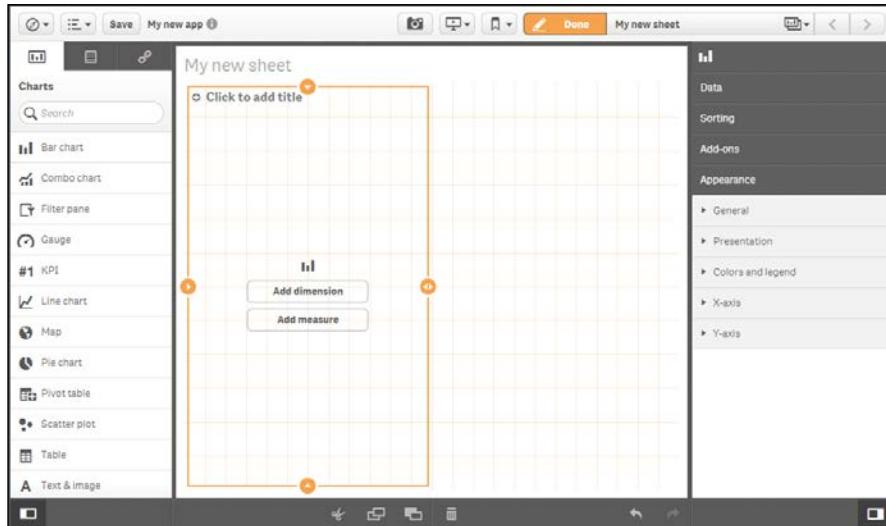
The Create new sheet button

3. Name it and hit *Enter*. You have now created an empty sheet and need to put some visualizations on it.
4. Click on the newly created sheet.

Adding visualizations

At this stage, you are probably looking at an empty sheet with the text **The sheet is empty** located in the middle.

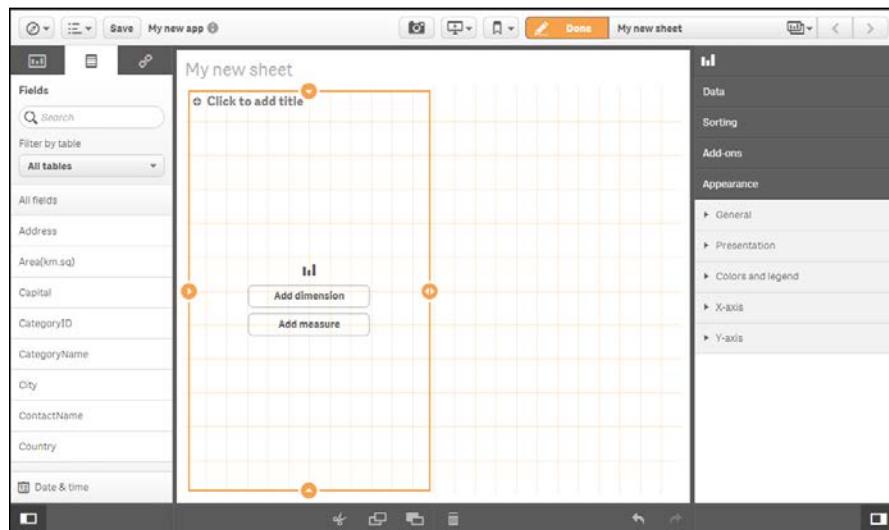
Click on the **Edit** button to the right in the toolbar to start adding things. Doing so will open the Assets panel to the left listing a number of object types: **Bar chart**, **Combo chart**, **Filter pane**, and so on. Now, you can drag and drop an object type onto your sheet, thereby creating such an object. If you, for instance, drag a bar chart onto the sheet, you will create an empty bar chart:



Depending on where you drop it, it will use all of the sheet or just half the sheet. Move the object around before you drop it, and you'll see. You can also adjust its size at a later stage.

Once you have dropped it, the bar chart will clearly show that it needs a dimension and a measure in order to display properly. You can click on the buttons on the bar chart to define these, but you can also use the Assets panel on the left.

The Assets panel shows object types, but if you look carefully, you will see that there are three tabs at its top – one for object types, one for fields, and one for the predefined library entities. So, if you click on the middle icon, you will see a list of fields that can be used as dimensions or as measures:



The Assets panel now shows a list of fields

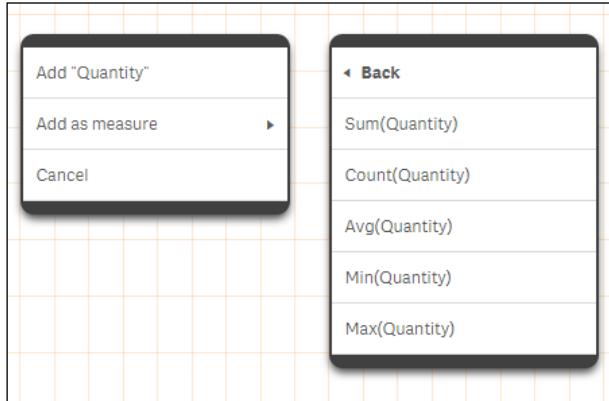
Adding dimensions and measures

You can now drag and drop fields onto the bar chart, thereby creating the dimension and the measure.

A dimension is a field with discrete values, for example, customer, product, or month. A chart will create one number per dimensional value; hence, a bar chart will create one bar per distinct value in the chosen field.

A measure is usually a number, for example, sum of sales or number of orders, and this will constitute the height of the bars.

When you drag a field onto the empty bar chart, Qlik Sense will ask you what you want to do with this field. You can add it (as a dimension), or you can use it inside an aggregation function (`Sum()`, `Count()`, or `Min()`) to form a measure:



Adding measures

Defining bar charts

When you have added both dimension and measure, the bar chart will appear as shown in the next screenshot. To the right, you will have the properties of the bar chart, where you can set its properties – the sort order, the colors, and so on. You can also define the dimension and the measure directly in the chart properties:



To see the final result, you need to click on **Done** in the toolbar, which takes you back to fullscreen.

Clicking on the **Save** button in Qlik Sense Desktop will save the application as a file with the extension **.qvf** in the application folder (`C:\Users\<user>\Documents\Qlik\Sense\Apps`). The file contains both data and script and it can be imported to other Qlik Sense installations. However, you may need to adjust the script so that it runs from the new location.

When you are done with the bar chart, you should click on **Save** and start creating your next visualization.

In the Server version of Qlik Sense, you don't have a **Save** button. The changes are saved automatically.

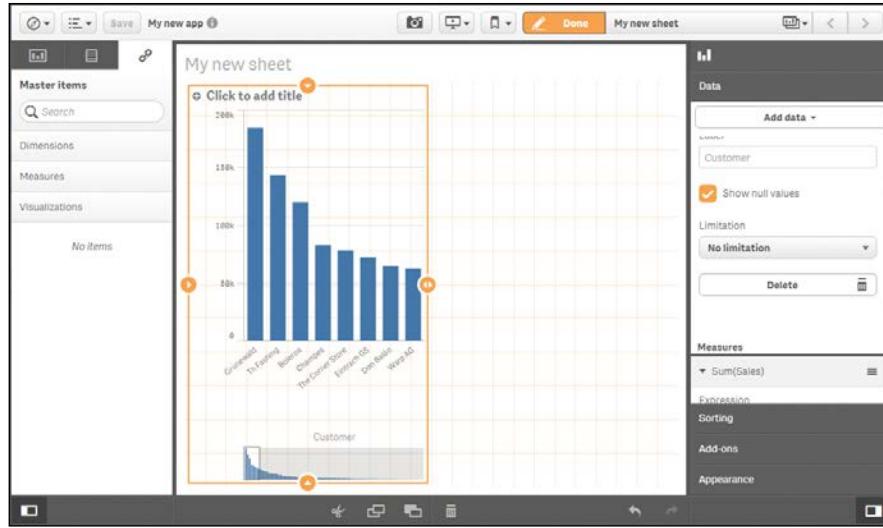
Storytelling

An exciting feature in Qlik Sense is **storytelling**. Storytelling is basically a presentation mode, where you can first prepare a presentation – like a slide show – and then present it. Storytelling is also an excellent way to present an application and create an overview of its content.

When you create an application, you can – in addition to the normal application development of course – also create a story that can be used by anyone who uses the application. However, we believe that the more common use case is that stories will be created not by the application developer, but rather by contributors – power users who choose to add elements to the application. Hence, storytelling is described in *Chapter 4, Contributing to Data Discovery*.

The application library

As previously mentioned, the Assets panel can show object types and fields. However, it has a third tab for predefined library entities. If you click on this tab, you will see the application library:



The library contains entities that have been predefined and that can simplify the Qlik Sense usage for a business user. Dimensions, measures, or entire visualizations can be stored in the library.

You do not *need* to use the library – nothing has to be predefined for Qlik Sense to work. However, if you want to reuse formulas or you have a situation where your task is to deliver an app to a business user, it is a good idea to use the library.

Which fields should be exposed?

Often, you have many fields in an app, of which maybe only a few should be exposed as dimensions. Then, you should use the library to define the fields that are appropriate as dimensions, and name them in a way that they can be easily understood.

A dimension can also be a group of fields that is exposed as a drill-down group. It can also be a formula using an `Aggr()` function that defines an array of values. In both these cases, it is a good idea to define the dimension in the library.

Defining KPIs

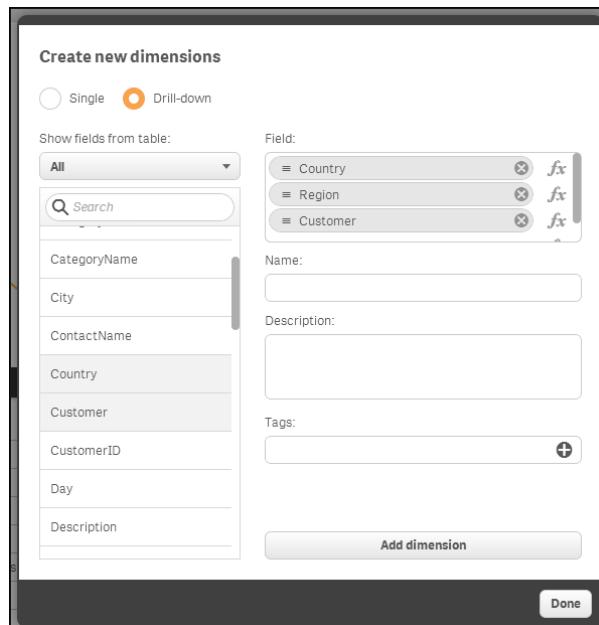
Measures are formulas that define KPIs and other numbers relevant for the analysis. These are often used in multiple places in an app, so it is convenient to store the definition in one place only. Then, should you want to change the definition, you need to do it in the library only. Also, this is a way to ensure that there is only *one version of the truth*.

Creating library entries

Library entries can be created in several different ways. The most obvious way is to enter the library and click on the **Create new** button.

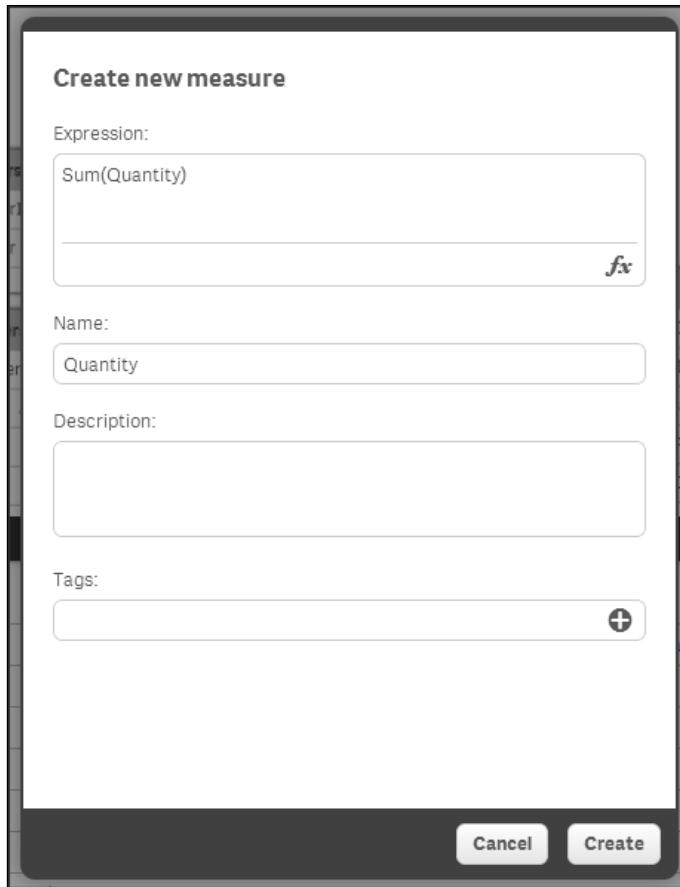
An alternative way is to do it from **Data model viewer**. Here, you can mark a field and click on the **Preview** button (in the bottom-left corner of the screen). You have the option of adding the field as dimension or measure.

Whichever way you choose to use when you create your dimension, you will see the following dialog where you define your dimensions:



The Create new dimension dialog

When you create a measure, you will see the following dialog. Make sure that you have an aggregation function, for example, `Sum()`, `Count()`, or `Min()`, wrapping the field reference:



The Create new measure dialog

Visualizations can only be entered into the library by the use of drag and drop, so you need to have created the visualization on the sheet first.

Best practices in data visualization

In the new world of ever increasing data volumes, the ability to visually communicate insights from data is an important skill set. Both the structure of an app and the chosen visualizations affect how data is perceived.

An app can contain many sheets, and the order of the sheets and what you put in them is the first consideration. The best practices can be summarized in three words: **dashboard, analysis, and report (DAR)**.

Dashboard

The dashboard is where the high-level perception takes place. It is usually the first sheet or the landing page, and it should give just the most important information and have the least amount of interactivity. Its main purpose is to help users get an overview and scan for status changes. The users can see at a glance whether things are working or not. It's a starting point, such as a table of contents; the user gets an idea of what is available and then heads off to other parts of the app based on what they have seen.

Some advice on dashboards:

- Display data only on a high level
- Don't use too many KPIs
- Use no or very few filtering options
- The most important information should be large

Analysis

The analysis pages should be more interactive: they should help users explore the data and look for answers to questions they may have formed on the dashboard page. Analysis pages are where the user spends time and interacts with the data. Typically, each sheet has a theme and covers a specific aspect of the business. The analysis pages are where the user learns from data.

Some advice on the analysis pages:

- Allow filter panes or listboxes to facilitate selections
- Make sure an entire page is about a particular topic
- Use graphs

Reporting

The third type of sheet is the reporting sheet. It is not always necessary to have these, but often it is advantageous to gather the most granular information on dedicated sheets. This is the purpose of the reporting sheets: to give the most granular information with tabular data. This is where a user can spend a lot of time sorting and filtering through the details.

Further, users sometimes want to export or print data, and the starting point is often a reporting sheet. All Qlik Sense objects can be printed or exported, either as images, as data, or bundled in a PDF document.

Some advice on the reporting sheets:

- Display transactional data in tabular form
- Give the users the ability to view every detail, so they can assess what actions they need to take

Structuring a sheet

The structure within a sheet is also important. When you create an app, it's your job to prioritize information and display it in such a way that users better understand the data and find their way in the app.

The human eye scans most content in an F pattern. The first time we see a page, we read the first line, then a bit of the second line, and then work our way down the left-hand side of a page looking for keywords. This means the content at the top of the page is the most important real-estate on a page, especially the top-left. The top of the sheet gives the users an idea of what content a page may contain and the scent of whether or not they are on the right track to finding what they are looking for. So, label the sheets appropriately.

It is also important that users easily find objects they are looking for. This applies to filter panes and listboxes, where the user makes selections. If used in several sheets, they should be placed in the same place in all sheets, if possible. Further, given how a human eye scans the page, these objects should preferably be placed to the left.

Graphs and other visualizations

Visualization also includes choosing appropriate graphs. Getting the graph right is important; otherwise, the data can be misinterpreted. There are several highly regarded thought leaders who have written excellent reading material on this topic, for example, Edward Tufte and Stephen Few. If you have not read any book in this area and you intend to build business intelligence applications, we recommend that you do this. It will help you in your work.

Dimensions and measures

Dimensions and measures are sometimes confused for one another, but it is really quite simple. You should start by asking yourself, "What do I want to show?". The answer is usually sales, quantity, or some other number. This is your **measure**.

The second question you should ask yourself is, "How many times should this measure be calculated?" or "Per what do I want to show this measure?" The answer could be once per month, once per customer, once per supplier, or something similar. This is your **dimension**.

The dimension and the measure of a chart are the core of the visualization and often indicate what visualization to choose. It is important to understand which type of field is used as a dimension. For example, when showing trends over time, you should usually use a line chart or a bar chart. The same is also true for any dimension with an implicit, intrinsic order. By the same token, you should never use a line chart unless the dimension has an implicit order.

The fields used as dimensions can be classified into the following groups:

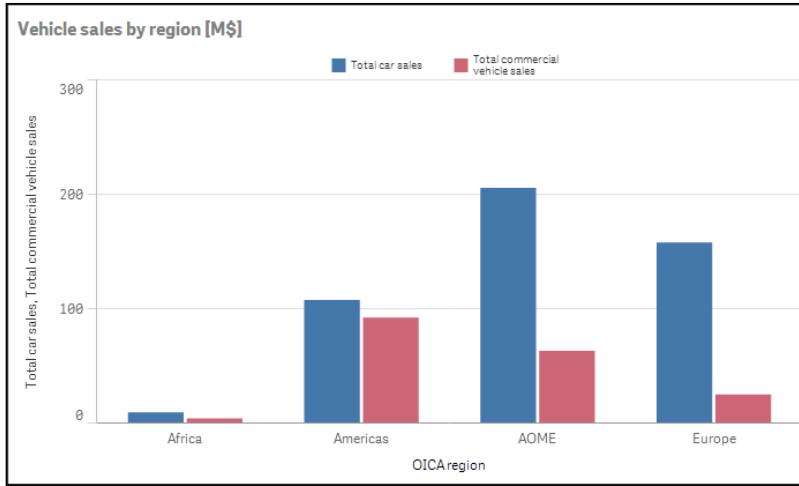
- **Nominals:** These are fields with qualitative values that *lack* intrinsic order, for example, product, customer, color, gender, and so on.
- **Ordinals:** These are fields with qualitative values that *have* intrinsic order, for example, ratings such as unsatisfied, neutral, or satisfied; that is, fields that have no numeric meaning.
- **Intervals:** These are fields with quantitative values that *lack* a natural zero. They are like coordinates, for example, date, time, longitude, temperature, and so on.
- **Ratios:** These are fields with quantitative values that *have* a natural zero. They are like amounts, for example, quantity, sales, profit, weight, and so on.

With this classification in mind, it is easier to describe what you can and cannot do with some graph types.

The bar chart

The most common visualization is the bar chart. Bar charts can be used for almost any dimension, and it is easy to compare the sizes of two bars. Further, they are good for ordinal data, since the intrinsic order can be used. This also means that trends over time can easily be spotted.

If a second measure is added to the bar chart, you will get two bar series so that you can make comparisons both between the measures and along the dimension. For example, in the following chart, you can compare not only the sales regions, but also the commercial vehicle sales with total sales:



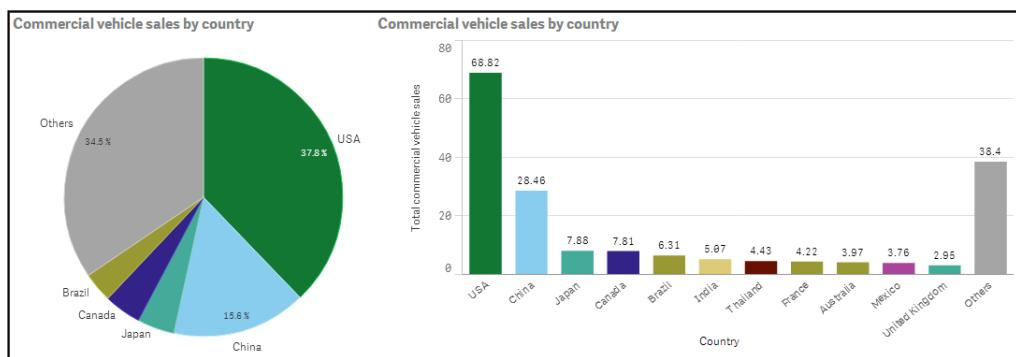
A bar chart is often the best visualization. By adding a second measure or a second dimension, you can get several series of bars.

The pie chart

The next visualization is the pie chart. This should only be used if the dimension is of the nominal type and you want to display the relative proportions. Pie charts are not good for ordinal data, since the order of the dimensional values isn't obvious.

Pie charts are, by some experts, considered a poor visualization, and a bar chart is indeed often a better alternative since it conveys the information more efficiently.

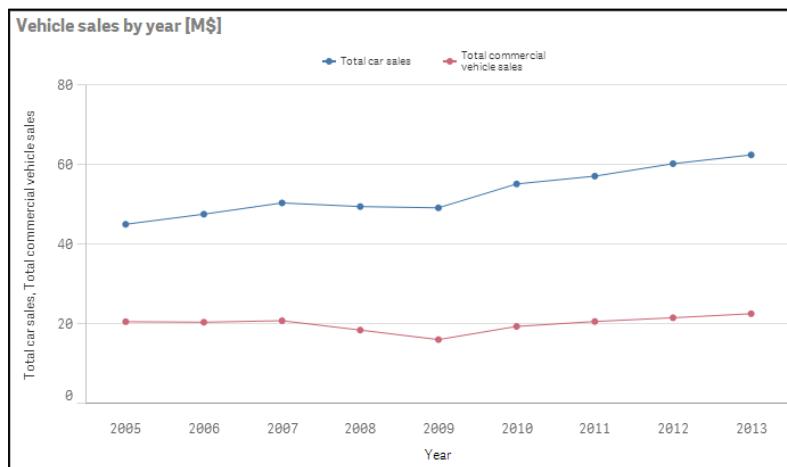
But pie charts are still useful to convey an overview of the relative sizes. For instance, in the following pie chart, you can clearly see that the combined sales in the USA and China constitute more than 50 percent – something that is not at all clear if you instead look at the corresponding bar chart:



Hence, a bar chart is often easier to read, but, in this case, the pie chart is better at showing the relative proportions of the largest countries. However, it can sometimes be hard to judge the relative sizes of the slices in a pie chart if there is only a small difference between them. Then, a bar chart is a better choice.

The line chart

The next visualization is the line chart. This should only be used if the first dimension is of the ordinal or interval type. Line charts are particularly useful for showing a change over time. Several lines can be used, either using a second dimension or by adding measures:



Line charts are good when you want to analyze trends over time.

The KPI object

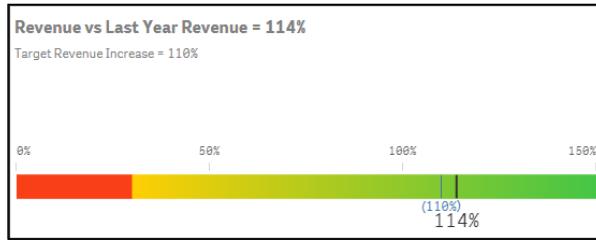
A new object in Qlik Sense is the KPI object. With this, you can display a measure or a KPI on a dashboard and label it so that it is clear what it is:



This allows you to create an overview so the user quickly and easily sees the main information. But you cannot show the number over a dimension, for example, per month or per customer, since there is no dimension in this object. It is just a number that is valid for the entire selected dataset.

The gauge

The gauge object is similar to the KPI object, in that it shows a measure and has no dimension. It should be used the same way as the KPI object—to create an overview.



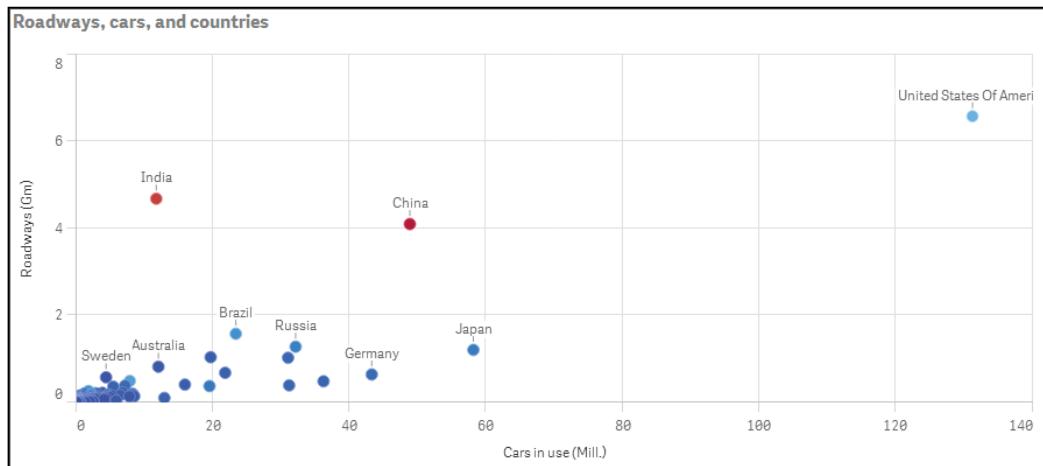
The difference with a KPI object is that a gauge can also show a basic graphical element—either a radial display like a speedometer or a linear display like a thermometer.

The scatter chart

Scatter charts are good if you want to compare two different quantitative measures for the same dimension, that is, pairs of data per some dimension. Such plots are useful to find clusters of values, linear arrangements, gaps, and values that are much different from the norm. These are the kinds of patterns that are meaningful in correlation relationships.

The unique strength of this chart type is its ability to encode values along two quantitative scales used as two axes. Note that the logical dimension is *not* used as an axis. Instead, two *measures* are used as axes.

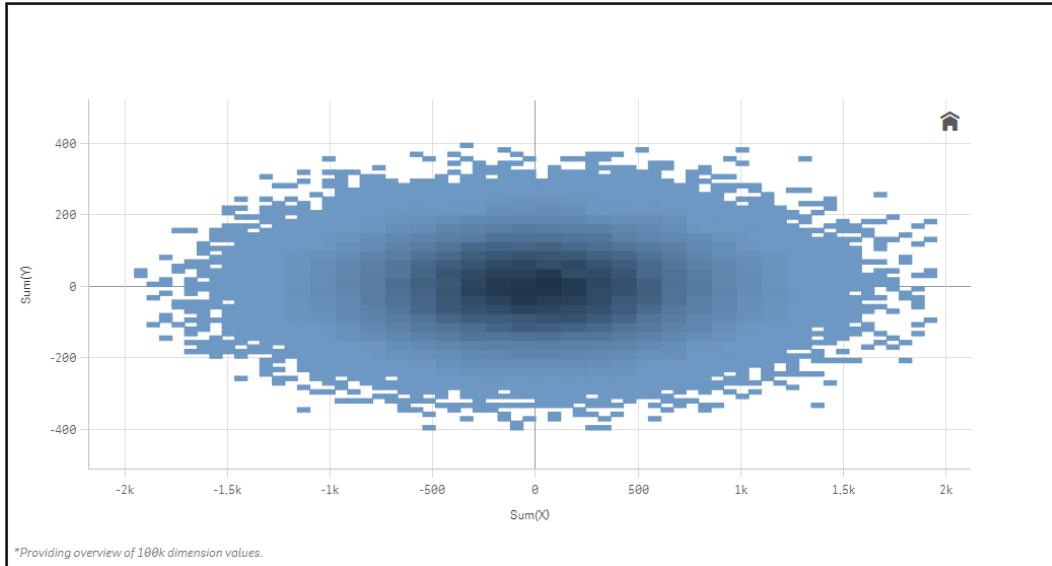
The dimension, which should normally be of the nominal type, defines the number of points in the scatter chart:



The preceding scatter chart shows the amount of roads per country (in million kilometers) versus the number of cars (in million units). The color indicates the country's population.

If you have a scatter chart with large amounts of data, Qlik Sense uses an algorithm to create an overview of the data, as shown in the following screenshot—it pixelates the data points and color codes the density of the data points.

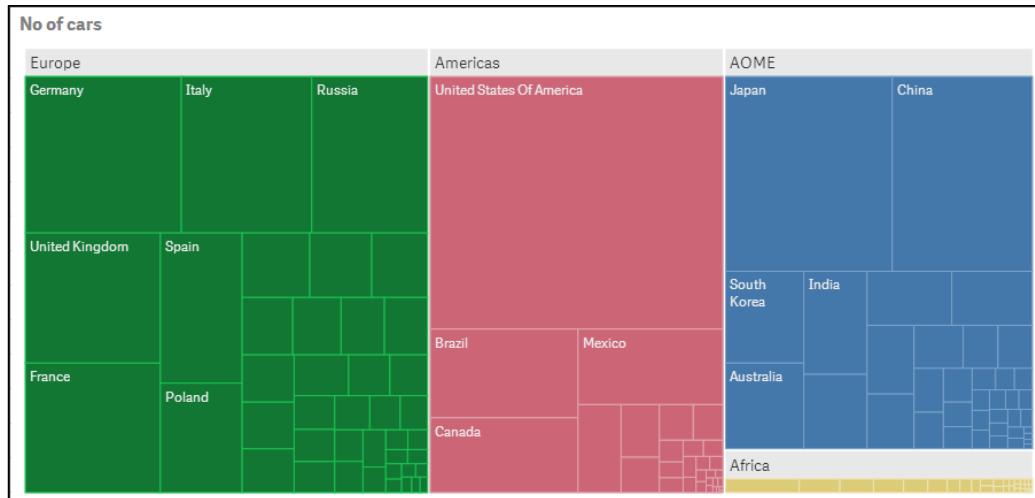
Not only does this result in a better overview of a large number of data points, it also minimizes the amount of information that needs to be transferred from the server to the client.



However, if you drill down in the chart by zooming or making selections so that the number of data points is reduced to less than 1,000, the data will be shown as individual data points.

The tree map

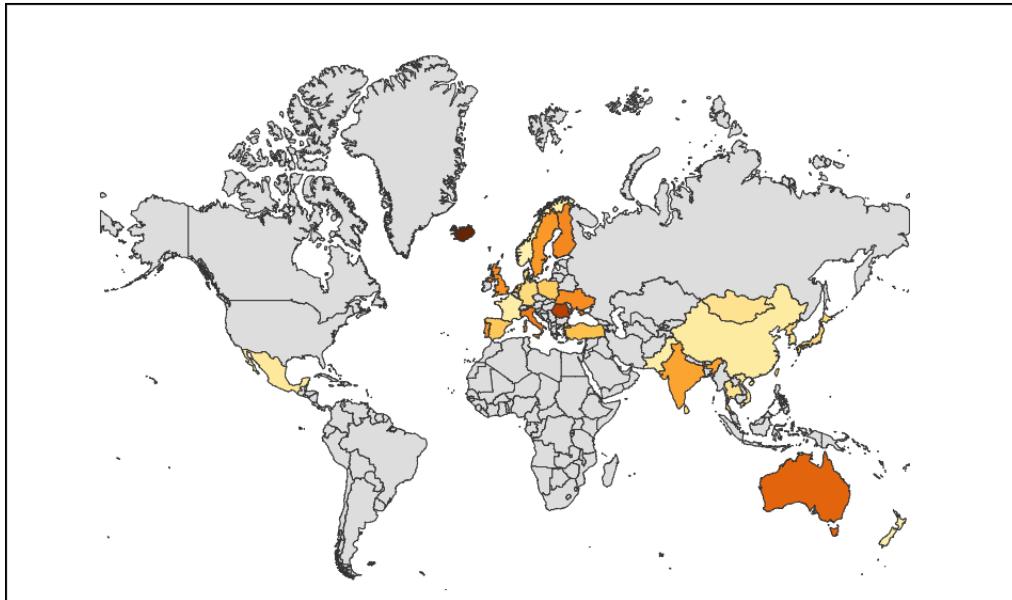
Another chart type to mention is the tree map (called block chart in QlikView). This is an excellent visualization if you have two or more dimensions and one single measure. The reason why it is called tree map is that it was originally designed to display hierarchical relationships that have a structure like a tree. In Qlik Sense, you can use it for nonhierarchical relationships that have no resemblance with trees whatsoever. Again, the dimensions should be of the nominal type:



A tree map showing the number of cars in different countries

The geographical map

The last chart type to mention is the geographical map. This visualization allows you to display regions and areas on a map of a country or region. Also, here the dimension is of the nominal type:



In Qlik Sense, you can connect maps to the data. Just as all other charts, a geographical map can be used for both input and output. Selections in other objects will affect how the geographical areas are displayed, and selections can be made directly in the map.

Tables

There are two table types in Qlik Sense – the standard table and the pivot table. They are similar to each other in that, just as charts, they can have both dimensions and measures. But they are slightly different in appearance and capabilities.

The pivot table is excellent for grouping data and showing it in a compact way. Dimensions can be used both as rows and columns, and rows can be collapsed into the above dimension:

| Sales Rep ▾ | Year ▾ Measures | | | |
|-----------------|-----------------|---------------|-------------|---------------|
| | 2006 | | 2007 | |
| | Budget \$ | Actual Amount | Budget \$ | Actual Amount |
| ⊕ Edward Smith | \$41,793 | \$114,923 | \$47,363 | \$56,069 |
| ⊖ Brenda Kegler | \$625,729 | \$1,149,330 | \$101,128 | \$199,230 |
| Oysters | \$105,807 | \$116,781 | \$17,709 | \$38,121 |
| Cheese | \$414,192 | \$729,785 | \$17,709 | \$38,121 |
| Bologna | \$264,974 | \$425,666 | \$83,419 | \$161,109 |
| Cereal | \$137,468 | \$129,823 | - | - |
| ⊕ Max Blagburn | \$1,620,736 | \$465,953 | \$1,407,075 | \$749,431 |
| ⊕ Teresa Lynch | \$476,841 | \$166,425 | \$139,496 | \$108,195 |

The standard table is different. All data is displayed in one long table, and its advantage is that you have full freedom in how to sort the data any way you want. In addition, you can make searches directly in the table:

| Sales Rep | Product Sub Group | Year | Budget \$ | Actual Amount |
|---------------|-------------------|------|-----------|---------------|
| Totals | | | | |
| Brenda Kegler | Bologna | 2006 | 264974.49 | 425666.36 |
| Brenda Kegler | Bologna | 2007 | 83419.39 | 161109.44 |
| Brenda Kegler | Cereal | 2006 | 137467.76 | 129823.25 |
| Brenda Kegler | Cheese | 2006 | 414192.45 | 729785.03 |
| Brenda Kegler | Cheese | 2007 | 17708.5 | 38120.57 |
| Brenda Kegler | Oysters | 2006 | 105806.65 | 116781.42 |
| Brenda Kegler | Oysters | 2007 | 17708.5 | 38120.57 |
| Edward Smith | Bologna | 2006 | 41792.8 | 114922.91 |

Both table types are good for showing details of the data, but to give the user an overview, a graphical visualization is better.

Sorting and colors

Once you have chosen the appropriate chart type, you should choose the appropriate chart settings, for example, scale, sorting, and appearance. Nominals should be sorted alphabetically or by some form of measure, for example, the size of the measure. The other types should be sorted according to the intrinsic sort order.

You also need to label the chart, for example, add a title, descriptions, *y* axis units, and so on.

Finally, you should also make sure to use the appropriate colors. But be careful here... bright colors are beautiful, but when it comes to data visualization, it is best practice not to use highly saturated colors. Instead, it is good to tone it down a bit. The main reason is that lighter colors are much easier on the eyes, so they show data better, for example, when displayed on large screens.

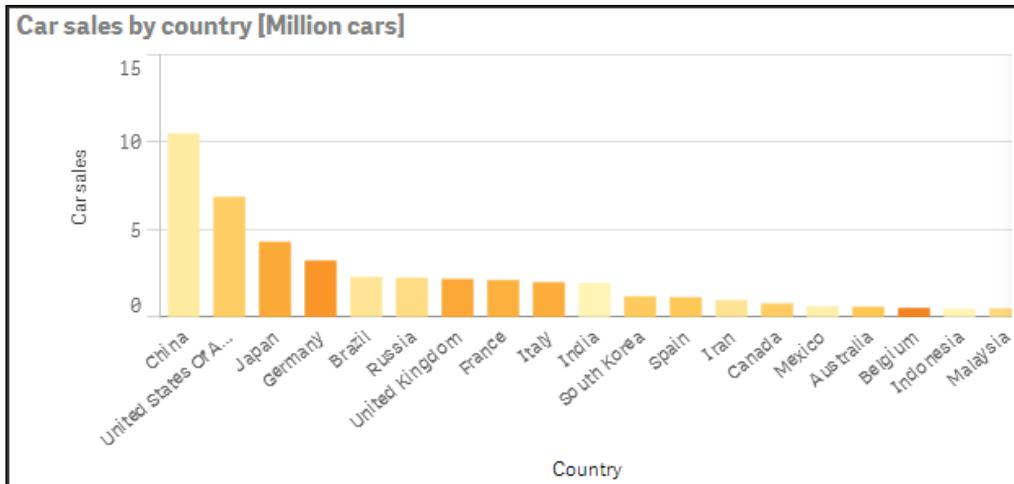
Further, bright colors draw attention, so they should only be used to highlight a specific field value or condition.



The color should *never* be a decoration only. It should *always* carry information, one way or another.

One way is to use the same color everywhere for the same dimensional value. This way, the user will easily identify the corresponding bars or slices in multiple visualizations.

Another way is to use a color that has an intensity that is in proportion to a specific calculation. For instance, in the following bar chart, the bars indicate the total number of sold cars in different countries and the color intensity indicates the number of sold cars per capita. This is one good way of using colors.



Migrating applications from QlikView® to Qlik Sense®

All QlikView applications since QlikView Version 8 can easily be migrated. However, the conversion is only partial. The data and the script will be converted, but nothing from the layout is used. Perform the following steps to migrate applications:

1. Move the QlikView app—the .qvw file—to your Qlik Sense Desktop app folder at C:\Users\<user>\Documents\Qlik\Sense\Apps. The file will then appear in your Desktop hub as a legacy app. Its name will have (qvw) after it:



2. You can now open the app and see the data model and the existing script.
3. Once you make changes, you will need to save these. This is when the conversion takes place. The old app with the new changes will be saved into a file with extension .qvf, and the old .qvw file will be renamed to *.qvw. backup.

Changes to the script

You might also need to make changes to the script. The structure of the script can remain the same, but all references to databases and files should be changed. In Qlik Sense, you need to use the data connections library. Hence, you must create the library entries that you need, and then replace connect strings and file references with references to the library.

Changes to the user interface

The modern layout in Qlik Sense with responsive design has very little to do with the old static layout in a QlikView app. A consequence is that you will need to recreate all objects – all charts, lists, and tables. In fact, you will often have to rethink your entire layout. If you are deploying your apps in an environment where users have tablets or smartphones, you would probably have had to do this anyway, since the old style QlikView apps display poorly on small screens.

The best way to do this is to have both Qlik Sense and QlikView displaying the same app simultaneously. Then you can go through the app sheet by sheet and decide how to design your new Qlik Sense app.

The syntax for formulas has not changed, so it is advisable to use copy and paste when moving complex formulas from QlikView objects into Qlik Sense objects.

Currently, there is no support for migration in the Qlik Sense server. So, if you want your old app on a server, you need to first convert it using Qlik Sense Desktop and then import the new file to the server.

Publishing your apps

When you have created an app, the next step is to make it available for other users. Perform the following steps if you have developed your app using Qlik Sense Desktop:

1. Import the app using the **Apps** sheet in the Qlik Sense Management Console. There, you will find an **Import** button at the bottom:

| Name | Owner | Published | Stream | Tags |
|----------------------------|-----------------------------|------------------|-----------------|------|
| Equity Sales Analysis | Henric Cronström (QTSEL...) | | | |
| Executive Dashboard | Henric Cronström (QTSEL...) | | | |
| Executive Dashboard | Henric Cronström (QTSEL...) | 2014-09-16 14:48 | Management | |
| License Monitor | sa_repository (INTERNAL...) | 2014-09-08 12:47 | Monitoring apps | |
| License Monitor | sa_repository (INTERNAL...) | 2015-05-27 12:19 | Monitoring apps | |
| License Monitor_2.0.0.0 | sa_repository (INTERNAL...) | | | |
| MFA Interactive | Henric Cronström (QTSEL...) | 2014-09-21 16:44 | Everyone | |
| MFA Pharma Interactive | Henric Cronström (QTSEL...) | | | |
| My new app | Henric Cronström (QTSEL...) | | | |
| Operations Monitor | sa_repository (INTERNAL...) | 2014-09-08 12:47 | Monitoring apps | |
| Operations Monitor | sa_repository (INTERNAL...) | 2015-05-27 12:19 | Monitoring apps | |
| Operations Monitor_2.0.0.0 | sa_repository (INTERNAL...) | | | |
| Procurement | Henric Cronström (QTSEL...) | | | |

2. Once the file is imported, you may also need to assign the correct owner: mark the file, click on **Edit**, and change the owner, if necessary. The file will then appear under **My Work** in the Qlik Sense hub.

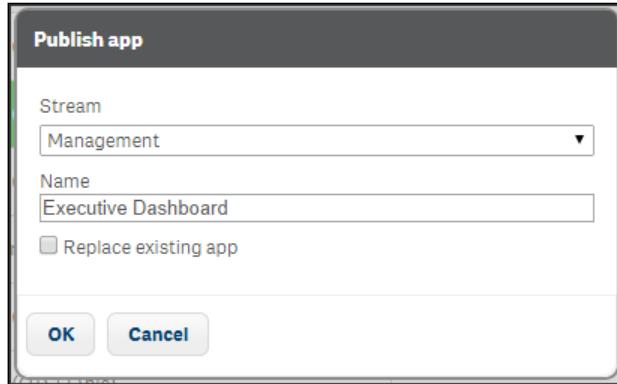


If you have developed the app using the Server version, it will already be under **My Work**.



3. Before you publish the app, it is recommended that you make a copy of the file. Mark the file on the **Apps** page in the Qlik Sense Management Console and select **Duplicate** in the **More actions** menu.
4. Now, you are ready to publish. You can publish the app by marking the file and clicking on the **Publish** button.
5. Publishing means that you move an app from your personal workspace to a stream, which means the file can be accessed by many people. The properties of the stream, including security rules, will then be applied. Note that the file will be removed from your personal workspace. This is the reason why it is a good idea to make a copy of it.

6. When publishing, you will be prompted to choose a stream for your application:



7. Currently, there is no way to "unpublish" an app. If you want to remove it from a stream, you need to delete the app.

Summary

In this chapter, we looked at the functions and commands you need to know to build engaging applications—both from a process perspective and, more practically, what you need to do to load the data and create an inviting user interface.

In the next chapter, we'll move into the basics of data modeling, which is an area you need to master in order to make advanced applications with multiple source tables.

6

Building Qlik Sense® Data Models

In the previous chapters, we looked at how to create a definition of which data to load. But we did not look at which considerations you should make on how to load and link different tables.

So, in this chapter, we will discuss the following topics:

- Data modeling
- How a data model reflects business processes

The QIX engine

The QIX engine is the core of the logic in Qlik Sense. All evaluations and calculations are made by this engine.

Every user selection implies a new logical situation. Other field values than the ones used earlier are possible; summations need to be made, so the charts and the KPIs get different values than what we had before. Everything needs to be recalculated, and the data model defines how this is done.

When the user makes a selection, Qlik Sense first evaluates which field values of other fields are possible and which rows are possible. In this evaluation, the key fields are used to propagate the selection to other tables. This is *the logical inference*. The second step is to calculate all formulas based on the possible rows. In this step, all *aggregations* are made.

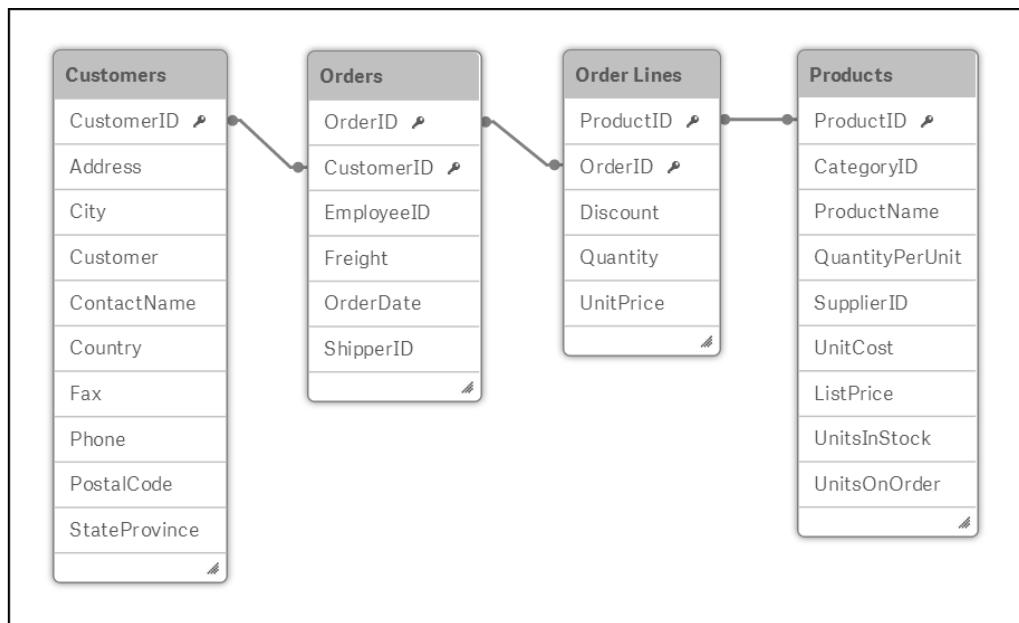
The data model defines *how* the QIX engine's logical inference and calculations will be made.

The Qlik Sense® data model

Data used in Qlik Sense needs to be in a tabular form, very much like a table in Excel. A column in the table is referred to as a *field* and every row is a *record*.

When data is loaded into Qlik Sense, it is stored in the QIX internal database. In the simplest case, the data is just one single table. However, more commonly, the data model consists of several tables with links between them. These define how the different tables relate to each other. It is, hence, a *relational model*.

In the previous chapter, we saw an example where four tables were used:
Customers, *Orders*, *Order lines*, and *Products*:



A simple data model made from four tables

This is in fact the core of a very common business application – a sales analysis based on the registered orders.

The structure is not a coincidence. Rather, the reason why it looks the way it does is that it is a reflection of the real business processes. The relations these four entities have in reality dictate the data model:

- A *customer may over time place several orders*. Hence, customers and orders should be stored in different tables, and the customer ID should be stored in the *Orders* table.

- *An order may contain several order lines.* Hence, orders and order lines should be stored in different tables, and the order ID should be stored in the Order lines table.
- *Several different order lines can refer to the same product.* Hence, products and order lines should be stored in different tables, and the product ID should be stored in the Order lines table.

You could add a number of additional tables, and for each table you will need to ask yourself what its relation is with the already existing tables. The new table could be a list of countries to which the customers belong, of product groups, of shippers, of suppliers, of invoices, and so on. The business processes define whether a customer can belong to more than one country, whether a product can belong to more than one product group, and whether a single invoice can refer to more than one order.

Hence, you should look at the real-life entities and their relationships to understand how they should be modeled in the Qlik Sense data model.

Creating a multitable data model

Loading several tables is technically just a matter of using several Load or Select statements in the script. Joins, in the way a database manager knows it from SQL, are usually not needed.

Normally, each Load or Select statement creates one table in the data model. Hence, if you want to load four tables, you should have four different Load or Select statements, each defining the appropriate table.

There are, however, some exceptions to this rule:

- If a loaded table contains exactly the same set of fields as a previously loaded table, the new table will *not* be created as a separate table. Instead, the loaded records will be appended to the existing table, which most likely is what you want. This way you can add more data to an existing table.
- If the Load or Select statement is preceded by the Concatenate or Join keywords, the loaded table will be merged with an existing data table. The Concatenate keyword is used if you want to add records to the table, just like in the previous bullet, but the two tables have slightly different sets of fields. The Join keyword is used to create the product between two tables, that is, the same as a JOIN in SQL.
- If the Load or Select statement is preceded by the Generic keyword, the loaded table will be transformed into several data tables. This is a keyword you need to use if your table is a generic database, that is, if the second to last column contains an attribute name and the last column contains the attribute value.

- If the Load or Select statements are preceded by the Mapping keyword, the loaded table will not be used as a normal data table. The table will be used for other purposes.
- A previously loaded table can be deleted using the Drop command. This is especially useful if you make many transformations and need temporary tables.

Linking tables

Further, when loading multiple tables, the links between the tables are defined by the field names. At the end of the script run, the existing tables will be evaluated. If the same field name is found in more than one table, this field will be considered to be a field that links both the tables. This way, a data model is created. The logic in the script evaluation is hence identical in Qlik Sense and QlikView.

This means you need to make sure that the fields you want to use as links between the different tables, the key fields, are named the same in all tables you want to link. You can do this using the Profiling dialog (refer to the previous chapter) or you can rename them yourself by editing the script.

For instance, if the key is called CustomerNo in one table and CustomerID in the other, you could rename them like this:

```
Load CustomerNo as CustomerID, ... From Table1 ... ;  
Load CustomerID, ... From Table2 ... ;
```

Also, it is important that you make sure that you don't have any unwanted links. For example, you may have a field called Description in two different tables. This is of course not a key, but rather just a short name for something that in one table may be a product description and in another a customer description. Also, here you need to rename the fields:

```
Load Description as Description1, ... From Table1 ... ;  
Load Description as Description2, ... From Table2 ... ;
```

The goal is to create a script that defines a logical, coherent data model that corresponds to the business situation.

Structuring your data

In a database, there are rules about where different entities are stored. For instance, everything about the customers should be stored in the `Customers` table. A customer identifier is stored in this table, which means that the necessary data can be retrieved by a simple lookup in the `Customers` table. So, if you need to refer to a customer from another table, you just store this identifier in the other table.

Normalization

The identifier needs to be unique in the `Customers` table, so that every record can be addressed. Here the field is called a *primary key*. In other tables, duplicates may exist. For example, several records in the `Orders` table might refer to the same customer. If this is the case, the key is called a *foreign key*.

A consequence of this setup is that a database can have many tables, often thousands. One table for customers, one for orders, one for order lines, one for products, one for product groups, one for shippers, one for invoices, and so on. Each table contains one entity type, and each instance of this entity has exactly one row in the table – one record.

In the customer example, it means that each customer is stored in one row only, and all the attributes of this customer are stored in the different columns of this row. This is called *normalization*.

The main goal with normalization is to avoid redundancy. In a transactional database, it is important that a piece of information is stored in only one place. Not only does it take less space, it also speeds up the update process and minimizes errors. You can imagine the confusion if, for instance, a customer address is stored in several places and the different occurrences contain different addresses. Which version should you trust?

So, the source data is often highly normalized. But does this mean that the Qlik Sense data model should be normalized too?

Yes and No.

The Qlik Sense data model does not need to be as normalized as the source data. Redundancy is not a problem, and if duplicating data improves the user experience, you should definitely consider using duplicate data. The data model should be optimized for user experience and performance, not for minimal size.

But *some* normalization has great advantages – structuring the data in entities (tables) simplifies the data modeling work. It also makes maintenance simpler, since a data model and a script can be understood by a developer who has never seen it before. Finally, the QIX engine works better with a normalized model. It is easier to make it calculate numbers correctly and avoid double counting, which is a real problem when you start to de-normalize.

So the bottom line is that you should have *some* normalization, but it does not need to be as strict as in the source data.

The main case in which you need to de-normalize is if you use the same entity in different places in the data model.

For example, you may use a table listing external organizations in the context of supplier, shipper, as well as customer. This means the Organization table is used in different roles. In such a case, you should load the organization table three times: first as a supplier, then as a shipper, and finally as a customer, linking to the three different foreign keys.

Another common situation is that you have several dates in your data model: OrderDate, RequiredDate, ShippingDate, and InvoiceDate. In other words, the date is used in different *roles*. Also, here you should load the dimension – the calendar – several times, once per date field.

Another reason to de-normalize is for optimization purposes. One of the cases would be if you have several very large tables linked to each other, for example, if you have an order headers table as well as an order lines table, and both are large (millions of records). From a performance perspective, this is not optimal. The QIX engine will need more CPU cycles than if the information of the two tables had been stored in one single table. So, you might want to join both the tables for performance reasons.

A small word of warning though, joining tables is not always safe. This operation may lead to a record being duplicated on multiple records, and if the record holds a number, the summation made by the QIX engine will lead to an erroneous result – the number will be counted twice. In the case of order headers and order lines, you know that an order line can belong to one order header only, so the numbers in the order lines table will not be duplicated. Hence, it is safe to join here.

However, if you have a number in the orders table, it will be duplicated. But luckily, this table rarely contains any numbers.

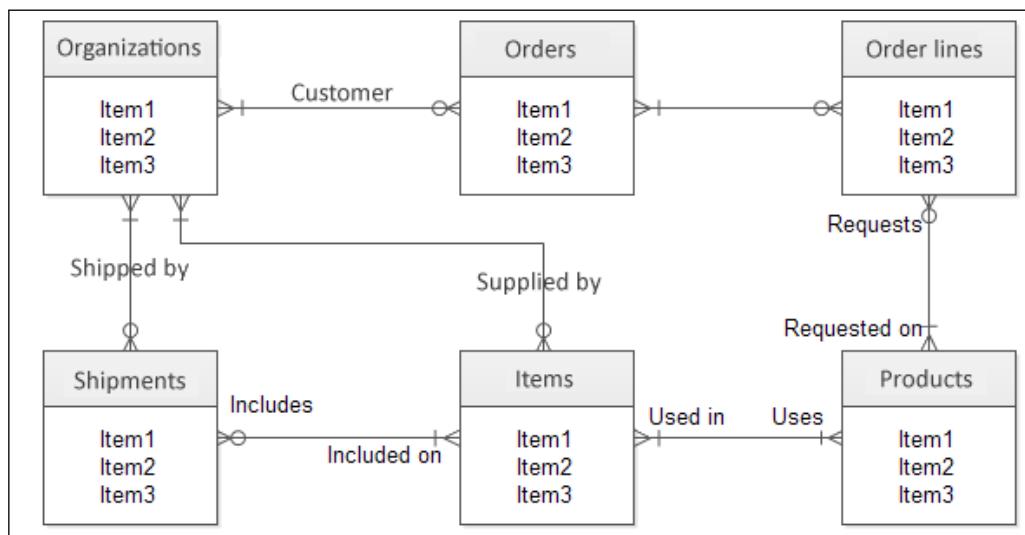
Star schema and snowflake schema

The normalization is usually quite different in the source data model and in the analytical model. For the source data, one often talks about *transaction tables* and *master tables*. The transaction tables are the ones that store orders, invoices, and other transactions. In these, new records are typically added every hour, minute, or even second.

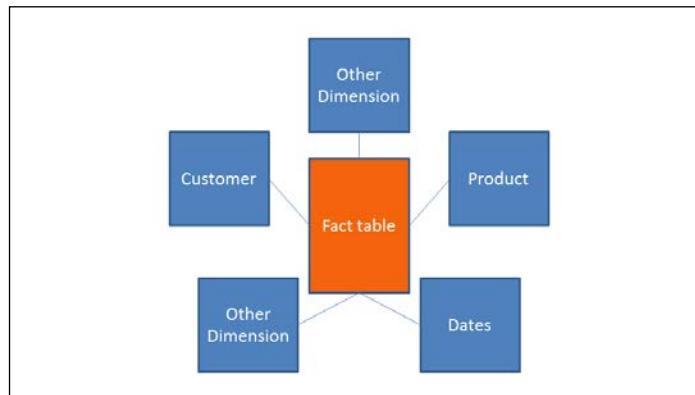
This is very different from the master tables, where new records are added much more rarely: *Products*, *Customers*, and the *Calendar* are typical master tables.

Master tables are often used for many purposes and are usually linked to several transaction tables, which makes the data model look as if it has circular references. This is, however, not a problem, since every link means a separate lookup in the master table.

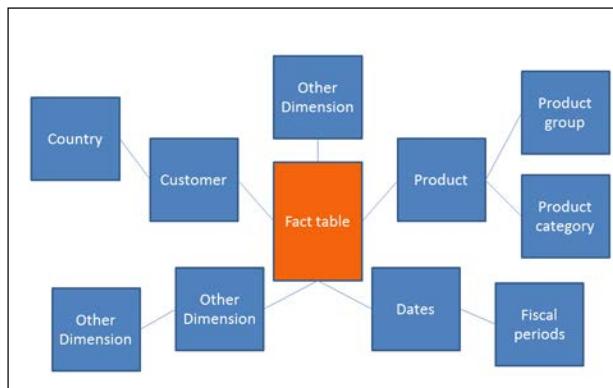
For example, in the following relational model, the *Organizations* table is linked to the transactional data through three keys: *Customer*, *Shipped by*, and *Supplied by*. This means that a specific shipment item can be linked to several organizations—one customer, one shipper, and one supplier:



In an analytical model, the tables are used in a different way. The transactions are joined into one single *fact table*, and the master tables are loaded as *dimensions*. The reason for this is partly historical. Older hypercube tools could not work unless all metrics were collected into a fact table. In addition, they could use hierarchical dimensions surrounding the fact table. The model will then look like a star; hence the name *star schema*:



This model has only one layer of dimensions—all the tables are directly linked to the fact table. But if the model instead uses dimensions in two or more levels, the model is called a *snowflake schema*:



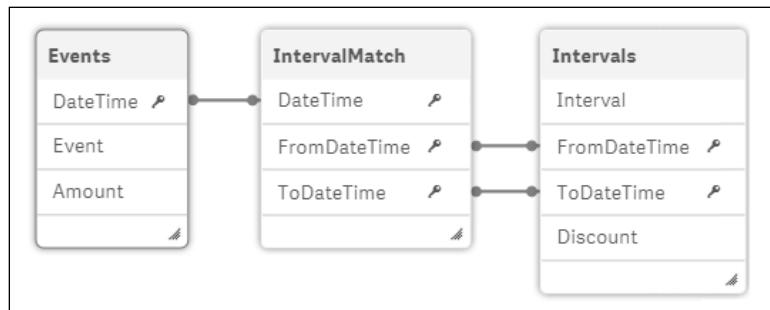
For Qlik Sense, the difference is minimal. All three data models can be used—provided that master tables used in several roles are also loaded several times. It is also possible to have metrics in any table, not just the fact table.

A star schema is, however, both simple and efficient, so we strongly recommend using this as an initial model for your data. It is then easy to build further and make it more complex.

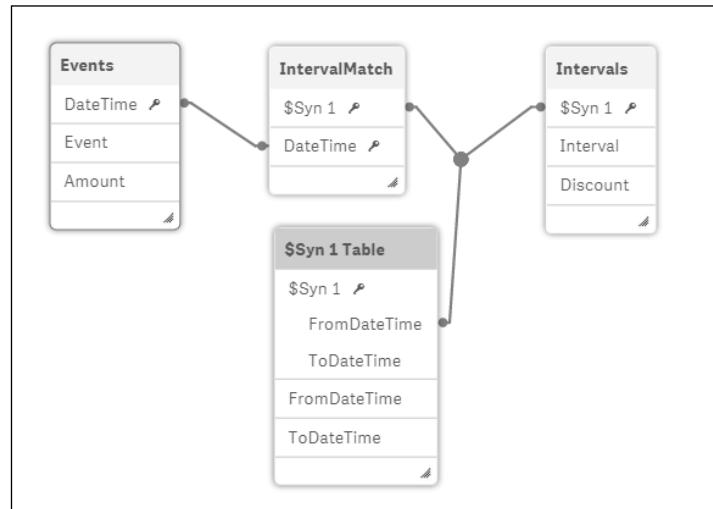
Pitfalls in the data model

When you create your data model, you should look out for two potential problems: synthetic keys and circular references.

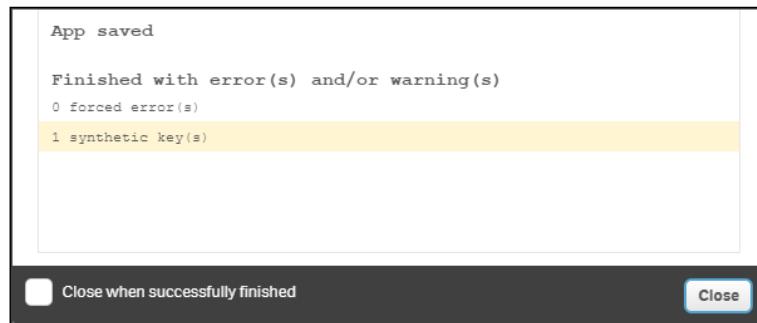
Synthetic keys are automatically created if you have multiple keys between two tables. They are not by themselves bad, but they could be a warning sign of a bad data model. If you have made a mistake when loading the data, the first sign is often one or several synthetic keys. Then, you need to go back and investigate why this has happened:



The preceding diagram shows a synthetic key modeled the way you loaded the data. It is a correct one that you don't need to change. Internally, it is stored differently; refer to the following diagram. In the data model viewer (as shown in the following diagram), you can toggle between these two views:

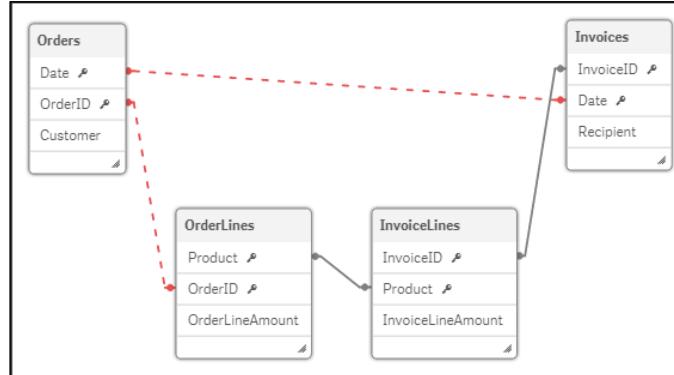


Qlik Sense will, at the end of the script run, warn you about these potential problems, as shown in the following screenshot:

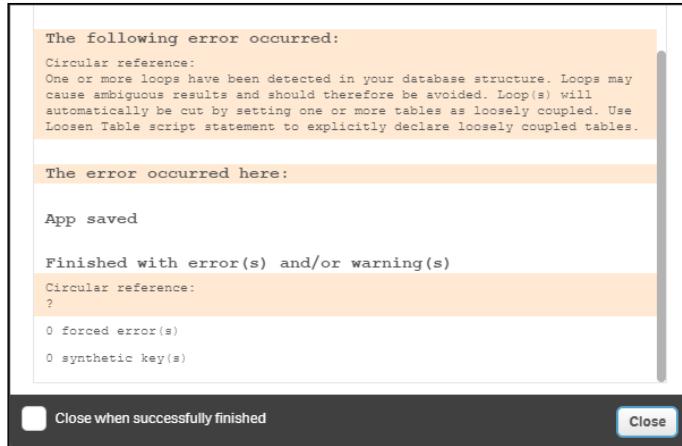


Usually, you do not want synthetic keys—you want one single key in each table link. However, if you know that you have multiple keys linking two tables, and that this is the way you want your data model, then there is no problem in having synthetic keys.

The second potential problem is circular references. This happens if you load data in such a way that the links between the tables form a loop. The following diagram is a typical example:



The circular reference from a data modeling perspective is an *error* and not just a warning, and you will get an error message at the end of the script run:

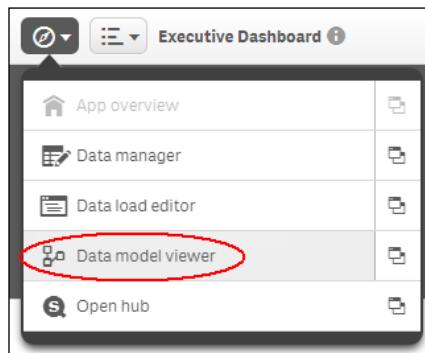


If you have a circular reference, you must rethink your data model with the goal of removing the loop.

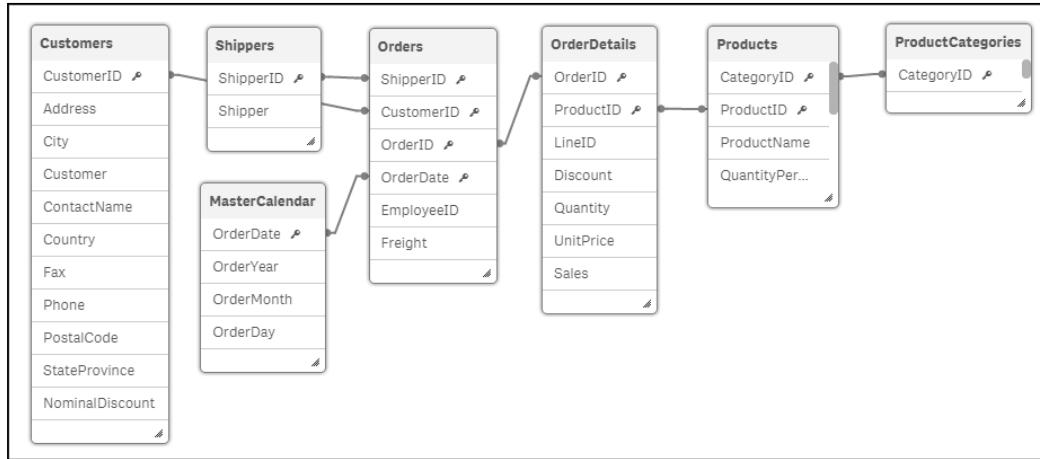
It could sometimes be difficult to figure out how to remove a circular reference, but a good advice is to look at every link in your data model and ask, "Are these two fields *really* the same thing? Or do the fields have different roles?." In the preceding screenshot, you have a circular reference where the Date field has two different roles: one is the date when the order arrived, and the other is the date when the invoice was sent. These two dates need not necessarily be the same. Hence, they should not be linked but instead loaded as two different fields.

The data model viewer

The script defines the data model, but if you want to view it graphically, you should use data model viewer. This is opened from the toolbar menu, as shown in the following screenshot:

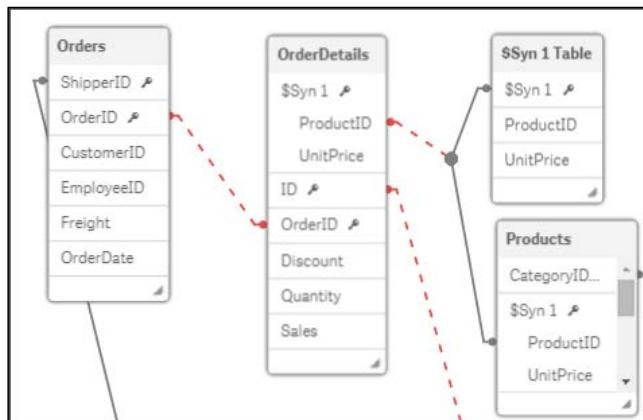


Clicking on the icon to the right in the menu will open **Data model viewer** in a new tab. Once this is open, you can visually see what the data model looks like. If you have more than one table, they should be linked by the key fields and should look something similar to the following screenshot:



The **Data model viewer** is an excellent tool to get an overview of the data model. It is also a very good debugging tool in the application development process.

Check whether the data model looks the way you want it to and make sure that you have no circular references. Circular references will be marked with red links and synthetic keys will be named \$Syn:



Using preview mode

The **Data model viewer** option has a useful additional feature, preview mode. This mode allows you to preview both the data and metadata of a field.

Select a field in a table and click on the **Preview** button to the lower-left corner of the screen. This opens the **Preview** panel in which you can see data about the field and some sample data records from the table. In addition, you can define dimensions and measures based on the chosen field, as shown in the following screenshot:



Summary

In this chapter, we looked at the functions and commands you need in order to create a logical and coherent data model that reflects your business processes.

In the next chapter, we'll move away from app creation and start examining how a Qlik Sense server can be deployed in the cloud.

7

Qlik Sense® Apps in the Cloud

In the previous chapters, we looked at how to go about creating Qlik Sense apps.

In this chapter, we look at sharing those apps with colleagues and friends in the cloud. We will cover the following topics:

- Why using the cloud makes sense
- Sharing Qlik Sense Desktop apps in the cloud
- Creating Qlik Sense apps directly in the cloud
- Maintaining Qlik Sense Cloud apps
- Adding context through external data from Qlik DataMarket

Why use the cloud?

Deploying software in the cloud removes infrastructure challenges and provides access to data and computing power, software, and services from virtually anywhere. For Qlik Sense Enterprise customers, this might mean a large organization choosing to deploy Qlik Sense apps on a *private* (internal) cloud, using a platform such as AWS, for example. This could enable them to scale up their applications in a cost-effective and flexible manner. Deploying using a private cloud model is out of the scope of this chapter; however, it instead covers how Qlik Sense Desktop users can also get the benefits of the *public* cloud in two ways—through sharing apps and adding data content.

Cloud sharing

The key value of cloud enablement for Qlik Sense is bringing together data and people by deploying apps in the cloud. This fosters collaboration through the sharing of insights and allows more people to create and discover through Qlik Sense.

Qlik Sense Cloud allows developers to share dashboards and storyboards for free via the Web for up to five users. Qlik Sense Cloud also offers the ability to create apps in the cloud, removing the need to use the desktop, and meaning that apps can be developed without the need to download a Windows executable, on any device running an HTML5-enabled web browser.

Cloud content

Qlik Sense developers can use the cloud as a source of data to broaden the scope of their Qlik Sense apps via Qlik DataMarket.

Qlik DataMarket provides and integrates third-party data with the Qlik Sense visual analytics platform, much of it for free. With data-as-a-service, app developers can use a comprehensive library of external data directly within Qlik Sense, allowing them to augment and cross-reference their internal data to gain context and so drive new insights.

Qlik DataMarket comes preconfigured with many ready-to-use data sources via a subscription model. Data sources include business demographics, currencies, population, economic indicators, development indicators, and weather, all offered at various levels of granularity, some down to postal code.

Using Qlik Sense® apps in the cloud

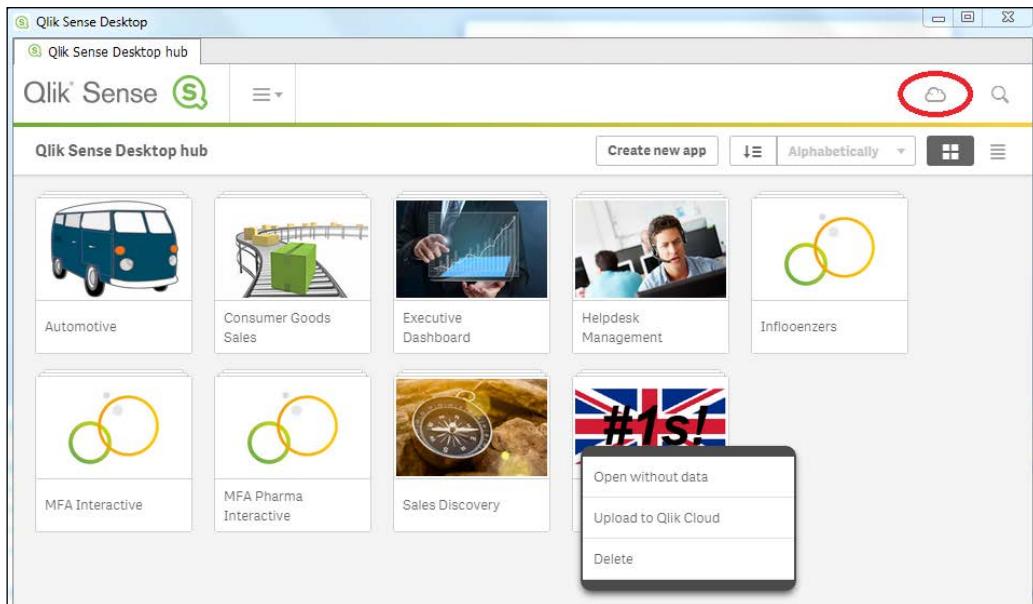
So, you've built a Qlik Sense Desktop app and you want to share it with others, either colleagues or friends, so you can find insights together. The cloud is the best way to do that, and doing so is built directly into Qlik Sense.

Qlik Sense isn't just for work, playing with data can be fun too. In this chapter, I will share a Qlik Sense app exploring UK music singles chart data going back to 1952.

Uploading an app from the desktop

To upload an app, you first need to log on to Qlik Sense Cloud on the Web. From the Qlik Sense hub, there are two ways to open Qlik Sense Cloud: either by clicking on the cloud image in the top-right corner of the window or by right-clicking on the app itself. Both the actions will take you to the Qlik Sense Cloud Hub in a browser. At this point, if you haven't done so already, you'll have to register.

The following screenshot shows the option for in-built cloud uploading:



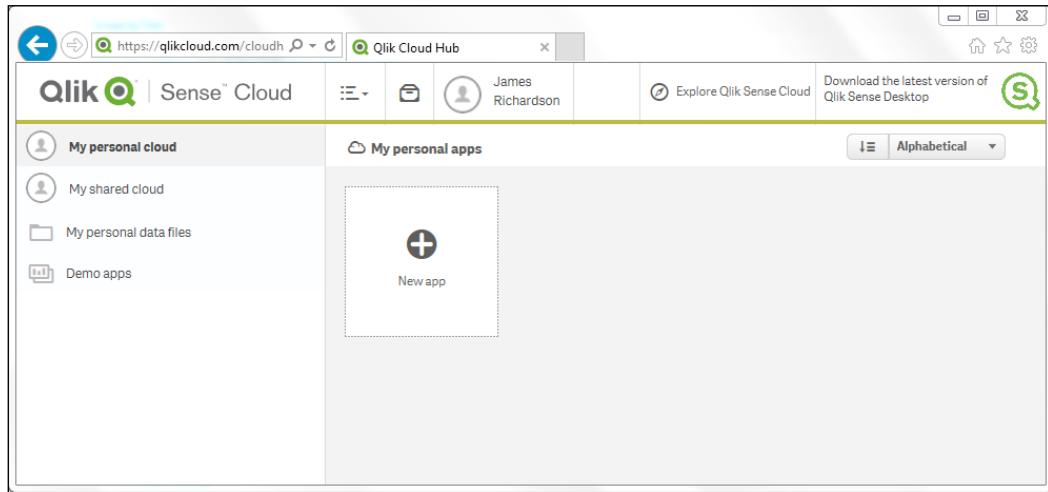
Options to upload to Qlik Sense Cloud from the desktop hub

As you'll see when you log in, Qlik Sense Cloud is intentionally simple, presenting just a few options. The actions you can take are:

- Uploading Qlik Sense apps
- Uploading personal data files and creating apps directly in the cloud
- Sharing apps with up to five people
- Managing your Qlik Cloud profile

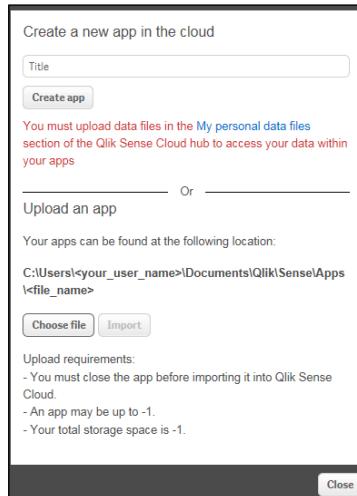
Qlik Sense® Apps in the Cloud

The following screenshot shows the Qlik Sense Cloud Hub:



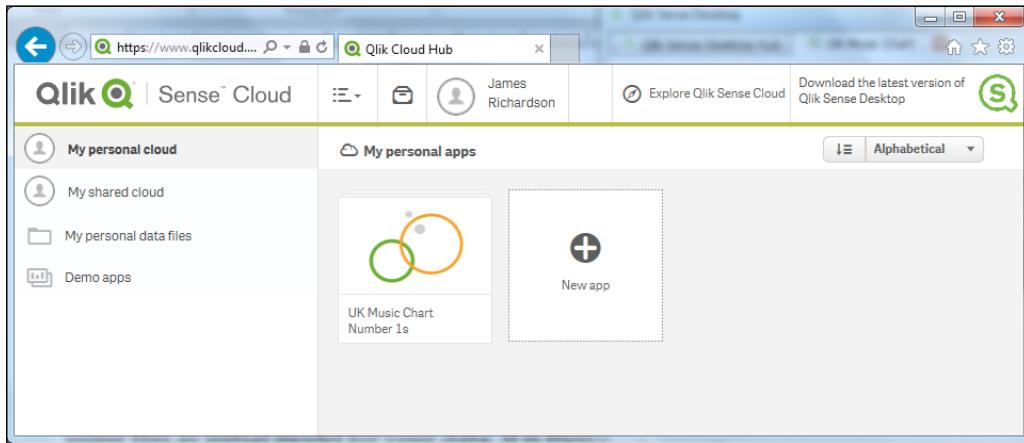
Qlik Sense Cloud Hub options

Uploading an app you have built on the desktop is simple. Click on **New app**, navigate to the file in the directory, and import. The uploaded QVF file brings the data model, sheets, and stories with it to the cloud for sharing. Note that the source data that was loaded into the app is not uploaded to Qlik Sense Cloud. This has implications for maintaining the app – more on this in a later section.



In the following screenshot, you can see the music chart app after upload. You'll notice that when uploaded, the app appears with the default thumbnail picture (rather than the custom image). Qlik Sense Cloud Hub does not currently support custom thumbnail images.

While not visible in the cloud hub, within the cloud app's information pane, you can show the custom image by selecting it from the media library. The media library is imported on upload, and includes thumbnails and any graphic files you've added to stories in the desktop app:



After the upload, you can use the app in exactly the same way as if it was running on a desktop. There's no difference; it's just Qlik Sense running on a server hosted by Qlik in the cloud. Modifications to the app can still be made using the **Edit** button, but only when the app is in the **My personal cloud** stream (more on this later).

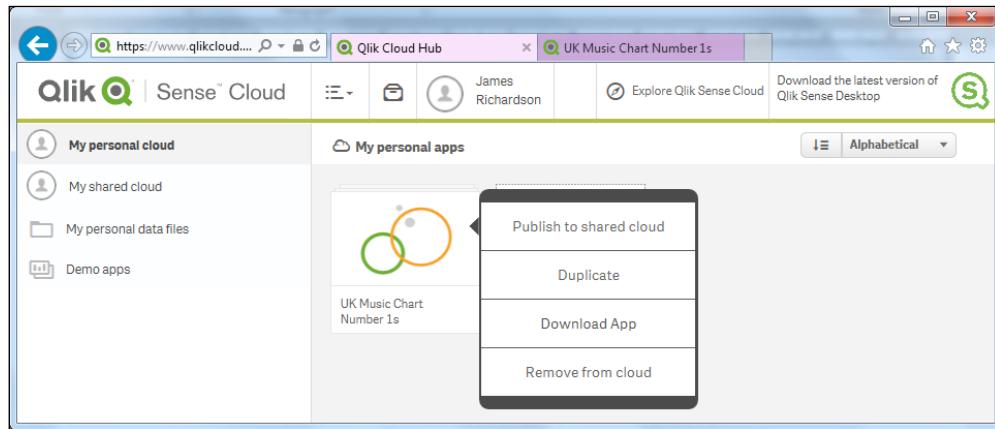
Note that, unless you selected the publish option at upload, only you can see it—it's not been shared with anyone else yet.

Creating an app in Qlik Sense® Cloud

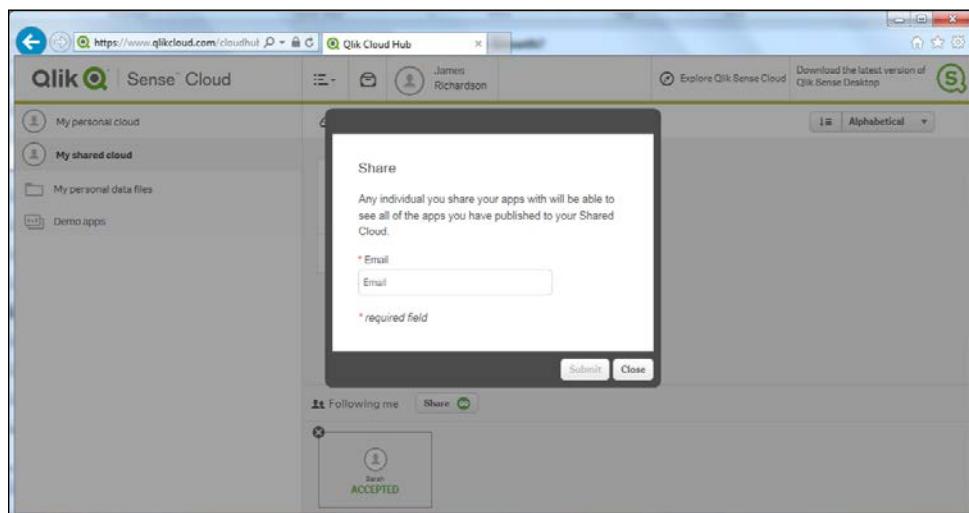
Qlik Sense apps can also be created directly in Qlik Sense Cloud. The only difference in doing so is that data must first be loaded into the **My personal data files** folder on the Qlik Sense Cloud Hub in order to be available for loading.

Sharing an app in Qlik Sense® Cloud

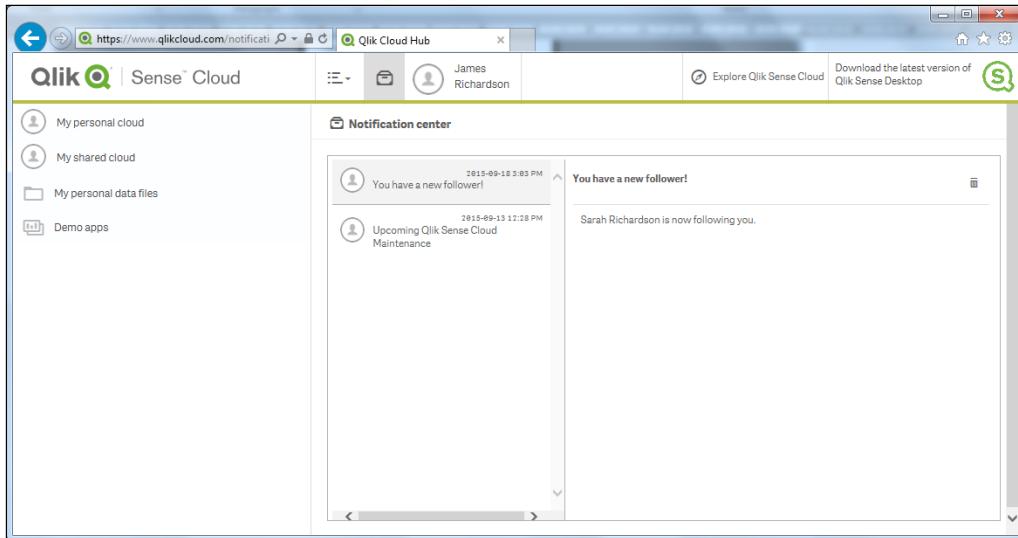
Qlik Sense Cloud is designed to enable people to share apps. To do this, a cloud app needs to be published, moving it from the **My personal cloud** stream to the **My shared cloud** stream. To do so, right-click on the app icon in the cloud hub and select **Publish to shared cloud**, as shown in the following screenshot:



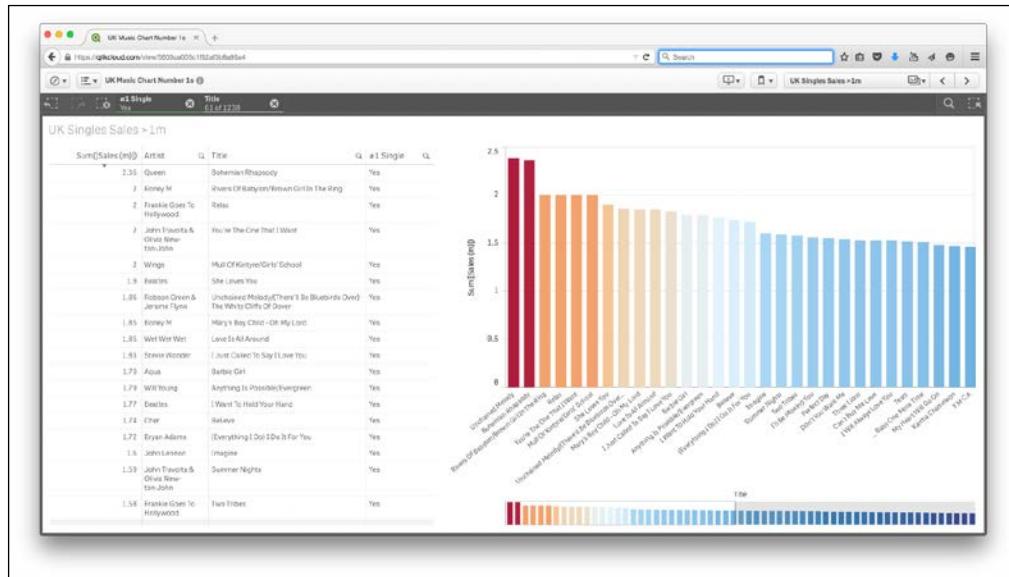
Once published, you can invite up to five people to access Qlik Sense apps in your shared cloud stream. Note that anyone you share it with will be able to see all the apps you have published. To do so, click on **share** and enter their e-mail address as shown in the following screenshot. When people accept the invitation to share and register on Qlik Sense Cloud, they appear as followers at the bottom of the **My shared cloud** stream:



By selecting the **Notification center** (in-tray icon), you can see who's viewed apps and other alerts about Qlik Sense Cloud, as shown in the following screenshot:



People that you share apps with can use those apps, making selections, and so on, but they cannot share them with others or modify them. You can see in the following screenshot (from my follower Sarah's Mac) that there is no edit pen icon on the shared app:



Maintaining Qlik Sense® Cloud apps

Some thought needs to go into maintaining apps in Qlik Sense Cloud:

- Just as with Qlik Sense Server, in Qlik Sense Cloud, when an app has been published to a shared stream, it can no longer be modified. To make changes, it must be *unpublished*. When this happens, your followers will no longer be able to see the app, until you decide to publish it again.
- For uploaded apps, rather than those created in the cloud, it's important to note that any modifications made to the cloud version of an app (for example, adding new sheets, and so on) will not be reflected in the original Desktop version. The simplest way to deal with this is to download the Cloud version to the Desktop, by right-clicking on the app in the Qlik Sense Cloud Hub. Ideally, you should decide which version (Cloud or Desktop) is to be the master on uploading in order to avoid version management issues.

If there are changes to the source data that need reflecting in a Qlik Sense Cloud app, there are differences depending on how the cloud app was created.

- If the app was built directly in Qlik Sense Cloud using uploaded data sources, then the easiest approach is to take down the data, using the **Remove from cloud** right-click option, and then upload the new dataset before reloading. If the data file has the same name and structure, the reload will work with no problems.
- If the app was uploaded directly from the desktop, then it is a little more complex, as the source data will not be in the cloud. In this case, there are two options:
 - Remove the app from the cloud, reload the changed data on the desktop, and re-upload the app.
 - Upload the changed data to the **My personal data files** space on Qlik Sense Cloud and then modify the load script. The file path in the load script will need to be amended so that each section points to the uploaded data files, which reads `lib://<user>/<file>`.

Using the Qlik DataMarket® content

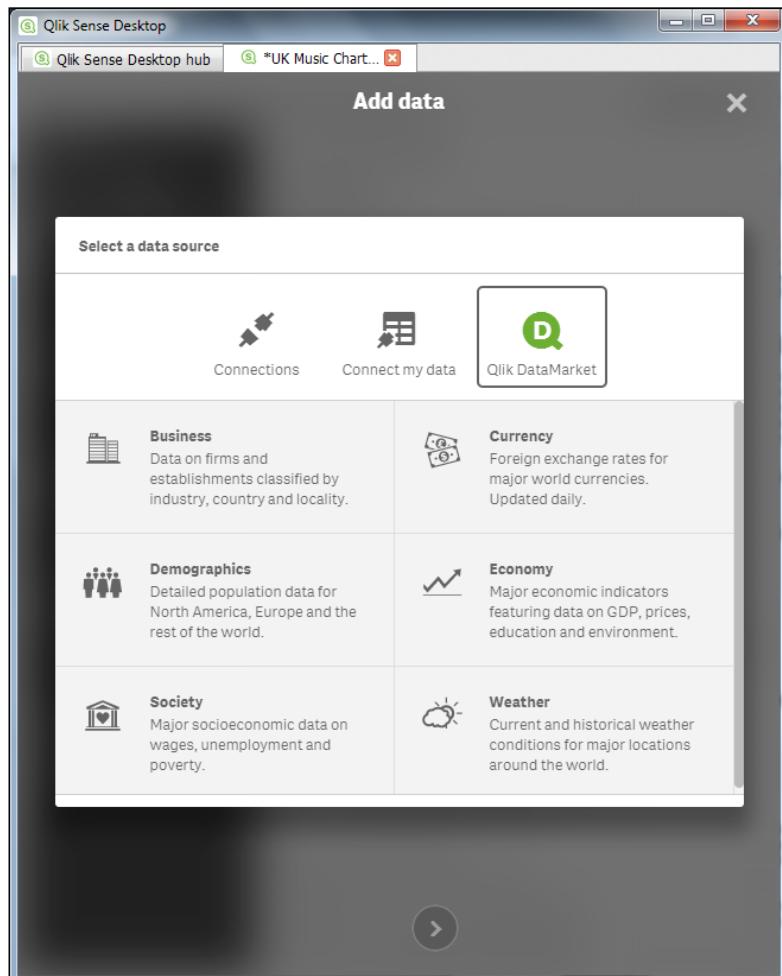
Adding externally sourced data to any business intelligence app can yield new insights. For example, looking at how the weather affects the sales of certain products could be hugely beneficial to retailers. Much data is freely available on the Web, but it is rarely in an easily consumable form. This is exactly the problem Qlik DataMarket solves, by providing curated, normalized datasets that can be easily loaded into Qlik Sense available in the cloud.

Note that Qlik DataMarket works as a source for Qlik Sense irrespective of whether you are running in the cloud or not.

Adding the QlikMarket® data

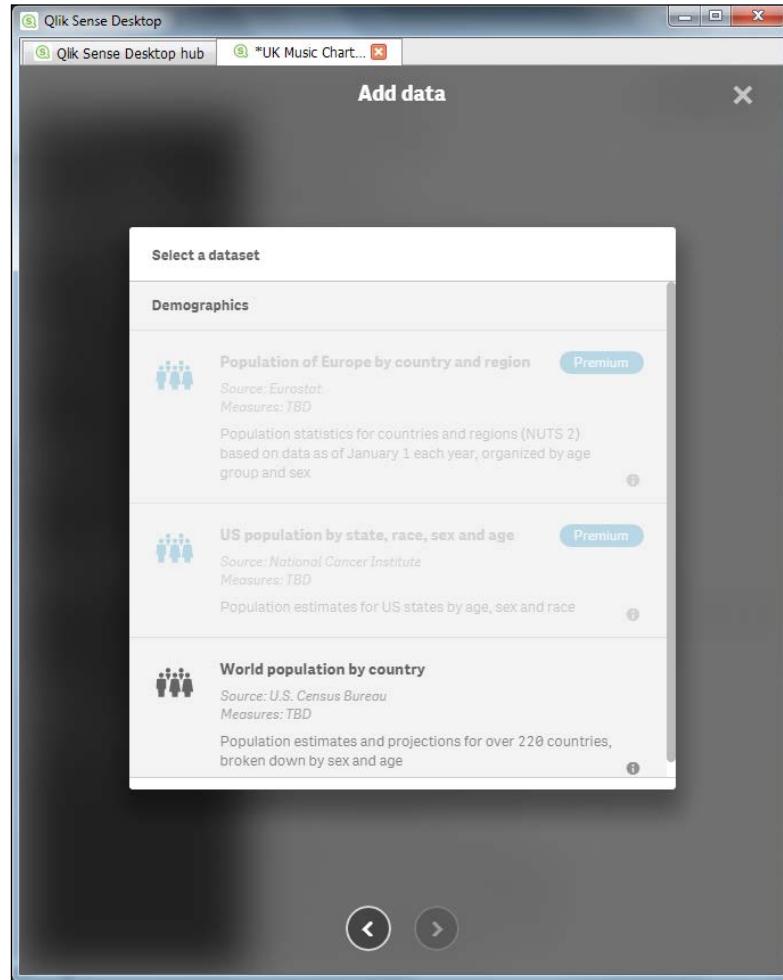
The UK music chart app contains a field showing the country, the musician, or the group it originated from. Perhaps it would be interesting to know which countries have had more hits per head of population. It is easy to do this in Qlik Sense, and can be done directly with cloud deployed apps too.

First, within the app, open the **Data Manager** and click on **Add data** and select **Qlik DataMarket**. This shows you a menu with the categories of data available. In this case, select **Demographics** as shown in the following screenshot:



Qlik DataMarket data categories

At this point, the available datasets containing demographic information are shown, including those that are premium (that is, paid for) and those that are free. In this case, the free dataset **World population by country** is what is needed:



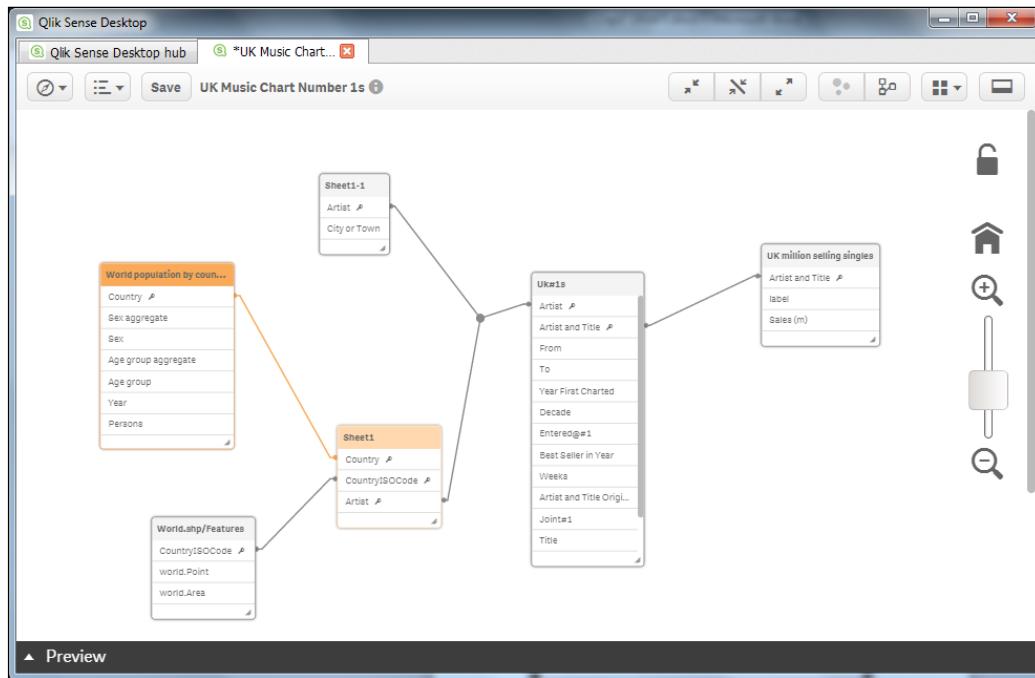
At this point, the data from the set is selected before loading:

The screenshot shows the 'Add data' dialog in Qlik Sense Desktop. The title bar says 'Add data'. The main area is titled 'Select data to load' and contains four sections: 'World population by country', 'Sex', 'Age group', and 'Year'.
World population by country: Contains checkboxes for 'Country' and a list of countries: Afghanistan, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antigua and Barbuda, Argentina, Armenia, and Aruba.
Sex: Contains checkboxes for 'Sex' and 'Sex ag...', and radio buttons for 'Female' and 'Male'.
Age group: Contains checkboxes for 'Age group' and 'Age gr...', and radio buttons for '55-59', '60-64', '65-69', '70-74', '75-79', '80-84', '85-89', '90-94', '95-99', and '100+'.
Year: Contains checkboxes for 'Year' and 'Most recent', and a radio button for 'All time'.
Below these sections is a preview table:

| Country | Age group aggregate | Age group | Sex aggregate | Sex | Year | Persons |
|----------------------|---------------------|-----------|---------------|-----|------|---------|
| Aruba | Total | | | | 2015 | 112162 |
| Antigua and Barbuda | Total | | | | 2015 | 92436 |
| United Arab Emirates | Total | | | | 2015 | 5779760 |

At the bottom are three buttons: a left arrow, a right arrow labeled 'Profile', and a right arrow labeled 'Load and finish'.

In the app, there is already a field called **Country**, which Qlik Sense uses as a key in order to allow you to work with the data, as the **Data Model Viewer** shows:



A few clicks and the app is enriched with content that provides context for analysis and insight, enabling users to ask more questions based on a broader scope and increased context.

Summary

In this chapter, we looked at sharing Qlik Sense apps in the cloud, and at using Qlik's Data-as-a-Service offering to add context to apps.

In the next chapter, we'll look at extending the use of Qlik Sense through the Qlik Analytic Platform (QAP).

8

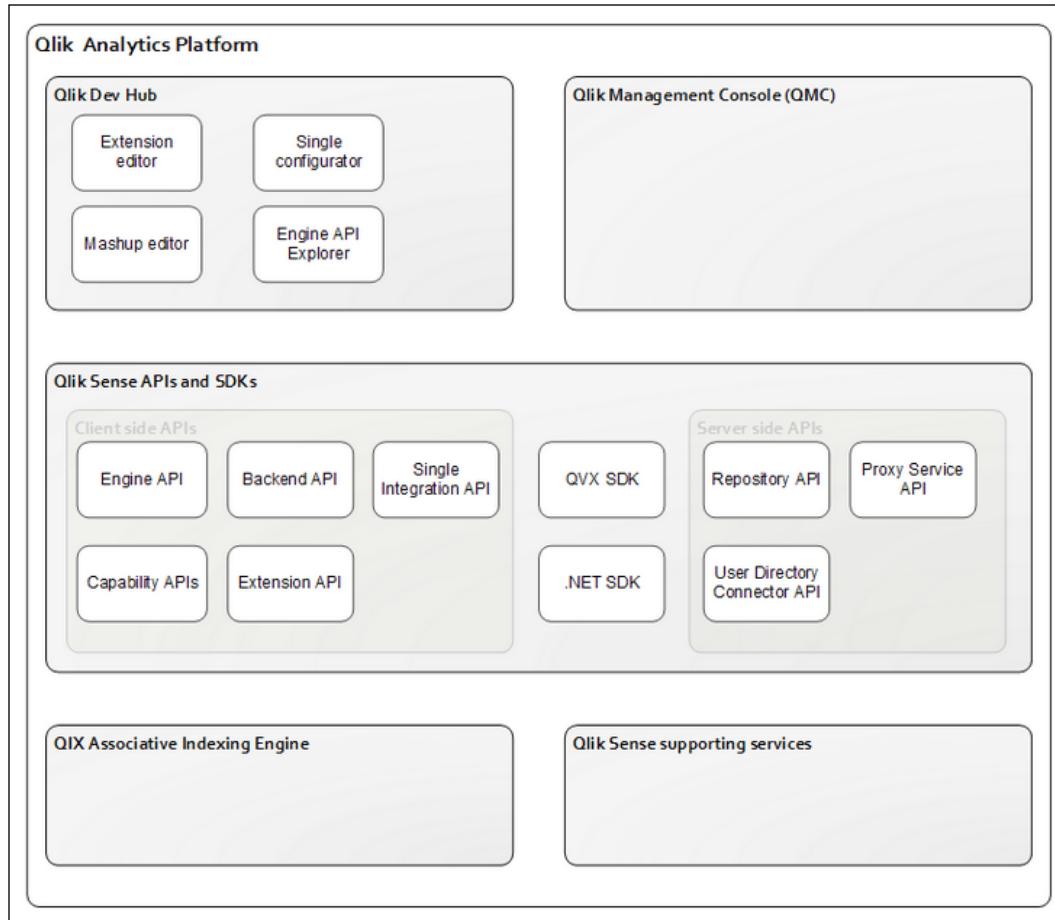
Extending the Qlik® Analytic Platform

In the previous chapters, we outlined the various capabilities of Qlik Sense and their use. One of the advantages of Qlik Sense is that it is built on open API's **Qlik Analytic Platform (QAP)**, that allows the customers and partners to extend their analytic solutions. This chapter will provide an overview and some interesting examples of how to enrich your solutions with QAP. It is not meant to replace the *Qlik Sense for Developers help* documentation, which can be found at <https://help.qlik.com/sense/2.1/en-us/developer/#Home-developer.htm>.

With this said, in this chapter, we will share some interesting examples and resources in the following key areas:

- Web mashups
- Extending Qlik Sense client
- Developer community - branch

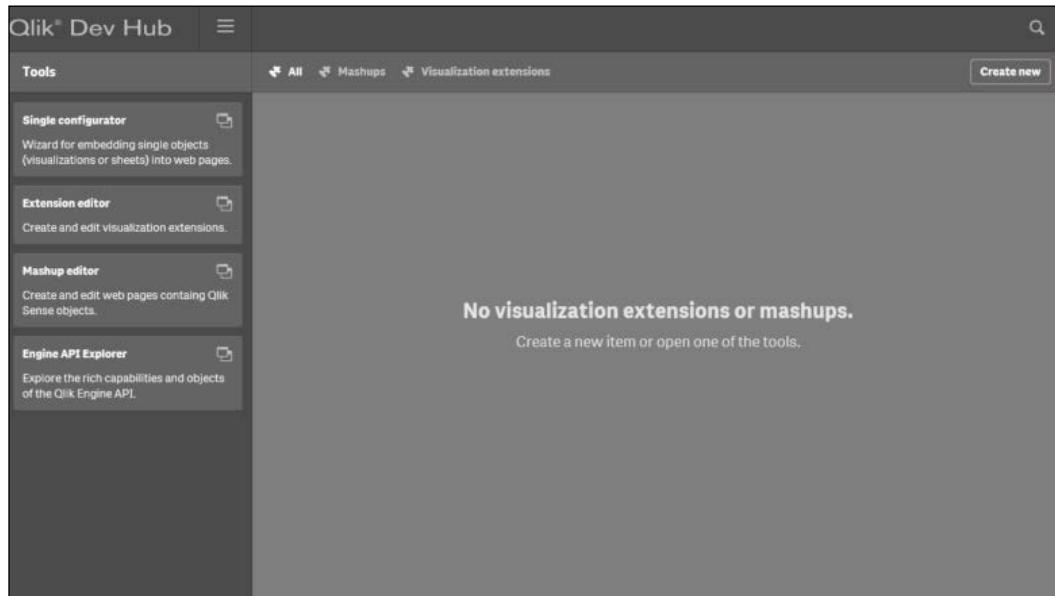
QAP is made up of the following three layers, which include the client layer (Qlik Hub and Qlik Management Console), API, the SDK layer, and finally the Engine layer, which contains the QIX engine and all the supporting services:



We will explore each of these layers through specific examples on how QAP is used to extend your data discovery solutions.

Qlik® Dev Hub

Any discussion of QAP would be remiss without reviewing the new Qlik Developer Hub which was released in Qlik Sense 2.1:



The **Qlik® Dev Hub**, shown in the preceding screenshot, was developed specifically to simplify access and development using Qlik's APIs, but it is not a replacement for the rudimentary concepts of JavaScript, HTML (Hyper Text Markup Language), and CSS (Cascading Style Sheets). The **Qlik Dev Hub** can be accessed via any Qlik Sense Server, or Qlik Sense Desktop via a browser. The link is <https://<servername>/dev-hub/>, and for the desktop version of Qlik Sense, it is <https://localhost:4848/dev-hub/>. This chapter will focus primarily on Qlik Sense Enterprise. Dev Hub provides four key tools for extending Qlik Sense solutions. They include the following:

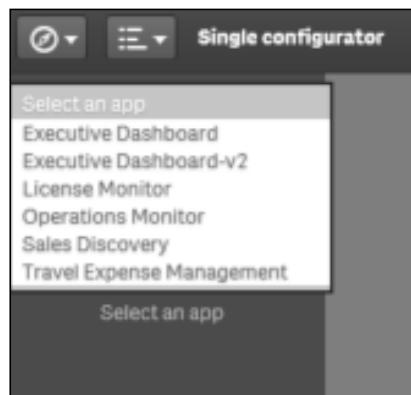
- **Single configurator:** A Qlik Sense tool that provides an easy way of creating simple mashup pages by returning a URL that will resolve to a Qlik Sense object.
- **Extension editor:** An editor for JavaScript files and QEXT files. It assists you with creating new visualization extensions as well as editing existing ones.

- **Mashup editor:** An editor for JavaScript files and HTML files. It assists you with creating your own mashups displaying Qlik Sense data on your website. You can use the templates provided with Qlik Sense to get started with building your own mashups.
- **Engine API Explorer:** A tool that helps you explore the capabilities of Qlik Engine API.

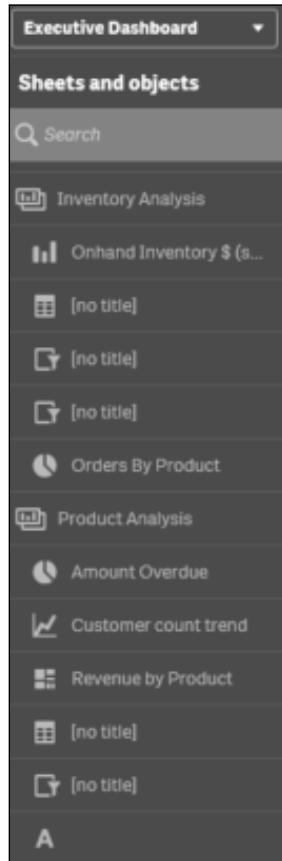
With this said, now let's take a closer look at the most common solution extensions that can be generated via Dev Hub.

Web mashups

One of the most common requests is: can Qlik Sense create a web mashup with Qlik Sense objects? There are two options based on the level of control and interactivity required of the Qlik Sense objects. The first is the single configurator which provides an easy way to create simple mashup pages without having to create any code. It simply generates a URL that returns a complete HTML page with an embedded Qlik Sense visualization. To create a Qlik Sense object link, select the **Single configurator** as shown in the following screenshot. Once selected, you will need to select the application you wish to access for the mashup. In this example, the **Executive Dashboard** application was selected. It is important to note that a developer should access a published application that is in a stream that aligns with the requirements of the resulting mashup. There is additional information on administration in *Chapter 9, Administering Qlik Sense®*.

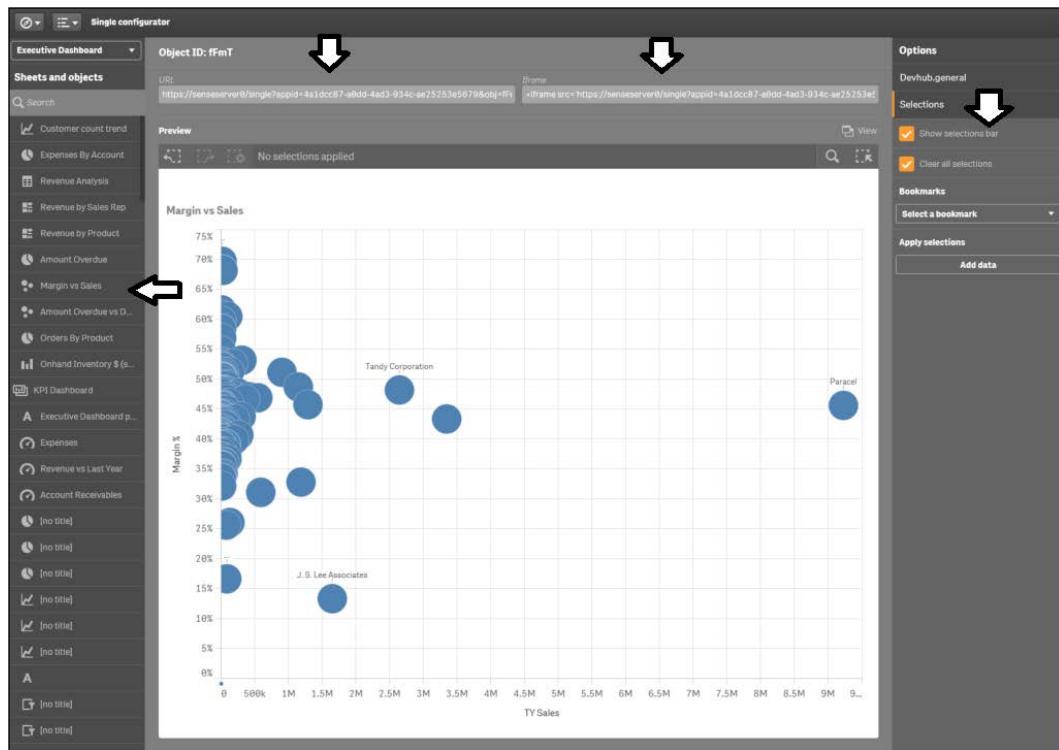


Once selected, a listing will be generated of all the sheets and respective objects. A key point to highlight is that *both* sheets and individual objects are available based on your web page requirements.

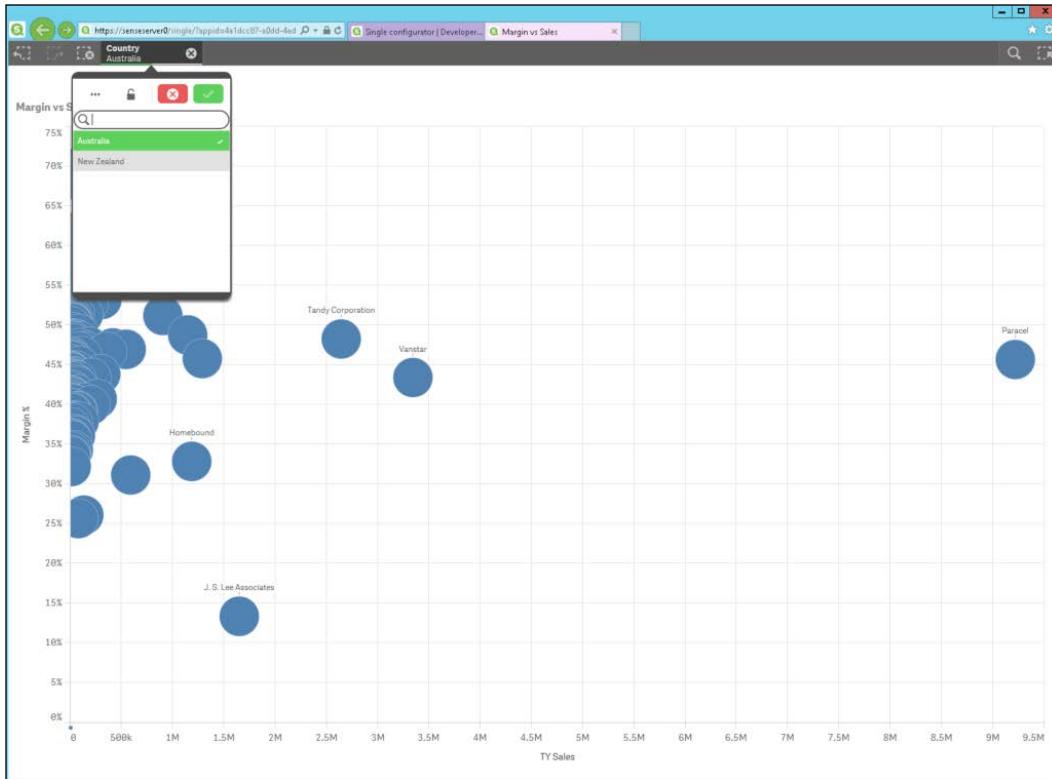


For this example, **Margin vs Sales** was chosen with selection bars showing so that a user can interact with this visualization.

In the following screenshot, you can see the use of the **Single configurator** generating the URL and the Iframe code for the **Margin vs Sales** by Sales Rep scatter chart that is contained in the **Executive Dashboard** Qlik Sense application:



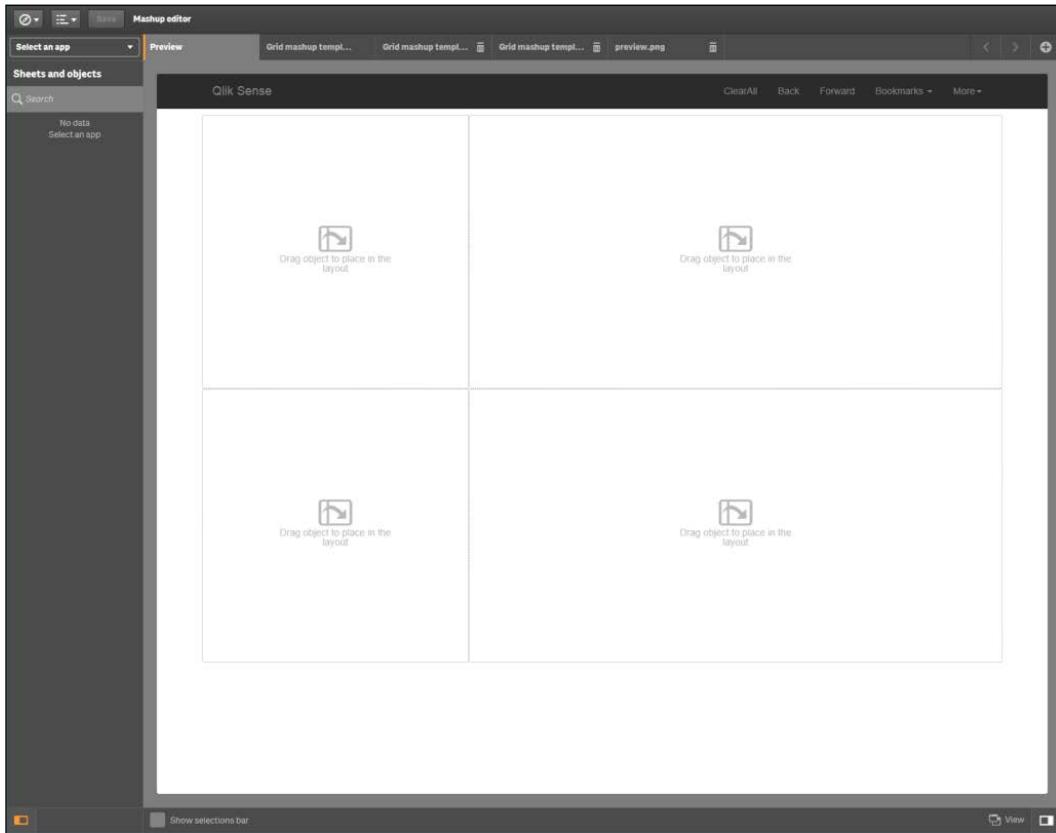
The resulting URL can then be embedded into a web page. Dev Hub offers a convenient **View** option so that a developer can see the results of the generated URL outside the **Single configurator** editor.



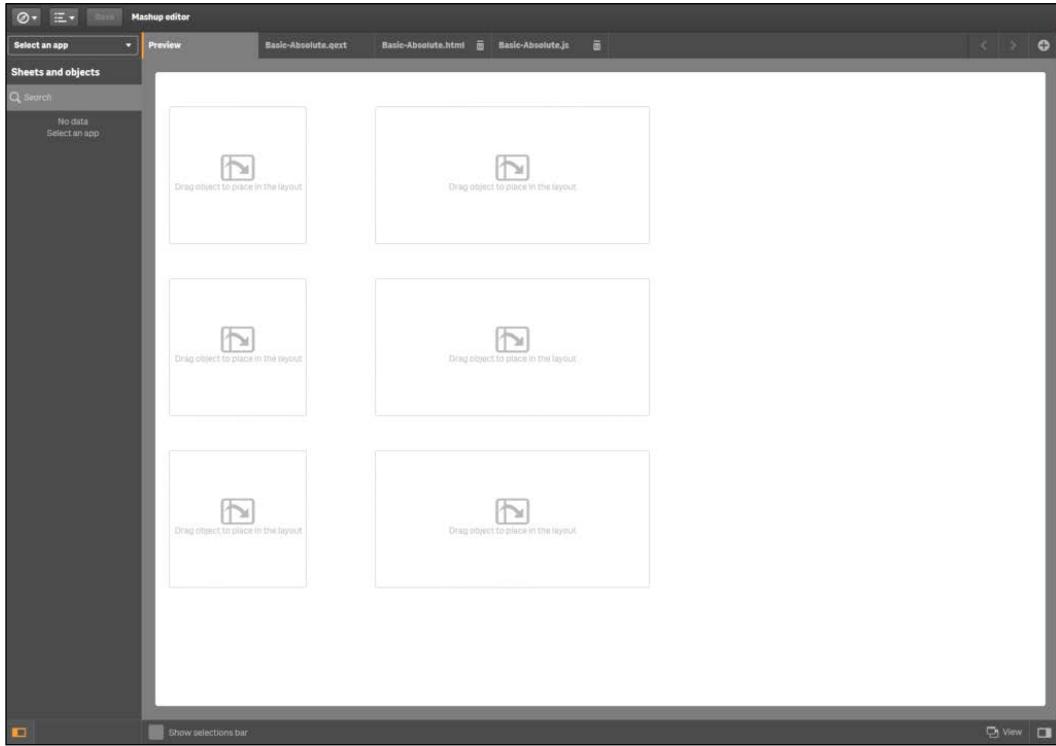
Additionally, multiple single Qlik Sense objects can be embedded and they will share common selections and interactions because of the QIX engine. Please note that not only can the visualizations be embedded, but the entire Qlik Sense sheets can be embedded as well. For more information, please refer to the *Qlik Sense for Developers* help site.

Now, let's move onto more complex mashup requirements with the **Mashup editor**. To start the creation of a new mashup, select **Create new** and immediately you will notice that the **Mashup editor** provides the following four pre-built templates to ease your development:

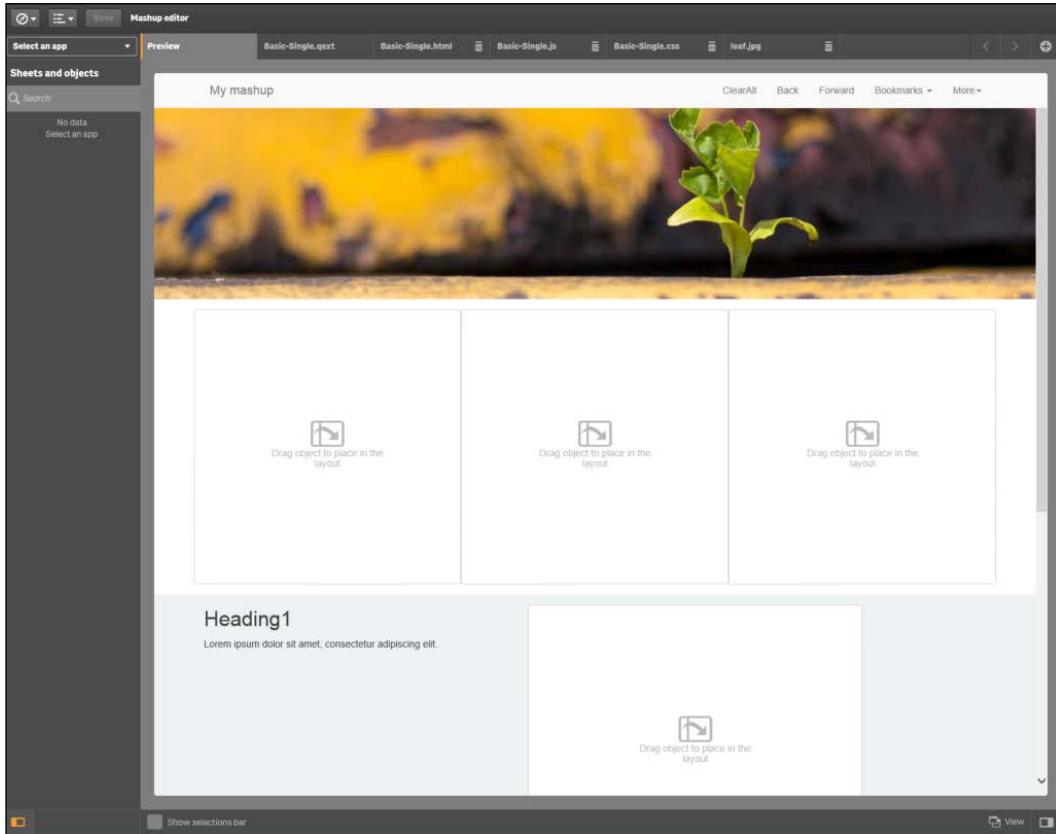
- Grid mashup template:



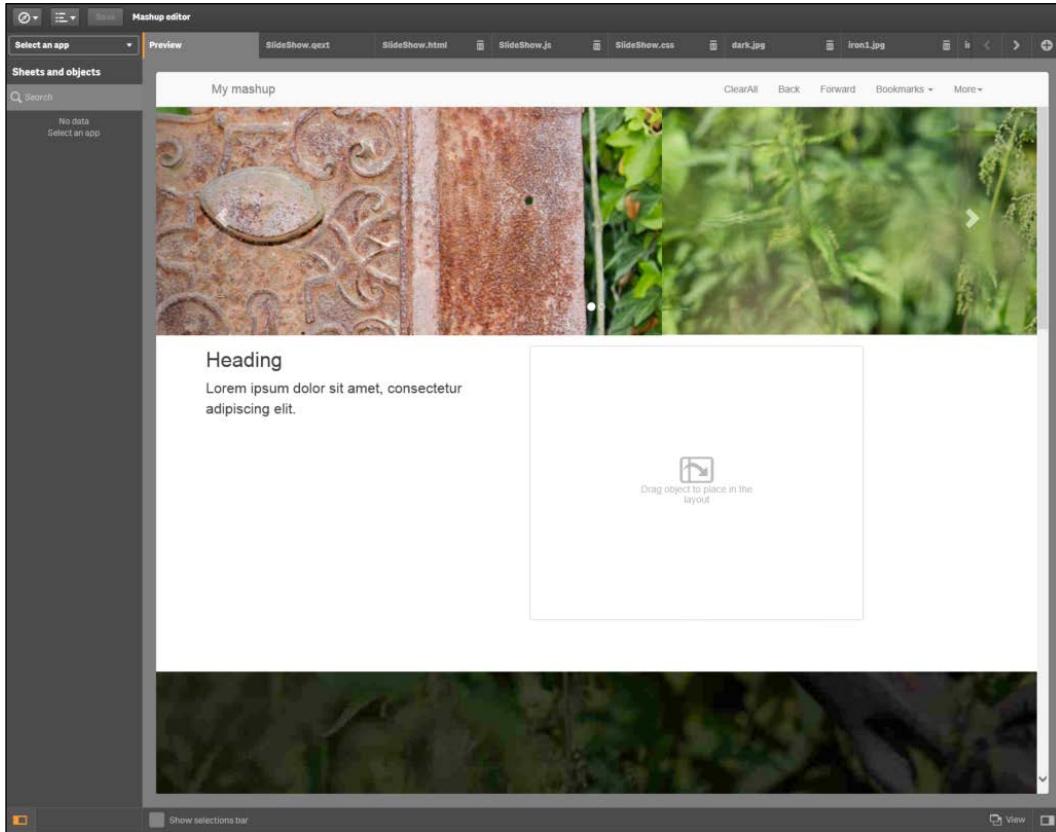
- Basic mashup template with absolute positioning:



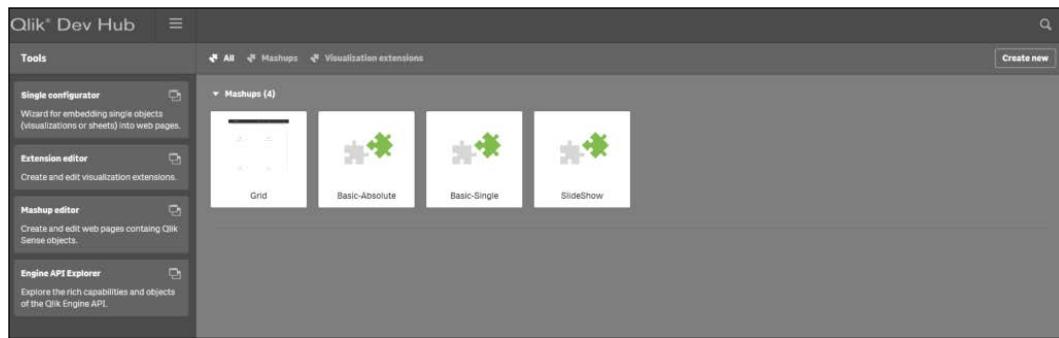
- Basic-single mashup:



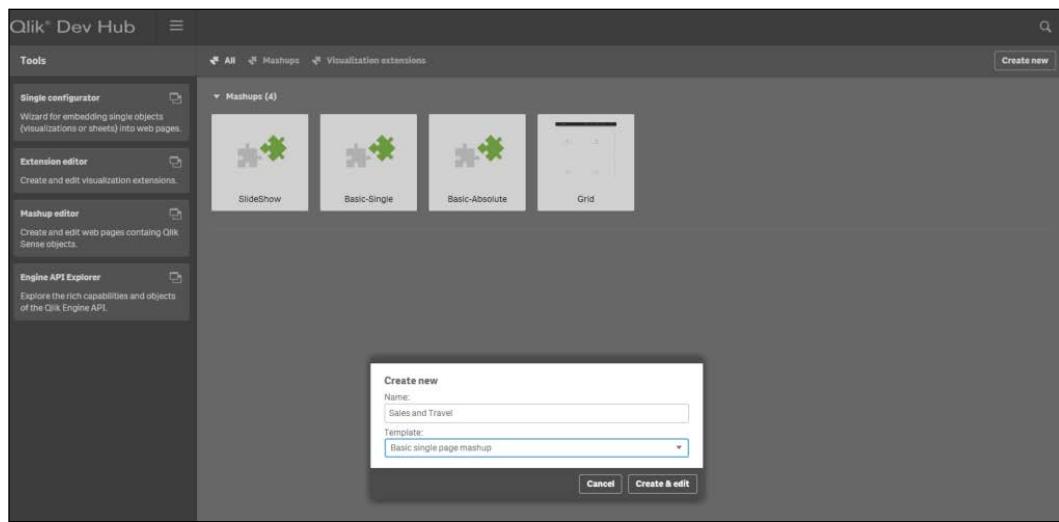
- SlideShow mashup



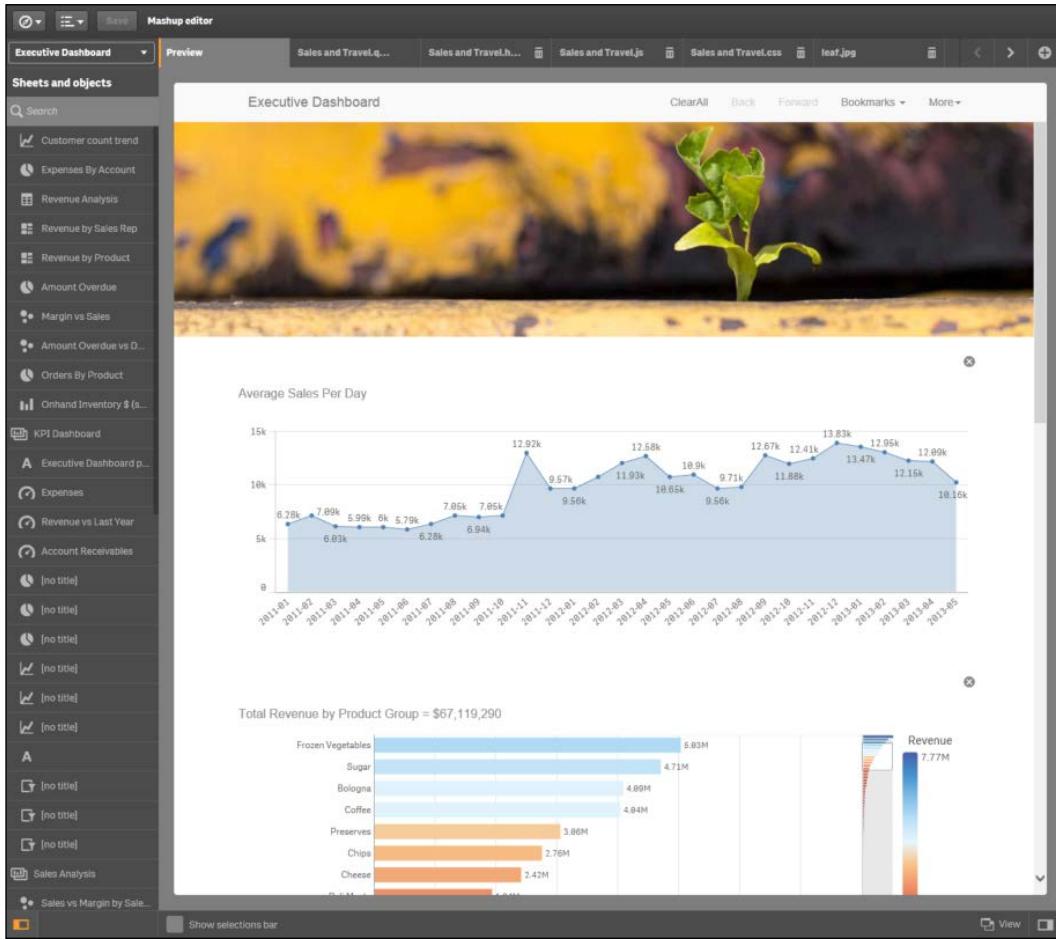
Once created, all these mashups are stored in the content store of the Qlik Sense server. Dev Hub provides an easy way to filter mashups versus visualization extensions. The following are the four templates that were generated:



Now, let's dig a bit deeper as we build out the Basic-single mashup with the creation of a sales and travel mashup that spans the two Qlik Sense applications:



With the template created, we are ready to begin. One of the advantages of working in the Dev Hub is that you can create mashups that can span different Qlik Sense applications, in this case, the **Executive Dashboard** that contains sales analysis visualizations and **Travel Expense Management** which tracks travel and food expenses. From the **Executive Dashboard** app, we have dragged **Average Sales Per Day**, **Total Revenue by Product Group**, and finally, **Revenue by Sales Rep**:



Now, let's combine these Qlik Sense sales objects with **Expense by Expense Type**, and **Employee Airfare Expense** from the **Travel Expense Management** app:

The screenshot shows the Qlik Sense Mashup editor interface. On the left, there is a sidebar titled "Sheets and objects" containing a list of objects from the "Travel Expense Management" app, such as "Actual: \$398,500 Budget", "Actual vs Budget", "Total Expenses", "Travel Expenses", "Food Expenses", "Actual: \$45,721 Budget", "Expenses by Expense T...", "[no title]", "Data as of 12/31/2013", "Airfare", "Average Employee Airfare", "Employee Airfare Expenses", "Average employee airfares", "[no title]", "Food Expenses", "Average Employee Dail...", "[no title]", and "Average Employee Dail...".

The main area displays two charts. The top chart is a treemap visualization titled "Executive Dashboard" with sections labeled "Convenience Stores", "Restaurants & Cafes", and "Bottle Shops", each containing names like Ken Roberts, Martha Richard, and Kathy Clinton. Below this is a bar chart titled "Heading1" with the subtitle "Lorem ipsum dolor sit amet, consectetur adipiscing elit." It is titled "Expenses by Expense Type" and shows the following data:

| Expense Type | Amount |
|----------------|-----------|
| Airfare | \$104,388 |
| Hotel | \$73,795 |
| Mileage | \$44,011 |
| Car Rental | \$14,989 |
| Dinner | \$38,735 |
| Parking/Tolls | \$17,194 |
| Transportation | \$16,914 |
| Lunch | \$10,975 |
| Breakfast | \$4,812 |
| Gasoline | \$1,568 |

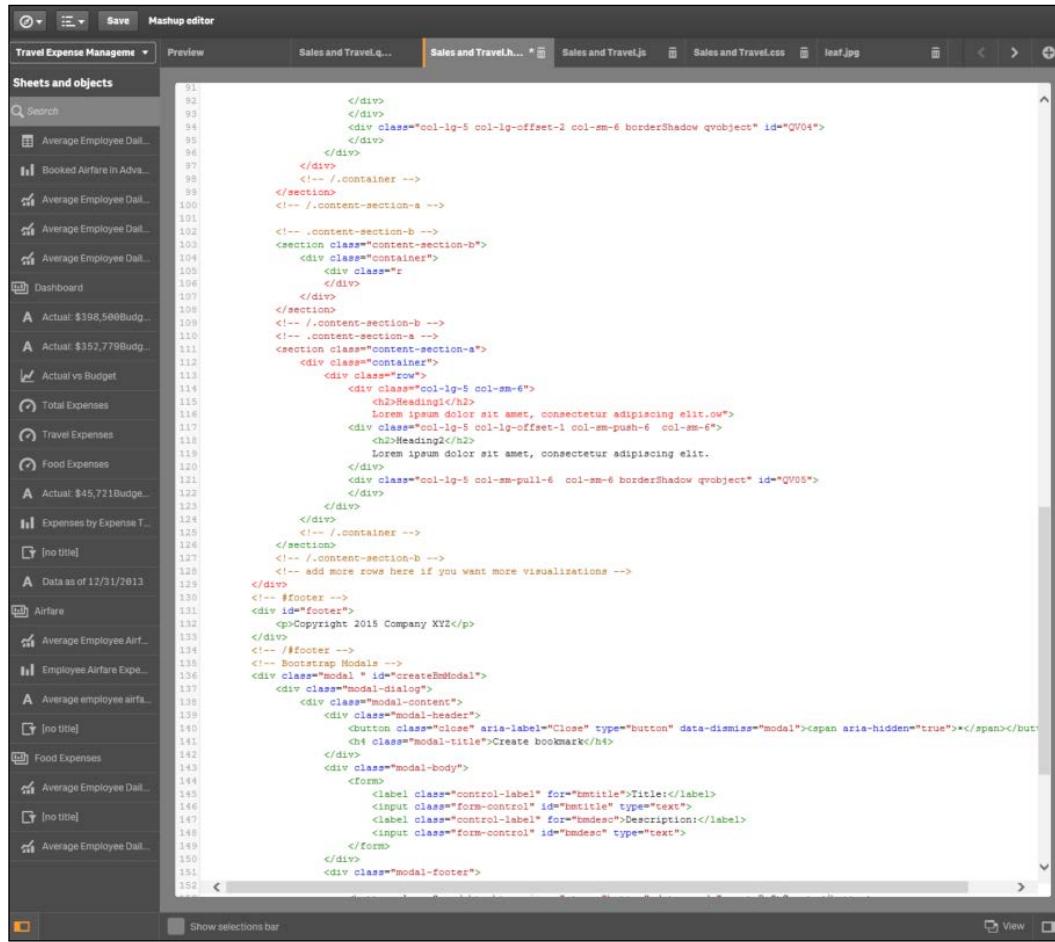
The bottom chart is a stacked bar chart titled "Heading2" with the subtitle "Lorem ipsum dolor sit amet, consectetur adipiscing elit." It is titled "Employee Airfare Expenses when Booked/Not Booked Mo..." and shows the following data:

| Employee | Measures | Value |
|------------------|-------------------------------|-----------|
| Jada Hepker | Booked Airfare in Advance | \$104,388 |
| Jada Hepker | Airfare Not Booked in Advance | \$73,795 |
| Dominique Tinnel | Booked Airfare in Advance | \$104,388 |
| Dominique Tinnel | Airfare Not Booked in Advance | \$73,795 |
| Grace Pannell | Booked Airfare in Advance | \$104,388 |
| Grace Pannell | Airfare Not Booked in Advance | \$73,795 |
| Brook Provencio | Booked Airfare in Advance | \$104,388 |
| Brook Provencio | Airfare Not Booked in Advance | \$73,795 |
| Johnie Myatt | Booked Airfare in Advance | \$104,388 |
| Johnie Myatt | Airfare Not Booked in Advance | \$73,795 |
| Angela Betterton | Booked Airfare in Advance | \$104,388 |
| Angela Betterton | Airfare Not Booked in Advance | \$73,795 |
| Dayle Kinzel | Booked Airfare in Advance | \$104,388 |
| Dayle Kinzel | Airfare Not Booked in Advance | \$73,795 |
| Ashley Vince | Booked Airfare in Advance | \$104,388 |
| Ashley Vince | Airfare Not Booked in Advance | \$73,795 |
| Hedwig Pelton | Booked Airfare in Advance | \$104,388 |
| Hedwig Pelton | Airfare Not Booked in Advance | \$73,795 |

With the key Qlik Sense objects all in place, we can focus on changing the headings for the mashup page. Fortunately, the **Mashup editor** generates the following four files for this template. They include:

- Sales and Travel.qext
- Sales and Travel.html
- Sales and Travel.js
- Sales and Travel.css

To make changes to all the headers and other formatting, just edit `Sales.html` and `Travel.html`:



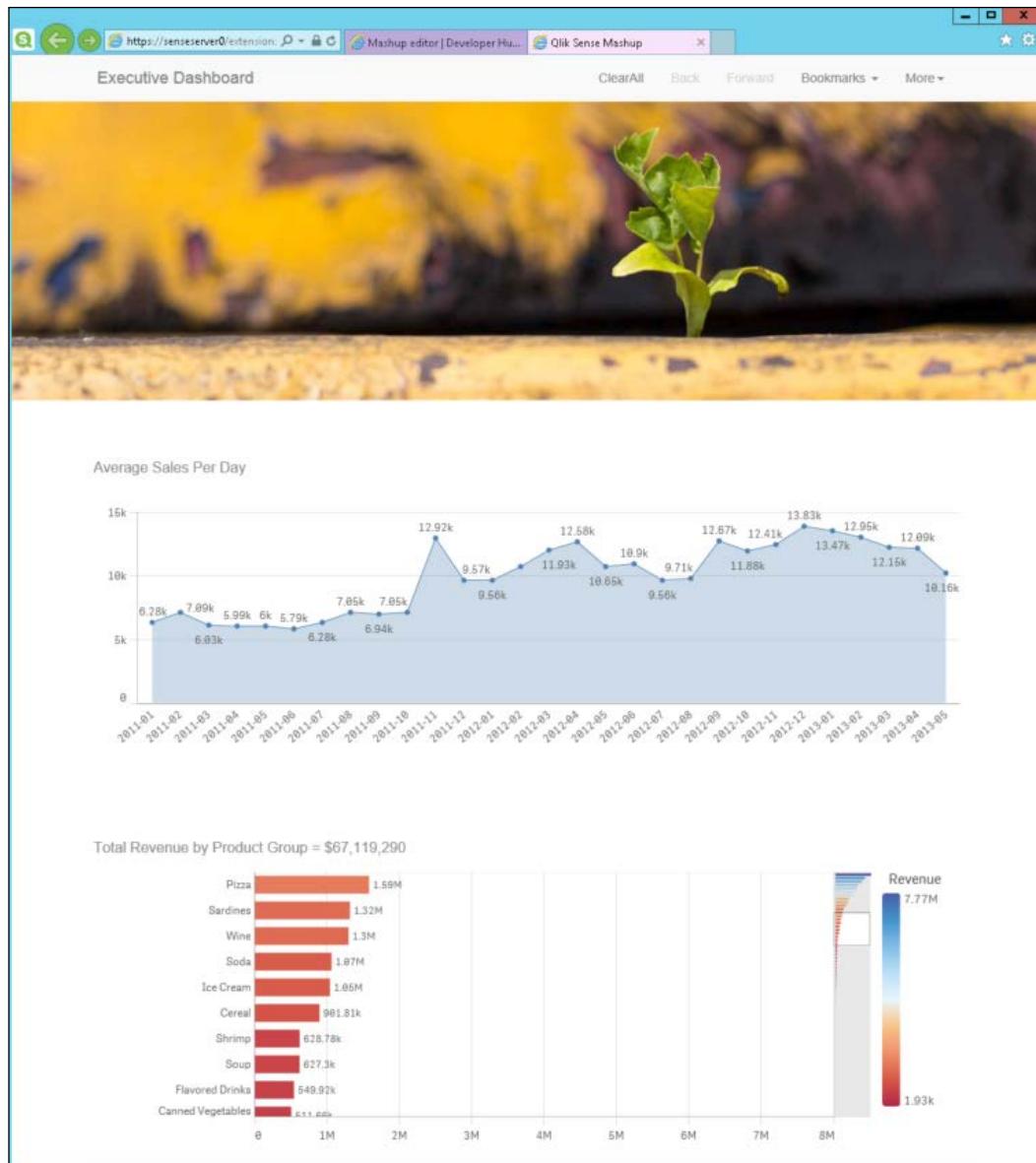
```

1. </div>
2. </div>
3. <div class="col-lg-5 col-lg-offset-2 col-sm-6 borderShadow qvobject" id="QV04">
4. </div>
5. </div>
6. </div>
7. <!-- /.container -->
8. </section>
9. <!-- /.content-section-a -->
10. <!-- .content-section-b -->
11. <section class="content-section-b">
12. <div class="container">
13.   <div class="r">
14.     </div>
15.   </div>
16. </section>
17. <!-- /.content-section-b -->
18. <!-- .content-section-a -->
19. <section class="content-section-a">
20.   <div class="container">
21.     <div class="row">
22.       <div class="col-lg-5 col-sm-6">
23.         <h2>Heading1</h2>
24.         Lorem ipsum dolor sit amet, consectetur adipisicing elit.ow>
25.       <div class="col-lg-5 col-lg-offset-1 col-sm-push-6 col-sm-6">
26.         <h2>Heading2</h2>
27.         Lorem ipsum dolor sit amet, consectetur adipisicing elit.
28.       </div>
29.     <div class="col-lg-5 col-sm-pull-6 col-sm-6 borderShadow qvobject" id="QV05">
30.     </div>
31.   </div>
32. </div>
33. <!-- /.content -->
34. </section>
35. <!-- /.content-section-b -->
36. <!-- add more rows here if you want more visualizations -->
37. </div>
38. <!-- #footer -->
39. <div id="footer">
40.   <p>Copyright 2015 Company XYZ</p>
41. </div>
42. <!-- #footer -->
43. <!-- Bootstrap Modals -->
44. <div class="modal" id="createRnModal">
45.   <div class="modal-dialog">
46.     <div class="modal-content">
47.       <div class="modal-header">
48.         <button class="close" aria-label="Close" type="button" data-dismiss="modal"><span aria-hidden="true">&amptimes</span></button>
49.       <div class="modal-title">Create bookmark</div>
50.       <div class="modal-body">
51.         <form>
52.           <label class="control-label" for="rnTitle">Title:</label>
53.           <input class="form-control" id="rnTitle" type="text">
54.           <label class="control-label" for="rnDesc">Description:</label>
55.           <input class="form-control" id="rnDesc" type="text">
56.         </form>
57.       </div>
58.     </div>
59.   </div>
60. </div>
61. <div class="modal-footer">
62.   </div>
63. </div>
64. <!-- Show selections bar -->
65. <div>
66. </div>
67. <div>
68. </div>
69. <div>
70. </div>
71. <div>
72. </div>
73. <div>
74. </div>
75. <div>
76. </div>
77. <div>
78. </div>
79. <div>
80. </div>
81. <div>
82. </div>
83. <div>
84. </div>
85. <div>
86. </div>
87. <div>
88. </div>
89. <div>
90. </div>
91. <div>
92. </div>
93. <div>
94. </div>
95. <div>
96. </div>
97. <div>
98. </div>
99. <div>
100. </div>
101. <div>
102. </div>
103. <div>
104. </div>
105. <div>
106. </div>
107. <div>
108. </div>
109. <div>
110. </div>
111. <div>
112. </div>
113. <div>
114. </div>
115. <div>
116. </div>
117. <div>
118. </div>
119. <div>
120. </div>
121. <div>
122. </div>
123. <div>
124. </div>
125. <div>
126. </div>
127. <div>
128. </div>
129. <div>
130. </div>
131. <div>
132. </div>
133. <div>
134. </div>
135. <div>
136. </div>
137. <div>
138. </div>
139. <div>
140. </div>
141. <div>
142. </div>
143. <div>
144. </div>
145. <div>
146. </div>
147. <div>
148. </div>
149. <div>
150. </div>
151. <div>
152. </div>

```

Extending the Qlik® Analytic Platform

In the case of this example, the Header 1 and Header 2 titles were removed because the Qlik Sense app object titles were self-explanatory. Additionally, because this is a generated HTML file, it can be edited for additional formatting and content. The following is the resulting web page that is previewed from the **Mashup editor**:

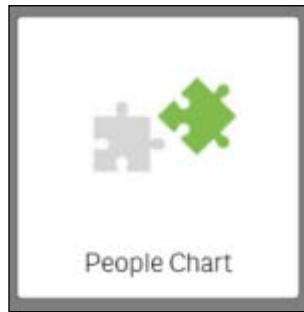


Extending the Qlik Sense® client

Now, let's turn our attention to extending the visualization objects in Qlik Sense. One of the advantages of an open API is that it can be extended easily to include external visualizations. In our example, we will explore adding a People Chart to Qlik Sense. This example is available in the following locations:

- Qlik Sense Desktop: ... \Users\<UserName>\Documents\Qlik\Examples\Extensions
- Qlik Sense: ... \ProgramData\Qlik\Examples\Extensions

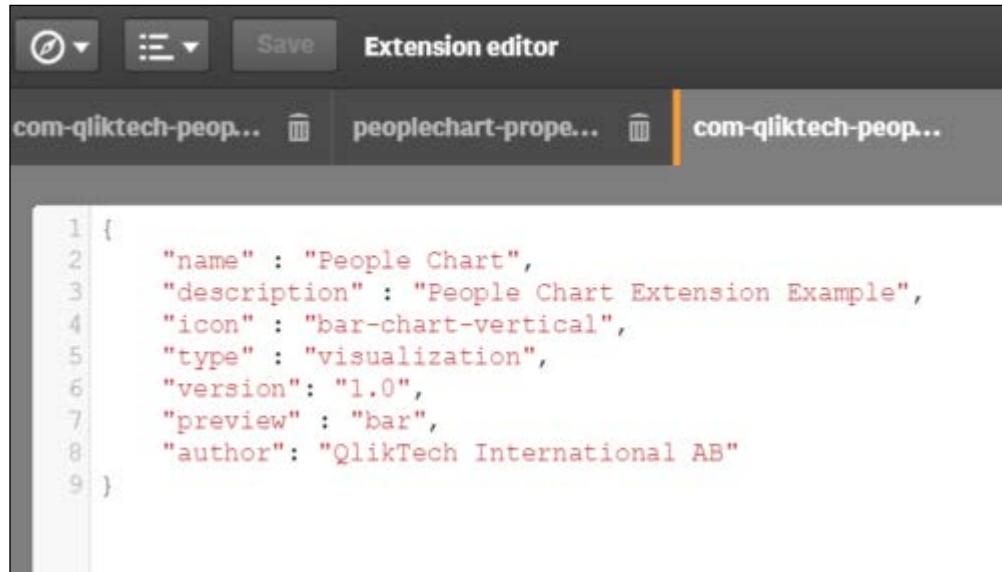
We start the process by reviewing the People Chart that is installed with Qlik Sense:



This visualization extension is made up of the following four files:

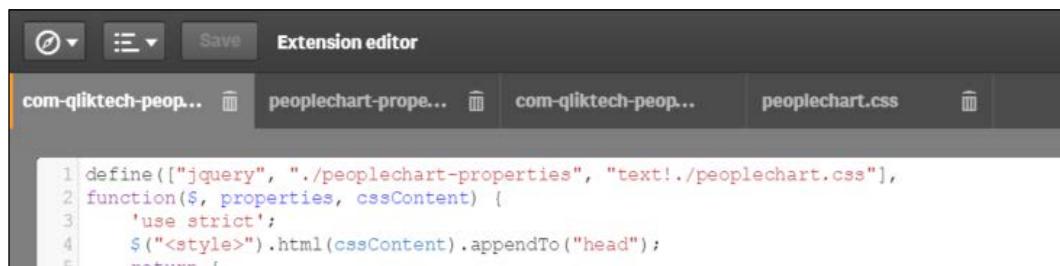
- `peoplechart-properties.js`: This JavaScript sets the properties in which the rendering and drawing scripts will operate
- `com-qliktech-peoplechart.js`: This JavaScript pulls together the properties, rendering, and drawing scripts for execution
- `com-qliktech-peoplechart.qext`: This file is primarily used to document the extension name, description, type, and so on
- `peoplechart.css`: Cascading Style Sheets (CSS) describes how HTML elements of the extensions will be displayed

The first step in defining this extension is to edit the applicable fields of the `com-qliktech-peoplechart.qext` file to set a default title, description, icon, and type:



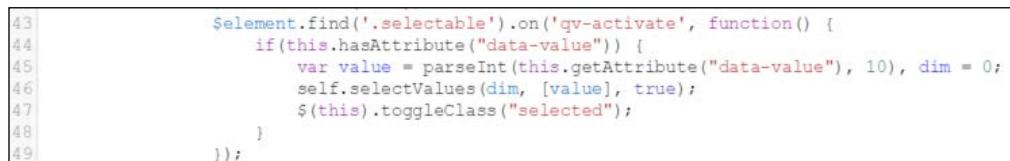
```
1 {
2     "name" : "People Chart",
3     "description" : "People Chart Extension Example",
4     "icon" : "bar-chart-vertical",
5     "type" : "visualization",
6     "version": "1.0",
7     "preview" : "bar",
8     "author": "QlikTech International AB"
9 }
```

Once this is completed, we need to define the properties for the extension in our Java code. In this example, we will do this in the `com-qliktech-peoplechart.js` file, which is loaded in the `define` statement:



```
1 define(["jquery", "./peoplechart-properties", "text!./peoplechart.css"],
2 function($, properties, cssContent) {
3     'use strict';
4     $("<style>").html(cssContent).appendTo("head");
5     return $;
```

The next step is to enable selections. To accomplish this, we use the `selectValues` function to reuse Qlik Sense standard selection UI. Also, make sure to set the selected CSS class on the selected elements:



```
43     $element.find('.selectable').on('qv-activate', function() {
44         if(this.hasAttribute("data-value")) {
45             var value = parseInt(this.getAttribute("data-value"), 10), dim = 0;
46             self.selectValues(dim, [value], true);
47             $(this).toggleClass("selected");
48         }
49     });
50 };
```

Now that it is completed, we need to implement the `paint` method. In the `paint` method, we create the HTML for our extension based on the data in the `layout` parameter. Then, we set the content of the `$element` parameter to display the extension content. It is also important to tag your elements with `class='selectable'` `data-value= '0'`:

```

20  paint : function($element, layout) {
21      var self = this, html = "", measure = layout.qHyperCube.qMeasureInfo, w = $element.width(), h = $element.height();
22      if(measure > qData.qMax) {
23          layout.qHyperCalc.function(key, row) {
24              if(dimensions < 11) {
25                  //dimension is first, measure second
26                  var dim = row[0], max = row[1];
27                  if(dim.qLabel > "") {
28                      dim.qText = layout.qHyperCube.qDimensionInfo[0].otherLabel;
29                  }
30                  html += "<div class='label'>" + dim.qText + " / " + max.qText + "</div>";
31                  //negative elementrowheights are not selectable
32                  if(dim.qElementRowHeight < -1) {
33                      html += "class='selectable' data-value='0' ";
34                  }
35                  html += "<div class='base' style='width:" + Math.round(w * (max.qNum / vmax)) + "px"><span>" + dim.qLabel + "</span></div>";
36              }
37          }
38      }
39      $element.html(html);
40  }
41
42

```

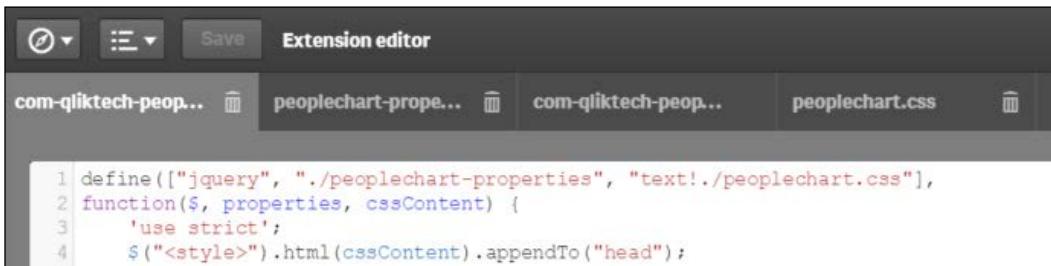
Additionally, let's make sure that this object is available for storytelling by setting `canTakeSnapshot` to true:

```

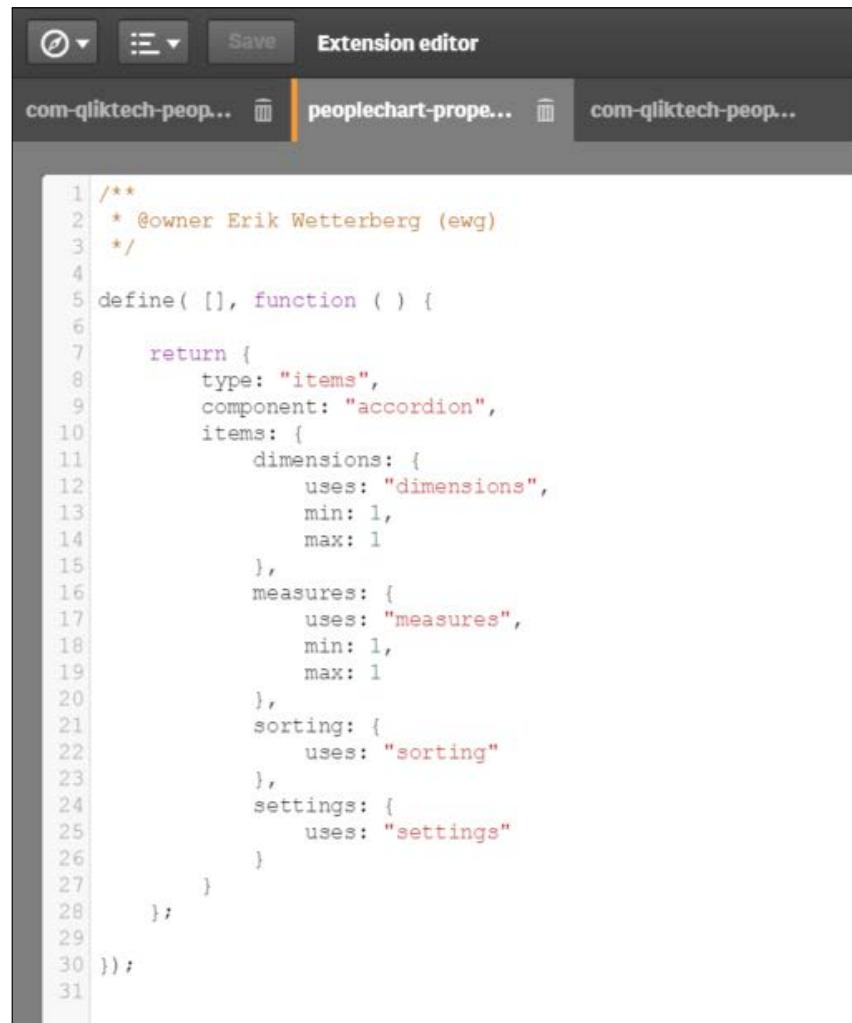
17     snapshot : {
18         canTakeSnapshot : true
19     },

```

It is a good programming practice to keep your styling in a separate CSS file. Qlik Sense sets the CSS class `qv-object-[extension name]` on your extensions. You should prefix your CSS rules with that. You then load your CSS file with RequireJS and add its content to the HTML page:



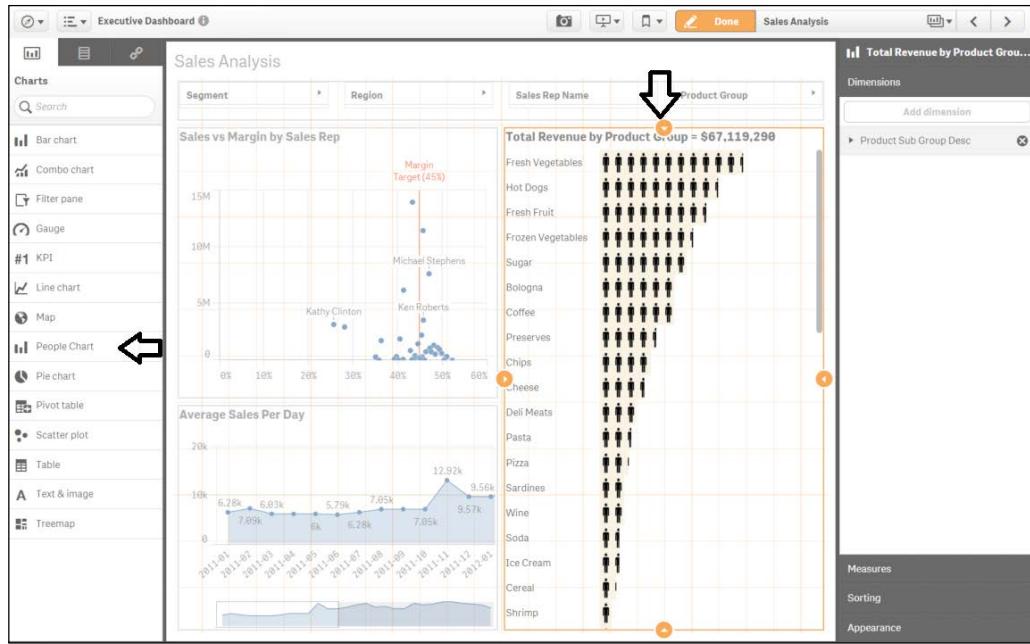
Finally, in the `peoplechart-properties.jss`, we need to define the accordions for reuse on the properties panel and the minimum requirements of one dimension and one measure to render this object:



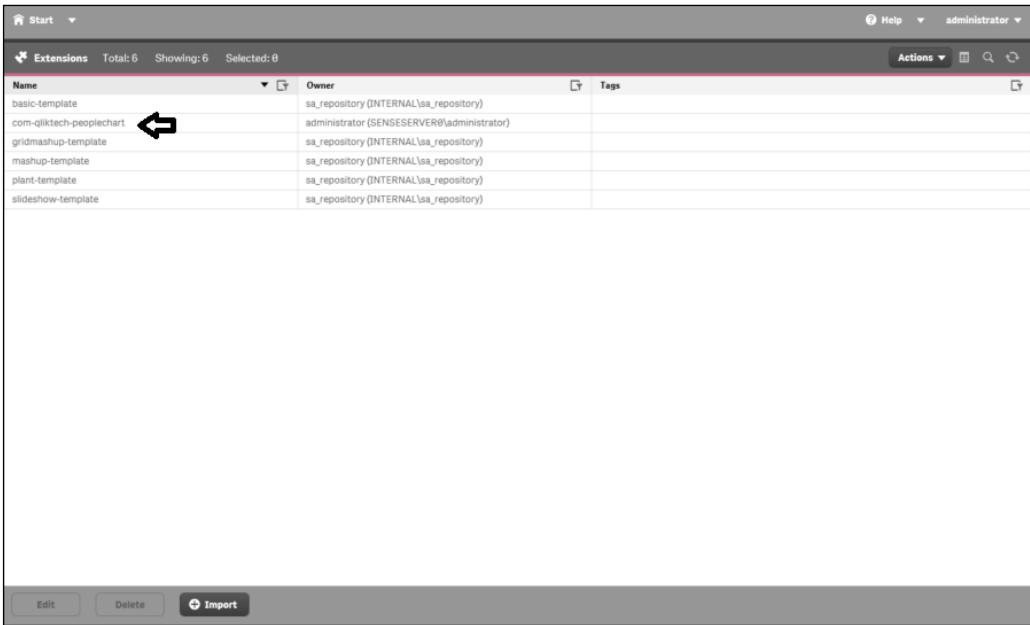
The screenshot shows the Qlik Sense Extension editor interface. The title bar says "Extension editor". Below it, there are three tabs: "com-qliktech-peop...", "peoplechart-prope...", and "com-qliktech-peop...". The middle tab, "peoplechart-prope...", is selected and highlighted with an orange border. The main area of the editor contains the following JavaScript code:

```
1 /**
2  * @owner Erik Wetterberg (ewg)
3 */
4
5 define( [], function () {
6
7     return {
8         type: "items",
9         component: "accordion",
10        items: {
11            dimensions: {
12                uses: "dimensions",
13                min: 1,
14                max: 1
15            },
16            measures: {
17                uses: "measures",
18                min: 1,
19                max: 1
20            },
21            sorting: {
22                uses: "sorting"
23            },
24            settings: {
25                uses: "settings"
26            }
27        }
28    };
29 });
30 });
31 );
```

Once the extension is saved, it is available for use by developers in the **Qlik Sense Chart Library**. In the following screenshot, you can see the Word Cloud chart type is available and was used to convert the horizontal bar chart of **Total Revenue by Product Group** into a Word Cloud:



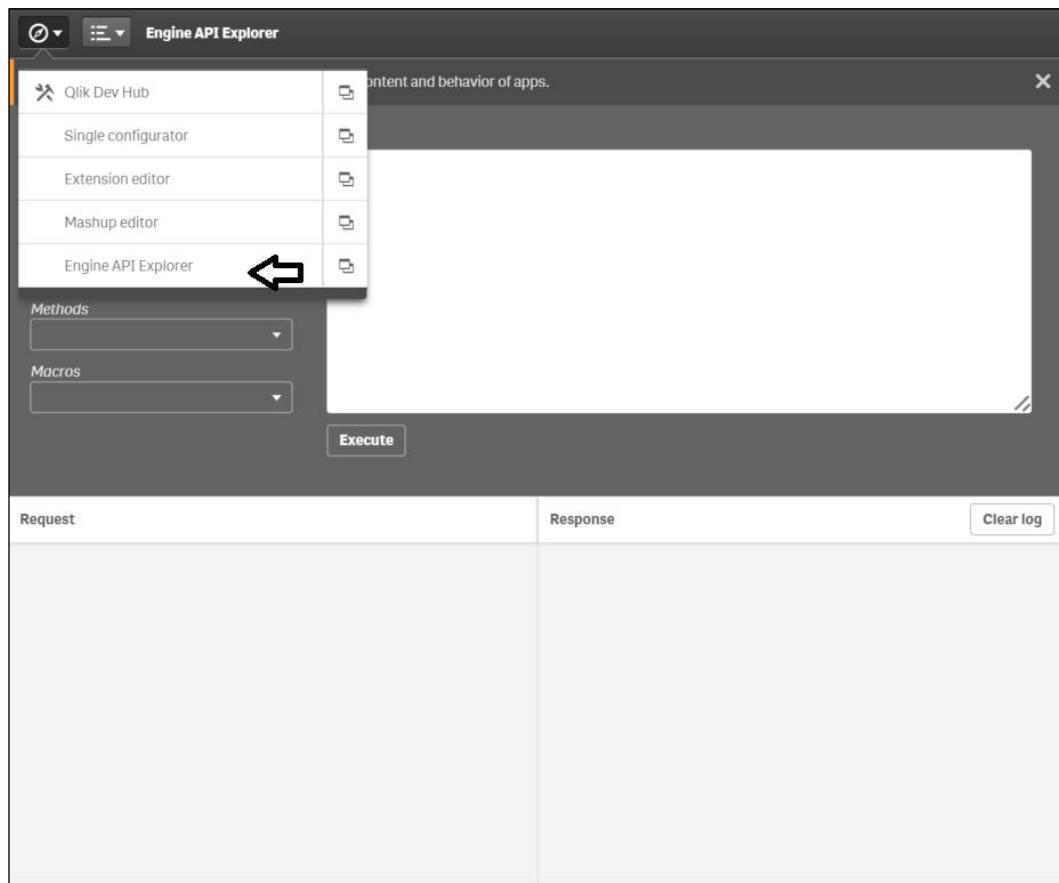
Additionally, please note that all extensions are stored in the Qlik Sense content store and are managed by the QMC. Additional information on administration is available in *Chapter 9, Administering Qlik Sense®*.



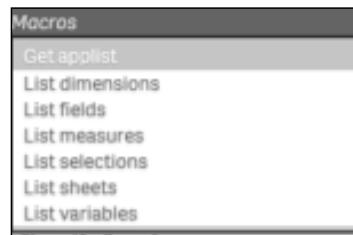
Now, let's turn our attention to the final area of Dev Hub, that is, **Engine API Explorer**.

Engine API Explorer

Qlik Engine API Explorer is a tool that allows developers to send messages and receive answers from the QIX engine. This provides an easy way for developers to form handles, methods, and macros to send the QIX engine and test the results before coding an application. To access the **Engine API Explorer**, select the **Engine API Explorer** from the **Qlik Dev Hub** menu:



The first step is to connect to a Qlik Sense app. To do this, we first must know what applications are available on this server. Fortunately, there is a full list of macros available to help explore QIX Engine:



In our case, we will be using the `Get applist` to see what applications are available to explore on this server. Based on the response, the **Executive Dashboard** is available:

Request

```
{
  "handle": -1,
  "method": "GetDocList",
  "params": [],
  "id": 3,
  "jsonrpc": "2.0"
}
```

Response

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "result": {
    "qDocList": [
      {
        "qDocName": "Executive Dashboard",
        "qConnectedUsers": 0,
        "qFileTime": 0,
        "qFileSize": 0,
        "qDocID": "4a1dc057-a0dd-4ad3-934c-ae25253e5679",
        "qLastModifiedDate": "2015-11-18T14:43:11.703Z",
        "published": true,
        "publishTime": "2015-11-18T14:43:07.537Z",
        "privileges": [
          "read"
        ],
        "description": "This Qlik Sense application was developed for the Executive team to monitor and analyze key metrics of company performance. This application is updated monthly. For additional information please review the Application Story."
      },
      {
        "qDocName": "Executive Dashboard-v2",
        "qConnectedUsers": 0,
        "qFileTime": 0,
        "qFileSize": 0,
        "qDocID": "88089685-7065-430b-bdde-7e585d424c50",
        "qMeta": {
          "modifiedDate": "2015-11-18T14:42:29.963Z",
          "published": false,
          "publishTime": "1753-01-01T00:00:00.000Z",
          "privileges": [
            "read",
            "update",
            "delete"
          ]
        },
        "description": "This Qlik Sense application was developed for the Executive"
      }
    ]
  }
}
```

Now we are ready to connect to the **Executive Dashboard** app. Note the send commands generated and the engine response. If satisfied with the response, the developer can then copy and paste into their development environment:

The screenshot shows the Engine API Explorer interface. In the top left, there are icons for refresh and search, followed by the title "Engine API Explorer". Below the title is a note: "NOTE: The Engine API explorer might change the content and behavior of apps." On the left side, there are four dropdown menus: "App" (set to "Executive Dashboard"), "Handles", "Methods", and "Macros". Below these is a "Connect" button. To the right of the dropdowns is a large white area labeled "Send". At the bottom of the left sidebar is an "Execute" button. The main area is divided into two sections: "Request" and "Response". The "Request" section contains the following JSON:

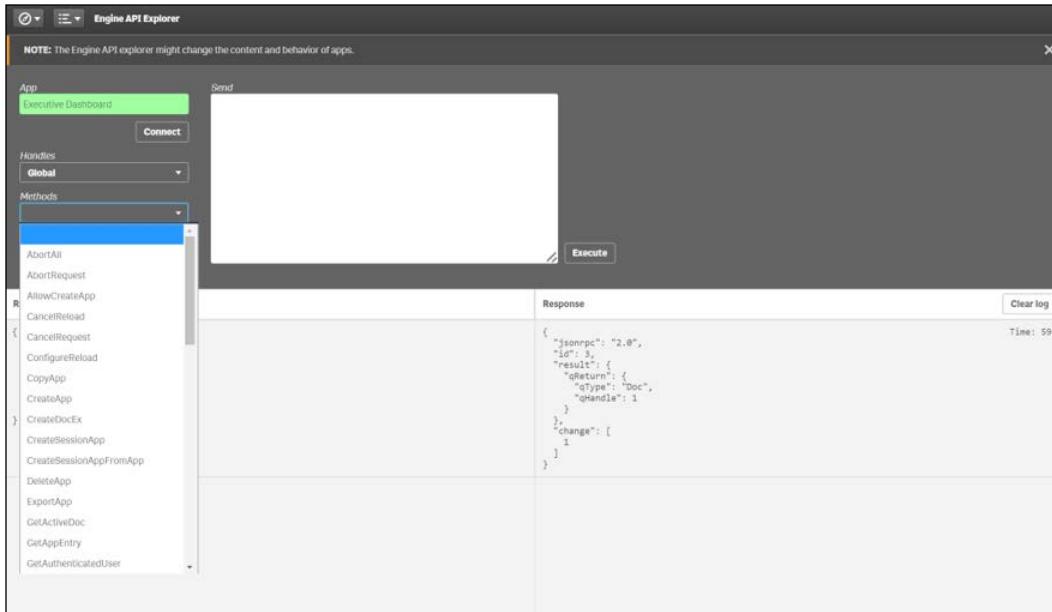
```
{  
    "method": "OpenDoc",  
    "handle": -1,  
    "params": [  
        "Executive Dashboard"  
    ],  
    "id": 7,  
    "jsonrpc": "2.0"  
}
```

The "Response" section shows the JSON returned by the engine:

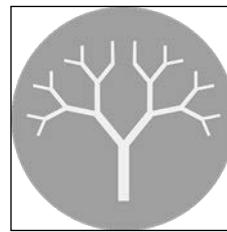
```
{  
    "jsonrpc": "2.0",  
    "id": 7,  
    "result": {  
        "qReturn": {  
            "qType": "Doc",  
            "qHandle": 1  
        }  
    }  
}
```

On the far right of the "Response" section, it says "Time: 35". At the bottom right of the main area is a "Clear log" button.

Now that we are connected, there is a global method to manage every aspect of a Qlik Sense app from outside the Qlik Sense client. This includes the full application life cycle from creation, maintenance, versioning, to deletion:



Now that we have explored the Qlik Analytic Platform through the Dev Hub, let's turn our attention to the value of having an open API through a developers community called Qlik Branch:



Developer community – Qlik Branch

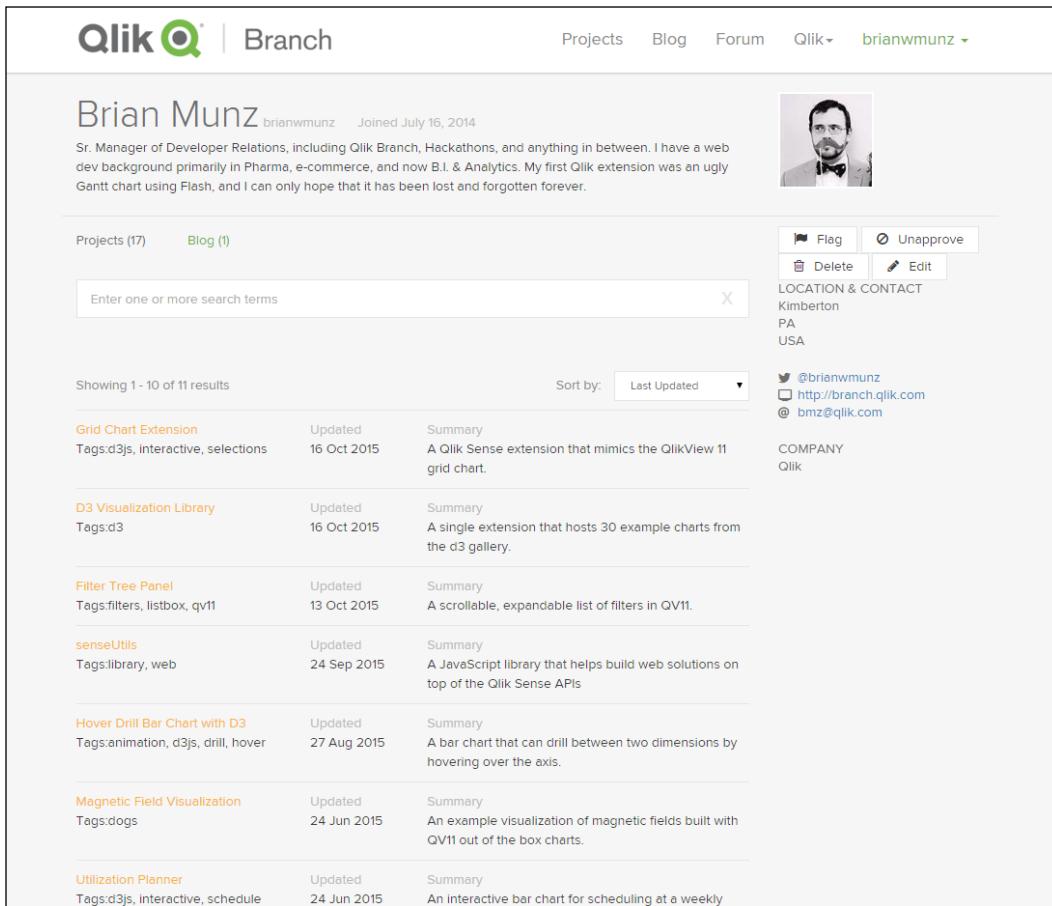
Qlik Branch (<http://branch.qlik.com>) is an open source community specifically designed for *developers*. It is a place to share and collaborate on projects and innovations created with Qlik products with an open source philosophy. All projects posted are required to have the code readily accessible, and they must be downloadable directly from the site. In short, everything on the site is free to use and free to modify in the spirit of open source:

The screenshot shows the Qlik Branch website interface. At the top, there is a navigation bar with the Qlik logo, the word "Branch", and links for "Projects", "Blog", "Forum", "Qlik+", and a user profile for "brianwmunz". Below the navigation is a search bar with the placeholder "Enter one or more search terms". A sorting dropdown is set to "Last Updated". The main content area displays a list of projects:

| Project Name | Description | Tags | Points | Views | Comments | Last Updated |
|---------------------------------|--|-----------------------------------|--------|-------|----------|--------------|
| D3 Visualization Library | A single extension that hosts 30 example charts from the d3 gallery. | Tags: d3 | 27 | 15405 | 35 | 16 Oct 2015 |
| qsVariable | Sense Variable extension | Tags: extension, sense, variable | 21 | 10432 | 46 | 9 Nov 2015 |
| deltaViz self service dashboard | deltaViz dashboard extension for Qlik Sense | Tags: sense dashboard | 18 | 17766 | 56 | 23 Oct 2015 |
| Qlik Sense SVG Reader | Maps data on built-in maps or custom SVGs | Tags: floorplan, maps, sense, svg | 15 | 4543 | 17 | 25 Aug 2015 |

Additionally, Qlik Branch is a place to find and download solutions for your projects and/or network with the developers to help extend your Qlik Sense or QlikView solutions. The site contains a wide variety of projects leveraging the APIs of Qlik Sense, including visualizations, web mashups, server automation, and connectors, to name just a few.

As a developer, Qlik Branch is a great place to get started and involved in the growing community. Many of the projects on the site could serve as a great starting place for development efforts. Additionally, a developer can stay up to date with API-related news, ask questions in the forum, join the public Slack channel, as well as educate themselves with the resources currently available or planned for the future. Furthermore, a more enterprising developer could make a name for themselves by sharing their expertise and creating valuable content, potentially driving business their way as the customers look to implement:



The screenshot shows the Qlik Branch website interface. At the top, there is a navigation bar with links for Projects, Blog, Forum, Qlik, and a user dropdown for brianwmunz. Below the navigation, the user's profile is displayed, featuring a portrait photo of Brian Munz, his name, and the date he joined (July 16, 2014). A bio describes him as Sr. Manager of Developer Relations. To the right of the profile are buttons for Flag, Unapprove, Delete, and Edit. Below the profile, there is a search bar with placeholder text "Enter one or more search terms". On the left, there are links for "Projects (17)" and "Blog (1)". On the right, there is a section titled "LOCATION & CONTACT" listing Kimberton, PA, USA, along with social media links (@brianwmunz, http://branchqlik.com, and bmz@qlik.com). The main content area displays a list of 11 projects, each with a title, updated date, summary, and tags. The projects listed are:

| Project Title | Updated | Summary | Tags | Company |
|-------------------------------|-------------|--|-------------------------------|---------|
| Grid Chart Extension | 16 Oct 2015 | A Qlik Sense extension that mimics the QlikView 11 grid chart. | d3js, interactive, selections | Qlik |
| D3 Visualization Library | 16 Oct 2015 | A single extension that hosts 30 example charts from the d3 gallery. | d3 | |
| Filter Tree Panel | 13 Oct 2015 | A scrollable, expandable list of filters in QV11. | filters, listbox, qv11 | |
| senseUtils | 24 Sep 2015 | A JavaScript library that helps build web solutions on top of the Qlik Sense APIs. | library, web | |
| Hover Drill Bar Chart with D3 | 27 Aug 2015 | A bar chart that can drill between two dimensions by hovering over the axis. | animation, d3js, drill, hover | |
| Magnetic Field Visualization | 24 Jun 2015 | An example visualization of magnetic fields built with QV11 out of the box charts. | dogs | |
| Utilization Planner | 24 Jun 2015 | An interactive bar chart for scheduling at a weekly | d3js, interactive, schedule | |

We look forward to you joining the thousands of developers who have joined together in building world class visual analytic solutions.

Summary

In summary, one of the strengths of Qlik Sense is that it is built on open API's Qlik Analytic Platform (QAP) that allows customers and partners to extend their analytic solutions. This chapter has provided an overview and some interesting examples of how to enrich your solutions with QAP. It is not meant to replace the *Qlik Sense for Developers help* documentation which can be found at <https://help.qlik.com/sense/2.1/en-us/developer/#Home-developer.htm>, but rather serve as an introduction for power users who seek to expand their skill sets, as well as developers who are new to Qlik Sense.

In the next chapter of this book, we will explore the key features of administrating your Qlik Sense environment.

9

Administering Qlik Sense®

Having established how to develop attractive and engaging applications with Qlik Sense, it's time to turn our attention from authors and business users. Instead, we will consider the requirements of administrators. In this chapter, we will move away from data and analysis to what's needed to run a Qlik Sense installation.

In this chapter, you will find information about the following topics:

- Architecture
- Clustering and nodes
- Licenses and tokens
- Streams and security concepts

The Qlik Sense® architecture

Qlik Sense has an architecture that is different from the QlikView Server architecture. Some components are very similar; others are very different. Hence, even if you know the QlikView architecture, you need to look at the following sections. In them, you will find an overview of some of the concepts in Qlik Sense.

Services

When you install the Qlik Sense server, you will install seven services. These are the cornerstones of the architecture. They can be deployed in different ways to suit different deployment purposes.

The Qlik Sense services are as follows:

- **Qlik Sense Engine Service (The QIX engine):** This is the application service, which handles all application calculations and logic. Everything that concerns the data analysis is handled by this service.
- **Qlik Sense Printing Service:** This manages the Qlik Sense exports, reporting, and printing. This is new for Qlik Sense Version 2.
- **Qlik Sense Proxy Service (QPS):** This manages the Qlik Sense authentication, session handling, and load balancing.
- **Qlik Sense Repository Service (QRS):** This manages persistence of apps and the synchronization of licensing, security, and service configuration data.
- **Qlik Sense Repository Database:** This service runs a PostgreSQL database used by the QRS.
- **Qlik Sense Scheduler Service (QSS):** This manages the scheduled reloads of Qlik Sense apps as well as other types of events, for example, task chaining.
- **Qlik Sense Service Dispatcher:** This is a service controller that is used to launch and manage additional Qlik Sense services.

In a standard installation, all seven services run on the same computer, and this works fine as long as the load on the server doesn't become too heavy.

The services can run under any account, but should preferably run under an account dedicated to the Qlik Sense services.

Clients

Qlik Sense has two different clients: the hub and the management console (QMC).

The hub is used to access, edit, and publish apps. It always runs in a web browser, regardless of whether you use a desktop computer, tablet, or smartphone to access it.



The basic Qlik Sense architecture

Qlik Management Console (QMC) is used for all types of administration. QMC is a web page found at https://<computer_name>/qmc/.

A link to this is installed in your Start menu during the installation.

The screenshot shows the Qlik Management Console (QMC) start page. On the left, there is a sidebar with several sections: **MANAGE CONTENT** (Apps, Content libraries, Data connections, App objects, Streams, Tasks, Users); **MANAGE RESOURCES** (Audit, Security rules, Custom properties, License and tokens, Extensions, Tags, User directory connectors); **GOVERNANCE** (Monitoring apps); and **CONFIGURE SYSTEM** (Nodes, Engines). The main area contains six cards with the following information:

- Tasks (5)**: You can see the task status from the overview and you can delete, start, stop, enable or disable the tasks. Tasks are used to reload the data in an app or to import users from a user directory.
- Apps (20)**: You can see all the available apps from the overview and you can edit, delete, publish, or duplicate the apps. An app can be reused, modified, and shared with others.
- App objects (90)**: You can see all sheets and stories that belong to all of the apps. You can view the publishing status of the app objects, delete them or change their owner.
- Users (6)**: You can see all the available users from the overview. You import new users by using the User directory connectors. Once imported, you can assign the users different roles. You must also define security rules to give the users access to resources.
- Streams (4)**: You can see all the available streams from the overview, and you can add, edit, and delete streams. From an authorization perspective, a stream is a collection of apps that a group of users have read or publish access to. By default, Qlik Sense includes two streams: Everyone and Monitoring apps.
- License and tokens**: The License Enabling File (LEF) determines the number of available tokens. You use the tokens on different access types and allocate the access types to users. The License usage summary page displays the token availability and how the tokens are distributed to different access types. The licensing is designed to allow you to adjust license usage as required.

The QMC start page

In QMC, you can manage and monitor everything for your installation: apps, streams, security, users, and so on.

To the left, you have the four main groups: tools to manage the content, tools to manage resources, tools for governance, and tools to configure the system.

The QMC is a multiuser environment, designed for the delegation of administration of, for example, streams to authors, if this fits a company's work process.

Applications

The apps are subject-specific; files that contain data, prepared visualizations, load script, and so on. This is where the analysis is done. From a user's perspective, an app is organized into sheets, sheet objects (visualizations), bookmarks, and stories. An app can be private or published to a stream.

If you want to access an app to do analysis, you can access it through the hub. However, if you want to perform any administrative task, such as importing or publishing an app, you can do it through QMC.

Nodes

Qlik Sense's site has an architecture that allows a distributed deployment. In other words, you can have several computers, each with a Qlik Sense installation, that work together and are managed as one coherent server. In such a configuration, each computer is called a **node** and the entire installation is called a **cluster**.

The installation can be configured so that data is synchronized between the different nodes, and so that the appropriate server is used for the client request. The purpose is, of course, to increase the system resilience and deployment flexibility.

Streams

The next important concept in Qlik Sense is **streams**. A stream is a dynamic, collaborative workgroup that is used when publishing applications. Hence, when you publish an app to a stream, you publish it to a group of people.

A stream has members, security rules, and tags. It enables the user to read or publish apps, sheets, and stories. The users who have publishing rights to a stream create the content for that specific stream, and the users who have read access are the consumers of the apps.

The screenshot shows the Qlik Sense Streams sheet. At the top, there are navigation links for 'Start' and 'Help', and a dropdown for 'Henric Cronström'. Below that is a toolbar with icons for 'Actions', 'Search', and 'Refresh'. The main area displays a table of streams with columns for 'Name' and 'Tags'. There are four rows: 'Everyone' (selected), 'HR', 'Management', and 'Monitoring apps'. At the bottom of the table are buttons for 'Edit', 'Delete', and '+ Create new'.

| Name | Tags |
|-----------------|------|
| Everyone | |
| HR | |
| Management | |
| Monitoring apps | |

The Streams sheet

Deployment and licensing

Deploying a Qlik Sense server is usually straightforward, but there are still a couple of things to think of. The first question is about clustering.

Single node or multinode

Normally, you should just install the Qlik Sense server, making sure that it is set as a central node during the installation. Then, you will get a single node installation. However, sooner or later you need to ask yourself a question about clustering, "How many servers do you want in your cluster?" This book is not a comprehensive guide to clustering issues; it will only point out the basics and the questions you need to ask.

In a standard installation, all seven services run on the same computer, and this works fine as long as the load on the server doesn't become too heavy. However, as soon as your installation starts to grow, you may need more computers to handle the load. If so, you can set up a cluster so that you have additional computers running only some of the services and still manage the entire cluster as if it was only one computer.

By far, the most common case is that Qlik Sense engine (the QIX engine) has a very large load, either due to many users or because some applications are large. Then, it might be a good idea to add one or several computers and use a separate Qlik Sense engine on each computer.

Another case is that you have several physical locations and want one node in each location, with the same content on each node. This way, the users always use the local node.

One of the computers must be set as the central node, that is, as the master. Here, you enter your license key and manage the entire cluster. Data will be synchronized from this node to other nodes.

It is possible to use the same entry point—the Qlik Sense proxy—for the entire cluster so that users don't notice that there are, in fact, several computers.

On the central node, it is recommended that you have a dedicated QPS and QIX that are used specifically for the QMC and not for the hub.

In addition, the central node must have the QSS installed even if other nodes with schedulers are added. This is because the scheduler on the central node is considered to be the master scheduler, which coordinates all scheduler activities within the site.

Hence, when defining your deployment strategy, you should try to answer some questions:

- What is the estimated number of computers needed to handle the number of apps and users you expect?
- Should the users use the same proxy so that you can set up rules for load balancing? Or should they use different entry points in the different locations?
- Do you want/need separate computers that are only used to run jobs, for example, to refresh the apps?

The answers will help you decide whether you need a cluster of Qlik Sense servers. If you don't know, or if it is your first server in a cluster, you should just install the Qlik Sense server making sure that it is set as the central node during the installation.

License and access passes

The first thing you need to do after installing Qlik Sense is to enter the license key and make sure that you get a valid **License Enabling File (LEF)** from the Qlik license activation server. However, this is not enough to get going. You also need to assign a license to yourself. Alternatively, to express this in the correct terminology, you need to allocate *a token as a user access pass for yourself*.

This is done by navigating to **License and tokens | User access**. Here, you can click on the **Allocate** button, select a user, and click on **Allocate**. This means you have given this user unlimited access to the Qlik Sense server. Unlimited means unlimited from a license perspective – you may still define restrictions on this user from a security perspective.

| Name | User directory | Status | Last used | License and tokens |
|------------------|----------------|-----------|------------------|--|
| Henric Cronström | QTSEL | Allocated | 2015-09-16 17... | License usage summary ✓ User access allocations User access rules Login access rules |

User access has been granted to one of the authors

Tokens

The reason for this procedure is Qlik Sense's flexible licensing model. In the QlikView license model, you bought Named CALs or some other license from Qlik, and that was then the license you had. To convert from one type of CAL to another was not possible, unless you contacted Qlik.

In the Qlik Sense model, you have a greater degree of freedom. Here, *you* decide how you want to allocate the licenses you bought. Some are allocated to the equivalent of a Named CAL, while some are allocated to another license type. As a consequence, you don't buy licenses. Instead you buy **tokens**, which is a kind of currency that you can convert into licenses at a later stage. In the initial configuration, no tokens are assigned to be used and hence, the need to allocate a token to yourself.

Another consequence is that the terminology has changed. A Named CAL is no longer called Named CAL. Instead, it is called a **user access pass**. So in the preceding case, you have effectively given yourself a Named CAL.

The Qlik Sense user access pass – and the QlikView Named CALs for that matter – is a general unlimited license that should be given to frequent users, that is, users that analyze data regularly, many times every month or perhaps even daily.

In Qlik Sense, there is a second license type that is designed to cover the needs of infrequent users. It is similar to the Usage CAL that exists in QlikView. It is called **login access pass**. A login access pass is equivalent to one login per month, that is, the login counter is refreshed so that a new login is possible every 28th day.

You can create login access passes in batches of 10, and 10 login access passes cost 1 token. These 10 logins can be used anyway you want. They can, for example, be used by 10 different people that log in once per month or by one single user who logs in 10 times every month.

| Name | Allocated tokens | Used login access... | Remaining login ac... |
|--------------------|------------------|----------------------|-----------------------|
| Finance department | 3 | 0 | 30 |
| Guests | 3 | 0 | 30 |
| HR | 1 | 0 | 10 |

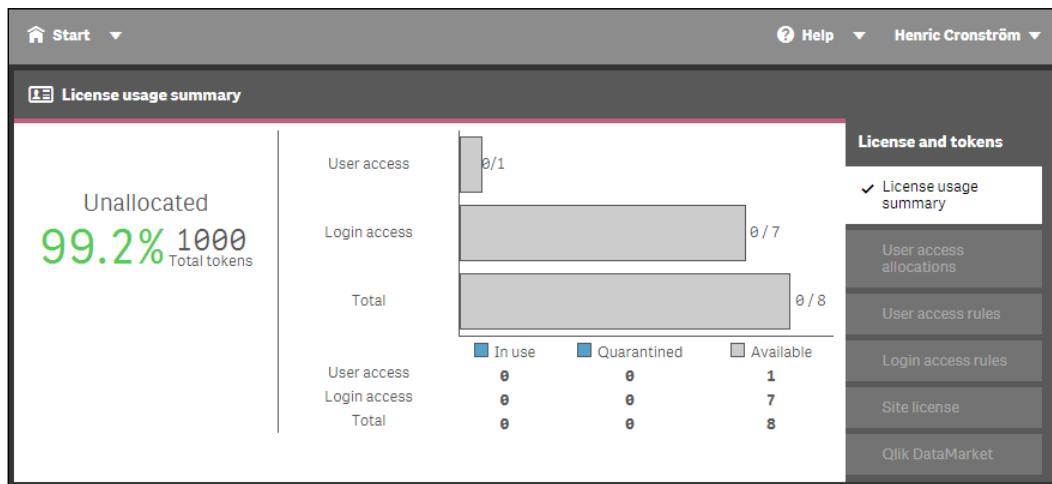
Login access passes can be created in pools for different groups of people

Access rules

Typically, you would use several tokens to create a pool of logins that can be dedicated for a group of people within your domain. You can create access rules both for user access passes and login access passes, and you should do this for your own benefit. This means that you can, for instance, say that anyone in the finance department will get a user access pass when logging on, whereas the users of another department will share the login access passes of a specific pool.

Hence, when you create the user and login accesses you want, you can get an overview of the **License usage summary** page, where you can clearly see the number of used tokens and how many you have left to allocate.

You can clearly see how many tokens you allocated to licenses, and how these are distributed over the two access types in the following screenshot:



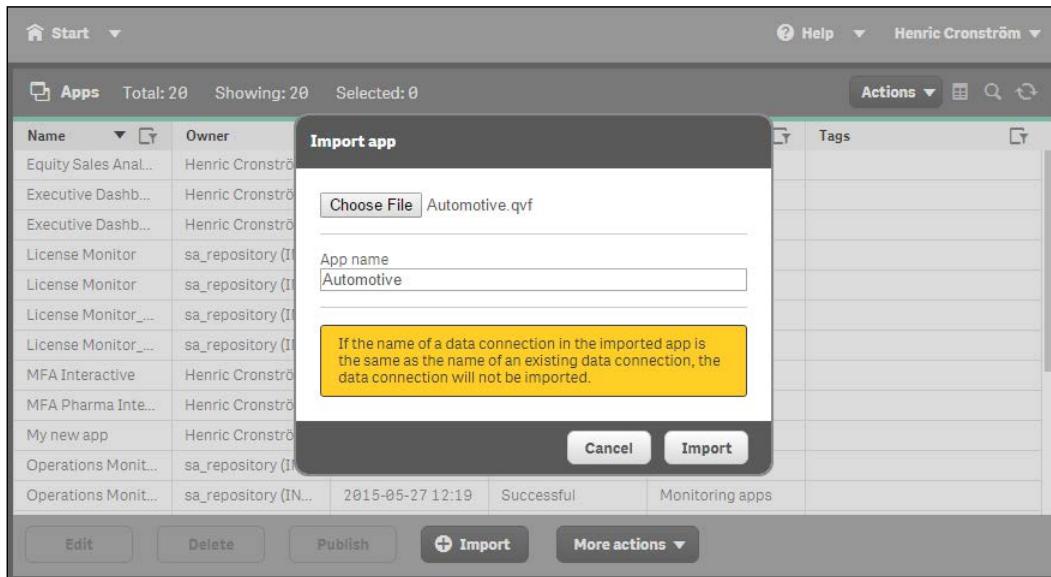
The License usage summary in QMC

Management and monitoring

So far, this chapter has dealt with managing the installation and the licenses, but very little has been mentioned about the real purpose of the Qlik Sense administration, which is, How to handle data and the analysis of data? How to handle the applications, the users, the data connections, the distribution, and so on? This section will cover these areas.

Importing and managing apps

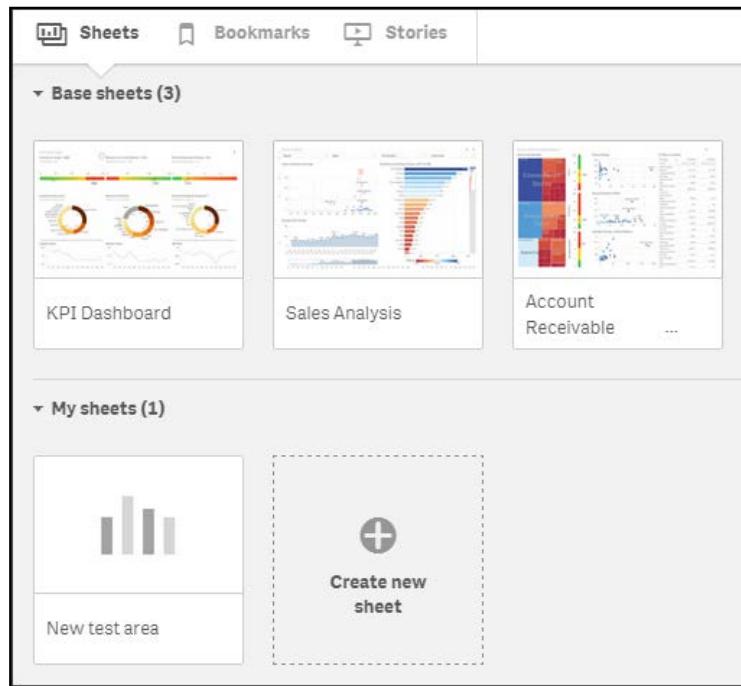
Once the Qlik Sense server is deployed, you might want to import an app that you have created in Qlik Sense Desktop. This is something that you can do in the **Apps** sheet. Look for the **Import** button at the bottom of the screen.



Importing an app created in Qlik Sense Desktop in the Apps screen

Once imported, you can set the owner of the app. Then, it will appear in the **My work** area in the hub of the owner. However, the app is still not published, which means other users cannot see it.

When you publish it, you move the app from **My work** to another stream, and once it is published, its layout is fixed and cannot be changed.

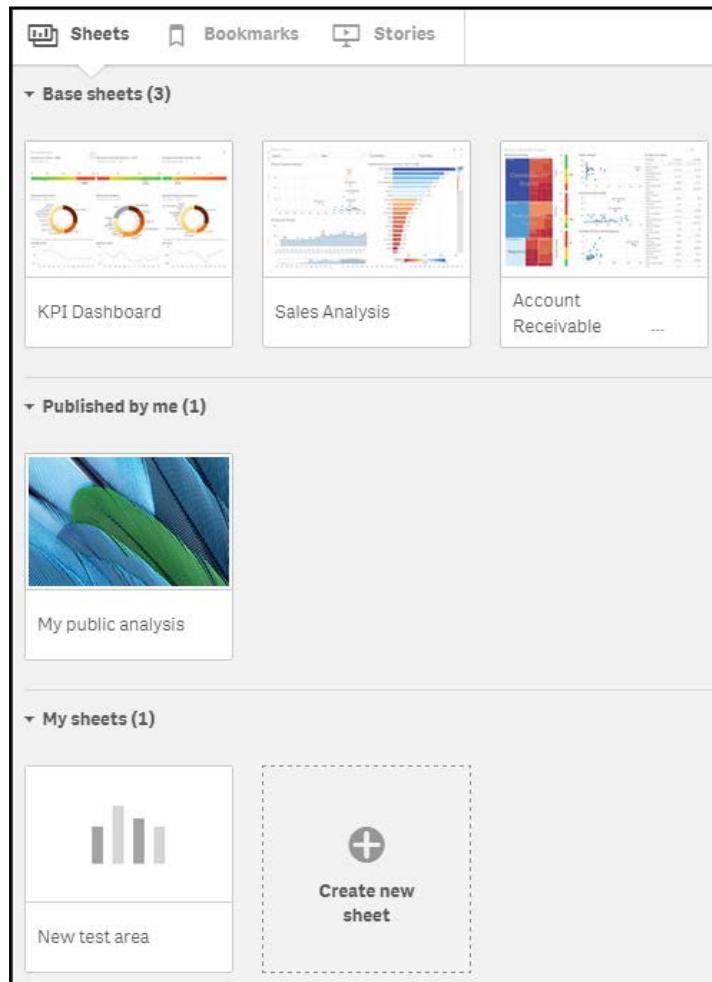


Once an app is published, the app overview in the hub changes

This is obvious if you look at the app overview in the hub. Here, you now have two rows of sheets: one with sheets that are fixed and public, and another with private sheets that aren't visible to other people.

You can also see this difference on the sheet listing the app objects. This sheet lists all sheets and stories, and QMC clearly indicates whether an object is public and who the owner is.

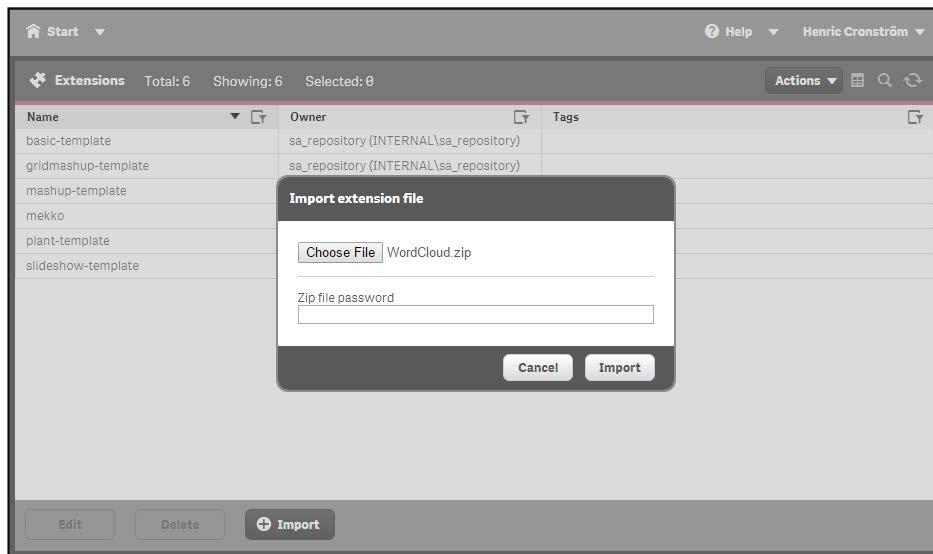
The user who creates an app is automatically designated as the owner of the app and its app objects. The app objects are published when the app they belong to is published. However, the users can add private app objects to the apps and share them by publishing the app objects from Qlik Sense. When this is done, the app overview in the hub gets three rows of sheets, as shown in the following screenshot:



Importing extensions

As you saw in the previous chapter, it is very simple to create additional visualizations, extensions, in Qlik Sense. To use them, you need to import them to your Qlik Sense server.

All you need to do is to navigate to the **Extensions** sheet. Look for the **Import** button at the bottom of the screen. By clicking on this button, you can browse to the location of the ZIP file containing the extension and import it.



Users and user directories

As soon as you want to manage the Qlik Sense server in terms of ownership and access rights, you need to have your users defined. Normally, these are already defined in a user directory, for example, in Windows Active Directory. Hence, you want to reuse these definitions.

On the **User directory connectors** page, you can define several sources for your users and user groups. You need to do this and sync at least one of them before you can start distributing licenses and access rights to your user groups.

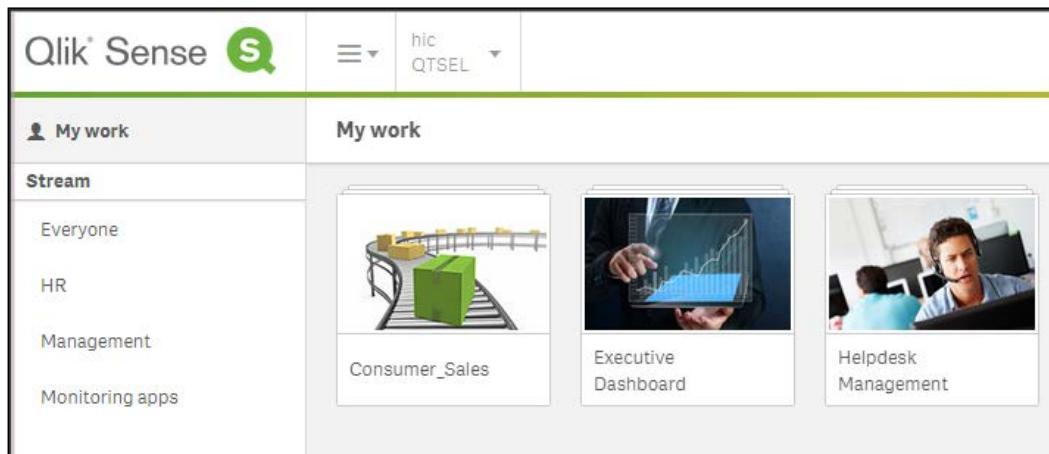
The users are managed on the **Users** sheet. However, when you first start Qlik Sense, the list of users is fairly short: just you and a couple of system users. To populate the list of users, you have two options:

- Define a user directory connector and sync the users in it
- Define rules for the access passes so that the users can assign licenses to themselves without you having to do it

Defining streams

Once you have created or imported an app, you may want to publish it. Publishing an app means that you move it from your personal workspace to a stream of your choice.

You have already seen the streams in the hub, where they appear to the left as groups of apps. **My work** is your personal stream that no one else can see.



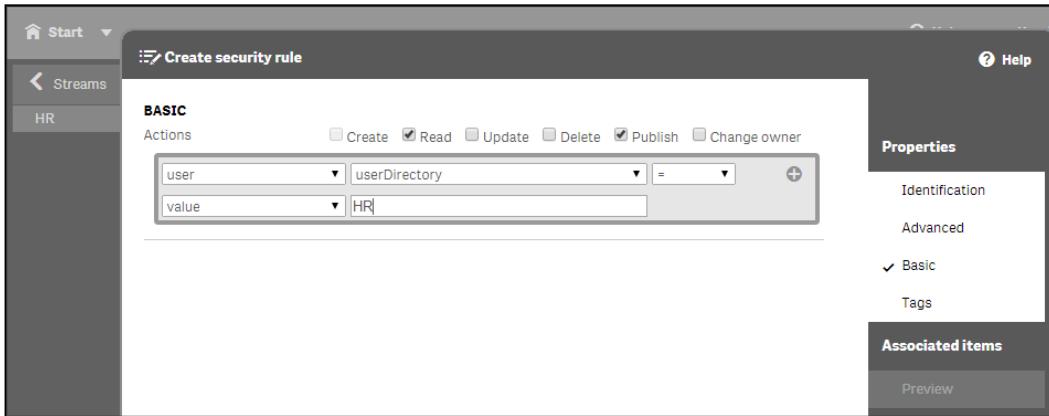
The screenshot shows the Qlik Sense hub interface. At the top, there's a navigation bar with the Qlik Sense logo, a search icon, and user information ('hic QTSEL'). Below the header, a sidebar on the left is titled 'My work' and lists several streams: 'Everyone', 'HR', 'Management', and 'Monitoring apps'. To the right of the sidebar, a main area is titled 'My work' and displays three published apps: 'Consumer_Sales' (with an image of a green cube on a conveyor belt), 'Executive Dashboard' (with an image of a person pointing at a screen with a chart), and 'Helpdesk Management' (with an image of a person wearing a headset). Each app card includes its name and a small thumbnail.

Streams, as seen from the hub

An app can be published to only one stream. By default, Qlik Sense includes a stream called **Everyone**, and you can create any number of additional streams from the **Streams** sheet. You should most likely create one stream for each distinct user group. Use the **Create new** button in the upper-right corner of the screen.

When creating your stream, you have the option to add a security rule, making the stream available only to some users. This is a very important security feature. One obvious example is, if you have a set of apps that should be seen only by the human resources department. Then, you should create a stream for this group and use the user information from the directory service to give access to this stream.

Another common case is if you want to delegate the administration of a stream to a group of users. The following screenshot shows a security rule that grants access to the **Human Resources** stream and to all users belonging to the **HR** user directory:



Connectivity management

Connectivity means the connection to source data. Source data can be ERP systems of different kinds, file folders, web addresses containing tables, and so on. When running a Qlik Sense script, data is pulled from the different sources into the Qlik Sense app so that it can be analyzed at a later stage.

With Qlik Sense, it is easy to get an overview of all data connections used, something that used to be a challenge. By opening the **Data Connections** sheet, you get a list of the data connections used in different apps.

The data connections can be managed and security can be set separately for different connections. It is, for example, possible to prevent some users from using a specific data connector. This way, you can control the usage and ensure that data is used in the correct way.

Tasks

On the **Tasks** page, you define the jobs that need to run in the background. These are of two kinds: **reload tasks** and **user synchronizations**.

The reload tasks are necessary to refresh data in the apps, which means you need to set up tasks so that they are refreshed with the frequency you want. Most apps should be refreshed once a day, but some others only need to be refreshed once a month. There are both advantages and disadvantages with a frequent refresh of the data. If it is refreshed rarely, for example, once per month, the users will not have the latest data.

On the other hand, if you refresh data too often, such as once per hour, you will have a heavy load on your server handling the reload tasks. You will also create a situation where two users in a meeting may have different opinions about what the correct number for a specific KPI is, since they looked at two different versions of the app. One looked at the app an hour ago, and the other just 10 minutes ago. This does not create an understanding; rather, it creates confusion, since you have two versions of the truth.

You should ask yourself whether the users benefit most from having as fresh data as possible, or whether they benefit more from having one truth. A good balance is to have one refresh per day. The users will learn this, and refer to the numbers as today's numbers and yesterday's numbers.

User synchronization is necessary to refresh data from the directory service, so that Qlik Sense is aware of any changes made to groups and users.

A task can be triggered by either a scheduler or the completion of another task. This way, you can get task chaining.

System management

The group to the bottom left in QMC relates to system settings. Here, you can configure the nodes, engines, proxies, schedulers, repositories, sync rules, and certificates. With these, you can configure how the Qlik Sense server should work on different computers. You can do really advanced things here, but this is beyond the scope of this book.

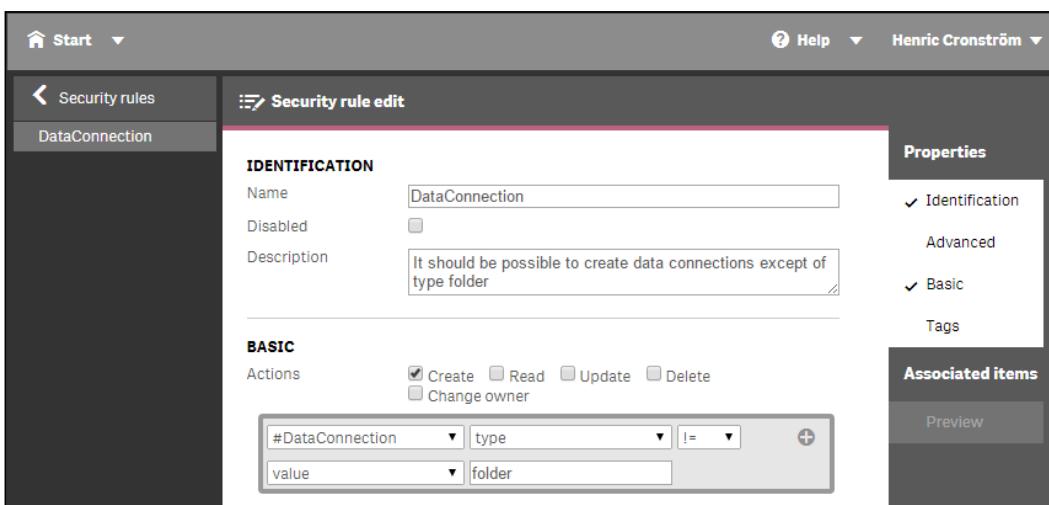
Security rules

You can set access control for most of the preceding settings, for example, only some users should be able to see a specific application; only some users should be allowed to use a specific data connection; all users should be allowed to create data connectors to databases, but not to file folders; only some users are allowed to log in using a specific pool of login access passes; and so on.

When doing so, you should think of the following user types:

- **Developer:** These are users who are allowed to create apps, sheets, stories, objects, and who can use and create data connections
- **Contributor:** These are users who are allowed to create stories and sheets for published apps but are not allowed to create new apps
- **Consumer:** These are users who can only use apps, sheet, stories, objects, and so on but are not allowed to create content

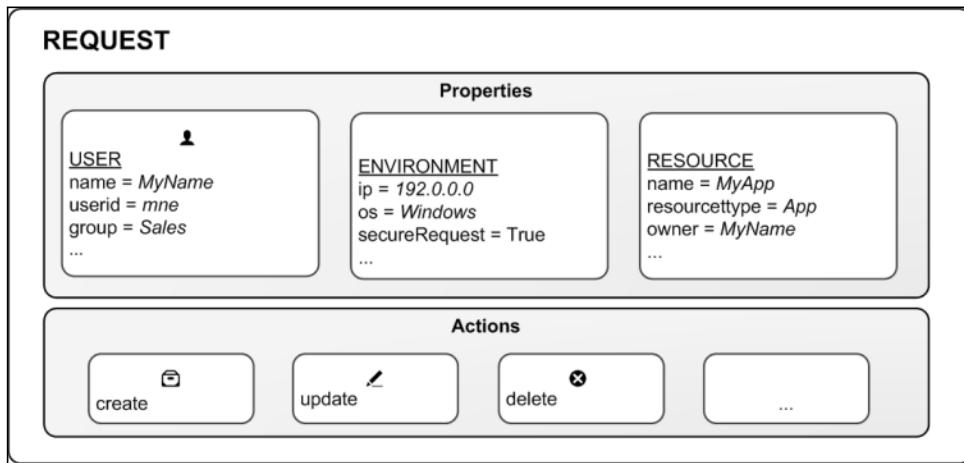
These rules are called security rules, even though they do not always pertain to true security. They can be edited on each sheet, for example, the rules for streams can be edited on the **Streams** sheet, but there is also an overview: the security rules have a sheet of their own where they can be edited.



When you create a security rule using the basic interface, you create a property-value pair that grants users the right to do something. In the preceding screenshot, all users are granted the right to create data connections that aren't file folders.

The rules are property-based and the properties are used to describe the parties involved in an access request. In the usual case, the parties involved are the user making the request, the environment the request is made from, and the resource the request applies to.

Each property is defined in a property-value pair such as **group = Sales** or **resourcetype = App**. Each request, in turn, includes the property-value pairs for the users, environments, and resources involved in the request together with the action that the person making the request wishes to perform on the resource, for example, create, update, or delete.



The four components in security rules: user, environment, resource, and action

You can create rules based on the property-value pairs. This means requests for an action on a resource are granted only if the property value of the requester matches the property-value conditions defined in a security rule for that resource.

A rule can read as a sentence in the following way: *Allow the requester to [action] the [resource] provided that [conditions]*.

Each rule must describe the action and the resource or resources the action should be applied to. If you don't define any rules for a resource, no users will have access to that resource.

By design, security rules are written to include, not exclude, users. Users who are not included in security rules will be denied access. So, security rules must be created to enable users to interact with Qlik Sense content, data connections, and other resources.

Hence, the rules define when access is granted, and there is no rule that can deny a user access. If there is a rule that allows the user to do something, they are allowed to do so. So, if you want to deny a user something, you must delete the rule that grants access, or edit the rule.

Monitoring

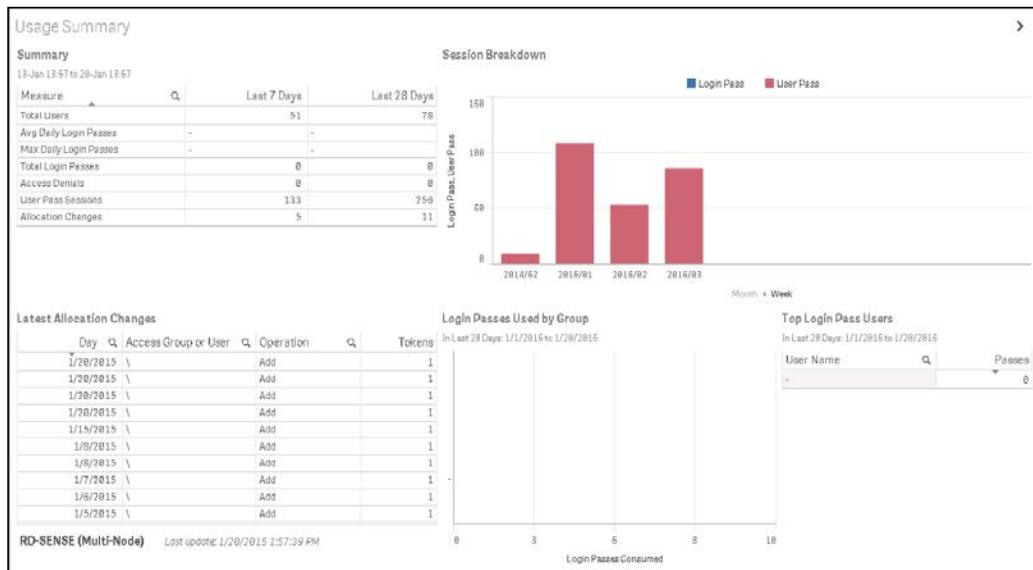
Delivered together with Qlik Sense, you will also find two monitoring Qlik Sense applications: the **License Monitor** and **Operations Monitor**. These read the log files of Qlik Sense and give you a good overview of the state of the Qlik Sense server.

The following screenshot shows the Operations Monitor:



The Operations Monitor

The following screenshot shows the License Monitor:



The License Monitor

Security

The security in Qlik Sense consists of many parts. In the QMC, there is a system with security rules for almost everything you can do, not only data access. There are also rules to change the setup or rights to publish apps or sheets. This implies protection of the platform, that is, how the Qlik Sense platform itself is protected and how it communicates and operates.

However, security as a concept goes beyond that. So let's start from the beginning.

Authentication and authorization

The two most basic concepts in security are authentication and authorization. Authentication answers the question, "Who is the user and how can the user prove it?" Authorization answers the question, "What does this specific user have access to, and what are they allowed to do?"

In Qlik Sense, authentication and authorization are two distinct, unconnected actions. In addition, the sources of information used for authentication do not have to be the same as for authorization, and vice versa.

Qlik Sense uses standard authentication protocols to authenticate every user requesting access, for example, Windows Integrated Authentication, HTTP headers, and ticketing. If you want a customized authentication, you can configure this in the proxy, but the details of this are beyond the scope of this book.

Authorization on the other hand, is the procedure of granting or denying user access to resources. A user perhaps has the right to see a resource or perhaps they don't. When it comes to data, the right to see data can be set with different granularity. A user may see an app or they may not; and once opened, the user may be restricted to see some parts of the app but not other parts.

Hence, authorization can be defined on several levels:

- Firstly, there is the administrator access control. Which rights are needed for the different roles and responsibilities of the administrators? This is controlled in the security rules as previously described.
- Secondly, there is app-level authorization. Is the user allowed to access the app? Which functions in the app is the user allowed to use, for example, printing, exporting, and creating snapshots?
- Thirdly, there is data-level authorization. Is the user allowed to see all of the data in the app or just parts of it?

Content security

Content security is a critical aspect of setting up and managing your Qlik Sense system. QMC enables you to centrally create and manage security rules for all your Qlik Sense resources. Security rules define what a user is allowed to do with a resource, for example, read, update, create, or delete.

Additionally, there is data reduction by a section access in the script that handles data-level authorization. The section access is an app-defined, data-driven security model, intimately connected with the data model. It allows the implementation of row- and field-level data security.

In data-level authorization, the authentication information also exists in the data model (albeit in a hidden part of it). It could be, for example, a username.

The selection propagates to all the other tables in the standard QlikView manner so that the appropriate records in all tables are excluded, wherein Qlik Sense reduces the scope for this user to only the possible records. This way, the user will only see data pertaining to the countries to which they are associated.



Summary

In this chapter, we have seen that with the QMC, you can manage a Qlik Sense installation very efficiently. It includes a wide range of functions that allow you to configure your system the way you want it, and it allows you to set access rights on not only apps but also on streams, licenses, and data connectors.

Since QMC is based on standard web technology, you can, in principle, use any browser to run it, and it integrates well with other systems used to manage software and hardware components. In addition, you can use APIs to create custom management utilities.

To end this book, we'll be looking at putting Qlik Sense into practice for active data discovery as we spend the last chapters analyzing the examples of sales, HR, travel expenses, and demographics.

10

Sales Discovery

Throughout this book, we have shared the driving forces in the creation of Qlik Sense and key capabilities to aid in helping organizations make better business decisions. This chapter is the first of four that will apply Qlik Sense to the challenges of analyzing sales performance within your organization. This example and many others are available for you to explore live at <http://sense-demo.qlik.com>. Please bookmark this link as additional demonstrations and examples are constantly being added and updated. Now, let's turn our attention to the following challenge of sales analysis and how Qlik Sense addresses this common business challenge.

In this chapter, we will cover the following topics:

- Common sales analysis problems
- The unique way Qlik Sense addresses these problems
- How the Sales Discovery application was built

The business problem

Analyzing sales information can be a difficult process for any organization, and is critical to meeting sales expectations and understanding customer demand signals. What makes sales analysis so difficult is that many perspectives can be taken on the enormous amount of information that is captured during the sales process.

Some key questions include:

- Who are our top customers?
- Who are our most productive sales representatives?
- How are our high margin products selling and to whom?

The key thing here is that during the analysis process, one answered question always leads to further questions depending on the results; in other words, the analysis process's diagnostics. These paths to discovery cannot be precalculated or anticipated. With this in mind, let's take a look at how the Sales Discovery application seeks to meet these requirements.

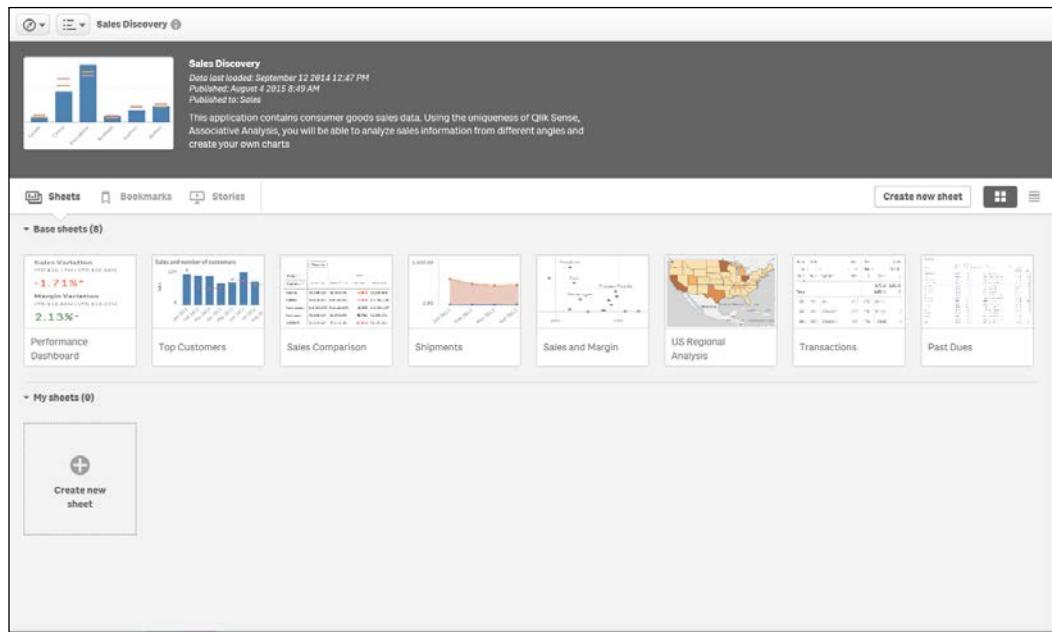
Application features

Qlik Sense's associative model allows users to answer the common questions outlined in the preceding section through the selection of elements in filter boxes, but more importantly, to drive follow-up questions. Often, this relies on Qlik Sense's ability to instantly identify the associated and nonassociated data, which is also known as The Power of Gray after the color assigned to nonassociated elements highlighted in *Chapter 3, Empowering Next Generation Data Discovery Consumers*. The following are two key beginner questions that will drive additional questions as the analysis begins.

Key questions include:

- Who are our top customers?
 - What are these customers buying?
 - Where are these customers buying from?
 - Are the products getting there?
 - Who are our bottom five customers?
- Can we cross-promote products?
 - Who are our most productive sales representatives?
 - What products are the most productive sales representatives selling?
 - Whom are they selling to?
 - Which regions are they being sold in?

Before we begin, let's review the main sheets in the Sales Discovery application. As noted in the following screenshot, the application is made up of the **Performance Dashboard**, **Top Customers**, **Shipments**, **Sales and Margin**, **US Regional Analysis**, **Transactions**, and finally, **Past Dues** sheets:



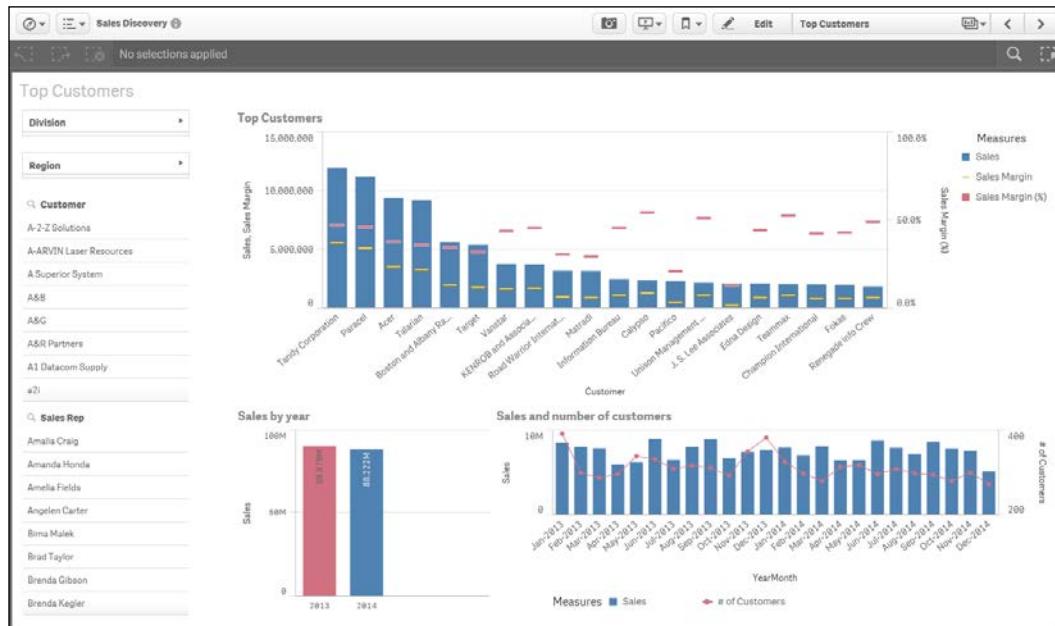
The application overview

Given the nature of the associative model, all filters are global, allowing a user to explore each application sheet in the context of the selected filters and associative results. Filters serve as a way to ask questions to the Qlik Sense application.

Sales Discovery

Who are our top customers?

So, with that said, let's begin with our first question, "Who are our top customers?". This is a typical question that can be handled by a number of BI solutions in the marketplace.

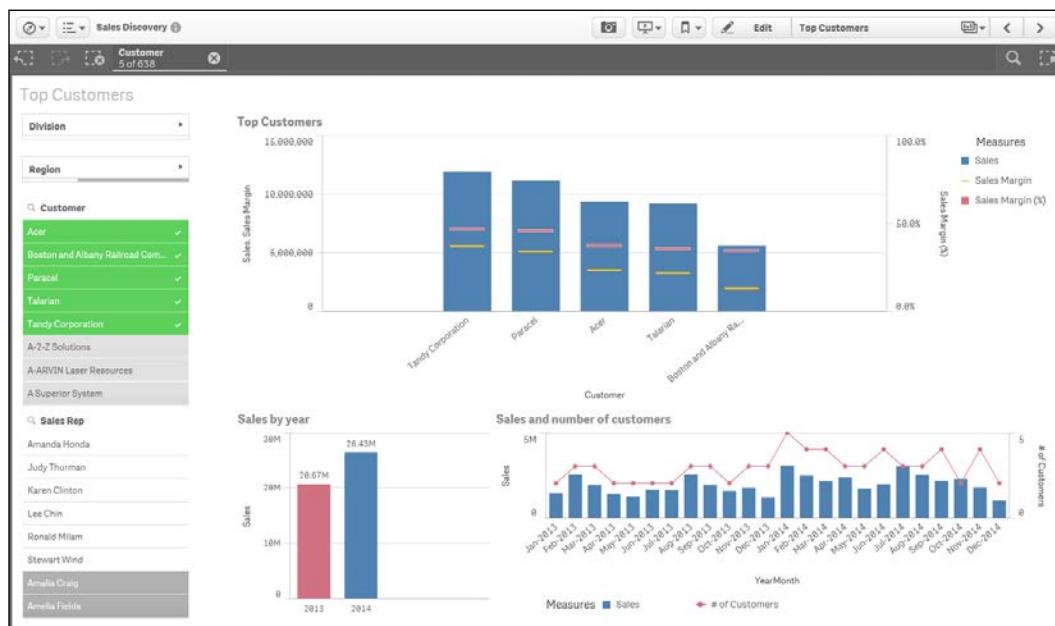


Our top customers

In the preceding screenshot, we can see that the top five customers in terms of sales are **Tandy Corporation, Paracel, Acer, Talarian, and Boston and Albany Railroad Company**.

The 360-degree customer view

Now is where things get interesting in Qlik Sense and the associative experience. Once we select these customers, as shown in the following screenshot, we get a 360-degree view of them across the application. Immediately, we can see which representatives have sold to these accounts, the trended revenue, year-on-year sales, as well as in what percentage of the regions these sales were made. The percentage of the regions (noted by the green arrow) where the sales were made is highlighted in the filter list shade, which shows approximately 25 percent of the regions:



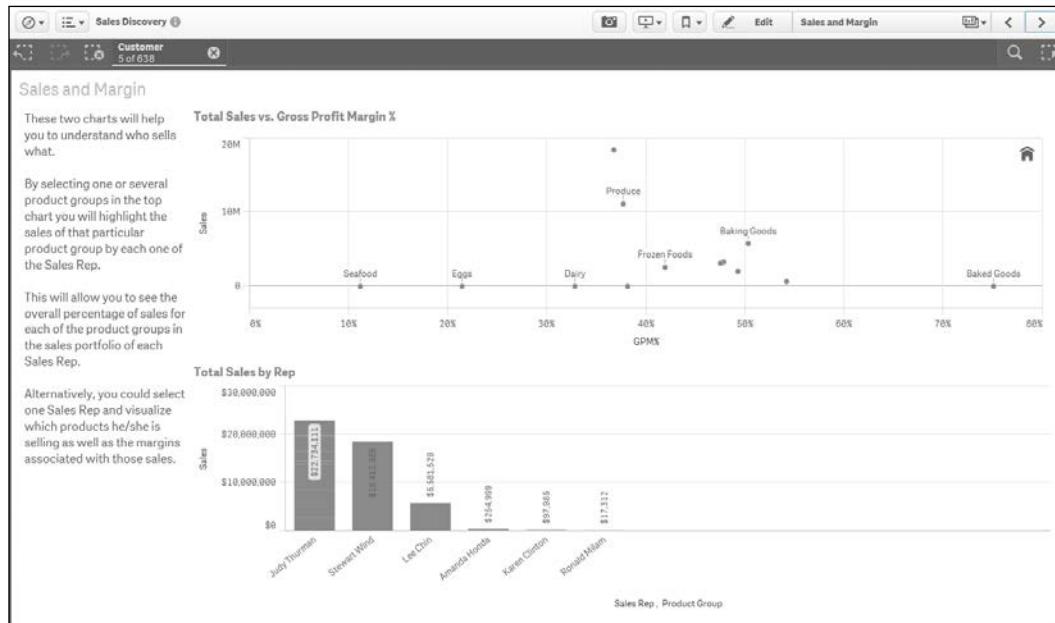
Top five customer sales

Filtering customers

The preceding information leads naturally to the next question: what are these customers buying and from where?

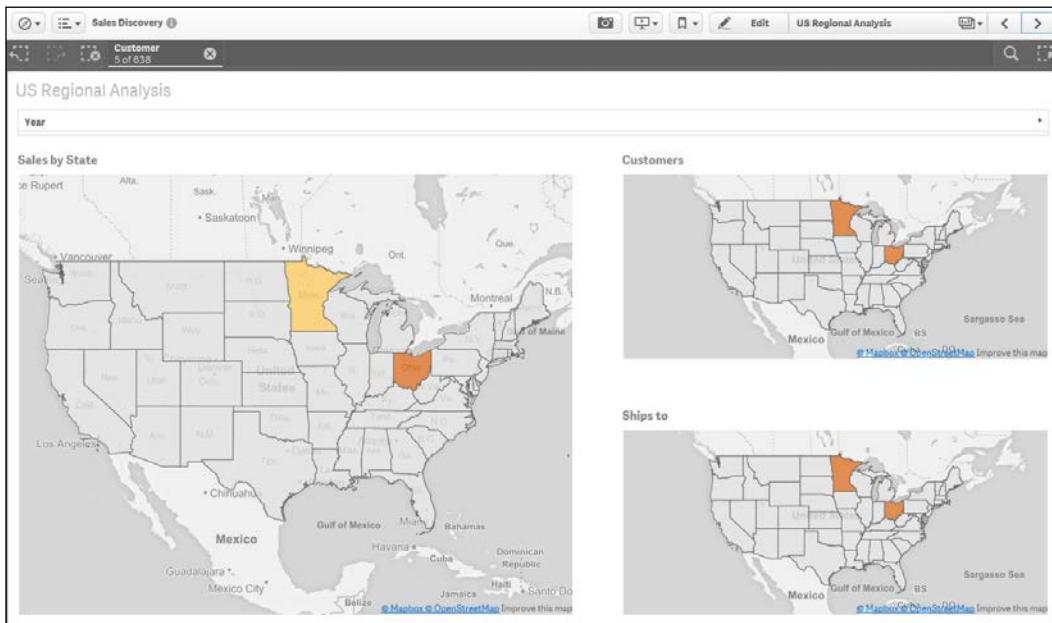
Sales Discovery

Again, because of Qlik's associative indexing engine, this information is linked together automatically. Based on this, let's view the impact that filtering these top five customers has on sales and gross margins, as shown in the following screenshot. Note that the customer filter box with selections is globally available at the top of the screen. In the **Sales and Margin** sheet, we can see that **Canned Foods** and **Produce** account for the largest sales, and **Baking Goods** has the highest gross margin with just over 50 percent.



What are these customers buying?

As we continue our analysis, the next question that is most likely to arise is where are these sales occurring? Again, this data is available in the sales transaction, and Qlik's associative indexing engine makes this easily available and interactive within the application. Note that in the next figure, the **US Regional Analysis** sheet displays the sales by states, customers, and the important shipments as well:



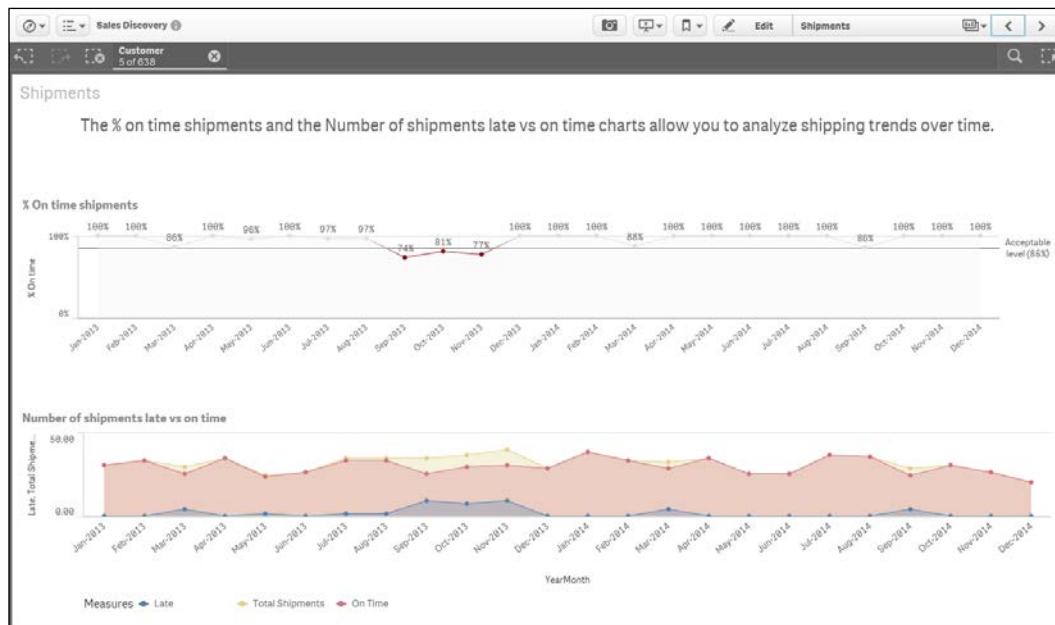
Where are these customers buying from?

Reviewing shipments for top customers

We can see in the preceding screenshot that Minnesota and Ohio account for all top five customer sales that are between 5.94 million and 11.89 million. After reviewing this sheet, a number of questions can arise and be analyzed. Let's follow one specific thought on shipments. Are products getting there?

Sales Discovery

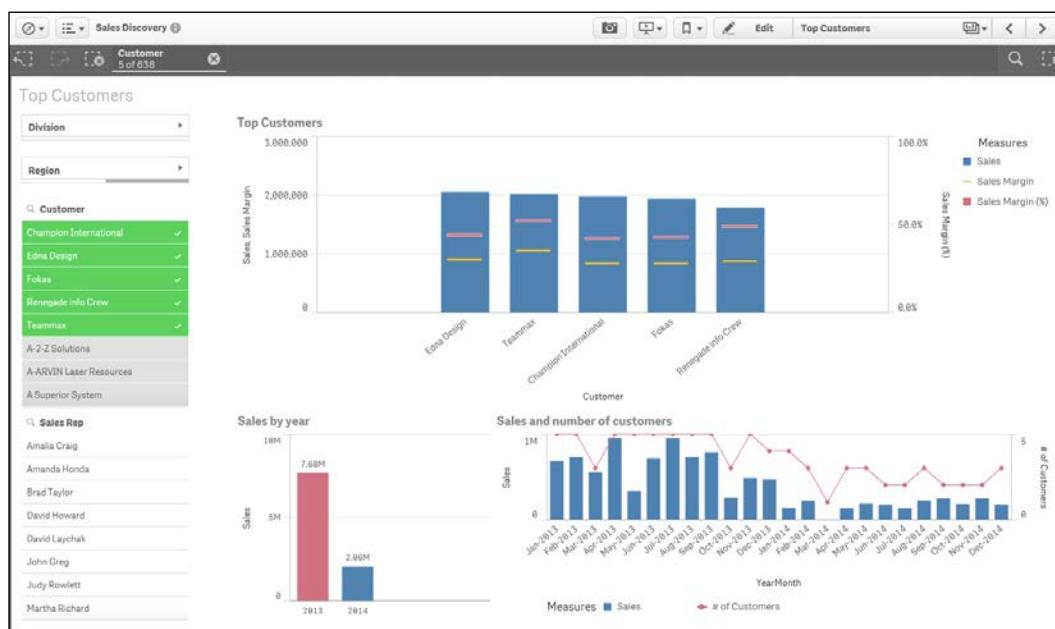
As we know, shipments play a critical role in a sales process because without shipping you cannot book revenue and continue to grow sales. With this in mind, let's turn our attention to the **Shipments** sheet as shown in the following screenshot. From here, we can see the trending shipment information on two levels: **% On time shipments** and **Number of shipments late vs on time**. Additionally, we see that the on-time shipment goal is 86%. Based on this, we can see problems in meeting these goals in September, October, and November 2014:



Are the products getting there?

Reviewing the bottom five customers

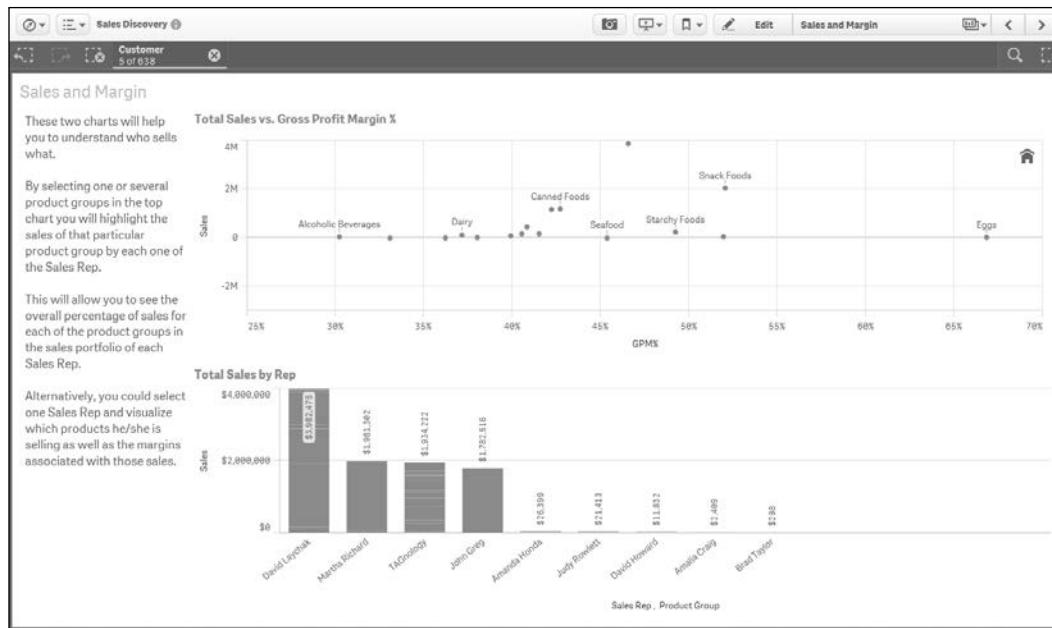
Now, let's turn our attention to an equally important topic: who are our bottom five customers and how can we increase sales to them? In the following screenshot, we can see the bottom five customers: **Edna Design, Teammax, Champion International, Fokas, and Renegade Info Crew**. Our sales to them are 2 million products or less and they purchase lower margin products:



Who are our bottom five customers?

Sales Discovery

While taking a note of this, let's dig in a bit deeper on the products they purchase. In the following screenshot, we can see that these customers purchase a large amount of **Produce** and **Snack Foods**. Now, the question arises—can we cross-promote products to increase our sales from these customers?

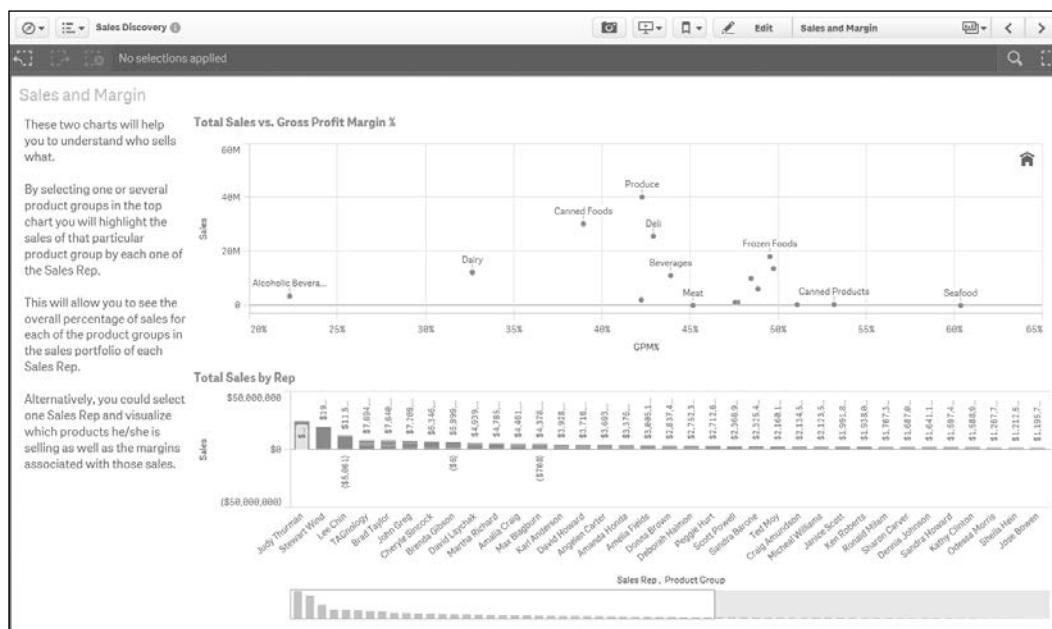


Can we cross-promote products?

Based on the information gleaned, we can see some opportunities to cross-promote products. For example, with the high purchase of **Produce** by these customers, perhaps a cross-promotional program that introduces **Eggs** (at a 67% margin) to them to supplement their produce may raise sales. Additionally, with strong sales of **Snack Foods**, perhaps we can expand the sales of **Baked Goods** (at a 52% margin) to these customers as well. Now, let's turn our attention to the analysis of sales representatives.

Who are our most productive sales representatives?

As often is the case, a key area for analysis is the performance of sales representatives. So, let's turn our attention to the **Sales and Margin** sheet in our Sales Discovery application, as shown in the following screenshot. Here, we can see that **Judy Thurman**, **Steward Wind**, and **Lee Chin** lead the sales team in terms of revenue:

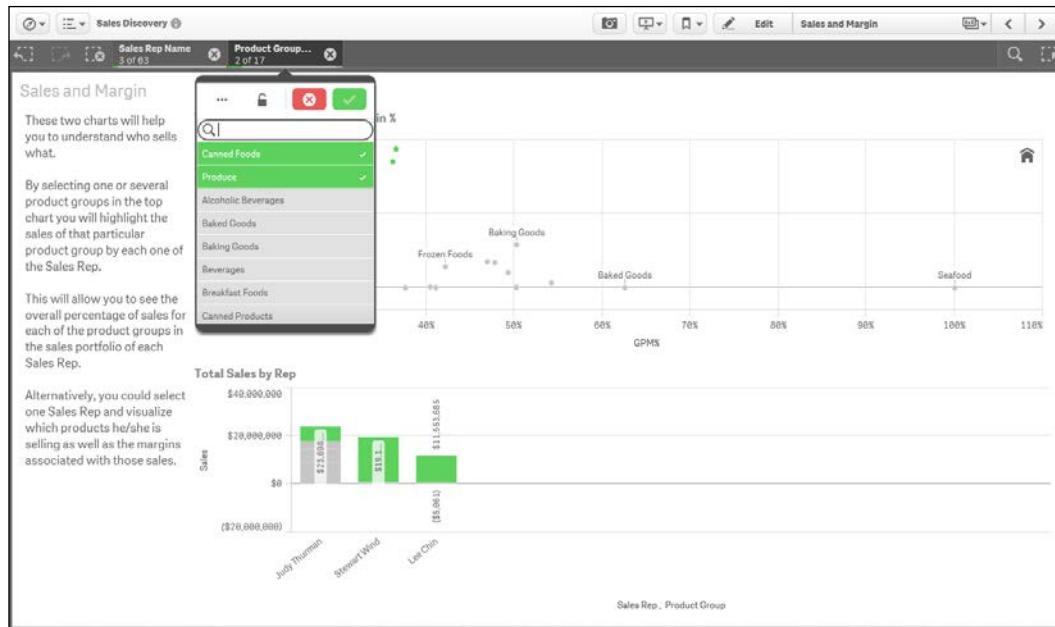


Who are our most productive sales reps?

Sales Discovery

Analyzing products

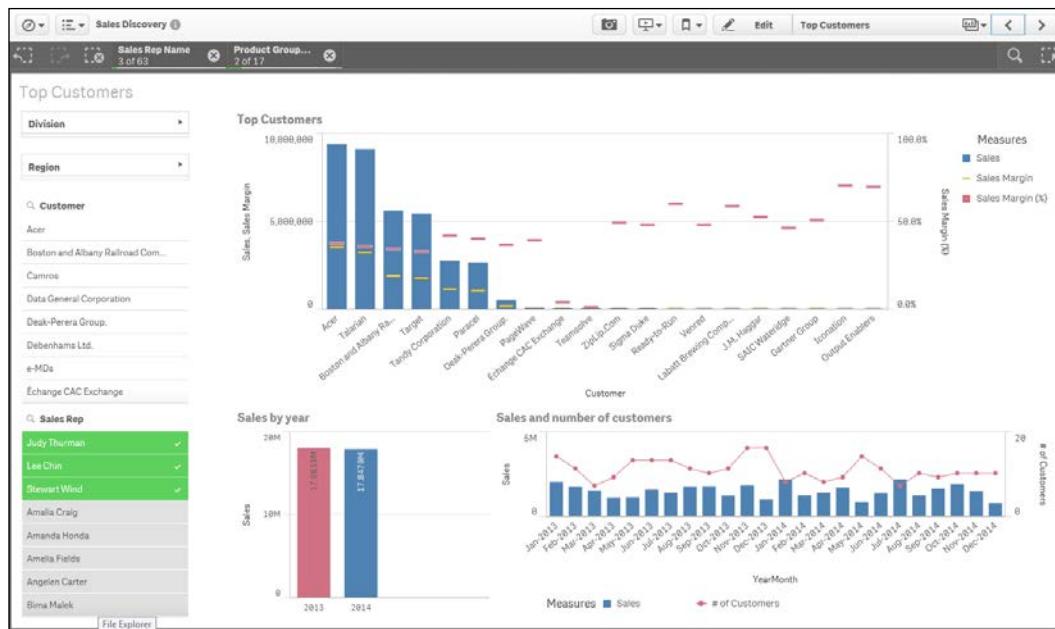
The next question that arises is what products are they selling? As we can see, **Canned Foods** and **Produce** are the top selling products. After identifying these sales representatives and top selling products, we will need to combine this information with an understanding of which customers are driving these sales.



What products are they selling?

Analyzing customer sales

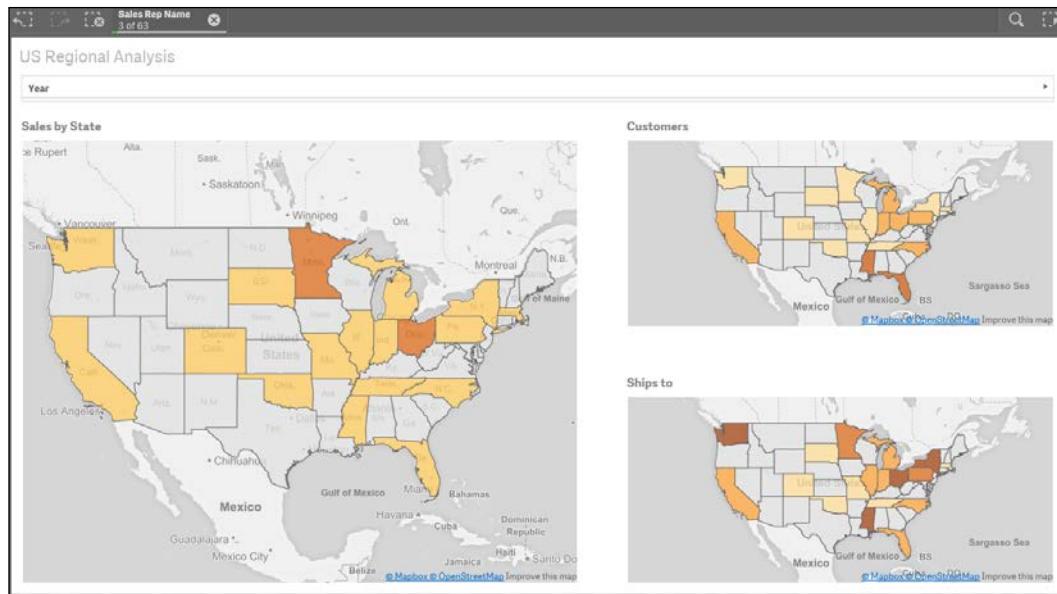
Navigating back to the **Top Customers** sheet, we can see from which customers these sales are generated. Perhaps, while working with these sales representatives, additional promotions can be developed to expand the sales of products such as **Canned Products** to these customers.



Who are they selling to?

Sales Discovery

The final area to help improve sales representative performance is to analyze where these products are being sold. In the **US Regional Analysis** sheet, we can see that **Sales by State**, **Customers**, and **Ships to** are nicely dispersed, and additional information is not necessary for the next step:

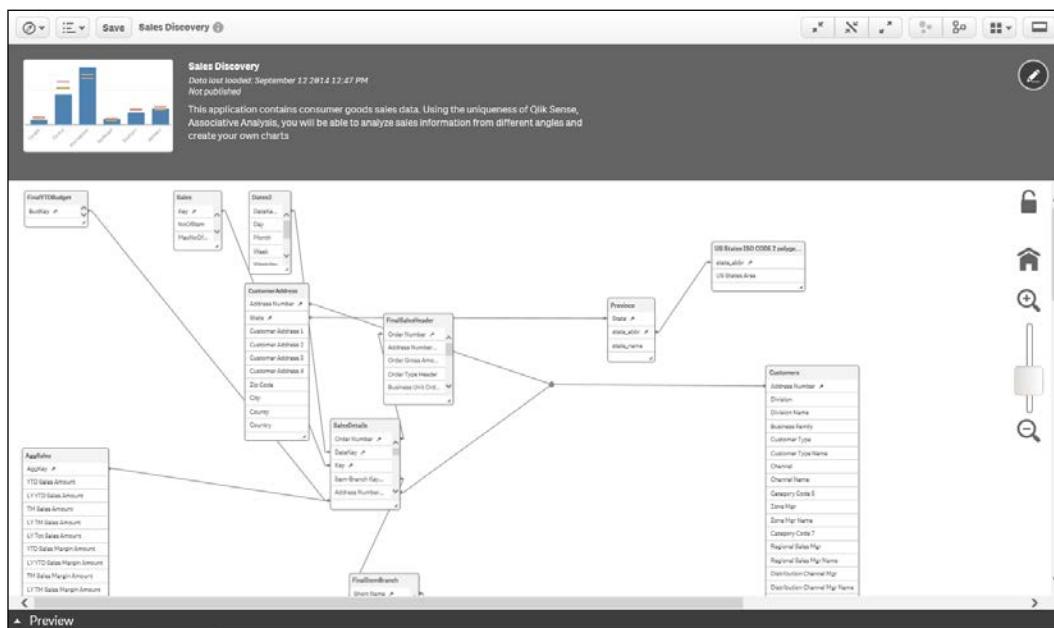


What regions are they selling to?

As you can see, the Sales Discovery application provides a 360-degree view of a sales analysis. This is primarily driven by Qlik's associative indexing engine that drives all Qlik-based applications. Additionally, like most analysis processes, the path to discovery of new information cannot be prestaged but rather unfolds based on the next question asked. This is where Qlik Sense excels in enabling a level of interaction with data to drive insight and is only limited by the data that is available. Now, let's turn our attention to how this application was built.

Building the application

Let's start our review of the heart of a Qlik Sense application, the data model. As you can see from the following screenshot, there are 12 tables in the Sales Discovery associative model. At the heart of this application is the `SalesDetails` table. All these tables were created through **Data Load Editor**, which was covered in *Chapter 5, Authoring Engaging Applications*. It is worth noting that Qlik and Qlik partners provide both general-purpose connectors and specialized connectors to access a broad array of data sources.



The Sales Discovery model

Let's dig a bit deeper into the key tables. The key tables that drive this application are covered in the upcoming sections.

Sales Discovery

The SalesDetails table

The `SalesDetails` table contains all the key information about the sales transaction for a specific order. This includes information such as the order number, date, and so on, as shown in the following screenshot:

| SalesDetails | | Preview of data | | | | | | | |
|--------------|---|-----------------|---------|--------------------------|-----------------|----------------|--------------------|----------|----------|
| Rows | 207976 | Order Number | DateKey | Key | Item-Branch Key | Address Number | AggKey | BudKey | Ship To |
| Fields | 54 | 200286 | 41476 | 7/21/2013 200286CR-10799 | CR-10799 | 10025919 | 100-10025919-28401 | 10025919 | 10025919 |
| Keys | 7 | 205053 | 41452 | 6/27/2013 205053CR-10799 | CR-10799 | 10025919 | 100-10025919-28401 | 10025919 | 100840 |
| Tags | \$key \$numeric \$integer \$timestamp \$date \$ascii \$text | 202518 | 41482 | 5/8/2013 202518CR-10799 | CR-10799 | 10025919 | 100-10025919-28401 | 10025919 | 100786 |
| | | 202147 | 41482 | 5/8/2013 202147CR-10799 | CR-10799 | 10025919 | 100-10025919-28401 | 10025919 | 100786 |
| | | 205654 | 41452 | 6/27/2013 205654CR-10731 | CR-10731 | 10025919 | 100-10025919-27550 | 10025919 | 100786 |
| | | 200285 | 41476 | 7/21/2013 200285CR-10384 | CR-10384 | 10009985 | 173-10009985-20910 | 10009985 | 101263 |

The SalesDetails table

The Customers table

The `Customers` table contains all the key information about the customer: channel, region, account management, and so on:

| Customers | | Preview of data | | | | | | | | | | |
|-----------|--|-----------------|----------|---------------|-----------------|---------------|--------------------|---------|--------------|-----------------|--------------|---------------|
| Rows | 638 | Address Number | Division | Division Name | Business Family | Customer Type | Customer Type Name | Channel | Channel Name | Category Code 5 | Zone Mgr | Zone Mgr Name |
| Fields | 29 | 10015793 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |
| Keys | 1 | 10019988 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |
| Tags | \$key \$ascii \$text \$numeric \$integer | 10007120 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |
| | | 10011981 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |
| | | 10009956 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |
| | | 10025248 | 2 | Domestic | R3 | G2 | Domestic | C7 | Distribution | ZM5 | Western Zone | |

The Customers table

The AggSales table

The `Aggsales` table contains all the sales KPI information and is associated with the model so that sales information is available by customer, product, region, and so on:

The AggSales table

US States ISO CODE 2 polygons

The US States ISO Code 2 polygons table drives the map visualization in the **US Regional Analysis** sheet. The key field is defined by the state, which drives the associative sections, and the field `US_States_Area` is an imported **Keyhole Markup Language (KML)** file that contains the map's geographic information. This is stored as blob data in the model, and the map object interprets this information when used in a sheet. This table is shown in the following screenshot:

The US States ISO CODE 2 polygons table

Analyzing the Sales Discovery Library

Now, let's turn our attention to what has been exposed in the Sales Discovery Library by the developer to facilitate the creation and sharing of personal sheets.

Dimensions

In the next screenshot, we can see the dimensions that were created. One particular dimension that needs attention is the **Region > Cust** dimension, which provides a drill-down navigation from **Region Name** to **Customer**. This capability usually requires extensive modeling or complex scripts in other BI software products, but with Qlik Sense, this is a simple selection process when creating a dimension. This is another example of the power of Qlik's associative indexing engine in action, but this time, easing the development of navigation within the application.

The screenshot shows the Qlik Sense Dimensions library on the left and a detailed view of the 'Region > Cust' dimension on the right. The library sidebar includes categories like Charts, Dimensions, Customer, Division, Month, Product Group, Product Line, Promised Delivery Date, Region, Sales Rep, State, Year, YearMonth, Measures, and Visualizations. The 'Region > Cust' dimension is selected, highlighted in grey. The main area displays the dimension's properties:

- Name:** Region > Cust
- Description:** Drill down from Region to State and then to Customers name.
- Dimension type:** Drill-down
- Fields:** Region Name, State, Customer
- Tags:** (empty)

At the bottom of the detail view are icons for trash, edit, copy, and refresh.

Dimensions

Measures

The next area to cover is **Measures**. These are calculated expressions that most often form the KPIs within an application. In the following screenshot, we can see the list of measures that are used and exposed to contributors to allow them to create private sheets. Note that hovering the pointer over any of these objects makes a preview popup appear to provide additional context. In this case, you can see how the measure is calculated. The following screenshot shows **Measures**:

The screenshot shows the 'Sales Discovery' application interface. On the left, there is a sidebar titled 'Library' with a search bar. Below the search bar are categories: Charts, Dimensions, Measures, GPM%, Margin Variation, Sales, Sales Goal, Sales LY YTD, Sales Quantity, Sales Variation, Sales vs Budget, Sales YTD, and Visualizations. The 'Measures' category is currently selected. To the right, a 'My new sheet' panel is open, displaying a card for the measure 'Avg Sales per customer'. The card includes a description: 'Total sales for a given period divided by the number of customers for the same period.', an expression: 'Sum([Sales Amount])/Count(distinct [Customer])', and two tags: 'customer' and 'sales'. Below the card is a grid area.

Measures

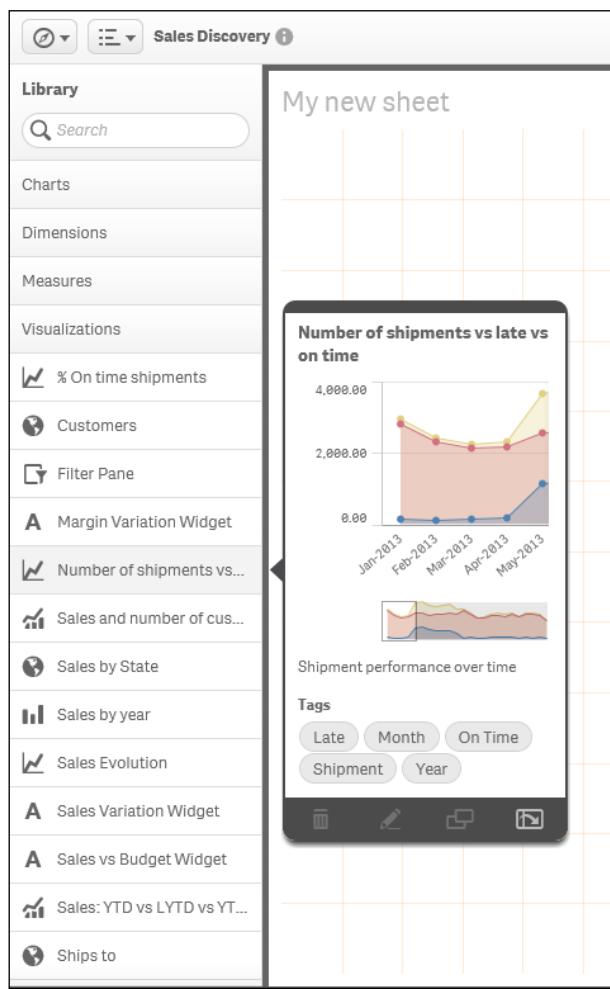
Sales Discovery

Additionally, the following table contains the measure definitions that directly tie to the KPIs used in this application:

| Measure | Calculation |
|------------------------|--|
| Avg Sales per customer | Sum ([Sales Amount]) / Count(distinct [Customer]) |
| GPM% | Sum ({< [Product Group Desc] = {*} >} [Sales Margin Amount]) / Sum ({< [Product Group Desc] = {*} >} [Sales Amount]) |
| Margin Variation | (Sum ([YTD Sales Margin Amount]) / sum ([LY YTD Sales Margin Amount])) - 1 |
| Sales | Sum ([Sales Amount]) |
| Sales Goal | Sum ([YTD Budget Amount]) |
| Sales LY YTD | Sum ([LY YTD Sales Amount]) |
| Sales Quantity | Sum ([Sales Quantity]) |
| Sales Variation | (sum ([YTD Sales Amount]) - sum ([LY YTD Sales Amount])) / sum ([LY YTD Sales Amount]) |
| Sales vs Budget | Sum ([YTD Sales Amount]) / Sum ([YTD Budget Amount]) - 1 |
| Sales YTD | Sum ([YTD Sales Amount]) |

Visualizations

The last category of objects in **Library (Master items)** is **Visualizations**. These are preformed visualizations that are typically the most popular or requested visualizations. They are defined to help facilitate a user's analysis and can be easily dragged and dropped onto a private sheet. Here, we see a trend line chart for **Number of shipments vs late vs on time**. Each of these visualizations contains predefined dimensions, measures, and chart definitions:



Visualizations

Summary

In summary, Qlik Sense provides unique capabilities to meet the challenging task of analyzing sales data. Without the capabilities offered by Qlik, this task can be difficult due to the size of the data and the many perspectives that can be taken in trying to understand customer buying habits, sales representative productivity, and the responsive nature of the organization in meeting customer needs. Qlik's associative indexing engine powers this exploration. This means meeting these requirements is no longer challenging at all.

In the next chapter, we will explore how Qlik Sense will meet the needs of Human Resource Discovery.

11

Human Resource Discovery

Just like the previous chapter, this chapter will show you how to apply Qlik Sense to the challenges of analyzing real data. This chapter's example and many others are available for you to explore on <http://sense-demo.qlik.com>. Again, make sure you bookmark this link, as more demonstrations and examples are constantly being added and updated.

This chapter is about the analysis of human resources data, and it covers the following topics:

- General information about common KPIs
- What a typical data model would look like
- An example of how to use the global selector
- Examples of dimensions and measures

The business problem

The term Human Resources analysis covers a wide area of KPIs that use data from a number of different data sources.

It could be that you want to analyze in-house data, for example, the efficiency of the recruitment process and the costs tied to it. It could just as well be analysis of external data, for example, different employee surveys or sentiment analysis on social media.

Just to give you an idea, we have compiled a list of some of the most common areas to investigate when preparing a Human Resources analysis:

- **Recruitment:** This measures the efficiency of the recruitment process, for example, what is the recruitment cost per employee? What is the average lead time to recruit?
- **Employee satisfaction and retention:** This measures employee loyalty, for example, what is the average satisfaction (as measured by a survey)? What is the employee turnover?
- **Training:** This covers the following questions as examples: What is the total expenditure on training? What percentage of the employees have gone through the training? What is the number of training hours per employee?
- **Health and safety:** This covers the following questions as examples: What is the number of accidents per year? How many employees are of adequate health and get safety training? How much does health and safety prevention cost?
- **Career and compensation:** These cover the following questions as examples: What is the average salary rate? How does it compare to the national average? How much is the salary cost compared to the sales turnover? What is the cost of social and medical insurances?

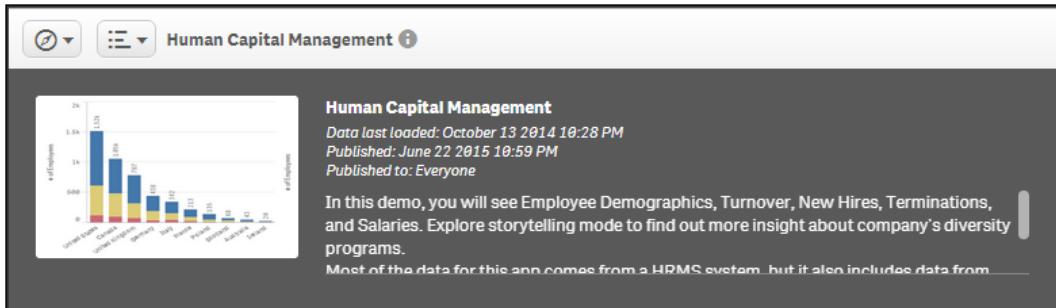
All of the preceding KPIs can be split by a dimension, month, department, position, tenure, age, and so on.

It might be that you don't have data for all the preceding KPIs, but we can assure you that if you do, you will find it worthwhile to analyze them.

Application features

On our demo site, we have a human resource app. You can find it on <http://sense-demoqlik.com>, by clicking on the **Human Capital Management** link. In it, you will find a subset of what you can analyze in HR data. Mainly, it analyzes training investments and employee satisfaction.

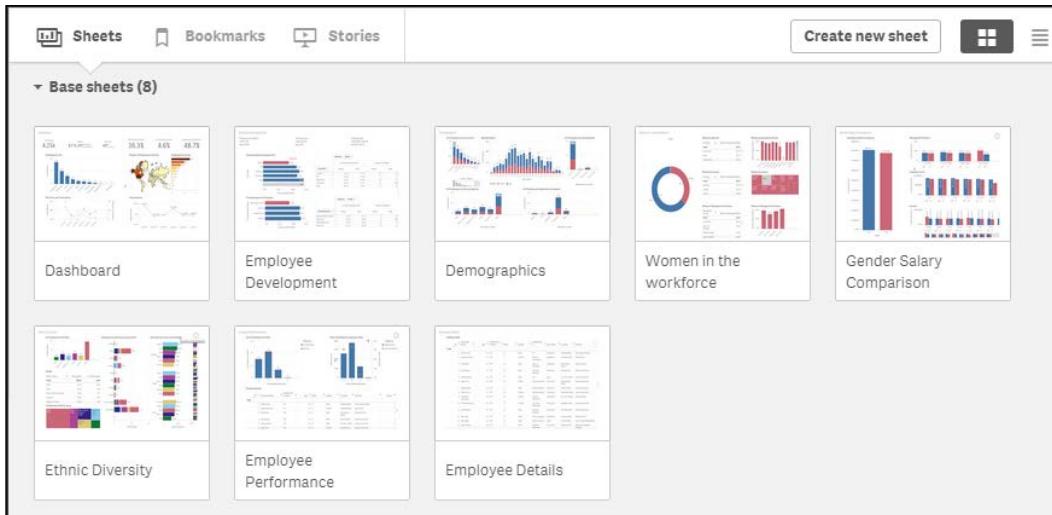
When you open the app, you will first see the app overview, with a small description of the app and a thumbnail in the form of a small bar chart, as shown in the following screenshot:



The overview of the Human Capital Management application

The following is this overview; you will see a number of sheets. These are created according to the dashboard analysis report principles described in *Chapter 5, Authoring Engaging Applications*. This means the leftmost sheet is an overview, very much like a dashboard, whereas the other sheets are prepared for analysis and detailed information.

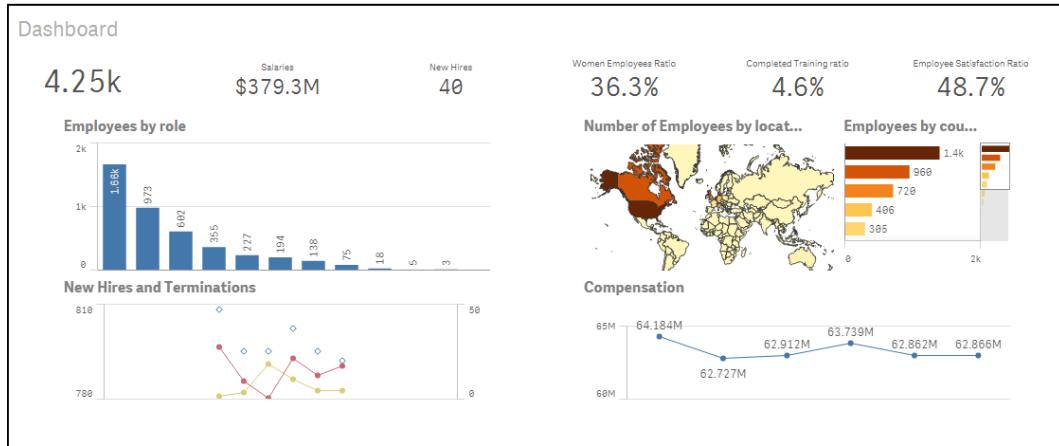
If you click on the **Stories** button, you will see that the app also contains a story – a story that can be used to present the data in the app. It can also be used as an introduction to the app the first time you open it.



The sheets on the app overview page

Sheets

The first sheet is called **Dashboard**, and if you open this, you will see several key numbers, a couple of charts, and a map. This overview is designed so that you can quickly get an overview and a brief understanding of the information without having to make any selections.



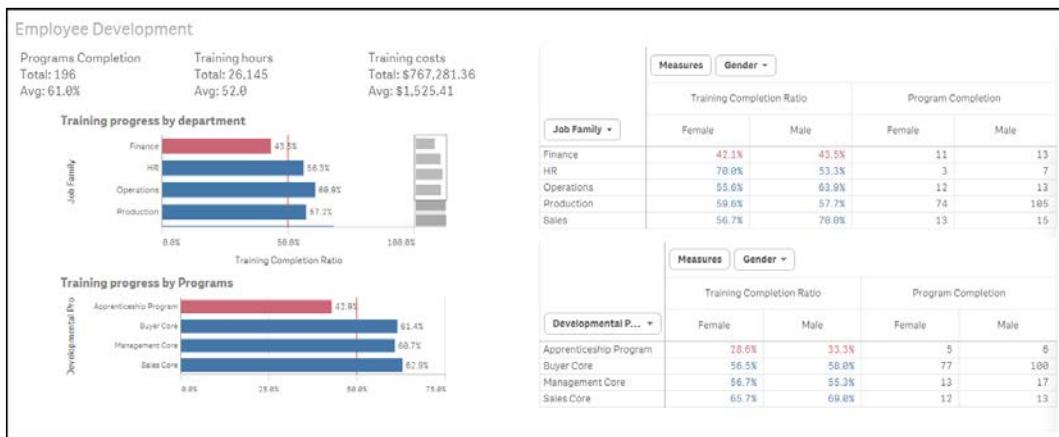
The first sheet – the Dashboard

The top-left chart and the map show the number of employees per role and per country. The two charts at the bottom show the number of hires, number of terminations, and total compensation over time. Note that this sheet does not contain any filter panes because it should not encourage making selections.

The other sheets contain more detailed information, ordered by topics, such as **Demographics**, **Ethnic Diversity**, and so on. The final sheet contains a table only, showing the details about what the application captures, should the user be interested in drilling down to the lowest level.

Training costs

This application contains information that covers only some of the KPIs mentioned in the previous section. One area that it covers well is training. Hence, our first question could be what the total expenditure on training is. The answer to this can easily be found from the **Employee Development** sheet.



The Employee Development sheet

In the top part of the sheet, you can find a textbox containing **Total:** under **Training costs**.

The next question is what the percentage of employees who have gone through the training is. The answer to this can be found from the same sheet. All charts on this sheet show the training completion ratio, split per department or program. By clicking on a chart, you can drill down to the data and explore how the numbers vary between departments, programs, job, course type, and course name.

When you analyze data in a Qlik Sense application, you will realize that there are many ways of using it. For example, say that you want to look at the training progress by gender, to see whether there is any difference between men and women. You have already found the charts showing training progress on the **Employee Development** sheet, but these only show the progress by department and by program.

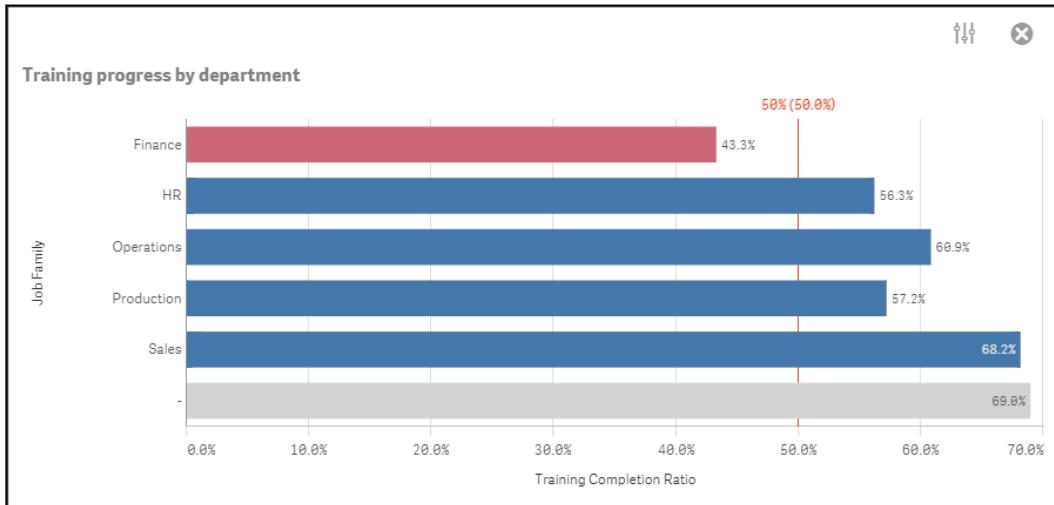
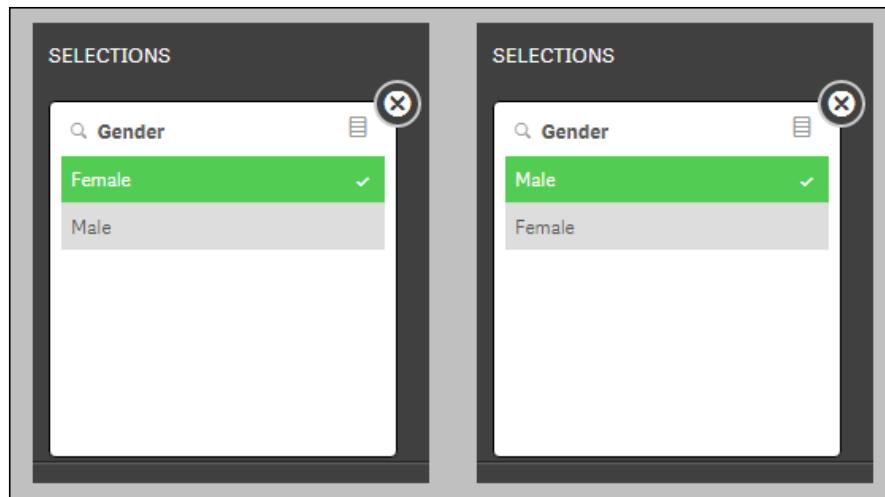


Chart showing training progress by department

Using the global selector

If you have been authorized to create your own visualizations, you can simply go to edit mode, duplicate the sheet, and then drag **Gender** onto the chart, thus replacing the existing dimension with **Gender**.

But even if you aren't allowed to change anything, you can still do your analysis. You just have to do it in a different way. What you could do is use the global selector (the rightmost button in the **SELECTIONS** bar) to select **Female** first, and then **Male**, as shown in the following screenshot:



Selecting Gender in the global selector

When you now close the global selector and return to your **Employee Development** sheet, you can toggle between **Male** and **Female** using the Step back and Step forward buttons to the left in the **SELECTIONS** bar, as shown in the following screenshot. There's also a Clear all selections button in this bar:



The left part of the Selections bar has the Step back, Step forward, and Clear all selections buttons.
Further right, the current selections are listed

This way, you can see how the chart changes as you toggle between the data of male and female employees.

The next question could be about employee compensation. For this, you need to go to the **Employee Performance** sheet. Here, you will find a table showing all employees and the compensation attached to them. By clicking on **Avg Compensation** in the table, you can sort the employees in ascending or descending order, and can thus get a good overview of the span.

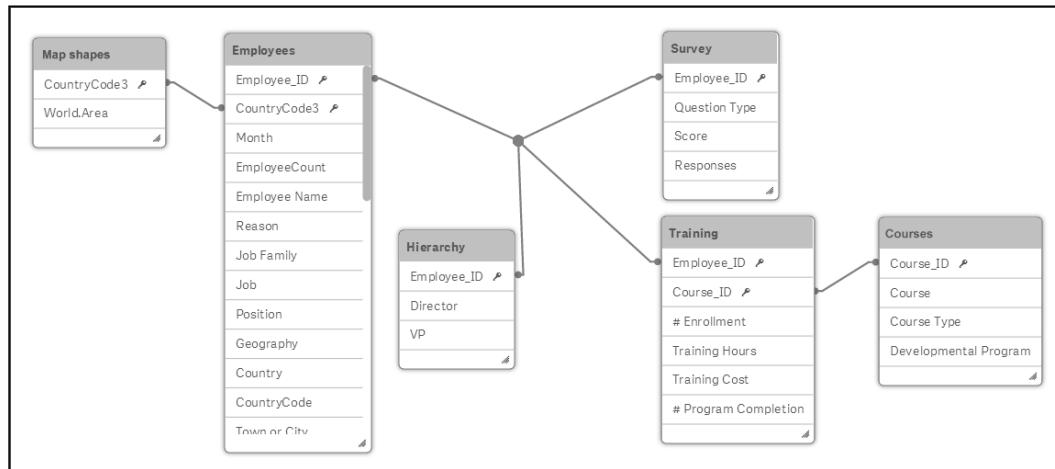
Avg Compensation is the rightmost column in the table, and due to the responsive design of Qlik Sense, it may be that this column is hidden. If so, just click the three dots on the edge of the table, and you can change the order of the columns so that this column becomes visible.

The screenshot shows a Qlik Sense interface with a table containing employee data. The columns are: Location, Position, Employment Status, Age, Band, Gender, Location, Avg Compensation, Position, Tenure, and Avg Compensation. A red arrow points to the 'Avg Compensation' column in the 'Columns' menu, which is currently being reordered. The 'Avg Compensation' column is highlighted in blue.

As users, we would probably also want to see a chart showing the average and total salary costs, split by department, but unfortunately, this has not been created by the app developer. However, in a real-life situation, a user should be empowered to create such charts. After all, it is impossible for an application developer to foresee all the needs of a user. Hence, this is a good example of the need for self-service data discovery.

How the application was developed

The data model for the **Human Capital Management** application looks similar to what is shown in the following diagram:



There are six tables in this application:

- **Employees**: This is the main table, which has one record per employee and month. It contains all the relevant information about the employee such as country, position, salary, and so on. It would probably be a cleaner data model if this table were split into one table containing employee information that doesn't change over time and another table with the time-dependent information.

However, since the QIX engine analyzes the data just as efficiently either way, we don't see any great benefit in spending time structuring the data more.

| Preview | | | | | | | | | |
|-----------|--|-----------------|--------------|----------|---------------|-----------------|-----------------------|------------|-------------|
| Employees | | Preview of data | | | | | | | |
| Rows | 4787 | Employee_ID | CountryCode3 | Month | EmployeeCount | Employee Name | Reason | Job Family | Job |
| Fields | 35 | 231 | GBR | Jun-2014 | 1 | Aaron Cohen | Import created action | Production | Engineering |
| Keys | 2 | 1298 | CAN | Jun-2014 | 2 | Abigail Kennedy | *New Job | Operations | Buyer |
| Tags | \$key \$numeric \$integer \$geoname \$ascii \$text | 1102 | USA | Jun-2014 | 3 | Abram Ruiz | Existing Position | Operations | Receiver |
| | | 49 | USA | Jun-2014 | 4 | Ada Morales | Existing Position | Production | Warehouse |
| | | 1013 | CAN | Jun-2014 | 5 | Adam Garrison | Existing Position | Sales | Sales |
| | | 728 | GBR | Jun-2014 | 6 | Adara Cruz | Import created action | Production | Warehouse |
| | | 1116 | CAN | Jun-2014 | 7 | Adda Heijman | Existing Position | Production | Engineering |

Preview of the Employees table

- **Hierarchy:** This table contains information about the hierarchy of the employee, such as who the manager of the employee is, and who the VP of the employee is.
- **Survey:** This table contains the results from an external survey made on employee satisfaction. Such surveys are usually made once in a year, so if the results from several surveys are kept in this table, the key needs to hold information not only about the employee but also about the year in which the survey was made.
- **Training:** This table contains information about the training sessions attended. Hence, if an employee has attended two courses, two records are stored. The table also contains costs associated with the training sessions.
- **Courses:** The possible courses are stored in this separate table.
- **Map shapes:** This table lists all countries. It has one record per country and could, in principle, hold demographic information about the country. However, in this case, it only holds the map information—the shapes of the countries—used in the map object, which is in the user interface.

Note that this application has fields used for measures in *several* tables: salary can be found in the `Employees` table, cost for training can be found in the `Training` table, and ratings from the survey can be found in the `Survey` table. This is in sharp contrast to classic BI tools, where all such facts need to be in one single table, the `Facts` table.

Dimensions

There are many fields that can be used as dimensions, and a large number of them have been added as dimensions to **Library** such as **Employee Name**, **Age Group**, **Department**, and so on.

In principle, any field that a user would want to use as a grouping symbol should be added as a dimension. However, you should not add keys with cryptic names or numbers that should be used as measures.



The dimensions in Library

A number of measures have also been defined, for example, **# of Employees**, **# of Women**, **Attrition**, **# of New Hires**, **Avg Compensation**, and so on.

It is important that the app developer writes the formula correctly, since this is something that could be difficult for a business user. A business user doesn't always have knowledge about the data model, which is something you need in order to get all the expressions right.

In the following table, you can find some of the measures defined in this app:

| Measure | Definition |
|------------------|--|
| # of Employees | Count (DISTINCT [EmployeeCount]) |
| # of Women | Count ({< [Gender] = {'Female'} >} DISTINCT EmployeeCount) |
| Attrition | Count ({< [Terminated Employee] = {'1'} >} DISTINCT [EmployeeCount]) |
| Avg Compensation | Avg ([Salary]) |

Human Resource Discovery

| Measure | Definition |
|-----------------------------|--|
| Completed Training ratio | Sum([# Program Completion]) / Count(DISTINCT EmployeeCount) |
| Employee Satisfaction Ratio | Avg(Score) |
| New Hires ratio | (Count ({< [New Hires] = {'1'} >} DISTINCT EmployeeCount) / Count(DISTINCT EmployeeCount)) |
| Terminations | Count ({< [Terminated Employee] = {'1'} >} DISTINCT EmployeeCount) |
| Wages Amount | Sum(Salary) |

Finally, there are also a number of visualizations added to **Library**. These are important, as they help a business user in the initial use of the app.

The most common bar charts and tree maps have been stored here: **Number of employees by role**, **Number of employees by management position**, and so on.



The list of visualizations in Library

Summary

In summary, the analysis of human resource data is easy when you use Qlik Sense's unique capabilities. Such an analysis can otherwise be difficult due to multiple and disparate data sources holding human resource data. Qlik's associative indexing engine powers this exploration and analysis is made easy for the user.

In the next chapter, we will look at how Qlik Sense can be used to analyze costs, or more specifically, travel expenses.

12

Travel Expense Discovery

The goal of this chapter is to continue our exploration of Qlik Sense with real data, and how it meets the needs of business discovery in your organization. The Qlik Sense application chosen for this chapter is a topic near and dear to most finance departments, Travel Expense Management. Like all the applications covered in this book, feel free to explore this application live at <http://sense-demoqlik.com>. With that said, let's turn our attention to the following challenge of travel expense management and how Qlik Sense addresses this common business challenge.

In this chapter, we will cover the following topics:

- Common travel expense analysis challenges
- The unique way Qlik Sense addresses these challenges
- How the application was built

The business problem

Expenses are a major line item of every global company. Traveling cost is a part of every sales and service cycle. Unfortunately, most expense tools capture the transaction but do little to help gain insights about how the expenses were spent, when, and most importantly how, to reduce these expenses when possible. Some key questions include:

- How are expenses tracked versus the budget?
- What is the actual amount spent to date?
- What is our largest expense type?
- How can we reduce expenses?

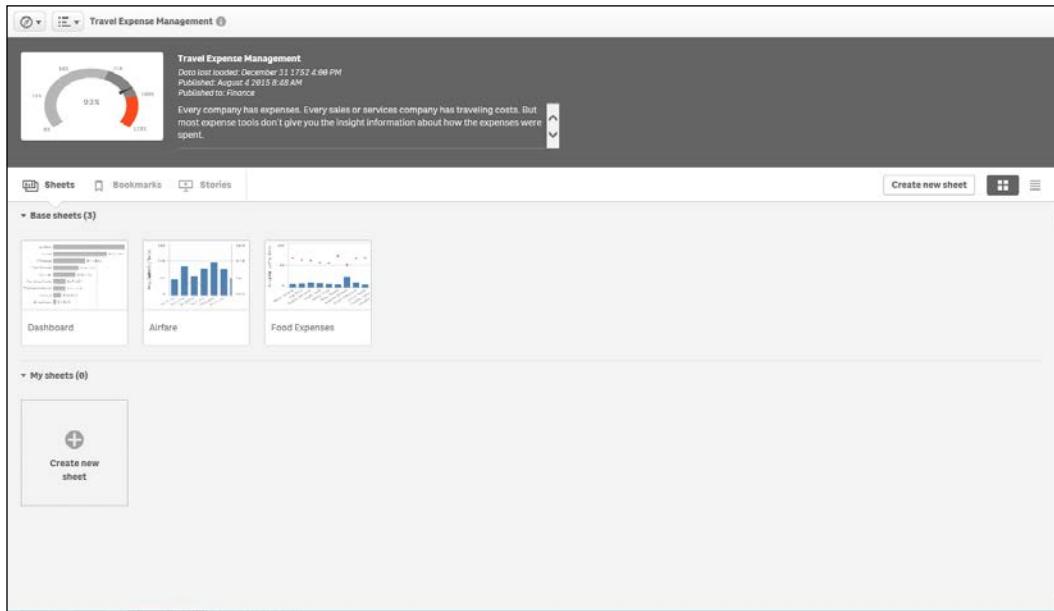
Application features

Now, let's take a look at the unique way Qlik Sense approaches solving the business problems mentioned in the previous section. Qlik Sense's associative model allows users to answer common questions through filters, but they can also address the more important follow up questions that arise. As you may recall, this type of analysis uses "The Power of Gray", named after the color Qlik Sense assigns to nonassociated elements (potential opportunities for improvement) highlighted in *Chapter 3, Empowering Next Generation Data Discovery Consumers*.

The key questions include:

- How are expenses tracked versus the budget?
- What is actual amount spent to date?
- Is my department over budgeted?
- What is our largest expense variance?
- What is the meal expense breakdown?
- How can we reduce expenses?

Before we begin, let's review the main sheets within the Travel Expense Management application. As noted in the following screenshot, the application is made up of three sheets: **Dashboard**, **Airfare**, and **Food Expenses**:



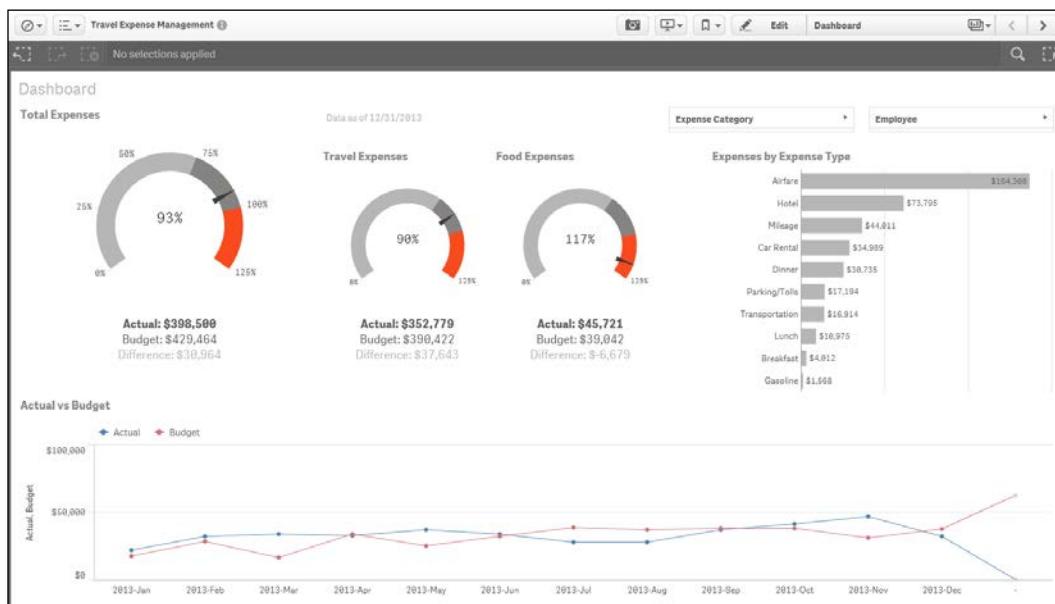
The Travel Expense Management overview

These sheets provide insights into the overall expense management, and the two largest expense categories of airfare and food. With that said, let's now turn our attention to our first question.

Tracking expenses

A key question is how to manage departmental expenses on a quarterly basis. How are expenses tracked versus what has been budgeted?

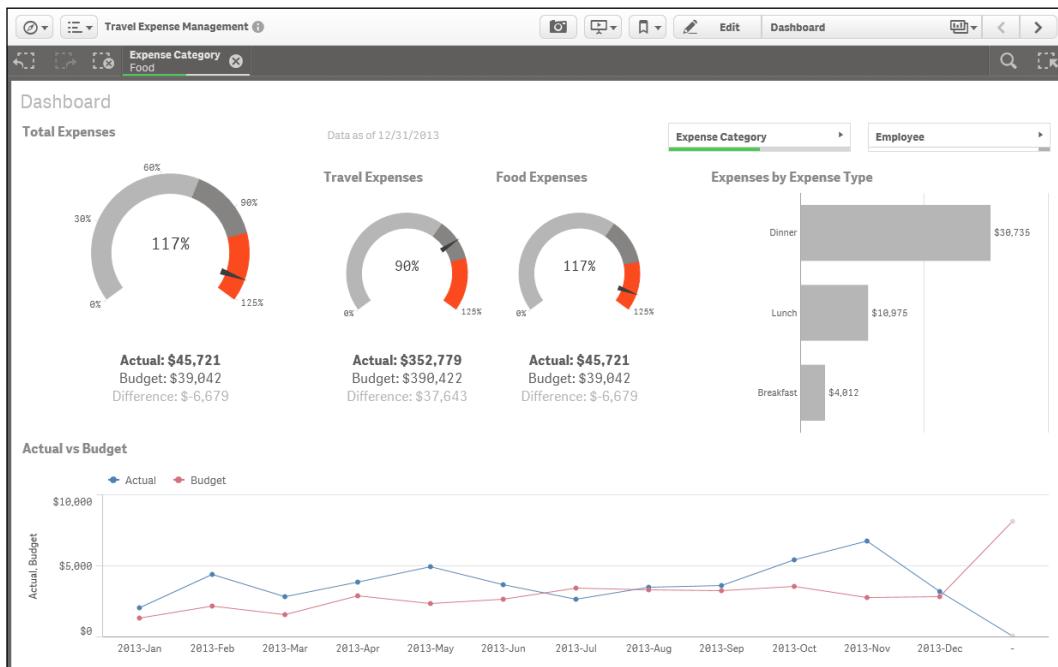
In the following screenshot, we can see in the sample application that **Total Expenses** is below budget by \$30,964. This is good news. Additionally, we see that the largest expense is **Airfare**, and what is more troubling is that **Food Expenses** is running \$6,679 over budget.



Expenses tracked versus budget

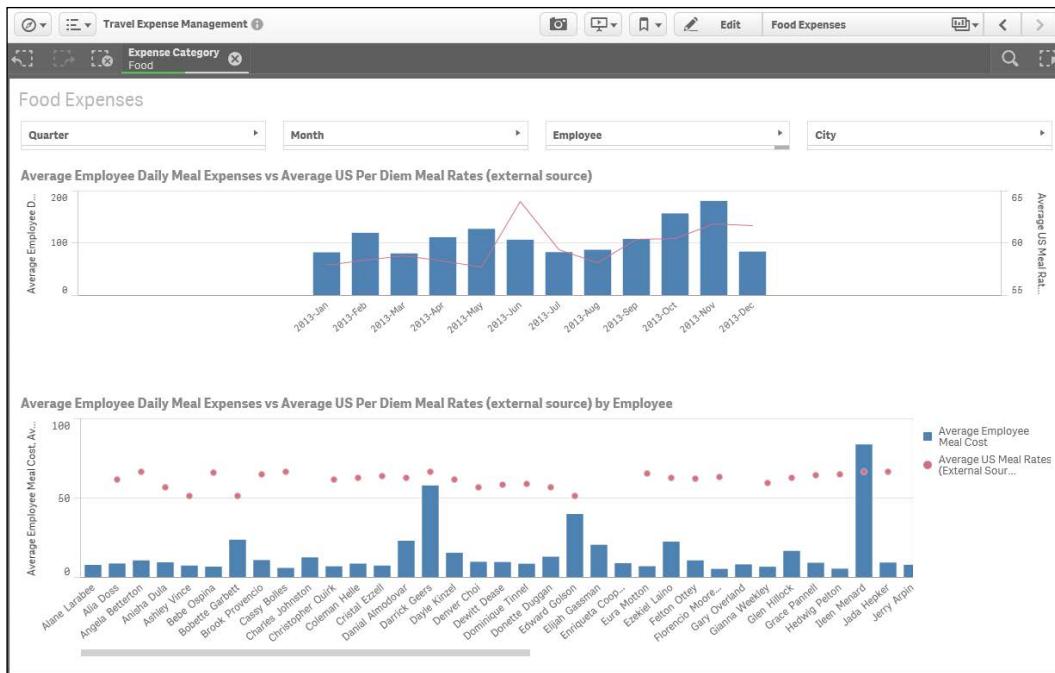
Analyzing expenses overspent

Taking a closer look at food expenses such as breakfast, lunch, and dinner, we can see that for most of 2013 (11 out of 12 months), employees spent more for meals than was budgeted. Exploring a little deeper, we can see that as you'd expect, **Dinner** takes up the majority of the expenses with \$30,735. What's more interesting is that the budget to actual variance starts to sharply grow in May, October, and November.



The meal expense breakdown

Now that we have highlighted a problem with food expenses, let's start to use the more detailed information that is available in the **Food Expenses** sheet, shown in the next screenshot. Additional external information is always helpful in variance analysis. In the following screenshot, we can see not only **Average Employee Daily Meal Expenses vs Average US Per Diem Meal Rates (external source)**, but also **Average Employee Daily Meal Expenses vs Average US Per Diem Meal Rates (external source)** by Employee on a monthly basis:



Average Employee expense versus Average US Meals expenses

Digging deeper into the data

Overall, the company seems to be performing well against the US average, but let's dig a bit deeper. For example, are there employees that do not spend on meals, which could be lowering the company average? To find this out, simply select the global filter (also known as the Selections tool) icon, as shown in the following screenshot:



Global filter

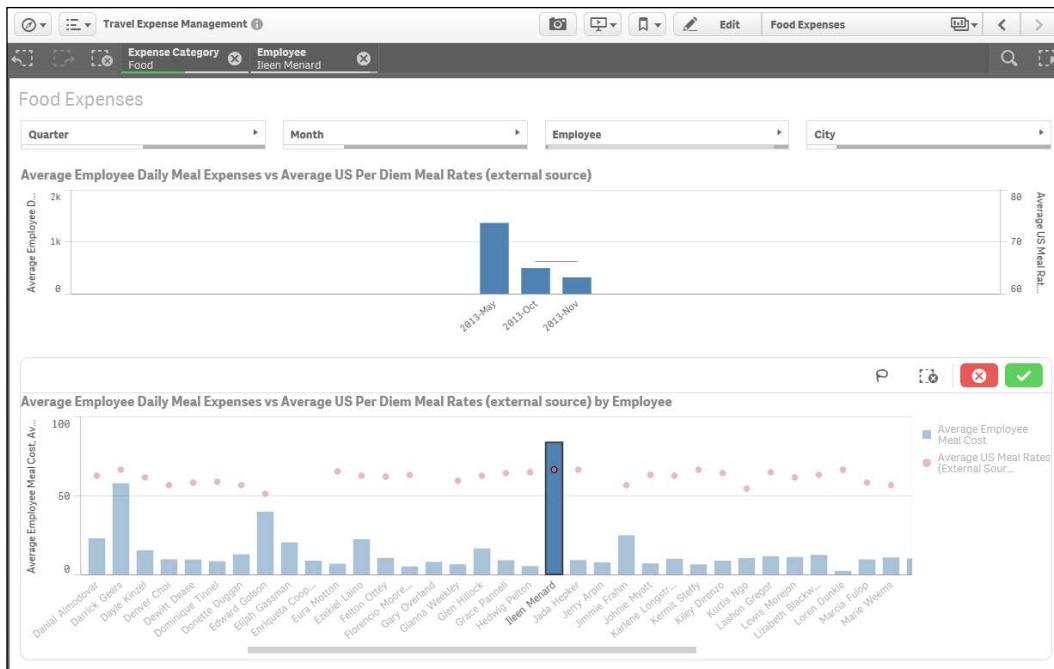
Travel Expense Discovery

Using "The Power of Gray" (nonassociated elements), we can see in the **Employee** dimension that four employees do not spend on their meals, as shown in the following screenshot:

The screenshot shows a Qlik Sense interface titled "Travel Expense Management". In the top left, there's a "CURRENT SELECTIONS" panel with a dropdown menu set to "Food". Below it, a search bar says "Expense Category". To the right, there are four lists of dimensions: "Destination", "Employee", "Expense Type", and "Month". The "Employee" list includes names like Trent Decary, Venise Fultz, Waldo Bigman, Wesley Alants, Zetta Siqueras, Carter Potti, Dustin Behling, Norine Everts, and Ollie Kigo. The "Expense Type" list includes Breakfast, Dinner, Lunch, Airfare, CarRental, Gasoline, Hotel, Mileage, Parking/Tolls, and Time compensation. The "Month" list includes Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, and Oct. The "Origin" list includes Atlanta, GA; Boston, MA; Mobile, AL; Orlando, FL; Philadelphia, PA; and Washington, DC. A search bar at the bottom right says "Search dimensions and fields".

Employees who are not spending on food

Knowing this, we can exit the global filter screen and continue our employee meal analysis. What started off as a travel expense analysis has, through Qlik Sense, narrowed down the analysis to an employee meal analysis. As we scroll through the employees, as shown in the following screenshot, we immediately get to an employee (**Ileen Menard**) who has exceeded the average US per diem allowance significantly, and by selecting **Ileen Menard**, you can see that May was the month with the significant variance. Additionally, the green/white/gray is shown in summary on the selection bar after the dimensions/members are selected in the global filter below:



Employees that exceed the average for US Meals per diem?

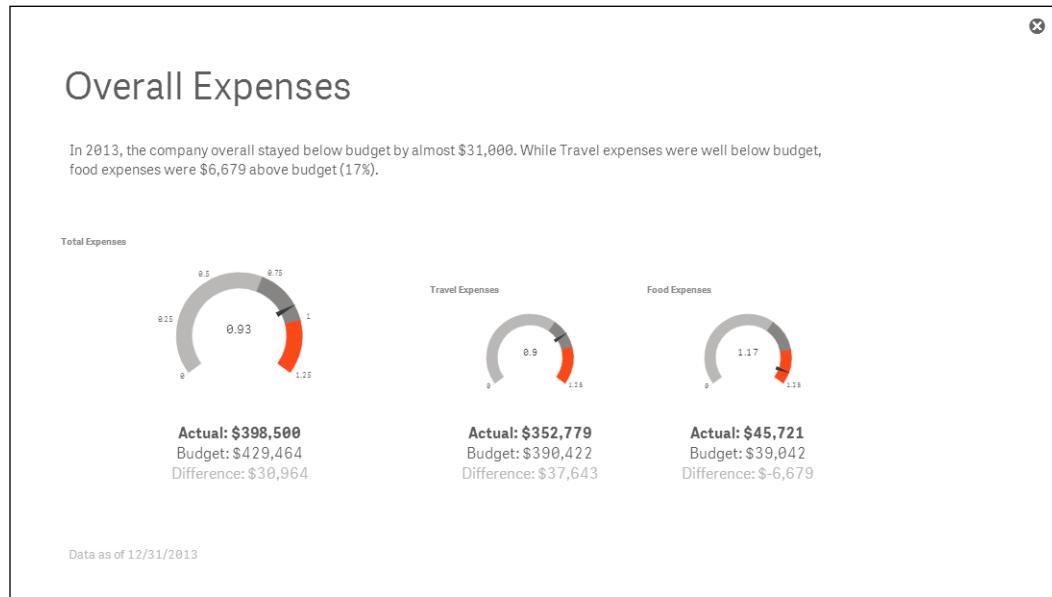
Creating an analysis story for travel expenses

Now that we've completed our analysis, let's create a **Travel Expense Analysis** story so that we can share our findings through our organization.

In *Chapter 4, Contributing to Data Discovery*, we reviewed the role of a contributor, and how to create a Qlik Sense story and publish it so that others may view their analysis. Based on the analysis discussed in the previous section, the travel expense story is made up of three sheets.

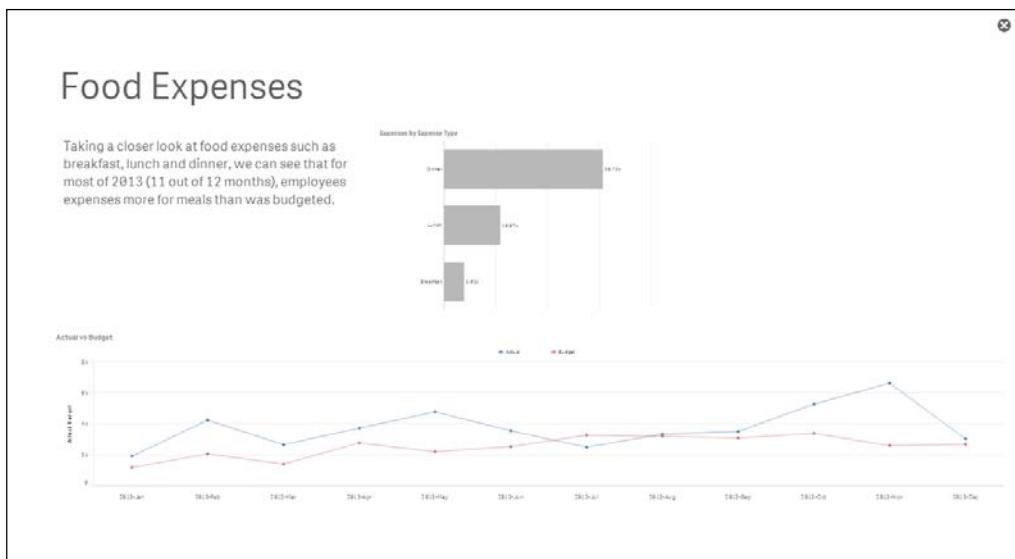
Creating an overview

In the **Overall Expenses** sheet shown in the following screenshot, you can see the **In 2013, the company overall stayed below budget by almost \$31,000. While Travel expenses were well below budget, food expenses were \$6,679 above budget (17%)** annotation as well as the key snapshot gauges of the actual to budget performance total, **Travel Expenses**, and **Food Expenses**:



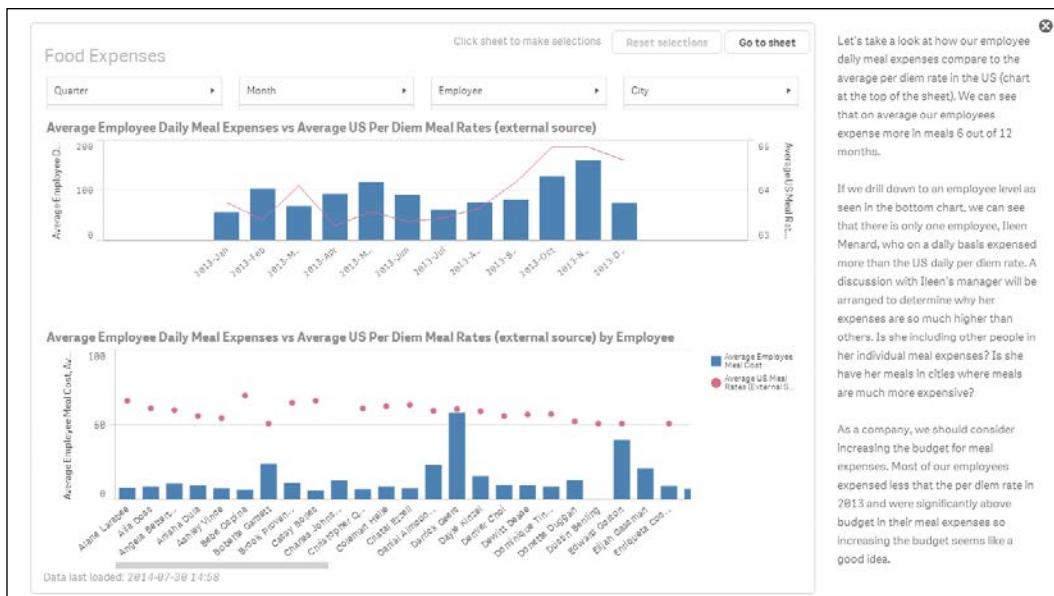
Sharing our analysis

Now that the overview of the analysis is complete, let's move onto the next step and share what was found in analyzing **Food Expenses**. The **Food Expenses** story sheet, shown in the following screenshot, highlights that for most of 2013 (11 out of 12 months), employees spent more for meals than was budgeted:



Finishing the story

With these two story sheets defined, a final sheet for the story will require a bit more interaction for the viewer. As noted in *Chapter 4, Contributing to Data Discovery*, this is achieved by embedding the **Food Expenses** sheet directly into the story, as shown in the following screenshot. This will allow the author to not only narrate the findings but also invite the viewer to explore these findings and others within the application:

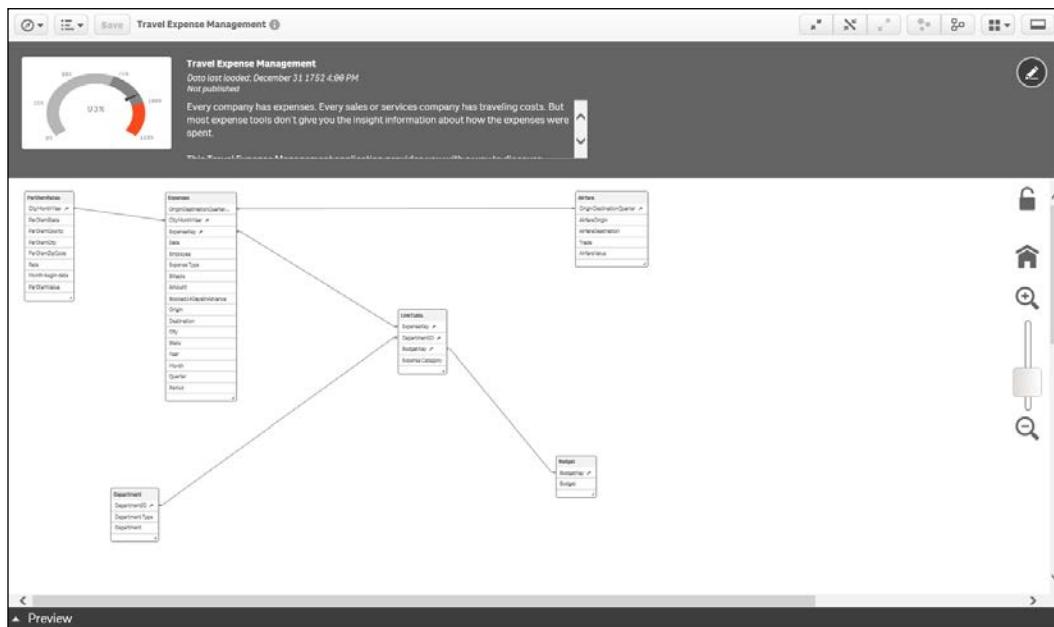


With the story created, the final annotation makes the following recommendation: **As a company, we should consider increasing the budget for meal expenses. Most of our employees expensed less than the per diem rate in 2013 and were significantly above budget in their meal expenses so increasing the budget seems like a good idea.**

Now that we have covered the application features, let's turn our attention to how it was built.

Developing the application

Let's start our review of the heart of a Qlik Sense application, the data model. As you can see from the following screenshot, there are six tables in the Travel Expense Management associative model. At the heart of this application is the Expenses table. These tables were created through **Data Load Editor**, which was covered in *Chapter 5, Authoring Engaging Applications*. It is worth noting that Qlik and Qlik partners provide both general-purpose connectors and specialized connectors to access a broad array of data sources.



Examining the key tables

Let's examine the key tables.

Expenses

The Expenses table contains all the key information (shown in the following screenshot) about the expense transaction of an employee. This includes information such as the date, employee name, expense type, and so on.

| Preview | | | | | | | |
|----------|---|-----------------------|--------------------------|--------------------|------------|------------------|--------------|
| Expenses | | Preview of data | | | | | |
| Rows | 18527 | CityMonthYear | OriginDestinationQuarter | ExpenseKey | Date | Employee | Expense Type |
| Fields | 17 | Las Vegas_Oct_2013 | _Q4 | Food_101_10/1/2013 | 10/17/2013 | Angela Betterton | Breakfast |
| Keys | 3 | Las Vegas_Oct_2013 | _Q4 | Food_101_10/1/2013 | 10/17/2013 | Angela Betterton | Dinner |
| Tags | \$key \$ascii \$text \$numeric \$integer \$timestamp \$date | Las Vegas_Oct_2013 | _Q4 | Food_101_10/1/2013 | 10/17/2013 | Angela Betterton | Dinner |
| | | Las Vegas_Oct_2013 | _Q4 | Food_101_10/1/2013 | 10/17/2013 | Angela Betterton | Dinner |
| | | Philadelphia_Nov_2013 | _Q4 | Food_101_11/1/2013 | 11/7/2013 | Angela Betterton | Breakfast |
| | | Philadelphia_Nov_2013 | _Q4 | Food_101_11/1/2013 | 11/7/2013 | Angela Betterton | Breakfast |

The Expenses table

PerDiemRates

The PerDiemsRates table contains all the key information (shown in the following screenshot) about state, city, month, rates, and so on:

| Preview | | | | | | |
|--------------|---|--|--------------|---------------|-----------------------------------|-------------|
| PerDiemRates | | Preview of data | | | | |
| Rows | 1994533 | CityMonthYear | PerDiemState | PerDiemCounty | PerDiemCity | PerDiemZipC |
| Fields | 8 | Riverhead / Ronkonkoma / Melville_Oct_2011 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |
| Keys | 1 | Riverhead / Ronkonkoma / Melville_Nov_2011 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |
| Tags | \$key \$ascii \$text \$numeric \$integer \$timestamp \$date | Riverhead / Ronkonkoma / Melville_Dec_2011 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |
| | | Riverhead / Ronkonkoma / Melville_Jan_2012 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |
| | | Riverhead / Ronkonkoma / Melville_Feb_2012 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |
| | | Riverhead / Ronkonkoma / Melville_Mar_2012 | New York | Suffolk | Riverhead / Ronkonkoma / Melville | 001 |

The PerDiemRates table

Airfare

The Airfare table contains all the key information (shown in the following screenshot) about the origin, destination, airfare value, and so on:

| Preview | | | | | |
|---------|--------------------------------|---|--------------------------|-----------------------|--------------|
| Airfare | | Preview of data | | | |
| Rows | 131634 | OriginDestinationQuarter | AirfareOrigin | AirfareDestination | Trade |
| Fields | 5 | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q1 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| Keys | 1 | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q2 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| Tags: | \$key \$ascii \$text \$numeric | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q3 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| | | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q4 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| | | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q1 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| | | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q2 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| | | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q3 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |
| | | Allentown/Bethlehem Area_Dallas/Fort Worth, TX_Q4 | Allentown/Bethlehem Area | Dallas/Fort Worth, TX | Average Fare |

The Airfare table

Department

The Department table contains all the key information (shown in the following screenshot) about the department ID, type, and department name:

| Preview | | | | | |
|------------|--|-----------------|-----------------|-----------------------|--|
| Department | | Preview of data | | | |
| Rows | 22 | DepartmentID | Department Type | Department | |
| Fields | 3 | 101 | C-level | C-level | |
| Keys | 1 | 102 | Finance | Finance | |
| Tags: | \$key \$numeric \$integer \$ascii \$text | 104 | HR | Human Resources | |
| | | 106 | Marketing | Marketing - Corporate | |
| | | 107 | Sales | Support | |
| | | 108 | Marketing | Marketing - Field | |

The Department table

Budget

The Budget table contains all the key information (shown in the following screenshot) about the budgeted amount using a compound key value that includes the expense type, department ID, and date:

| Budget | | Preview of data | |
|--------|--------------------------------|-------------------|--------|
| Rows | 504 | BudgetKey | Budget |
| Fields | 2 | Food_101_1/1/2013 | 18.1 |
| Keys | 1 | Food_101_2/1/2013 | 16.4 |
| Tags: | \$key \$ascii \$text \$numeric | Food_101_3/1/2013 | 27.4 |
| | | Food_101_4/1/2013 | 15.6 |
| | | Food_101_5/1/2013 | 31.7 |
| | | Food_101_6/1/2013 | 38.5 |

The Budget table

LinkTable

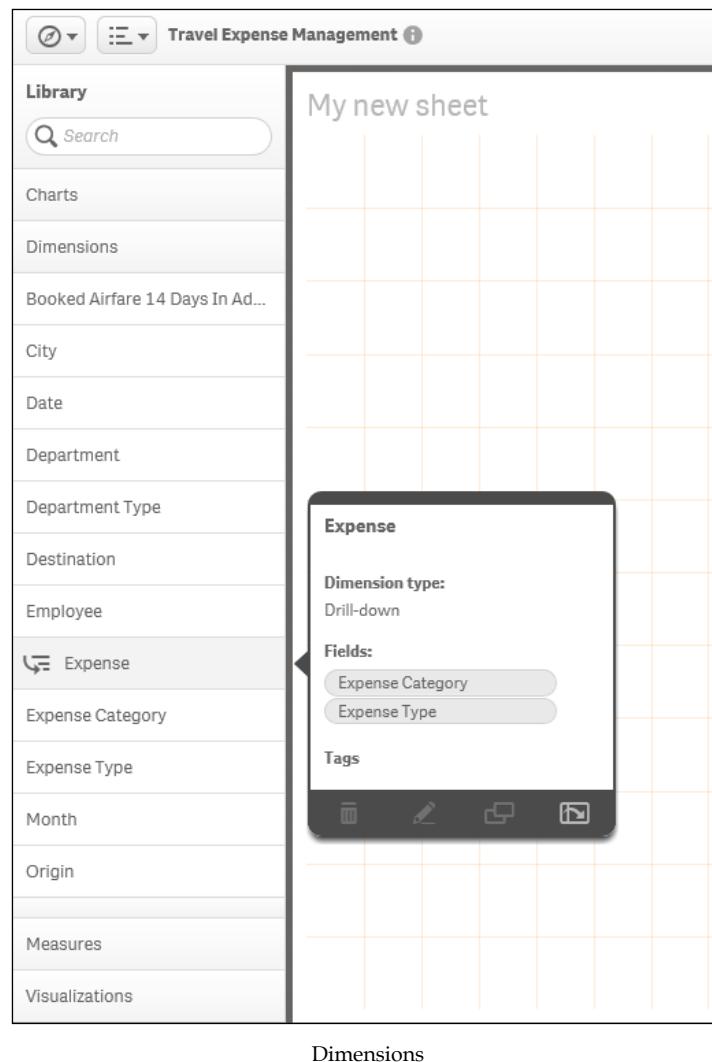
LinkTable contains all the keys (shown in the following screenshot) to link the expense, department, and budget tables:

| LinkTable | | Preview of data | | | |
|-----------|--|---------------------|--------------|---------------------|------------------|
| Rows | 504 | ExpenseKey | DepartmentID | BudgetKey | Expense Category |
| Fields | 4 | Travel_401_1/1/2013 | 401 | Travel_401_1/1/2013 | Travel |
| Keys | 3 | Food_401_1/1/2013 | 401 | Food_401_1/1/2013 | Food |
| Tags: | \$key \$ascii \$text \$numeric \$integer | Travel_301_1/1/2013 | 301 | Travel_301_1/1/2013 | Travel |
| | | Food_301_1/1/2013 | 301 | Food_301_1/1/2013 | Food |
| | | Travel_406_1/1/2013 | 406 | Travel_406_1/1/2013 | Travel |
| | | Travel_106_1/1/2013 | 106 | Travel_106_1/1/2013 | Travel |

LinkTable

Dimensions

Now, let's turn our attention to what has been exposed in Travel Expense Library by the developer to facilitate the creation and sharing of personal sheets. In the following screenshot, we can see the dimensions that were created. One particular dimension that is worth calling out is the **Expense** dimension, which provides a drill navigation from **ExpenseCategory** to **ExpenseType**. This capability usually requires extensive modeling or complex scripts in other BI software products, but with Qlik Sense, this is a simple selection process when creating the dimension. This is another example of the power of Qlik's associative indexing engine in action, but this time, easing the development of navigation within the application.



Dimensions

Measures

The next area to cover is **Measures**. These are calculated expressions that most often form the KPIs within an application. In the following screenshot, we can see a list of measures that are used and exposed to contributors to allow them to create private sheets. Note that hovering the pointer over any of these objects makes a preview popup appear to provide additional context. In this case, you can see how the measure **Actual - Food** is calculated.

The screenshot shows the Qlik Sense interface with the title bar "Travel Expense Management". On the left, there is a sidebar titled "Library" with a search bar. Below the search bar are categories: "Charts", "Dimensions", "Measures", "% of Budget", "% of Budget - Food", "% of Budget - Travel", "% of Budget 2", "Actual", "Actual - Food" (which is highlighted), "Actual - Travel", "Actual/Budget Difference", "Airfare Not Booked in Advance", "Average Employee Airfare", "Average Employee Daily Hotel...", "Average Employee Daily Meal...", and "Visualizations". The main area is titled "My new sheet" and contains a grid. A tooltip window is open over the "Actual - Food" measure, showing its details. The tooltip has a title "Actual - Food", a section "Expression:" with the formula "num(sum({<[Expense Category] = {Food}>} Amount), '\$#,##0')", and a "Tags" section with four icons: trash, edit, copy, and print.

Measures

Additionally, the following table contains the measure definitions that directly tie to the KPIs used in this application. Refer to the Qlik Sense online help for additional information on the Qlik Sense function, which is available at <https://help.qlik.com>.

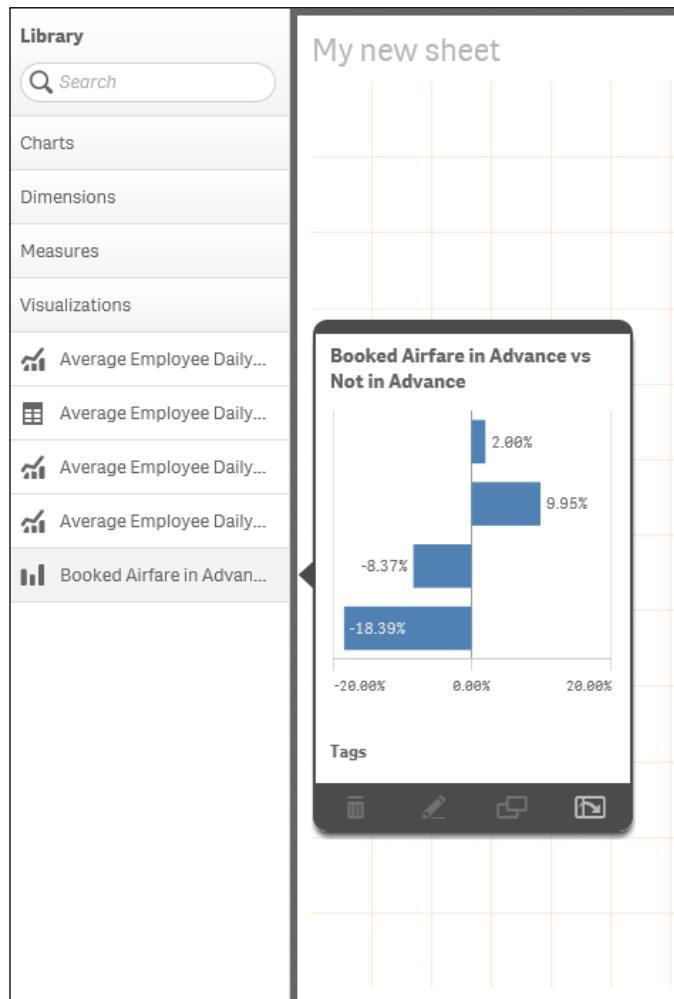
The measure expressions include:

| Measure | Calculation |
|--|--|
| % of Budget - Travel | num(sum({<[Expense Category] = {'Travel'}>} Amount) / sum({<[Expense Category] = {'Travel'}>} Budget), '#,##0%') |
| % of Budget 2 | num((Sum(Budget)/sum(Amount))-1, '#,##0%') Amount]) |
| Actual | Num(Sum(Amount), '\$#,##0') |
| Actual - Food | num(sum({<[Expense Category] = {'Food'}>} Amount), '\$#,##0') |
| Actual - Travel | num(sum({<[Expense Category] = {'Travel'}>} Amount), '\$#,##0') |
| Actual/Budget Difference | Num(Sum(Budget) - Sum(Amount), '\$#,##0') |
| Airfare Not Booked in Advance | Num(Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'No'}>} Amount), '\$#,##0.00') |
| Average Employee Airfare | Avg({<[Expense Type]={'Airfare'}>} Amount) |
| Average Employee Daily Hotel Cost | Sum({<[Expense Type]={'Hotel'}>} Amount) / Count(DISTINCT Employee) |
| Average Employee Daily Meal Cost | Sum({<[Expense Type]={'Breakfast', 'Lunch', 'Dinner'}>} Amount) / Count(DISTINCT Employee) |
| Average US Airfare (External Source) | Avg({<Trade={'Average Fare'}>} AirfareValue) |
| Average US Lodging Rates (External Source) | Avg({<Rate={'Lodging'}>} PerDiemValue) |
| Average US Meal Rates (External Source) | Avg({<Rate={'M&IE'}>} PerDiemValue) |
| Avg Airfare - Largest Carrier (External Source) | Avg({<Trade={'Average Fare - Largest Carrier'}>} AirfareValue) |
| Avg Airfare - Low Fare Carrier (External Source) | Avg({<Trade={'Average Fare - Low Fare Carrier'}>} AirfareValue) |

| Measure | Calculation |
|---------------------------|--|
| Booked Airfare in Advance | Num(Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'Yes'}>} Amount), '\$#,##0.00') |
| Booked Difference | Num(Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'Yes'}>} Amount) - Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'No'}>} Amount), '\$#,##0.00') |
| Booked Difference % | Num((Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'Yes'}>} Amount) / Avg({<[Expense Type]={'Airfare'}, Booked14DaysInAdvance={'No'}>} Amount)) - 1, '#,##0.00%') |
| Budget | Num(Sum(Budget), '\$#,##0') |
| Budget - Food | num(sum({<[Expense Category] = {'Food'}>} Budget), '\$#,##0') |
| Budget - Travel | num(sum({<[Expense Category] = {'Travel'}>} Budget), '\$#,##0') |
| Food Difference | num(sum({<[Expense Category] = {'Food'}>} Budget) - sum({<[Expense Category] = {'Food'}>} Amount), '\$#,##0') |
| Travel Difference | num(sum({<[Expense Category] = {'Travel'}>} Budget) - sum({<[Expense Category] = {'Travel'}>} Amount), '\$#,##0') |

Visualizations

The last category of objects in the **Library (Master items)** is **Visualizations**. These are preformed visualizations that are typically the most popular or requested. They are defined to help facilitate a user's analysis and can be easily dragged and dropped onto a private sheet. In the following screenshot, we see a horizontal bar chart that analyzes the variance in **Booked Airfare in Advance vs Not in Advance**. Each of these visualizations contains predefined dimensions, measures, and chart definitions.



The Travel Expense visualizations

Summary

In summary, Qlik Sense provides unique capabilities to meet the challenging task of analyzing and managing travel expenses. Without the capabilities offered by Qlik, this task can be difficult due to the size of the data and the many perspectives that can be taken in trying to understand airline purchasing, meal expense habits, and the impact on meeting corporate budget requirements. Qlik's associative indexing engine powers this exploration and means that meeting these requirements is no longer challenging at all.

In the next chapter, we will explore how Qlik Sense meets the needs of demographic data discovery.

13

Demographic Data Discovery

In this final chapter, we shall finish our exploration of real data with Qlik Sense by moving beyond the standard structures of the office and showing the full possibilities of the software for analysis of almost any kind of imaginable data. We'll therefore be looking at applying Qlik Sense to demographic data. As before, this example and many others are available for you to explore at <http://sense-demo.qlik.com>.

This chapter will cover the aspects necessary for demographic data discovery, including:

- General information about common KPIs
- Examples showing how to use the lasso selection in maps and scatter charts
- Examples of dimensions and measures

Problem analysis

With Qlik Sense, it is possible to analyze not only business data, but rather *any* data. One great example is demographic data – statistics of countries and regions on anything from age and gender to income and life expectancy.

Such data can be found on a number of Internet sites and downloaded for your convenience, for example, from the following websites:

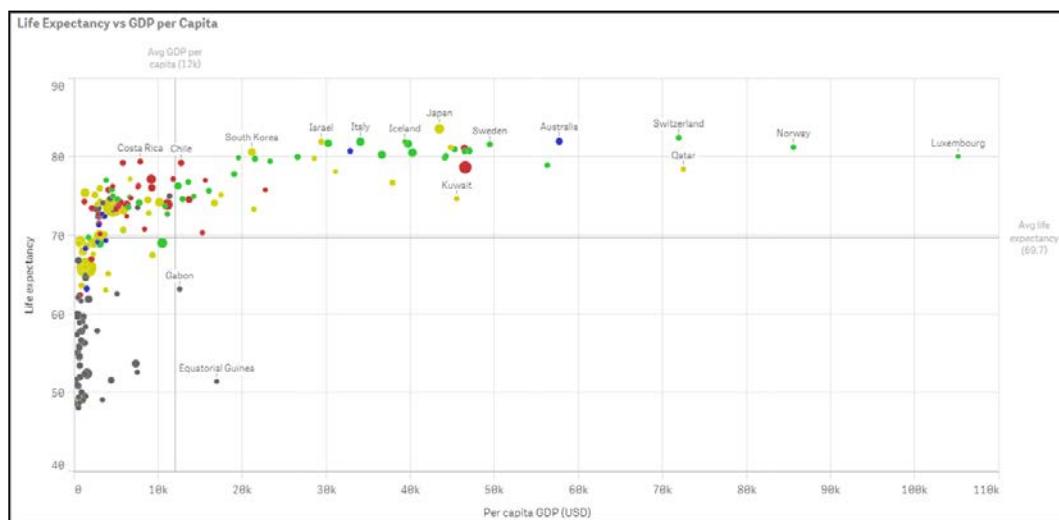
- United Nations (data.un.org)
- Federal government of the United States (www.data.gov)
- European Union (ec.europa.eu/eurostat)
- Qlik DataMarket (www.qlik.com/us/explore/products/qlik-datamarket)

Demographic Data Discovery

Demographic data is used and analyzed as-is by a number of nongovernmental organizations that need it for their activities. The common measures required are GDP per capita, population, unemployment rate, inflation, life expectancy, happiness, trade balance, labor cost, national debt, election results, and so on.

Often, interesting questions about correlations are asked; for example, how does happiness correlate with material standards and health? How are population growth and the number of children affected by factors such as life expectancy, poverty, and average salary? How has life expectancy improved over the years? If you haven't seen Hans Rosling's presentations on the Internet on this topic, we strongly recommend them. They show that data analysis is both important and fun.

Common dimensions in demographic data are country, region, gender, age group, ethnicity, and so on. An example can be seen in the following scatter chart, where you can see life expectancy and per capita GDP for different countries. Many developing countries are found in the lower-left quadrant, whereas the richer countries usually are found in the upper-right quadrant.



Life expectancy versus per capita GDP

You can clearly see that the two numbers are highly correlated – the higher the GDP, the higher the life expectancy.

These measures can often also be linked to your business data to enable a deeper understanding. For instance, you can divide your country sales by the population of the country, thereby getting a relative sales number, which tells you how well you sell in that country. With this number, you can make relevant comparisons of countries of different sizes.

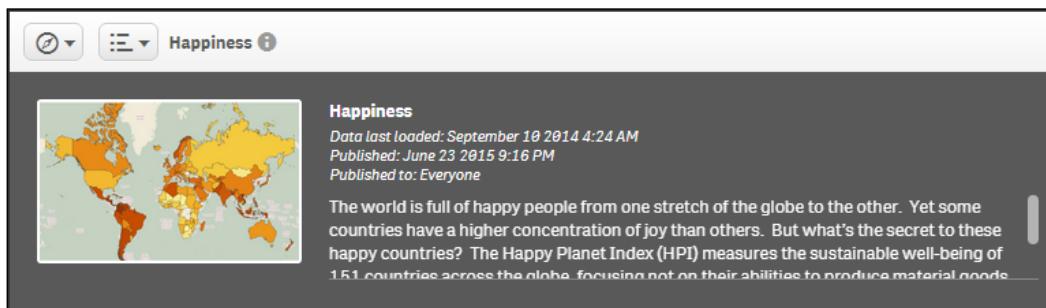
Alternatively, if you assume that the market space in the country is roughly proportional to the GDP, you can divide your sales by the GDP and use this number to compare market penetration between countries.

These numbers will answer questions such as, "How well are we selling in this country, given the potential?"

Application features

On our demo site, we have an app with a number of demographic measures per country. You can find it at <http://sense-demoqlik.com> under the name **Happiness**. It analyzes, among other demographic indexes, the **Happy Planet Index (HPI)** in a number of countries. You can learn more about this index at www.happyplanetindex.org.

This index measures the sustainable well-being of 151 countries across the globe, focusing not on their abilities to produce material goods and services, but rather on their abilities to produce long, happy, and sustainable lives for the people who live in them. A happy life doesn't have to come at the expense of our environment, and the HPI is used to promote a policy that puts the well-being of people and the planet first.

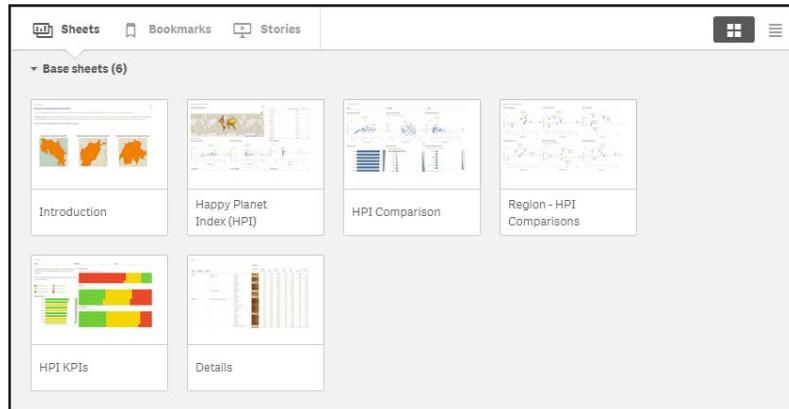


The app overview of the Happiness application

Below this overview, you will see a number of sheets. The leftmost sheet is an introduction, whereas the other sheets are prepared for analysis and detailed information.

Demographic Data Discovery

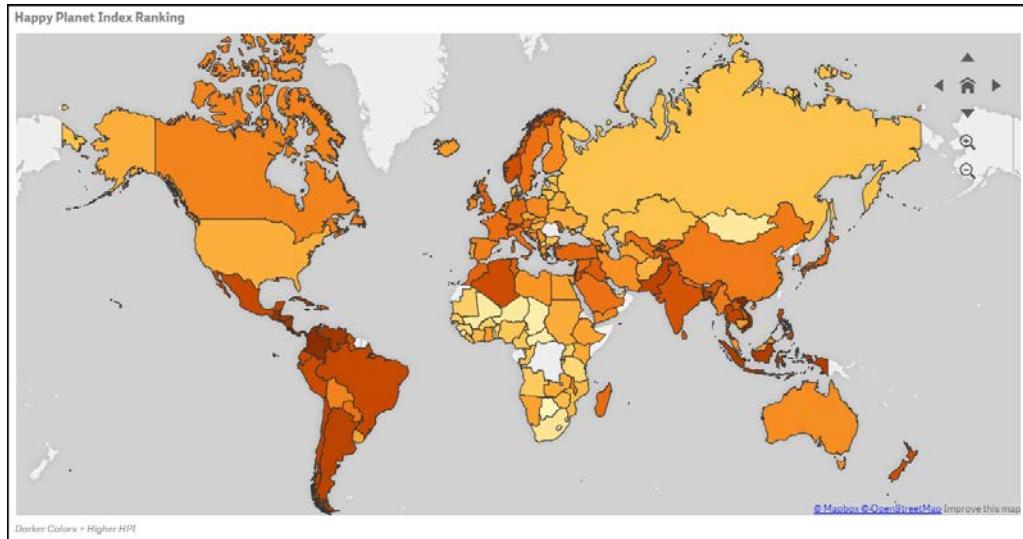
If you click on the **Stories** button to the left, you will see that the app also contains one story—a story that can be used to present data in the app. It can also be used as an introduction to the app the first time you open it.



The sheets on the app overview page

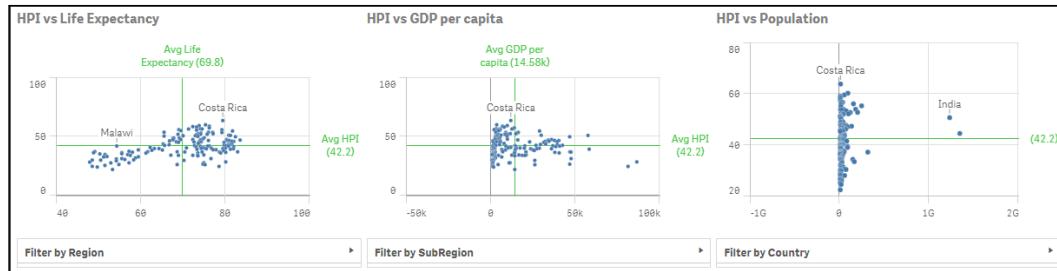
The first sheet is an introduction sheet that explains what the app is all about. The second sheet, which is the first one with traditional charts, is called **Happy Planet Index (HPI)**. On it, you will see the happiness index for all countries, first on a map, and then in a table.

The countries in the map are colored according to the happiness index—the darker the color, the higher the happiness index.



A map showing the happiness index per country

Below the map, there are three scatter charts showing the happiness index per country, plotted against the life expectancy, GDP per capita, and total population. These three charts are excellent tools to analyze any correlation between happiness and the mentioned demographic measures.



Scatter charts that show the correlation (or lack of correlation) between happiness and other demographic measures

Finally, at the bottom, you have three filter panes, allowing the user to choose only a region, subregion, or country to zoom in the numbers for a specific area.

The other sheets contain additional and more detailed information, ordered by topics. The final sheet contains a table showing the details, should the user be interested in drilling down to the lowest level.

Analysis

When looking at data in this app, the first question that pops up in the user's mind is usually, "Is there any correlation between happiness and x ?" To get a qualitative answer to this, you only need to browse through the scatter charts.

On the **Happy Planet Index (HPI)** sheet, you have three scatter charts. In the leftmost chart, **HPI vs Life Expectancy**, you can see a correlation between the two measures, at least for lower life expectancies. In the other two charts, however, there is no clear correlation.

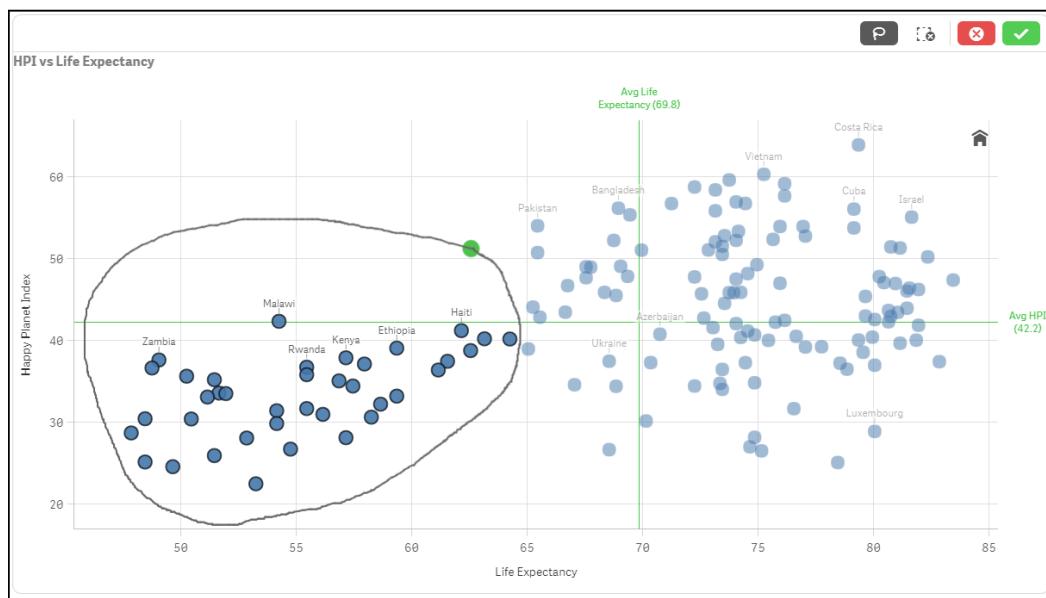
On the **HPI Comparison** sheet, you have three additional scatter charts. In the leftmost chart, **HPI vs Happy Life Years**, you can see a weak correlation between the two measures. The same is true for the rightmost chart, **HPI vs Global Footprint**, but in the chart in the middle (**HPI vs Governance**), there is no clear correlation.

However, as in all statistics, you have to be careful with your conclusions. Firstly, correlation does not imply causation. You have to look at many factors and use common sense to find the true cause and effect. In this case, it is just that the happiness index is an artificial index calculated from the life expectancy and ecological footprint among others, hence the correlation with happy life years and global footprint.

Using the lasso selector to make selections

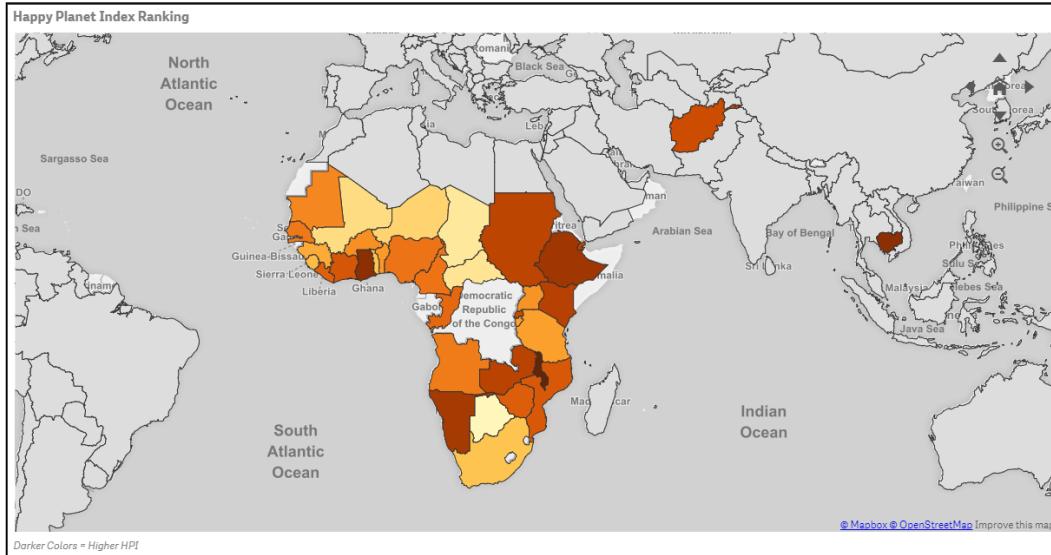
Now, let's explore the data. One question could be, "Where in the world do we find the countries with a low average life expectancy?" To answer this, you need to make a selection in the scatter chart showing life expectancy:

1. First, navigate to the **Happy Planet Index** sheet. Maximize the scatter chart that shows **HPI vs Life Expectancy** by clicking on the fullscreen arrow in the upper-right corner of the object.
2. Then, click on the chart so that the chart controls, including the lasso symbol, appear in the upper-right corner.
3. Next, click on the **Turn on lasso selection** option. Now you can draw a line around the points you want to select.
4. Finally, confirm your selection by clicking on the green tick mark in the upper-right corner.



Lasso selection in the scatter chart

If you now look at the map, you will see where these countries appear in the world. It's predominantly Africa and South Asia. If you click on the map, you can zoom in using the scroll wheel of the mouse. You can also pan the map.

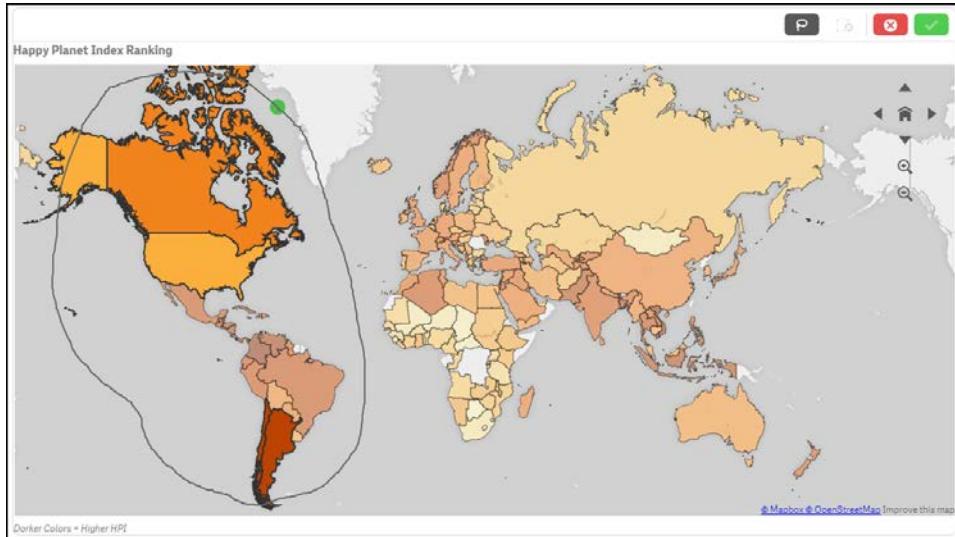


Countries with low life expectancy

Of course, you can also make a selection the other way round. Use the lasso selector in the map and see how the selected countries are distributed in the scatter chart. The way to do this is as follows:

1. Maximize the map.
2. Click somewhere in the map.
3. Click on the **Turn on lasso selection** option and encircle the part of the world you want to explore.

4. Finally, confirm your selection.



Making a lasso selection of America on the map

Using the global selector to make selections

You can also use the global selector to make selections. Just click on the global selector and make selections directly in the fields.

For instance, you may have a question like this, "Where in the world do I find the richest countries?" In such a case, perform the following steps:

1. Open the global selector. (This is found to the right in the toolbar with **Selections tool** as a popup.)
2. Find a field called **GDP/capita (\$PPP)**. To do this, you might first need to check **Show fields** in the global selector.
3. Once you have found this field, you can investigate it just by scrolling. You will then see that there are some countries with less than \$400 in GDP per capita, while the richest countries have more than \$80,000 in GDP per capita.

If you want to find the countries where the GDP is greater than \$10,000, perform the following steps:

1. Click on the search icon and type >10000.
2. Confirm the search by pressing *Enter*, and then confirm the selection by clicking on the green tick mark.

The screenshot shows a data visualization interface with four main panels:

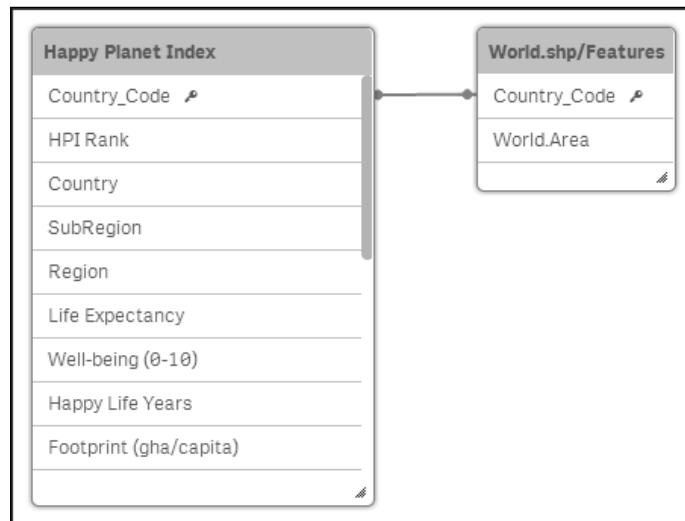
- APP DIMENSIONS**: A list of countries including Afghanistan, Albania, Algeria, Angola, Argentina, Armenia, Australia, and Austria.
- Country**: A list of country codes (Country_Code) and their corresponding Footprint (gha/capita) values.
- GDP/capita (\$PPP)**: A search results panel showing GDP values for various countries. A search query >10000 has been entered, and the results show values starting from 10565.184056308 up to 12169.055806245.
- Governance Rank**: A list of governance ranks from 1 to 7, with most entries marked as n/a.

Selecting the countries in the world with the highest GDP

If you now close the global selector and go back to the map and the scatter charts, you will be able to see where you find the richest countries, both on the map and in the scatter charts.

How the application was developed

The data model of the **Happiness** application is not very complicated:

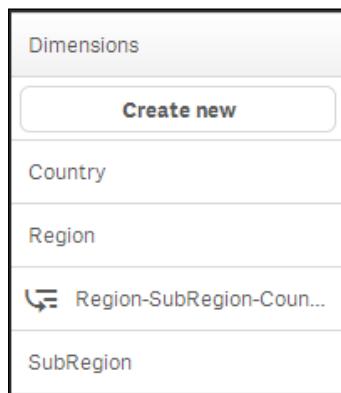


This is an extremely simple data model that only contains one table of real data, Happy Planet Index, and an additional table listing all countries, World.shp/Features. The second table has one record per country and holds the map information – the shapes of the country – used in the map object in the user interface.

In this app, the data table has exactly one record per country – a record that contains the relevant information for a given country at a given moment. However, this is not always the situation. More often, the data table contains data for countries over many points in time, for example, one record per combination of a country and a year. This will result in several lines per country.

Dimensions

There are not many fields that can be used as dimensions. The three available fields are region, subregion, and country. The world is split into 7 regions and 19 subregions. A country can only belong to one subregion and one region. These fields have been added to **Library**. In addition, a drill-down dimension has been created from the three fields.



The dimensions in Library

One way of adding dimensions could be by creating buckets based on one of the measures, for example, population. Countries could then be grouped under **Large**, **Medium**, and **Small** classes, which will be stored in a new field, **Population Class**.

Measures

A number of measures have also been defined, for example, GDP, happiness index, global footprint, life expectancy, and so on.

It is important that the app developer formulates the formulas correctly, since this is something that could be difficult for the business user. The business user doesn't always have knowledge about the data model, which is something you need in order to get all the expressions right.

In the following table, you can find some of the measures defined in this app:

| Measure | Definition |
|--------------------|------------------------------|
| GDP per Capita | Avg([GDP/capita]) |
| Global Footprint | Avg([Footprint]) |
| Governance Rank | Only [Governance Rank]) |
| Happy Life Years | Only([Happy Life Years]) |
| Happy Planet Index | Only([Happy Planet Index]) |
| HPI Rank | Only([HPI Rank]) |
| Population | Only(Population) |

Several of these measures can be defined differently. How you do this is very much a matter of taste. For instance, the measures where the `Only()` function is used can also be defined using `Sum()` or `Avg()`. As long as you only have a single number, all three functions will return the same answer.

But how do you want Qlik Sense to behave when there are several countries, for example, a region that should be represented by one value? For the **Population** measure, the obvious function to use should be `Sum()`. Then the total population of the region will be shown.

But if the source data contains several years, so that a single country has several records, you don't just want to sum the population. Then you would get numbers that are much larger than they should be. Instead, you might want to use `Sum(Population) / Count(distinct Year)` to create an average over all possible years.

In addition, for a rank, you wouldn't want to use `Sum()` because it would show an incorrect number. You could use `Avg()`, which will give the average rank between the countries. An average is clearly better, but it is still not mathematically correct. Then it might be better to use `Only()`, which doesn't return an answer at all when more than one country is involved.

Summary

The analysis of demographic data is easy when you use Qlik Sense. Obviously, this analysis can also be made with a number of other tools, since the data model is very simple. However, with Qlik Sense, it is easy to build further. Qlik's associative indexing engine powers the analysis and ensures that you can develop or change your apps quickly and easily. With Qlik Sense, data discovery and analysis is made easy.

With the end of this chapter, we have also reached the end of the book. We took you from the history of Qlik to how to develop applications, and finally gave you some examples of how applications might look.

We hope that after reading this book, you have acquired some skills that will be useful when you develop your own Qlik Sense applications. We also think you now have a better understanding of the thoughts behind Qlik Sense, and wish you good luck in your endeavors.

Welcome to the community of Qlik users!

Module 2

Qlik Sense Cookbook

Over 80 step-by-step recipes to tackle the everyday challenges faced by Qlik Sense developers

1

Getting Started with the Data

In this chapter, we will cover the basic tasks related with extracting data into a Qlik Sense application:

- ▶ Extracting data from databases and data files
- ▶ Extracting data from Web Files
- ▶ Activating the Legacy Mode in Qlik Sense® desktop
- ▶ Extracting data from custom databases
- ▶ Invoking help while in the data load editor or the expression editor
- ▶ Previewing data in the Data model viewer
- ▶ Creating a Master Library from the Data model viewer
- ▶ Using a Master Library in the Edit mode

Introduction

Data is the core aspect of any Business Intelligence application. It provides information that helps organizations to make decisions.

A Qlik Sense application is based on the data extracted from various sources, such as relational databases, CRM systems, ERP systems, and data files.

This chapter introduces the user to various methods of extracting data into a Qlik Sense application effectively. It is assumed that the reader is already acquainted with the concepts of ODBC, OLEDB, and relational databases. The chapter also provides an essential recipe for fetching the data into Qlik Sense from a SAP system. The SAP connector can be downloaded from the Qlik website and installed before working on the recipe. You need to acquire a valid license enabler file beforehand, in order to download the SAP connector.

The later part of the chapter focuses on a few recipes regarding the creation of a library and content.

Extracting data from databases and data files

The data within an organization is usually stored in relational databases and data files. Extracting data is the first step towards creating a data model. The following section demonstrates the steps to extract data from an MS Access database and a delimited (.csv) file. The procedure to extract data from other relational databases is the same as the process for extracting data from MS Access.

The dataset that we will use is available publicly and covers information about routes and fares of various transport systems in Hong Kong. The original data files have been downloaded from (<https://data.gov.hk/>) website. This dataset can also be obtained from the Packt Publishing website.

The data connections in the Qlik Sense data load editor save shortcuts leading to commonly used data sources, such as databases and data files. The following types of connections exist in Qlik Sense:

- ▶ ODBC database connection
- ▶ OLEDB database connection
- ▶ Folder connection
- ▶ Web file connection

This recipe deals with the ODBC, OLEDB, and Folder connections. The web file connection will be dealt with in a separate recipe.

Getting ready...

The dataset required for this recipe that is downloaded from the Packt Publishing website comes in a zipped folder called as *QlikSenseData*. Extract all the files from this zipped folder and save them on the hard drive at a desired location.

If you are connecting to the database using **Open Database Connectivity (ODBC)** then:

1. Install the relevant ODBC drivers on your system.

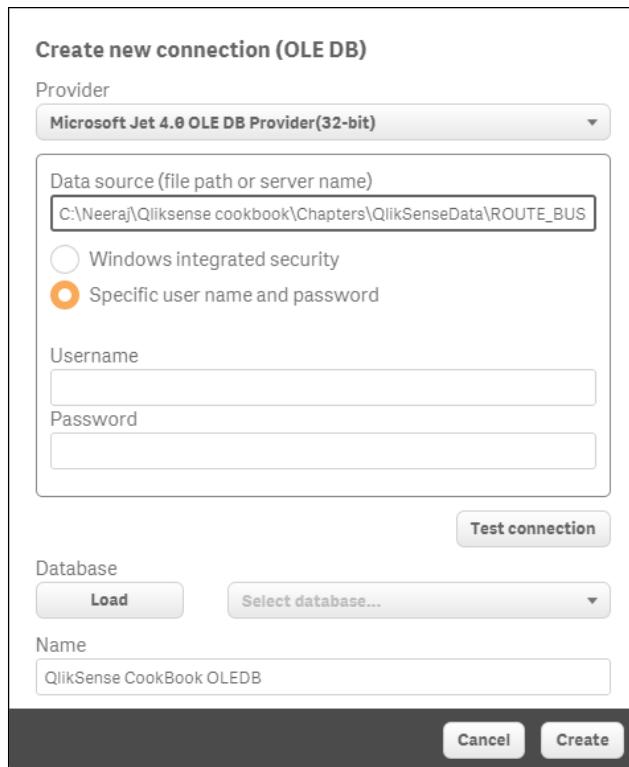
[ For the sake of our exercise, we need the MS Access drivers. The system DSN connection can be set up through the ODBC administrator under the **Administrative Tools** in **Control Panel**.]

2. While setting up the ODBC connection, select the ROUTE_BUS.mdb file as the Data Source from the QlikSenseData folder.
3. Name the ODBC DSN connection as HongKong_Buses.
4. Create a new Qlik Sense application and open the data load editor.
5. Click on the **Create New Connection** and select **ODBC**.
6. Select **HongKong_Buses** under **System DSN**.
7. Name the data connection as **Qlik Sense CookBook ODBC**.
8. The following image shows the details we enter in the **Create new connection (ODBC)** window:



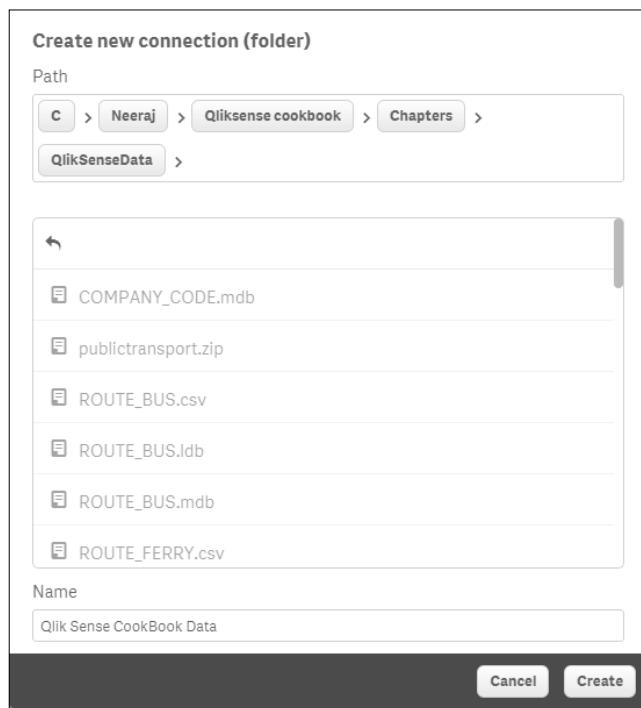
If you are connecting to the database using OLE DB connectivity, we can directly set this up through the editor:

1. Open the data load editor in Qlik Sense.
2. Click on the **Create New Connection** and select **OLE DB**.
3. Select the **Microsoft Jet 4.0 OLE DB Provider (32 Bit)** driver from the provider drop-down list.
4. Insert the **Data Source** file path, which in our case will be the path for the ROUTE_BUS .mdb file in the QlikSenseData folder.
5. Name the data connection as **QlikSense Cookbook OLE DB**.
6. The following image shows the details we enter in the **Create new connection (OLE DB)** window:

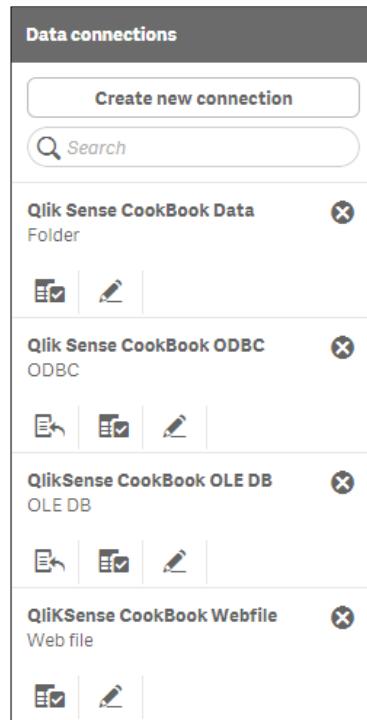


If you are extracting the data from a data file, such as .csv, perform the following steps:

1. Open the data load editor in Qlik Sense.
2. Click on **Create New Connection** and select **Folder**.
3. Select the location of the QlikSenseData folder which contains our data files. Alternatively, one can directly enter the path of the source folder under **Path**.
4. Name the data connection as Qlik Sense CookBook Data.
5. The following image shows the details we enter in the **Create new connection (folder)** window:



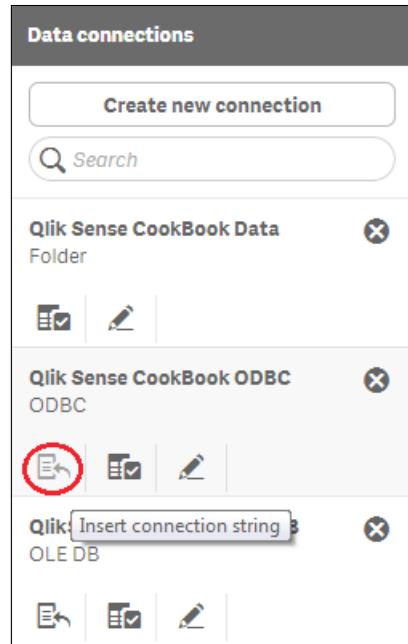
6. Once the connections are created in the Qlik Sense library, they will be seen as a list under **Data connections** in the data load editor, as shown in the following screenshot:



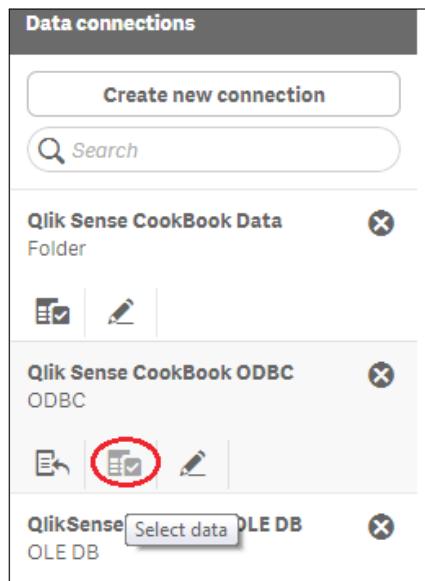
How to do it...

If you are working with an ODBC or an OLEDB data connection, follow the steps:

1. Insert the relevant data connection string to the script by clicking on **Insert connection string**, as shown in the following screenshot:



2. Next, click on **Select data** under **Data connections** to view and extract data from the ROUTE table in the MS Access database, as shown:



Getting Started with the Data

3. The preview of the ROUTE_BUS .mdb table will look like the following. The fields in the table can be excluded or renamed while working in the **Preview** window, as shown in the following screenshot:

The screenshot shows the 'Select data to load' interface in Qlik Sense. On the left, there are dropdown menus for 'Database' and 'Owner'. Below them is a 'Tables' section with a 'Filter tables' input and a list of tables: ROUTE (selected), MSysACEs, MSysObjects, MSysQueries, and MSysRelationships. The 'ROUTE' table has 17 columns. In the center, the 'Selections summary' shows '1 table' and '17 columns'. Below it, the 'Fields' section has tabs for 'Data preview' (selected) and 'Metadata'. The 'Data preview' tab displays a grid of data with columns: COMPANY_CODE, DISTRICT, FULL_FARE, LAST_UPDATE_DATE, LOC_END_NAMEC, and LOC_END_NAMEE. The data rows are: 1001 (KMB, 1, 1, 1), 1002 (KMB, 10, 10, 10), 1006 (KMB+CTB, 102P, 102P, 102P), 1008 (KMB+CTB, 103, 103, 103), and 1009 (KMB+CTB, 103P, 103P, 103P). At the bottom of the preview area, there is a script pane with the following code:

```
LOAD 'COMPANY_CODE',
      'DISTRICT',
      'FULL_FARE',
      'LAST_UPDATE_DATE',
      'LOC_END_NAMEC',
      'LOC_END_NAMEE',
      'LOC_END_NAMEP';

      INTO TABLE ROUTE_BUS;
```

At the bottom right of the preview window, there are 'Cancel' and 'Insert script' buttons.

4. Click on **Insert Script** in the **Preview** window. This will insert the connection string as well as load the statement to the script. Make sure that you delete the duplicate LIB CONNECT TO 'Qlik Sense CookBook ODBC'; statement from your script.
5. Load the data in your application by clicking on the **Load data** button.

Keep the **Close when successfully finished** option checked in the data load progress window. If the data is loaded successfully, then the window automatically closes or else the error encountered is highlighted.

1. On a similar note, in order to test the Qlik Sense data files, Click on the **Select data** option under the **Qlik Sense CookBook Data** connection.
2. Next, select the ROUTE_GMB .csv file from the QlikSenseData folder and load it in the application.
3. The preview of the ROUTE_GMB .csv table will look like the following screenshot. Make sure that you select **Embedded field names** under **Field names**. Note that the **Delimiter** in this case is automatically set to **Comma**.



4. Insert the script and then save and load it.

How it works...

The `LIB CONNECT TO` statement connects to a database using a stored data connection from the Qlik Sense library; thus, acting as a bridge between our application and the data source.

There's more...

This recipe aimed at extracting data from common data sources, such as RDBMSs and data files. Qlik Sense can also extract data from web files and custom data sources such as SAP. We will see this in the forthcoming section.

See also...

- ▶ [Creating a Master Library from the Data model viewer](#)

Extracting data from Web Files

Often, the data required for the purpose of reporting is not stored in a database, but instead needs to be fetched from a website. For example, customer location information specifically the geographic co-ordinates used in mapping analysis is not available internally within an organization. This information may be available on the web and can be extracted from there.

Getting ready...

When extracting the data from a web file:

1. Open an existing Qlik Sense application or create a new one.
2. Open the data load editor.
3. Click on **Create New Connection** and select **Web** file.
4. The **Select web file** window will open.
5. Insert the following URL from which you can fetch the data:
`http://www.csgnetwork.com/l1infotable.html`
6. Name the connection as **QlikSense Cookbook Webfile**, as shown:



How to do it...

1. In the list under **Data Connections**, select **QlikSense Cookbook Webfile** and click on **Select Data**. This will open up a preview window listing out all the tables from the web page. When you carefully examine the table contents, you realize that it is the second table **@2** that contains the location information.
2. Check the box next to **@2** and ensure that it is selected, so the correct table is shown in the preview. The user will need to change the value under **Field names** to **embedded field names**.
3. The preview of the table will look like the following screenshot:

The screenshot shows the Qlik Sense Data Modeler interface. In the top left, it says "Select data from http://www.csgnetwork.com/lainfotable.html". On the left, there's a "Tables" list with checkboxes for @1 through @10, and table @2 is checked and highlighted with a red circle. The "File format" is set to "HTML" and "Field names" to "No field names". The "Character set" is "Western European". On the right, there's a "Preview script" button. The main area shows a "Fields" grid with four columns: @1, @2, @3, and @4. Below the grid is a table with data:

| Country | Capital | Latitude | Longitude |
|----------------|-----------|----------|-----------|
| Afghanistan | Kabul | 34°0'N | 69°11'E |
| Albania | Tirane | 41°18'N | 19°49'E |
| Algeria | Algiers | 36°42'N | 03°38'E |
| American Samoa | Pago Pago | 14°16'S | 170°43'W |

Below the table is a "LOAD" script:

```

LOAD
@1,
@2,
@3,
@4
FROM [lib://QlikSense Cookbook Webfile]
(html, codepage is 1252, embedded labels, table is @2);

```

At the bottom right are "Cancel" and "Insert script" buttons.

4. Select all the fields from the table in the preview window. Click on **Insert script** to load the web data in the application.

5. Name the table as `Country_Location` and the script will read as follows:

`Country_Location:`

```

LOAD
Country,
Capital,
Latitude,
Longitude
FROM [lib://QlikSense Cookbook Webfile]
(html, codepage is 1252, embedded labels, table is @2);

```

6. Save and load the data. Once the script is successfully loaded, the data model viewer will show the loaded table.

How it works...

Qlik Sense connects to the web file using the stored data connection. Once connected it identifies the tables in the HTML source and lists them in the preview window.

Certain external websites require authentication in order to be accessed and Qlik Sense is unable to cope with websites that are secured in this manner. In order to get over this issue, we can use a third party data extraction tool. The extracted data can be stored in a data file, such as a **qvd**. The qvd file can then be used as a data source in the Qlik Sense application.

There's more...

Qlik Sense can also extract data from other data formats, such as XML. The underlying principles remain the same as explained in the preceding recipes.

See also...

- ▶ *Creating a Master Library from the Data model viewer*
- ▶ *Activating the Legacy Mode in Qlik Sense® desktop*

Activating the Legacy Mode in Qlik Sense® desktop

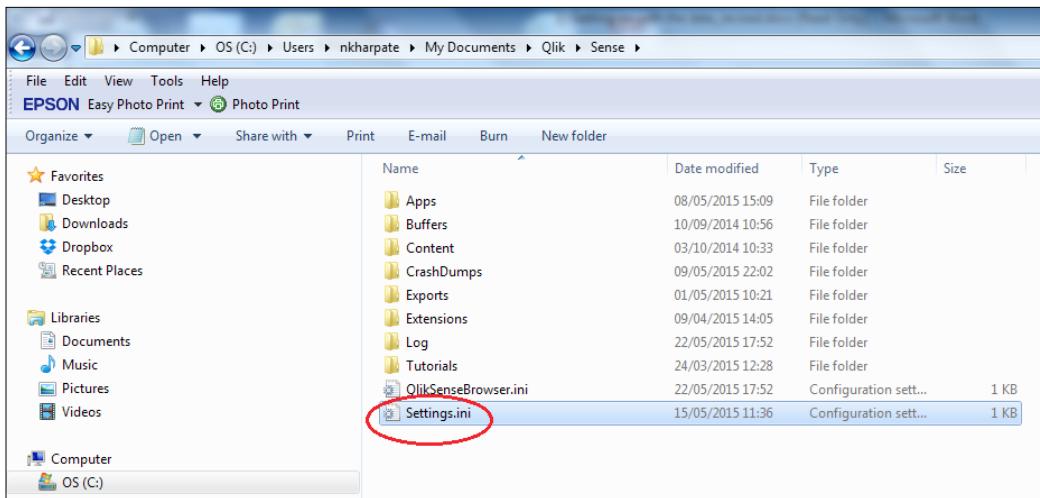
Qlik Sense is a developing product; hence, certain features are not active when running the Desktop version in its standard mode. A prime example of this is using the `Custom Connect` to statement to create the ODBC/OLEDB connection strings or attempting to connect to a custom database as SAP. Both these activities are not possible if Qlik Sense runs in its standard mode. In order to get these functionalities to run, we need to activate the legacy mode. However, one must note that enabling the legacy mode has security implications, if the application is deployed on the Sense server then one does not have control over the data connections in QMC (if the legacy mode is activated). The library security features may also be lost; moreover, the legacy mode does not work with Qlik Cloud either.

Getting ready...

Activating the Legacy Mode requires changing a parameter value in the `settings.ini` file for Qlik Sense.

How to do it...

1. Make sure that Qlik Sense Desktop is closed before opening the `settings.ini` file.
2. Open the `settings.ini` file that is by default stored under `C:\Users\{user}\Documents\Qlik\Sense\Settings.ini`.
3. Change `StandardReload=1` to `StandardReload=0`.
4. Save the file and start Qlik Sense Desktop in order to run it in a legacy mode, as shown:



How it works...

Changing the value for the `StandardReload` parameter in the `settings.ini` file enables the Legacy Mode in Qlik Sense. When running in the Legacy mode, any of the scripts in Qlik View can be directly used in Qlik Sense. This will also allow us to use the library connections.

There's more...

The Qlik Sense has the capability to use the same script that is found in any Qlikview file. One can also use a binary load statement in Qlik Sense in order to load the entire data model from an existing Qlikview file. All the `Custom Connect To` statements can only be used after we activate the legacy mode.

See also...

- ▶ [Extracting data from custom databases](#)

Extracting data from custom databases

The current version of Qlik Sense does not support the loading of data from custom databases, such as SAP or Salesforce. Nevertheless, it can still be achieved in a few simple steps. The following recipe explains the steps to load data from a SAP database.

Getting ready...

The **Custom connector** option under **Create new connection** is not available in the Qlik Sense data load editor. This feature is going to be introduced soon in a forthcoming release of the product.

The following recipe requires you to use another Qlik product named **Qlikview** in order to generate the extract script that is to be copied and used in the Qlik Sense application. Qlikview is free software that can be downloaded from the Qlik website. The recipe also requires the **SAP connector** for QlikView to be installed.

How to do it...

Once we install the SAP connector, the `RELOADSAPDD.qvw` and `ScriptBuilder.qvw` files are saved on the hard drive.

We will work along with the `RELOADSAPDD.qvw` file, which is stored at the `C:\ProgramData\QlikTech\CustomData\QvSAPConnector\ScriptBuilder` location.

In order to extract data from a custom database, such as SAP:

1. Activate the legacy mode as described in the recipe just prior to this.
2. Open the Qlikview file and input the SAP credentials to generate the connection string similar to the following:

```
CUSTOM CONNECT TO ""Provider=QvSAPConnector.dll;
ASHOST=192.168.210.166;SYSNR=00;CLIENT=100;KeepCasing=1;
NullDate=1;XUserId=UPJDRIRJJaSMVEVIXSFA;XPassword=
IQWOQIRNJbaMXUVMXLMGSEA;"";
```
3. Open Qlik Sense. Copy and paste the SAP Connection string from the script editor of the QlikView file to Qlik Sense.
4. Similarly, one can copy and paste the load script generated for any SAP table in a QlikView file to a Qlik Sense file.
5. Save and load data.
6. The data load editor with all the connection strings will appear, as shown in the following:

```

1 //Binary [C:\Neeraj\Projects\Chain Reaction\Document\Sales-Link.qvw];
2
3 SET ThousandSep ",";
4 SET DecimalSep ".";
5 SET MoneyThousandSep ",";
6 SET MoneyDecimalSep ".";
7 SET MoneyFormat="#,##0.00;-#,##0.00";
8 SET TimeFormat="hh:mm:ss";
9 SET DateFormat="DD/MM/YYYY";
10 SET TimestampFormat="DD/MM/YYYY hh:mm:ss(.fff)";
11 SET MonthNames="Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec";
12 SET DayNames="Mon;Tue;Wed;Thu;Fri;Sat;Sun";
13 SET LongMonthNames="January;February;March;April;May;June;July;August;September;October;November;December";
14 SET FirstWeekDay=1; //Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday;
15 SET FirstWeekDay=0;
16 SET BrokenWeeks=1;
17 SET ReferenceDay=0;
18 SET FirstMonthOfYear=1;
19 SET CollationLocale="en-IE";
20
21
22 CUSTOM CONNECT TO #Provider=QrSAPConnector.dll;AHOST=192.168.210.16;
23 SYNCH=0;CLIENT=100;FeeCasing=1;NullDate=1;XUserIId=UFDRIRJJSBMEVIXSF;
24 XPassword=fQWQzRNBaxKXU7XKLW3SER;"; 
25
26
27 DD02L: // tables
28 SQL SELECT TABNAME TABCLASS FROM DD02L
29 where ( TABCLASS = 'TRANS' or TABCLASS = 'POOL'
30 or TABCLASS = 'CUSTTER' or ( TABCLASS = 'VIEW' and ( VIEWCLASS = 'D' or VIEWCLASS = 'P' ) ) );
31 Store * from DD02L into qvd/DD02L.qvd;
32 drop table DD02L;
33

```

How it works...

The essence of the recipe is that the custom connections don't work in Qlik Sense, unless it is running in a Legacy mode. The user can copy the script generated in the QlikView file to the Qlik Sense Load script while running the application in the legacy mode, as this script cannot be generated directly in Qlik Sense.

There's more...

Qlik Sense can extract data from any data source that can be loaded by QlikView (such as Salesforce) in practically the same way as it is described in this recipe.

See also...

- ▶ Activating the Legacy Mode in Qlik Sense® desktop

Invoking help while in the data load editor or the expression editor

As a Qlik Sense developer, one often needs access to the help module in order to search for certain functions or simply understand their usage and syntax in detail. Help is available in the dropdown menu on the toolbar. However, when we use this option, it takes us to www.help.qlik.com/sense and then we again need to search for the keyword. It's not a huge effort but it would be more beneficial if we were taken directly to the information regarding the keyword or function we are looking for.

Getting ready...

For this recipe, we will use the `Automotive.qvf` file, which comes as a built in example when we install the Qlik Sense Desktop.

How to do it...

1. Open the `Automotive.qvf` file from the Qlik Sense desktop hub.
2. Open the data load editor and go to the **Territory data** tab.
3. Click the **Help** (?) button inside the data load editor. This will highlight the script so that all the keywords are then clickable links.
4. Click on the keyword `pick` in the script. This will take us to the correct place in the help file, as shown:

```
// Load the territory data
LOAD
    "English short name" as Territory,
    "OICA region",
    pick match("OICA region", 'Africa', 'Americas', 'AOME', 'Europe'), 'springgreen', 'lightskyblue'
    "Trading bloc",
    "ISO 3166-1 alpha-2",
    "ISO 3166-1 alpha-3" as [Territory code],
    "Calling code",
    "Area (Km2)",
    "Population (2012 or latest)" as Population,
    "GDP (US$ current)",
    "Currency name",
    "Currency code",
    "Capital city",
    "BBC country profile",
    "CIA World Factbook article",
    "Wikipedia country article"
FROM [lib://Data/Data sources/Territories.xlsx]
(xml, embedded labels, header is 7 lines, table is Territories);
```

There's more...

An alternative approach that can be used in Qlik Sense versions prior to 2.0.1 is as follows:

1. Highlight the key word `pick` in the script.
2. Press `ctrl + h`. This will take you directly to the content explaining **Pick** on the help page.

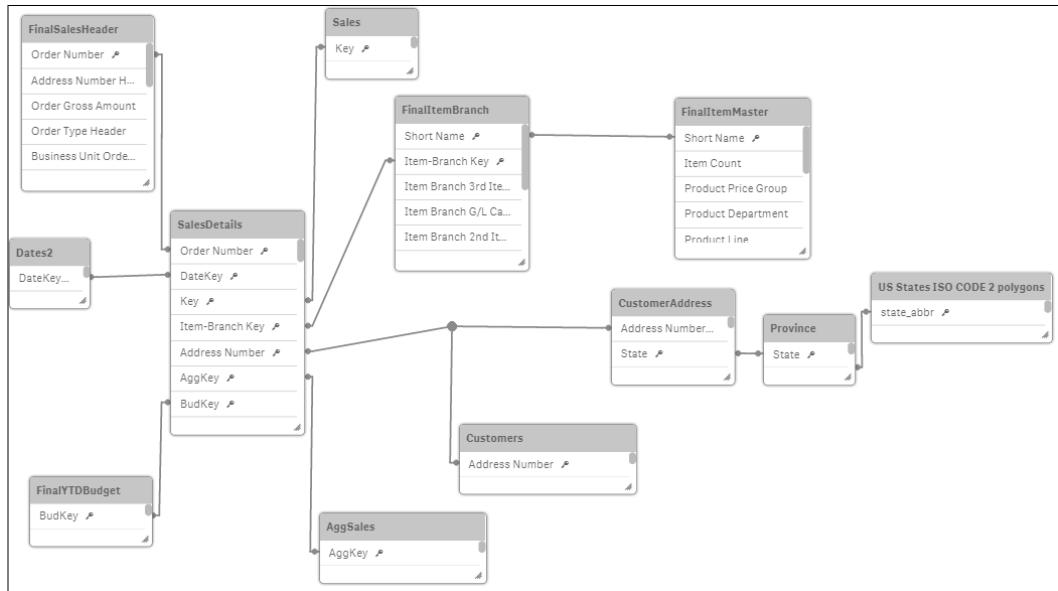
A list of useful shortcuts for Qlik Sense is given at the end of this book.

See also...

- ▶ Keyboard shortcuts in Qlik Sense® desktop

Previewing data in the Data model viewer

As any experienced Qlik developer will tell you, the data model viewer is a key component you will undoubtedly spend time using on your Qlik journey. Qlik Sense has brought with it some nice new features. We will also delve into the different insights that can be gleaned from the data model viewer:



Getting ready

For this recipe, we will make use of the `Data model viewer.qvf` application. This file is available for download on the Packt Publishing website.

How to do it...

1. Open the `Data model viewer.qvf` application that has been downloaded from the resource library.
2. Click on data model viewer in the Navigation dropdown on the toolbar.

How it works...

In this section we will see how the different types of data are viewed.

Viewing the data model

The data model consists of a number of tables joined by the key fields. The following screenshot contains functions that can be used to manipulate the layout of the data model:

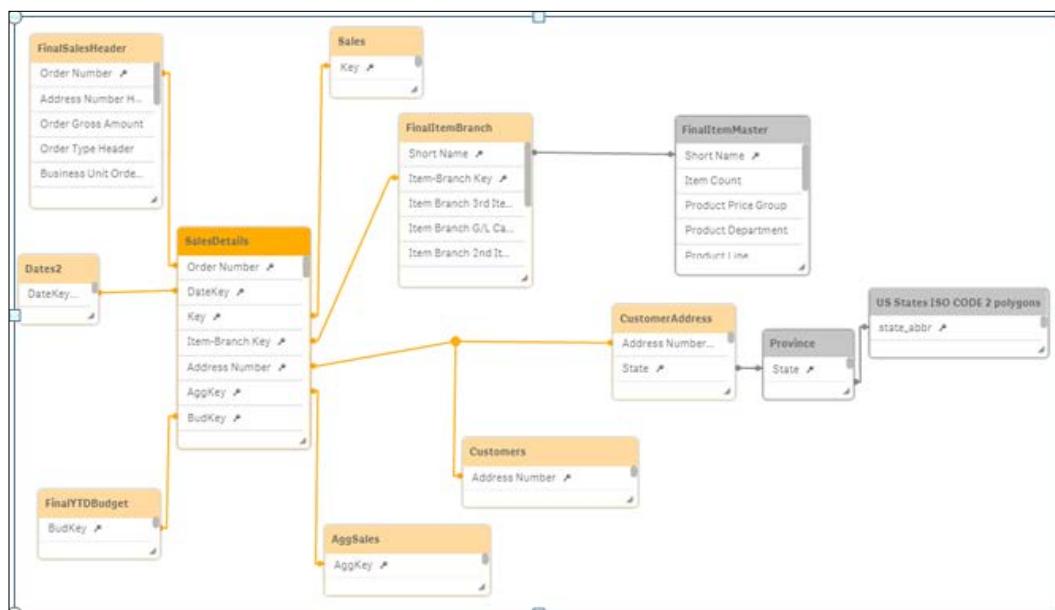


The detail of the available keys (from right to left) is given as follows:

- ▶ **Collapse all:** This reduces down the tables to just their headers; thus, hiding all the fields
- ▶ **Show linked fields:** Expands the tables enough to only display the key fields in each
- ▶ **Expand all:** Displays all the fields for each table
- ▶ **Internal Table viewer:** Shows the internal representation of the data model
- ▶ **Layout:** Provides options to auto align the table grid or space out across the screen
- ▶ **Preview:** Toggles the data preview screen to either on or off

Viewing the associations

Clicking on a table will highlight its associated tables in orange. The customer's table is selected in the following screenshot and the shared key here is **Address Number**:



Click on the **CustomerAddress** table to see a highlighted expansion, via the state key, as shown:

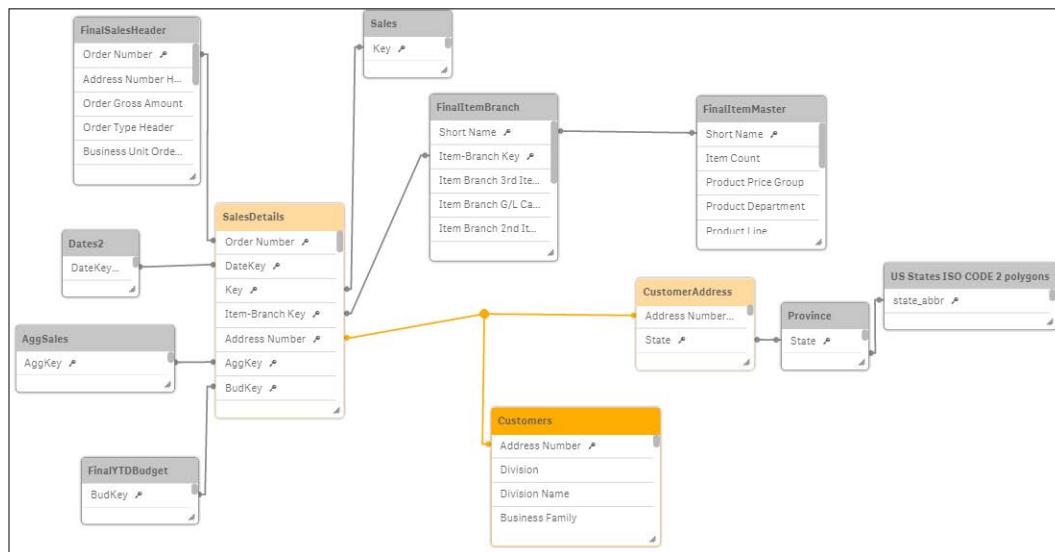


Table Meta Data

The data model viewer also provides information on the contents of each table.

Click the header of the customer address table then open the **Preview** pane by clicking the **Preview** button in the bottom left hand corner.

The following preview will be displayed at the bottom of the screen:

| Preview | | | | | | | | | | | | |
|-----------------|-----|------------------|-------|---------------------------------------|--------------------|--------------------|--------------------|------------------|---------------|--------|---------|--|
| CustomerAddress | | Overview of data | | | | | | | | | | |
| Rows | 688 | Address Number | State | Customer Address 1 | Customer Address 2 | Customer Address 3 | Customer Address 4 | Zip Code | City | County | Country | |
| Fields | | 10023778 | OR | PO Box 70 | | | | 97123 | Hillsboro | | US | |
| Keys | | 10025285 | TN | 2401 East 31st Street | PO Box 72538 | | | 37407 | Chattanooga | | US | |
| Tags | | 10025285 | TN | 2401 East 31st Street | PO Box 72538 | | | 37407 | Chattanooga | | US | |
| | | 10016807 | KS | Attention Accounts Payable Department | PO Box 12990 | | | 66282-2960 | Overland Park | | US | |
| | | 10010914 | KS | Attention Accounts Payable Department | PO Box 12990 | | | 66282-2960 | Overland Park | | US | |
| | | 10021911 | MS | Cnr Avalon Drv & Tasman Road | PO Box 20977 | | | Te Rapa Hamilton | | | US | |
| | | 10021911 | MS | Cnr Avalon Drv & Tasman Road | PO Box 20977 | | | Te Rapa Hamilton | | | US | |

Along with a small snippet of the table contents, the far left table also provides some high level table information about the number of rows, fields, keys as well as any tags.

Next, click the **Address Number** field from the **Customers** table in the data model viewer.

You can now see more detailed information about the individual field.

These are:

- ▶ Density
- ▶ Subset ratio
- ▶ Has duplicates
- ▶ Total distinct values
- ▶ Present distinct values non-null values
- ▶ Tags

This information is very helpful when we are debugging issues. If a count does not return the expected result, you may want to ensure that there are no duplicates.

If a selection is not filtered correctly you may want to check the sub-set ratio of the key and so on.

There's more...

Double clicking a table header in the data model viewer will either collapse or expand the table fully.

Creating a Master Library from the Data model viewer

To help reduce the repetition and developer error, Qlik has introduced a master library where we can store reusable items, such as dimensions, measures and even whole visualizations. For people experienced in Qlik's other products such as QlikView, just think; "no more linked objects and storing expressions in variables!"

It is easy to think of library items in a self-service context. Don't get me wrong; ultimately you will have to decide what will be published; from your data model to the world for their own analysis purposes. Having said that, the secret sauce of this recipe is in saving your own time.

It is a productivity hack that implies; "automation is to your time what compound interest is to money". While it is not an exact parallel, this is a nice concept to frame the usefulness of timesaving functions in Qlik Sense. The effective use of the library saves time spent on scrolling down field lists, rewriting expressions over and over, applying a single change in multiple places, and so on.

Once you have saved enough time to eclipse the setup investment, the value of taking this approach can only compound with continuous development.

Getting ready

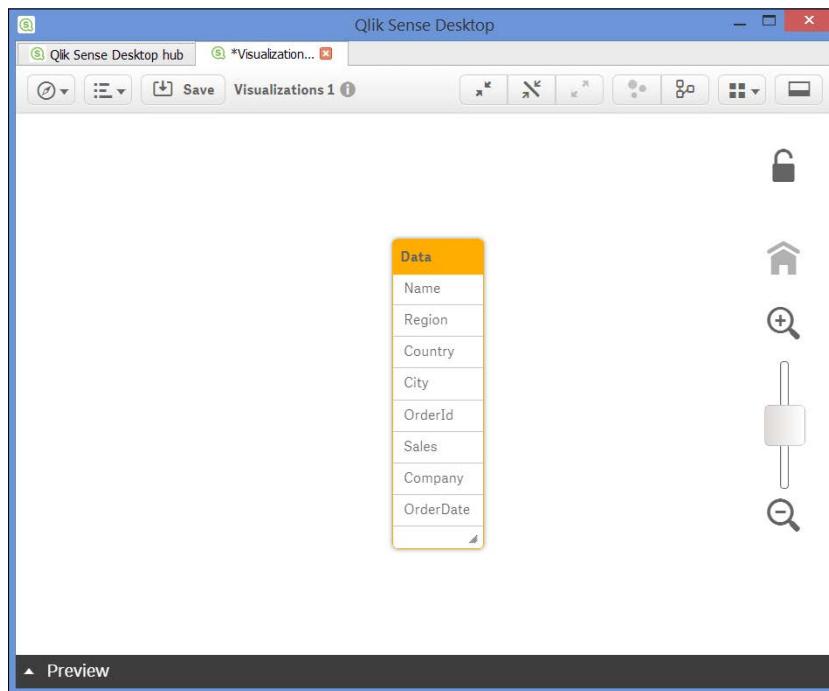
1. Create a new Qlik Sense application and name it Master Library.
2. Open the data load editor.
3. Enter the following script and load the data by clicking on the  button.
(The script is available in a separate text file that can be downloaded from the Packt Publishing website):

Data:

```
LOAD * INLINE [  
    Name, Region, Country, City, OrderId, Sales, Company,  
    OrderDate  
    Wooten, C, Mozambique, Carmen, 1, 45.55, Est Nunc  
    Laoreet LLC, 22/12/14  
    Blankenship, Delta, Cayman Islands, Sapele, 2, 95.76,  
    Lorem Donec Inc., 17/01/15  
    Sheppard, Wyoming, Vatican City State, Cheyenne, 3,  
    38.31, Lobortis, 07/08/14  
    Goddard, H, Curaçao, San Francisco, 4, 86.33, Non Inc.,  
    07/09/14  
    Galloway, Aragón, Trinidad & Tobago, Zaragoza, 5,  
    85.80, Diam Proin., 21/01/15  
    Kirsten, Tamil Nadu, Wallis & Futuna, Neyveli, 6,  
    28.47, Mollis Non Limited, 03/05/14  
    Holland, Cartago, Falkland Islands, San Diego, 7, 1.34,  
    Ullamcorper Inc., 17/07/14  
    Thaddeus, BC, Canada, Oliver, 8, 59.04, Ante Nunc  
    Mauris Ltd, 17/02/15  
    Lareina, CA, Spain, San Diego, 9, 4.55, Pellentesque  
    Tincidunt Limited, 29/07/14  
    Jescie, Vienna, Monaco, Vienna, 10, 54.20, Ultricies  
    Ligula Consulting, 16/06/14  
    Logan, IL, Saint Barthélemy, Paris, 11, 91.31, Mi  
    Foundation, 13/12/14  
    Shannon, CG, Nepal, Aberystwyth, 12, 80.86, Auctor Non  
    LLC, 03/05/14  
    Andrew, SO, Argentina, Sokoto, 13, 88.78, Scelerisque  
    Mollis Associates, 12/12/14  
    Jocelyn, WP, Tanzania, Konin, 14, 15.91, Ligula Tortor  
    Dictum Ltd, 22/08/14  
    Gordon, FL, Hong Kong, Miami, 15, 93.97, Suscipit Inc.,  
    12/05/14  
];
```

How to do it...

Once the data has been loaded, you can check the results by opening the data model viewer through the navigation dropdown () in the top corner on the left hand side of the toolbar, as shown in the following screenshot:



You can find the **Preview** button to the bottom left of the screen. There are several other places in Qlik Sense where you can create master library items but the data model preview screen is the best, as it also lets you see the data first. Take a minute to browse the data you have loaded in data model viewer.

1. In the data model viewer, select the Data table by clicking on its header and then click the **Preview** button to view the fields and the field values loaded from the Data table.

2. The **Preview** window will appear as shown in the following:

The screenshot shows the Qlik Sense Desktop interface with a preview of data. The left sidebar lists fields: Name, Region, Country, City, OrderId, Sales, Company, and OrderDate. The preview table shows 15 rows of data with columns: Name, Region, Country, City, OrderId, Sales, Company, and OrderDate. The data includes entries like Wooten (Region C), Mozambique (Country), Carmen (City), etc.

| Preview of data | | | | | | | |
|---|-------------|------------|--------------------|---------------|---------|---------|----------------------|
| Data | Name | Region | Country | City | OrderId | Sales | Company |
| Rows 15 | Wooten | C | Mozambique | Carmen | 1 | \$45.55 | Est Nunc Laoreet LLC |
| Fields 8 | Blankenship | Delta | Cayman Islands | Seapele | 2 | \$95.76 | Lorem Donec Inc. |
| Tags: \$ascii \$text \$numeric \$integer \$timestamp \$date | Sheppard | Wyoming | Vatican City State | Cheyenne | 3 | \$38.31 | Lobortis |
| | Goddard | H | Curacao | San Francisco | 4 | \$86.33 | Non Inc. |
| | Galloway | Aragón | Trinidad & Tobago | Zaragoza | 5 | \$85.80 | Diam Proin. |
| | Kiraten | Tamil Nadu | Wallis & Futuna | Neyveli | 6 | \$28.47 | Mollis Non Limited |
| | Holland | Cartago | Falkland Islands | San Diego | 7 | \$1.34 | Ullamcorper Inc. |

3. Select the **Region** field from the table to get the preview as shown in the following:

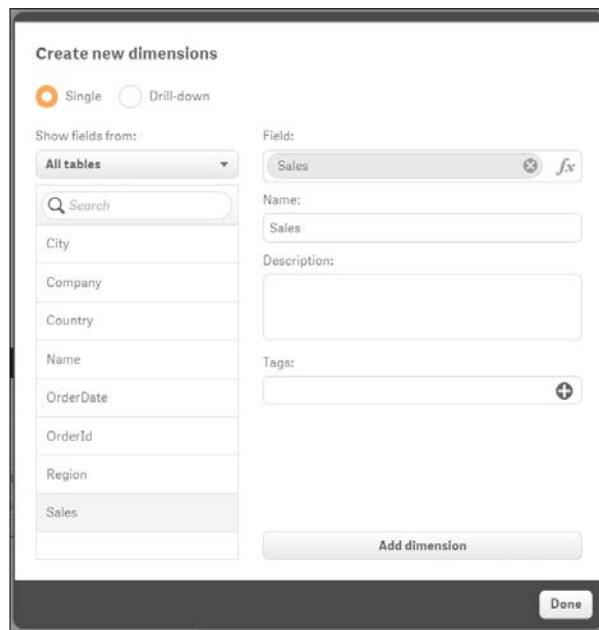
The screenshot shows the preview for the Region field. It includes buttons for "Add as dimension" and "Add as measure". The preview table shows metrics for the Region field: Density (100%), Subset ratio (100%), Total distinct values (6), Present distinct values (6), Non-null values (6), and Tags (\$text).

| Region | |
|-------------------------|--------|
| Density | 100% |
| Subset ratio | 100% |
| Total distinct values | 6 |
| Present distinct values | 6 |
| Non-null values | 6 |
| Tags | \$text |

4. Next, click the **Add as dimension** button.

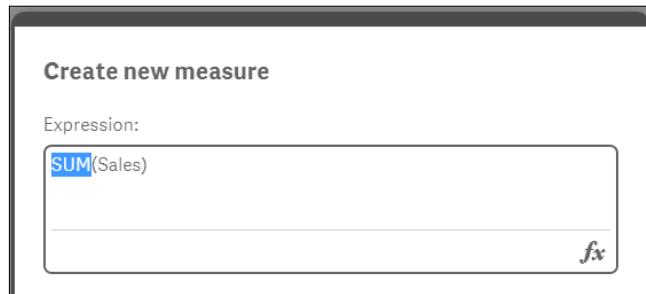
Getting Started with the Data

5. The following window appears. If you are likely to publish this dimension for consumption by users, you can enter a description here:



6. It is advised to use tags to make our life easier. Add the tag **Geo** and click on **+**.
7. Now click on **Add dimension** to create a Master dimension in the library.
8. Repeat this process for the **Country** and **City** fields.
9. Click on **Done** to go back to the data model viewer.

10. Finally, it's time to create a measure. Select the **Sales** field from the Data table in the data model viewer.
11. Click the **Add as measure** button. When we create a Master measure we need to make sure we use an aggregation function such as Sum, Avg, and so on, along with the selected field.
12. In the **Create new measure** window, type SUM in front of (**Sales**), as shown in the following image:

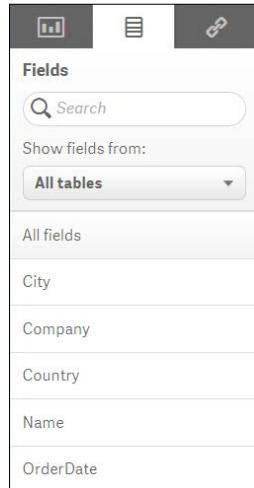


13. Click on **Create**.
14. Save the changes made in the master library by clicking on the **Save** button on the toolbar in the table preview. Exit the table preview by going to the App Overview.
15. Open (or create) a sheet and enter the edit mode by clicking on the **Edit** button.
16. Once you are in the edit mode, click the chain () icon on the left hand side of the asset panel to open the Master items menu.
17. To add visualizations, first create them in the user interface then drag them into the library.

While the Master item menu panel is very useful to speed up the development when defining the contents, it is easier to do it from the filters pane. In short, you can browse the entire contents of your data model and right-click on the most important fields to add the ones that will be frequently used.

How it works...

1. Right click on a field from the field's pane that you want to add to the master library.



2. Click on **Create Dimension**, enter a **Description** and any relevant **Tags**, click **Done** once finished:

A screenshot of a modal dialog box titled "Create new dimensions".

The dialog has two radio button options: "Single" (selected) and "Drill-down".

"Show fields from:" dropdown is set to "All tables".

"Field:" input field contains "Country".

"Name:" input field contains "Country".

"Description:" input field is empty.

"Tags:" input field is empty with a plus sign (+) icon.

At the bottom are two buttons: "Add dimension" and "Done".

There's more...

We can also create Master dimensions and measures through the GUI. In order to do this:

1. Open an existing sheet or create a new one.
2. Click on the Master items  icon.
3. Click on either Dimensions or Measures. This will enable an option to create new library items.

Using a Master Library in the Edit mode

As mentioned in the previous recipe, a great benefit of creating a master library is to save you time and reduce the complexity by applying global changes to your visualizations.

There are three main areas in the asset panel when editing a Qlik Sense sheet (Objects, Fields, and Master items). Clicking the  chain button opens the Master items pane.

From here, you can manage every aspect of the Master items, such as renaming, replacing, deleting, and editing.

Getting ready

You can continue to use the application from the previous recipe.

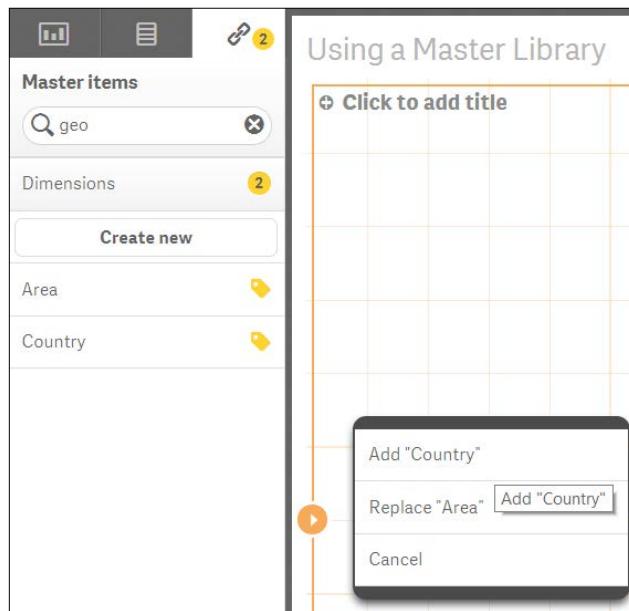
1. If you have not completed the previous recipe. Load the following in your data load editor:

```
LOAD * INLINE [  
Country, Area, Sales  
USA, North, 1000  
USA, North, 1200  
USA, South, 2500  
USA, South, 2500  
UK, North, 1000  
UK, North, 2500  
UK, South, 2000  
UK, South, 1900  
];
```

2. Add **Country** and **Area** as Master dimensions both with the tag **Geo**.
3. Add **Sales** as a Master measure.

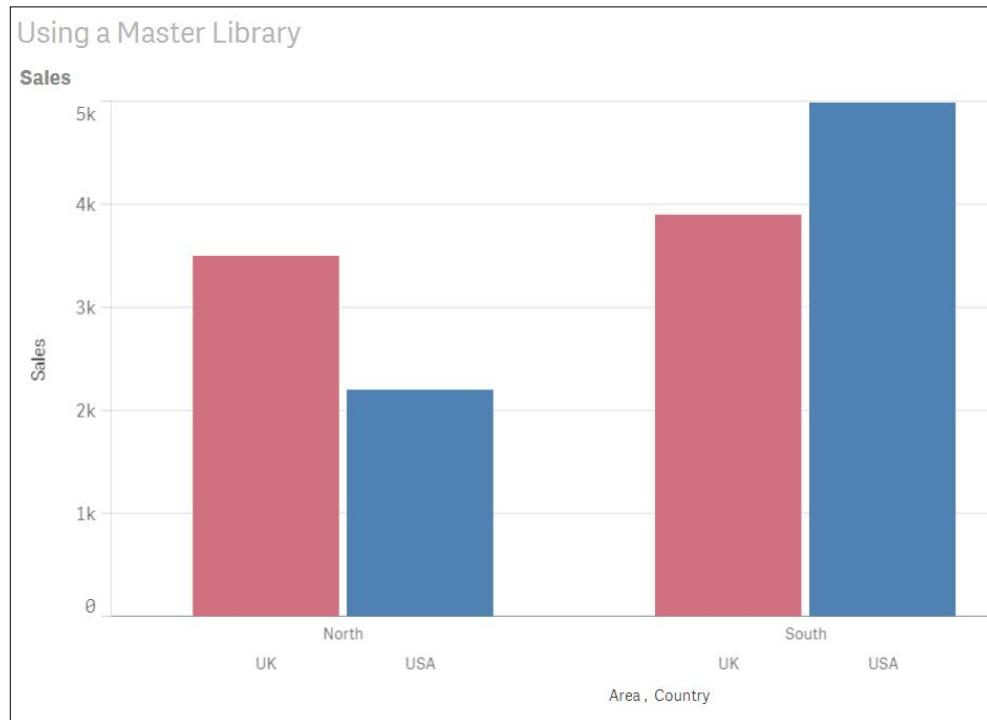
How to do it...

1. Open the **App overview** screen by clicking on the navigation dropdown on the toolbar at the top.
2. Create a new sheet or open an existing one.
3. Enter the edit mode by clicking on the  Edit button.
4. Click on the object pane button  and double click on the bar chart button. . The chart will be added to the main content area automatically.
5. Type Geo in the search box of the asset panel on the left of your screen. While there are no charts called **Geo**, the search has flagged up our two tagged dimensions in the master library pane with a yellow circle like this .
6. Next, drag the **Area** field to where it says **Add dimension**. Repeat the steps where the **Country** field selects **Add "Country"** when prompted, as shown:



7. Clear your search on Geo by pressing the  button.
8. Click on **Measures**.

9. Drag the **Sales** measure from the asset panel over to the add measure area of the chart. Voila! You have created your first visualization using Master dimensions and measures:



10. You can now drag this chart into the asset panel and it will become a master visualization.

There's more...

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. The same applies to deleting a field from the data model; the reference will remain a part of the Master item pane until it's updated from the edit screen.

Echoing a comment in the previous chapter regarding time saving and creating Master measures, replaces the need to write expressions as variables for reuse. Another piece of QlikView functionality that has been replicated and expanded upon is the concept of linked objects. Any updates you make in the Master visualization area will be applied globally.

If you rename a field in your script without moving the position it will be applied automatically to all the objects.

2

Visualizations

In this chapter, we will cover some visualization tips and tricks to create a compelling dashboard in Qlik Sense:

- ▶ Creating Snapshots
- ▶ Creating and adding content to a story
- ▶ Adding embedded sheets to the story
- ▶ Highlighting the performance measure in a bar chart
- ▶ Associating persistent colors to field values
- ▶ Using the `colormix1` function
- ▶ Composition
- ▶ Relationships
- ▶ Comparison
- ▶ Distribution
- ▶ Structuring visualizations

Introduction

A typical Qlik Sense application should always follow the **Dashboard Analysis Reporting (DAR)** methodology. This methodology focuses on developing a Dashboard sheet followed by an analysis sheet and then a reports sheet. The Dashboard projects the high-level figures of the business; the analysis sheet gives more control to the end user to filter the data, while the Reports sheet has the detailed information at a granular level. For more information on the DAR concept, visit:

- ▶ <https://community.qlik.com/blogs/qlikviewdesignblog/2013/11/08/dar-methodology>

While this concept can be easily implemented within the application, one often tends to forget the best design practices that help in making the applications more engaging for the users. An optimal design will convey the right information to the right people at the right time. This will elevate the decision making process within the organization.

The following chapter focuses on some of the key concepts in data visualization that will help the users take their Qlik Sense design capabilities to the next level.

It also discusses the importance of choosing the right visualization for the right purpose. Some useful blogs written by the experts in data visualization can be accessed by the users to enhance their knowledge:

- ▶ <http://www.perceptualedge.com/blog/>
- ▶ <http://global.qlik.com/uk/blog/authors/patrik-lundblad>

Creating Snapshots

Snapshots are an exciting feature in Qlik Sense that enables the users to capture the point in time state of the data object. Snapshots work as insights for a story which will be discussed in later recipes.

Getting ready

For the sake of this exercise, we will make use of the `Automotive.qvf` Qlik Sense application. This application is downloaded as a sample file with the default Qlik Sense desktop installation and can be accessed through the Qlik Sense hub.

The sample files may differ by region. If the `Automotive.qvf` application is not available in the Qlik Sense hub, it can be downloaded from the Packt Publishing website.

Perform the following steps once you download the application from the Packt Publishing website:

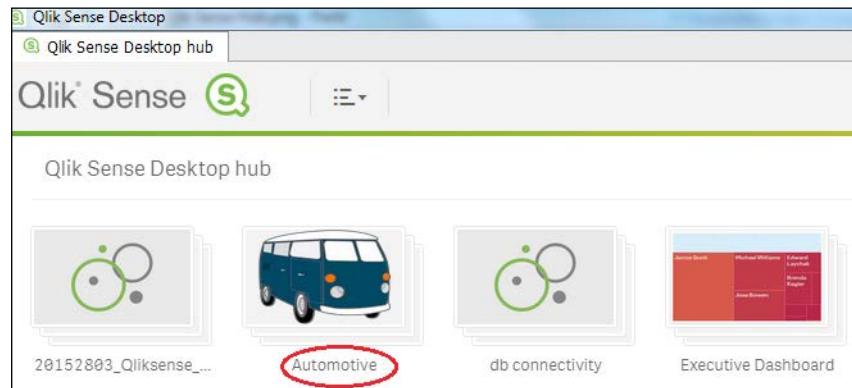
1. Copy the `.qvf` file to `C:\Users\<user>\Documents\Qlik\Sense\Apps` folder.
2. Open Qlik Sense desktop and the app will appear in the hub.

How to do it...

Qlik Sense provides you the opportunity to take a single snapshot of a selected object or take several snapshots of multiple objects at the same time.

In order to take Snapshots,

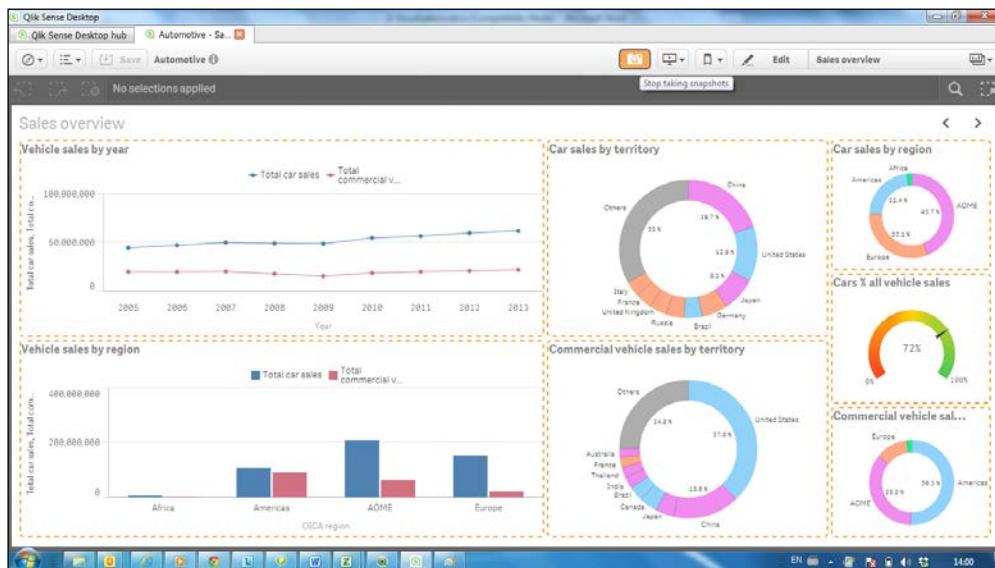
1. Open the **Automotive.qvf** application from the Qlik Sense hub:



2. Open the **Sales Overview** sheet and select the trendline chart **Vehicle Sales by year**. Right-click on the object to display the options and select to take a snapshot.
3. The snapshot is saved within the snapshot library and is titled with the same name as the object.

In order to take multiple snapshots at the same time, use the following steps:

1. While you are still in the sheet view, click on the toolbar. The objects for which we can take snapshots are marked in broken orange lines, as shown:



2. Next, click on any object whose snapshot you want to take. As we click around on the sheet objects, a snapshot gets saved in the library automatically and a snapshot image  is displayed at the top right corner for every individual object along with the count of snapshots attached to the object.
3. For our exercise, we will save snapshots for **Car Sales by Industry** and **Vehicle sales by Region**.
4. As mentioned, the snapshots are again saved within the snapshot library and are titled with the same name as the objects.

How it works...

Snapshots are usually taken by the users when they want to store the point-in-time picture of an object corresponding to any selections. Snapshots are synonymous with taking a static picture of the object on the screen. As this is a static picture, it does not get updated with the change in data or with the change in state of the individual Qlik Sense object. The state and selections within a snapshot will not be updated after a data reload.

A Snapshot can be called a sibling of another Qlik Sense feature called **Bookmarks**. The difference being, Bookmarks capture the state of selections within an application, while Snapshots store the state of objects as it was at a particular point of time. The data projected by a bookmark gets updated on data reload.

There's more...

Snapshots form the basis of creating stories in Qlik Sense. We deal with stories in the following sections.

See also

- ▶ *Creating and adding content to a story*

Creating and adding content to a story

Qlik Sense introduces the concept of storytelling within the application. The data story interface helps the user to collate all the important observations and insights from the application to create a convincing narrative and present it to the intended audience in the form of a slideshow.

Getting ready

As in the previous recipe, we will again make use of the `Automotive.qvf` Qlik Sense application.

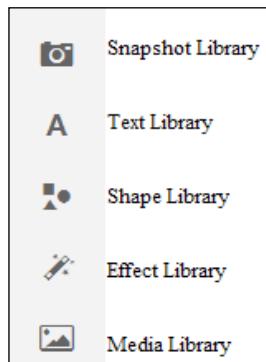
How to do it...

To create a story, perform the following steps:

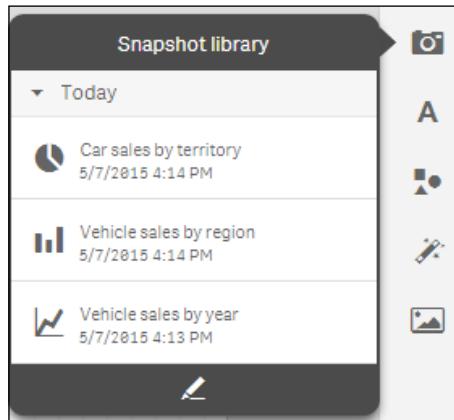
1. Open the `Automotive.qvf` application.
2. While you are still on the **App overview** page, click  **Stories** on the toolbar.
3. Click on the  sign to create new story.
4. Add the story name as `Sales Overview` and description as `A narrative of the overall sales for the company`.
5. Click outside the description window to save the story.

Adding contents to a storyboard:

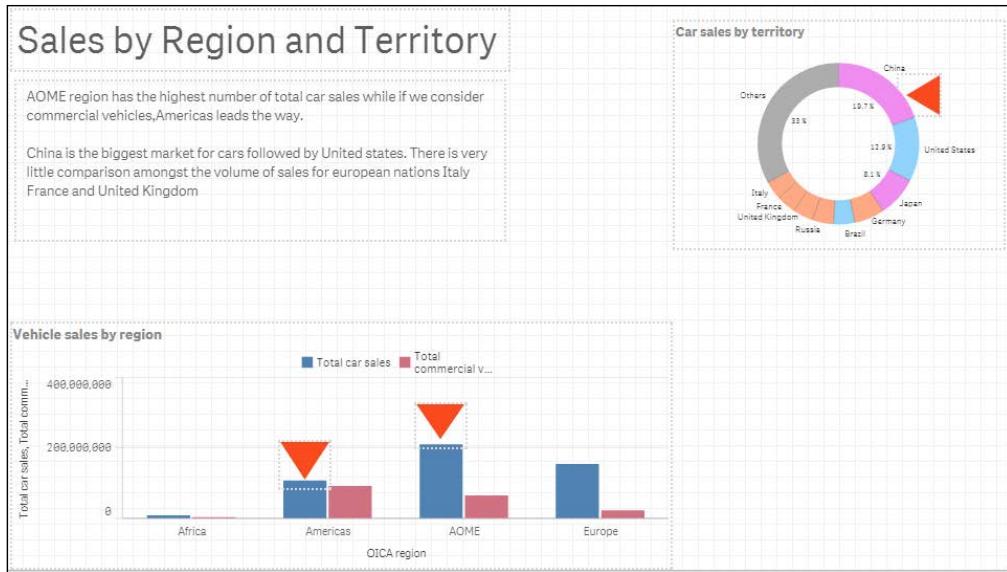
1. Open the storyboard for "Sales Overview" by clicking on the thumbnail.
2. The right side pane of the storyboard represents five libraries which serve as the source for the contents which we can use for our storyline:



3. Click on the **Snapshot library** icon. This will display the list of snapshots we took in the earlier recipe:



4. Drag and drop the **Vehicle sales by region** and **Car sales by territory** snapshot on the sheet.
5. Click on the **Text Library**. Drag and drop the **Title** box on the sheet.
6. Double click the **Title** box and add the title Sales by Region and Territory.
7. Click on the **Text Library**. Drag and drop the **Paragraph** box on the sheet.
8. Double click on the Paragraph box and add the following text:
 - ❑ AOME region has the highest number of total car sales while if we consider commercial vehicles, Americas leads the way.
 - ❑ China is the biggest market for cars followed by United States. There is very little comparison amongst the volume of sales for european nations Italy France and United Kingdom.
9. Click on the **Shapes** library. Drag and drop the and shapes on the sheet.
10. Save the story by clicking on **Save** on the toolbar.
11. The effective story interface should look like this:



12. The story can be played as a presentation by clicking on the button on the left-hand side vertical pane.

How it works...

The story with all the essential elements, namely the snapshots, commentary and the highlighters, conveys the essence of the data to the audience. The data added through the snapshots contains static point in time information. Since stories are native to Qlik Sense, there is no need to create separate PowerPoint files for presentations. Although with Qlik Sense version 2.1.1, one has the option to export the stories to PowerPoint so that they can be shared offline.

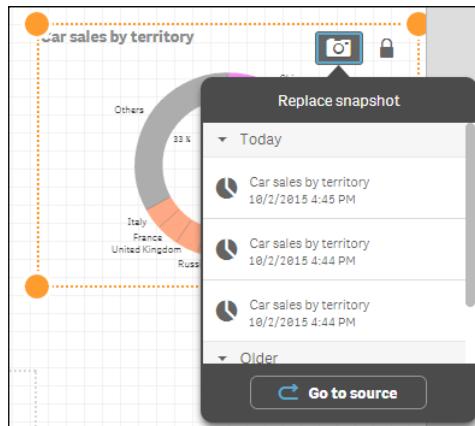
There's more...

While working in the story Edit mode, click on any individual object. On clicking the object we will find two options highlighted on the top right-hand corner. The first one is for **Replace snapshot** and the other one is for **Unlock the snapshot**. Use the following steps to explore the **Replace snapshot** and **Unlock the snapshot** functionalities:

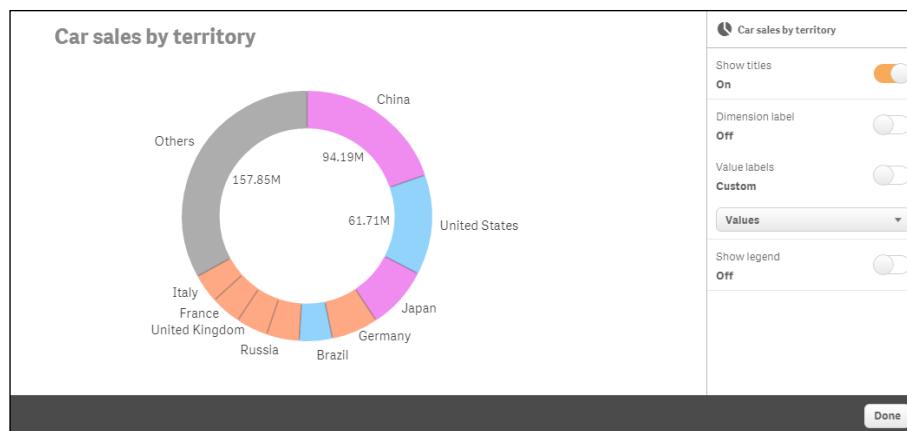
1. Click on **Replace snapshot**.

Visualizations

2. The following dropdown appears which lists all the snapshots captured for the original visualization. The snapshot in use is marked with a ✓ :



3. The user can replace the existing snapshot with a new one.
4. Alternatively, the user can click on the **Go to source** button in the dropdown that opens the original sheet where the visualization resides. New snapshots can then be created using the live data.
5. Click on **Return** button on the original sheet to return to the story.
6. Next, click on the **lock** button for visualization. This will unlock the snapshot and activate the edit **pencil** option.
7. Click on **pencil** to change the basic properties of the object. The modified properties for the object are specific to the story. The object on the Qlik Sense sheet still has the original properties, as shown:



A Qlik Sense storyline can also have embedded sheets. This is particularly useful if we want to showcase the entire content of the sheet on the slide.

See also

- ▶ *Adding embedded sheets to the story*

Adding embedded sheets to the story

Multiple sheets can be added to the story and the following section deals with the steps involved.

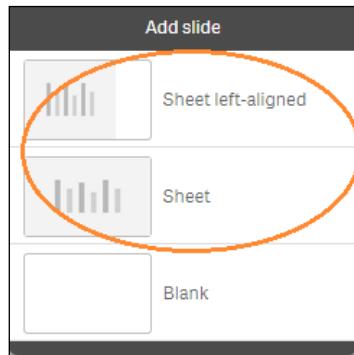
Getting ready

As in the previous recipe, we will again make use of the `Automotive.qvf` Qlik Sense application:

1. Open the `Automotive.qvf` application from the Qlik Sense hub.
2. Next, open the **Sales Overview** story, which was created in the previous recipe by clicking  **Stories** on the toolbar.

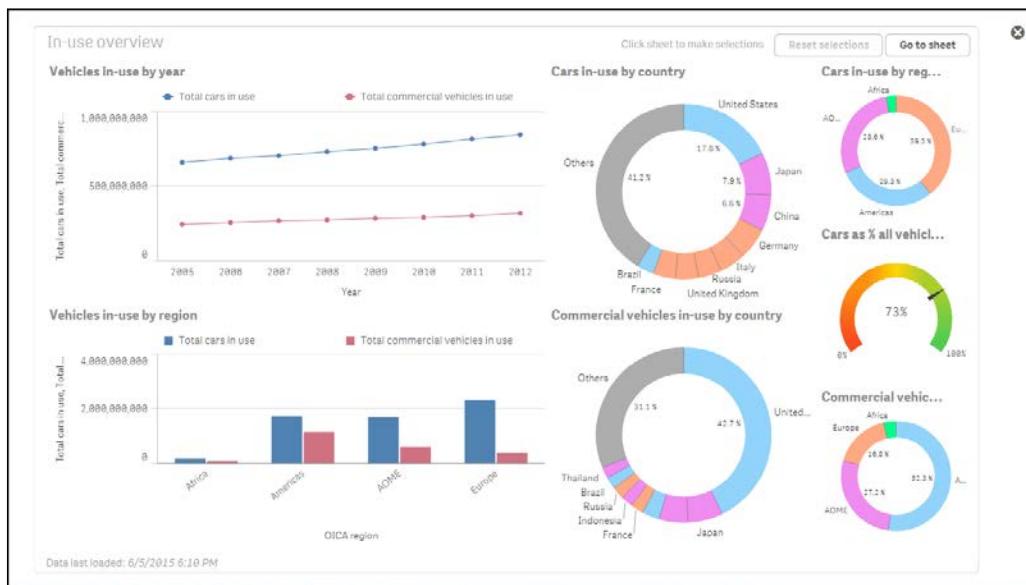
How to do it...

1. In the story view, click on the  image at the bottom left corner of the storyboard.
2. Select either **Sheet left-aligned** or **Sheet** from the **Add slide** pop up:



Visualizations

3. A slide with a sheet placeholder is added to the screen. Next, click on **Select Sheet** and select the **In-use overview** sheet from the dialog box, as shown in the following screenshot:



4. Save the file by clicking on the **Save** button.
5. The story can be played as a presentation by clicking on the **▶** button on the left-hand side vertical pane.

How it works...

When we embed a sheet into our story, it places all the contents of the desired sheet on the slide. The embedded sheet always has the same set of selections as the sheet in the sheet view.

There's more...

When we play the story, you will observe that we have two buttons at the top of the embedded sheet slide, namely **Reset selections** and **Go to sheet**. Selections can be made on the embedded sheet by clicking on each individual object. The **Reset selections** button clears the selections and the **Go to sheet** option takes the user back to the original sheet where new snapshots of objects can be taken. The **Go to sheet** button is also available in the edit-mode of the embedded sheet.

We can add effects and images to the story using the **Effects** and **Media library**, which is available in the storytelling view.

Highlighting the performance measure in a bar chart

One of the essential components of a Qlik Sense dashboard is the **Key Performance Indicators** or the **KPIs**. The KPIs indicate the health of the company based on specific measures. The information displayed in the KPI should stand out distinctly and demand attention. For example, one of the key KPIs that a CEO of the company may like to have on his dashboard is "Actuals vs Budget". A CEO is mostly interested in knowing if the company is below or above the budgeted figures. So, it makes sense to highlight the required information inside the visualization object. The following recipe explains and shows you how to do this in a bar chart.

Getting ready

A "Dial Gauge" is quite commonly used to display the key KPIs in Qlik Sense. However, the best design practices say that the "Bar chart" is the most effective way of conveying the information to the user. The following example makes use of a bar chart to strengthen this thought.

Perform the following steps to get started:

1. Create a new Qlik Sense application. Name it `Performance_Bar_Chart`.
2. Open the data load editor.
3. Load the following script, which contains information on the `Actuals` and `Budget` for four products. The script can be downloaded from the Packt Publishing website:

```
Products:  
LOAD * INLINE [  
    Product, Actuals, Budget  
    Footwear, 100000, 120000  
    Tyres, 180000, 150000  
    Mountain Bikes, 250000, 195000  
    Road Bikes, 200000, 225000  
];
```

How to do it...

The following steps highlight the performance measure in a bar chart:

1. Open the **App overview** and create a new sheet.
2. Create a bar chart on the sheet.
3. Add **Product** as the first dimension.

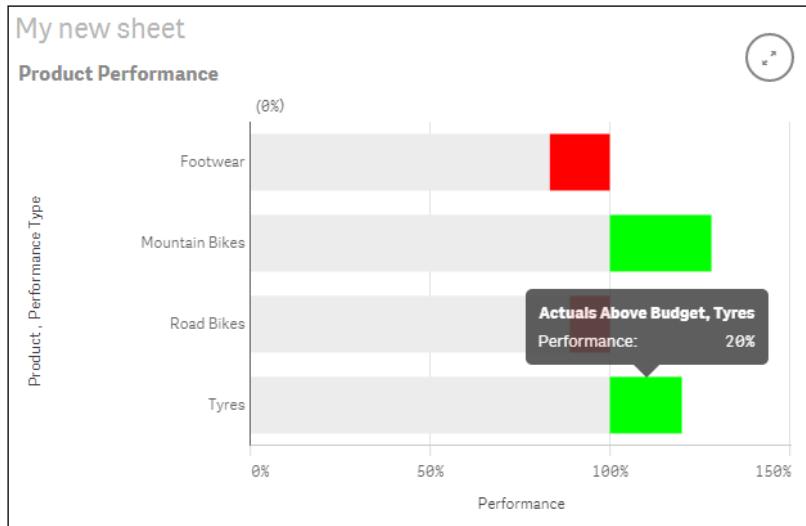
-
4. Under the Properties panel present on the right-hand side. Click on the **Add data** dropdown menu and select **Dimension**.
 5. Open the expression editor by clicking on .
 6. Add the following calculation as the second dimension and name it as **Performance Type**:

```
=ValueList ('Actuals Up To Budget', 'Actuals Below Budget', 'Actuals Above Budget')
```
 7. Click on **Add data** again and add the following measure to the object and call it as **Performance**:

```
if(ValueList ('Actuals Up To Budget', 'Actuals Below Budget', 'Actuals Above Budget')='Actuals Up To Budget', RangeMin(Sum(Budget), Sum(Actuals))/Sum(Budget) , if(ValueList ('Actuals Up To Budget', 'Actuals Below Budget', 'Actuals Above Budget')='Actuals Below Budget' , num( (RangeMax(Sum(Budget) - Sum(Actuals), 0))/Sum(Budget) , '$#,##0.00;-$#,##0.00') , (RangeMax(Sum(Actuals) - Sum(Budget), 0))/Sum(Budget) ) )
```
 8. Once we define the **Performance** measure, we will notice that just below the Expression box for the measure, we get a dropdown for number formatting. Under this dropdown, change the number format to **Number**. Next, we define the exact format of the number. To do this, switch off **Custom Formatting** and then under the dropdown below that select **Formatting** representation as **12%**.
 9. Under **Appearance**, click on **General** and add the **Product Performance** title.
 10. Under **Sorting**, set the sort-order for **Performance Type** as alphabetically and Descending.
 11. Under **Appearance**, click on **Presentation** and pick the style for the chart as **stacked** and **Horizontal**.
 12. Under **Colors** and **Legend**, switch off **Auto Colors** to activate custom colors.
 13. Along with the custom colors, a dropdown to define the colors is also activated. This is situated right below the colors switch. Under this dropdown select **By expression**.
 14. Add the following expression under the color expression:

```
if(ValueList ('Actuals Up To Budget', 'Actuals Below Budget', 'Actuals Above Budget')='Actuals Up To Budget', rgb(234,234,234) , if(ValueList ('Actuals Up To Budget', 'Actuals Below Budget', 'Actuals Above Budget')='Actuals Below Budget', rgb(255,0,0),rgb(0,255,0) ) )
```
 15. Make sure that **The expression is a color code** is checked.

16. The resulting chart will look like the following screenshot:



How it works...

The chart in this recipe shows the user the relative performance of each product. The colored segments highlight the extent by which a product has exceeded or failed to reach the budgeted value. The Green segment indicates that the product has fared well while the Red segment indicates that the product is below the budgeted figure.

The preceding example makes use of the `ValueList` function in both dimension and measure. For dimension, this results in three string values, namely "Actuals Up To Budget", "Actuals Below Budget", and "Actuals Above Budget" as row labels, which are further referenced in the measure.

The measure takes the values from the dimension and references them in a nested `IF` statement as an input to three aggregated calculations.

We use the `ValueList` function in this recipe as Qlik Sense doesn't allow you to have custom colors for each measure, which we needed in order to do the highlighting.

There's more...

The same information can be conveyed using a CapVentis Redmond Pie Gauge, the credit for which goes to Stephen Redmond, former CTO of Capventis. The Redmond Pie Gauge chart can be accessed on Qlik Branch at <http://branchqlik.com/projects/showthread.php?159-CapVentis-Redmond-Pie-Gauge-for-Qlik-Sense&highlight=redmond+pie+gauge>.

See also

- ▶ *Use the Colormix function*

Associating persistent colors to field values

The best practices say that a designer should avoid using bar charts with multi-colored bars or having too many colors in any of your chart objects. But at times, we need to cater to the demands of the organization and take an uncalled for approach to designing. The following recipe explains how to associate distinct field values to different colors in the Qlik Sense script.

Getting ready

This recipe serves to be a good example to demonstrate the use of `Pick` function in the script. Use the following steps to get started:

1. Create a new Qlik Sense application and name it `Persistent Colors`.
2. Open the data load editor.
3. Load the following script that contains information about the `Actuals` and `Budget` of four products. The script is available for download on the Packt Publishing website:

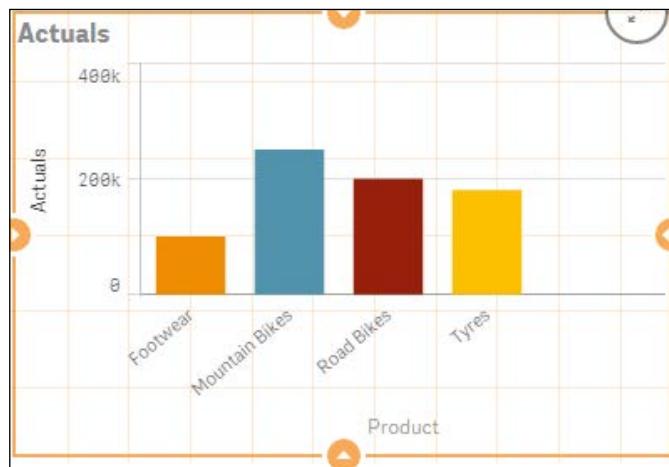
```
ProductsTemp:  
LOAD * INLINE [  
Product, Actuals, Budget  
Footwear, 100000, 120000  
Tyres, 180000, 150000  
Mountain Bikes, 250000, 195000  
Road Bikes, 200000, 225000  
];
```

```
Products:  
LOAD *,
```

```
pick(match("Product", 'Footwear', 'Tyres', 'Mountain  
Bikes', 'Road Bikes'), RGB(236,129,0),RGB(250,185,0),  
RGB(70,137,164), RGB(141,25,8)) as "Product color"  
  
RESIDENT ProductsTemp;  
  
Drop table ProductsTemp;
```

How to do it...

1. Open the **App overview** and create a new sheet.
2. Create a bar chart on the sheet.
3. Use **Product** as dimension.
4. Use **Sum (Actuals)** as measure. Label it as **Actuals**.
5. Under **Colors** and **Legend**, switch off **Auto Colors** to activate custom colors.
6. Along with the custom colors, a dropdown to define the colors is also activated. This is situated right below the colors switch. Under this dropdown, select **By expression**.
7. Add the following expression under the color expression:
=[Product color]
8. Make sure that **The expression is a color code** is checked.
9. The result would be as follows:



How it works...

The `Pick` function used in the script links values in the `Product` field to distinct RGB values. Each product is displayed in a different color bar when the `Product_color` field is used in the color expression of the chart.

There's more...

Persistent colors can also be obtained through the chart properties when we select the colors by dimension. However, using this approach we can't have custom colors but have to depend on the color scheme in Qlik Sense.

See also

- ▶ Use of `ColorMix1` function to establish a color gradient in charts

Using the `ColorMix1` function

Heat maps are a common requirement in most of the BI implementations. A `Colormix1` function helps to create a gradient between two colors. Look at the following recipe to understand the use of this function.

Getting ready

We will make use of a simple Inline load for this recipe. Perform the following steps to get started:

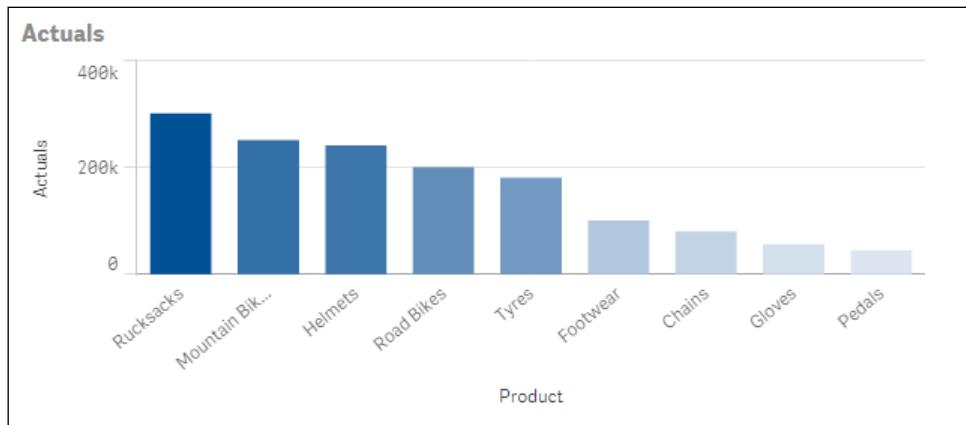
1. Create a new Qlik Sense application. Name it `HeatMaps_Colormix1`.
2. Open the data load editor.
3. Load the following script that gives you information about `Actuals` and `Budget` for products:

```
Products:  
LOAD * INLINE [  
    Product, Actuals, Budget  
    Footwear, 100000, 120000  
    Tyres, 180000, 150000  
    Mountain Bikes, 250000, 195000  
    Road Bikes, 200000, 225000  
    Chains, 80000, 90000  
    Helmets, 240000, 160001  
    Gloves, 56000, 125000
```

```
Pedals, 45000,100000  
Rucksacks, 300000,450000  
];
```

How to do it...

1. Open the **App overview** and create a new sheet.
2. Create a bar chart on the sheet
3. Use **Product** as dimension.
4. Use **Sum (Actuals)** as measure and label it as **Actuals**.
5. Switch off **Auto Colors** to activate custom colors under **Colors and Legend**.
6. Along with the custom colors, a dropdown to define the colors is also activated. This is situated right below the colors switch. Under this dropdown, select **By expression**.
7. Add the following expression under the color expression:
`colormix1(sum(Actuals) / ${=max(aggr(sum(Actuals),
Product))}, white(), RGB(0, 70, 140))`
8. Under **Sorting**, promote **Sales** above **Product** in the order of priority. This can be done by holding the  button and dragging it to **Sales** above **Priority**.
9. Set the **Sort** order for **Sales** as **Sort numerically and Descending**.
10. Make sure **The expression is a color code** is checked.
11. The result will be as follows:



How it works...

The colormix1 function creates a gradient between two colors using a number that varies from 0 to 1.

We know that the bar for the product with the highest value of Actuals will be the most intense. So to achieve a value between 0 and 1, we calculate the relative shares of each actual value against the highest actual value from the entire product range. That is "Actuals for each product"/"The highest Actuals value from the entire product range".

In our expression, the Colormix1 function helps to establish a gradient from white to RGB (0, 70, 140).

There's more...

A sequential color gradient across the chart can be obtained through the chart properties if we select the color by measure. However, we can't have custom colors if we use this approach and we will have to depend on the color scheme in Qlik Sense.

See also

Similar to Colormix1, we can also use the Colormix2 function, which gives us an option to have an intermediate color between the lower and upper limit color.

Composition

Composition can be defined as looking at a particular measure compared to the whole.

For example, In a "Sales by Region" chart, the sales for each singular region would be a discrete value while the total sales across all countries would be the "Whole".

Total sales can be divided into "Relative shares" for each region. Having information on "Relative Sales Percentages" as compared to the total sales has a greater impact rather than viewing just the plain sales figures. Eureka moments are much more likely when people use a tool to answer their own questions, which is a core belief behind the design of Qlik Sense.

As with everything else, data composition can be visualized in multiple ways. Understanding what you are trying to achieve will eventually dictate the best choice of visualization.

For example, depending on what matters, each of the following points will favor a different form of visualization:

- ▶ Relative differences
- ▶ Relative and absolute differences
- ▶ Share of the total

- ▶ Accumulation to the total (or subtraction)
- ▶ Breaking down components of components

As such, each example in the next four recipes will be supported by a goal, questions, and an analysis description, which is as follows:

- ▶ **Goal:** As a business analyst, I want to report on the best regions to focus our marketing strategy on
- ▶ **Question:** I want to see how our total revenue is shared this year across the various regions
- ▶ **Analysis:** How the total revenue is divided per region and if it is performing positively

Getting ready

Downloading the source files:



Downloading the example code

You can download the example source files from your account at <http://www.packtpub.com>, for all Packt books that you have purchased. If you have purchased the book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Use the following steps to get started:

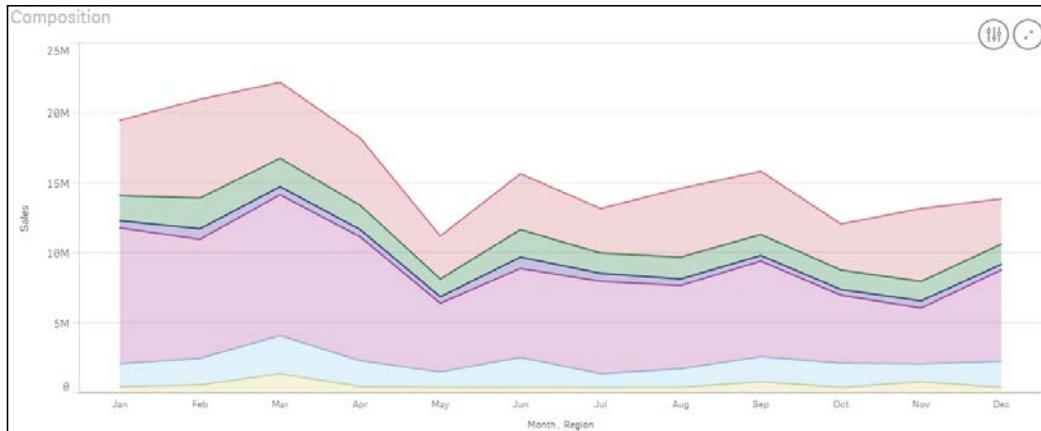
- ▶ Download Chapter 2 - Sales.qvf application from the Packt Publishing website
- ▶ Save the application at the following location: C:\Users\<user>\Documents\Qlik\Sense\Apps
- ▶ Open the application though the Qlik Sense hub

How to do it...

1. Click the button in the top right-hand corner in the application overview and click the **Create New Sheet** button. Name this sheet as Composition.
2. Go to the charts asset pane and double click the line chart button .
3. Add the following measure (m) and dimensions (d) in the same order as follows:
 - (m) Sum(Sales)
 - (d) Month
 - (d) Region

Visualizations

4. Select **Area** from the properties pane under the **Appearance | Presentation** menu.
5. Finally, tick the **Stacked area** box. The following screenshot is an example of the final visualization:



How it works...

Enabling the right property settings can turn a line chart into a stacked area chart. This clearly shows the differences when we analyze the relative and absolute composition of many time periods, as shown in the preceding example. If you had less time, say the last 3 years, then you would use the same approach, however; you will change the chart type to **Bar** instead of **Line** as the magnitude of change is more important than the change trend.

There's more...

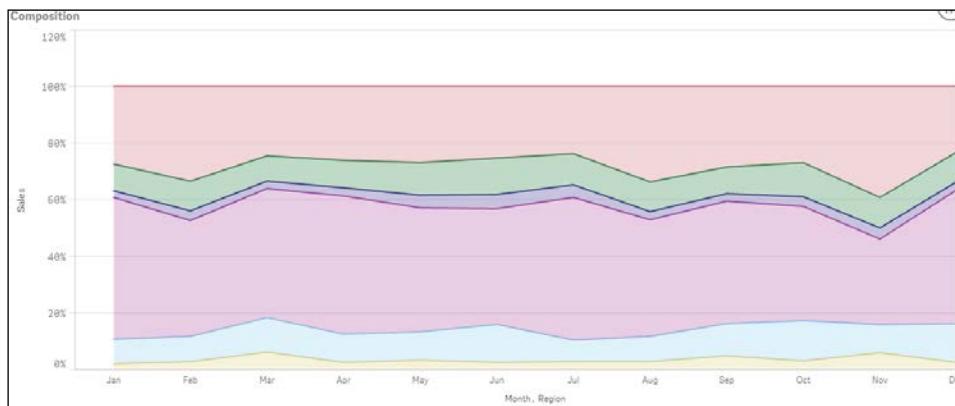
When looking at the composition in terms of accumulation or subtraction from the TOTAL, a good option for representation is the waterfall chart. If the only important differences are the relative differences are, then write your calculation as a percentage of Total.

To achieve this, follow the following steps:

1. Replace the **Sales** expression from the preceding recipe with the following:

```
Sum(Sales) / sum( TOTAL <Month> Sales) .
```

2. Once we define the preceding measure, we will notice that just below the expression box for the measure, we get a dropdown for number formatting. Under this dropdown, change the number format to **Number**. Next we define the exact format of the number. To do this, switch off custom formatting and then under the dropdown below that select **Formatting** representation as **12%**. This will produce the following 100 percent stacked area chart:



Relationships

Seeing relationships in data is something that is very difficult to achieve when we view data numerically. The following visualizations are the key to uncovering correlations, outliers and clusters in the data:

- ▶ **Goal:** Increase product subscriptions
- ▶ **Question:** Are there any differences in the relationship between the revenue and the Sales Quantity by product sub-group?
- ▶ **Analysis:** Here we will use a scatter graph to plot product sales that are grouped by product sub-group

Getting ready

We will make use of the same Chapter 2 – Sales.qvf application used in the "Composition" recipe.

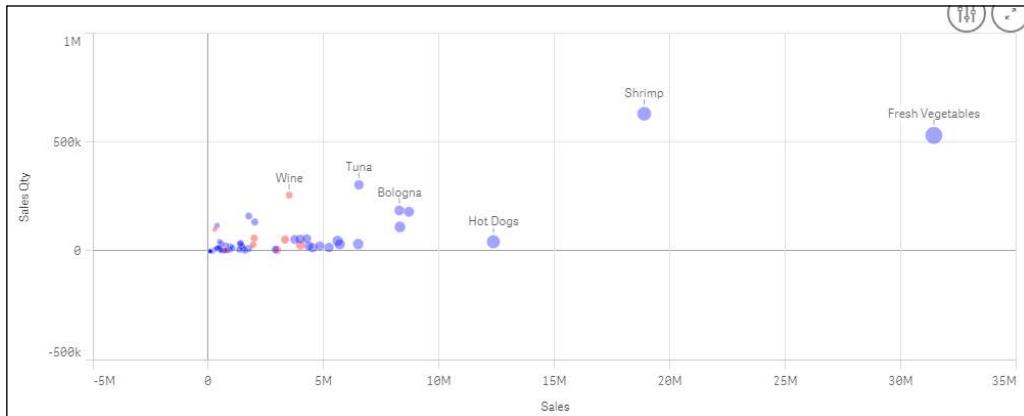
How to do it...

1. In the application overview click on the button in the top right-hand corner in order to create a new sheet and then click on the **Create New Sheet** button. Name this sheet Relationships.

Visualizations

2. Once inside the newly created sheet, go to the charts asset pane  and double-click on the scatter chart button .
3. Add the following measures (m) and dimensions (d) in exactly the same order as shown:
 - (m) Sum(Sales)
 - (m) Sum([Sales Qty])
 - (m) Sum(Margin)
 - (d) Product Sub Group
4. In the properties pane, under **Appearance | Colors** and **legend**, switch-off the **Auto colors**. Then select **By expression** from the drop-down menu.
5. Finally, add the following expression into the area provided below the dropdown menu:

```
IF( [Product  
Line] = 'Drink' , ARGB(100,255,0,0) , ARGB(100,0,0,255) )
```
6. The final visualization should resemble the following screenshot:



How it works...

Since the nature of product sub-group dimension is hierarchical, we can actually show two relationships. The first is between the different measures whereas the second looks at the relationships between different product sub-group categories by coloring them separately.

Comparison

The bar graph is one of the most common data visualizations. This is because it is simply the best way of comparing the difference in value across a single item.

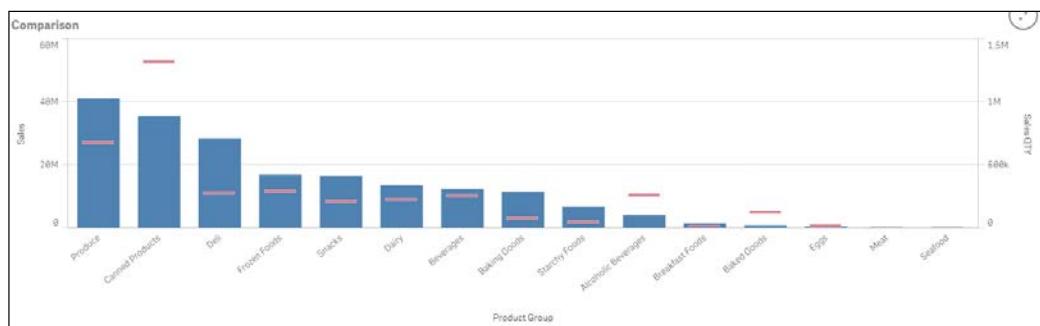
- ▶ **Goal:** Increase product subscriptions
- ▶ **Question:** Why does a sub set of similar products not respond as positively as others in the same market?
- ▶ **Analysis:** Combo chart

Getting ready

We will make use of the same Chapter 2 – Sales.qvf application used in the "Relationships" recipe.

How to do it...

1. From the application overview click the button in the top right-hand corner and click the **Create New Sheet** button. Name this sheet Comparison.
2. Once inside the newly created sheet, go to the charts asset pane  and double click the **Combo chart** button .
3. Add **Product Group** as a dimension.
4. Next add **Sum (Sales)** as the first measure. Label it **Sales**.
5. Add **sum ([Sales Qty])** as the second measure. Label it **Sales Qty**.
6. For the **Sales Qty** measure:
 - Change the default display-format for the expression from **Bars** to **Marker**
 - Right below the display format options, there is a dropdown to define the axis.
 - Set the axis to secondary. Just below the axis formats there is a markers style dropdown. Select the style as **Line**.
7. Under **Sorting**, promote **Sales** to the top of the list.
8. The visualization should resemble the following image:



How it works...

When it comes to comparing the magnitude of change of the values against each other, you really cannot beat a bar chart. When you need to compare multiple dimensions in the same visualization, a common approach is to stack them on top of each other. This option is available in the properties of the bar chart object.

However, this removes the length comparison we are so good at, thus making the view not as effective.

The preceding method of using symbols instead of additional bars still leaves a good focus on the comparative length to determine the magnitude of change. This is also a more efficient use of space than creating separate visualizations to cover additional analysis.

See also

- ▶ *Highlighting the performance measure in a bar chart*

Distribution

Distribution analysis takes a look at how quantitative values are distributed along an axis, from the lowest to the highest. The characteristics emerge while looking at the shape of the data, such as central tendency, shape and outliers:

- ▶ **Goal:** To understand, which demographics should be focused on for our marketing approach for a specific product group
- ▶ **Question:** The suitable age-range to target our new marketing campaign towards
- ▶ **Analysis:** Use a histogram to see a useful range from the mean age

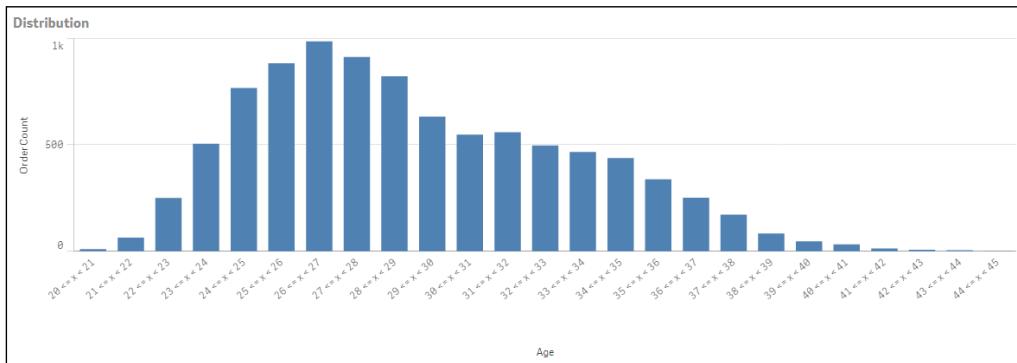
Getting ready

We will make use of the same Chapter 2 – Sales.qvf application which we have used in the "Comparison" recipe.

How to do it...

1. In the application overview, click on the button in the top right-hand corner and click on the **Create New Sheet** button. Name this sheet Distribution.
2. Once inside the newly created sheet, go to the charts asset pane  and double click on the bar chart button .
3. In the properties pane to the right of your screen, click on **Add** data and select **Dimension**.

4. Click on the  button for the input box of dimension and enter the following pre-calculated dimension. Label the dimension as Age.
 $=\text{Aggr}(\text{Class}(\text{Age}/19, 1), \text{Order})$
5. Add the following line as the measure and label it Order Count.
 $\text{Count}(\text{Order})$
6. In the Properties panel, select **Age** under **Sorting** and check **Sort numerically**. In the dropdown located right below the **Sort numerically** check box, select the order as **Ascending**.
7. The final distribution chart should look like the following screenshot:



How it works...

Distribution visualizations help you to analyze one or two variables spread along an axis starting from the lowest to the highest. The shape of the data will tell you about characteristics, such as the central tendency, shape and outliers.

Structuring visualizations

As discussed in the introduction, when choosing visualization you should start with knowing if you are looking at a comparison, composition, distribution, or relationship.

While this helps in answering a single question effectively, this is often to fulfill the goal that you want to see the information from different angles. Structuring visualizations to easily answer "the next question" keeps consistency in analysis.

While in Qlikview it is common to design a user interface with more interaction than simply filtering the data, Qlik Sense is built with a large focus on the business user and analyst. This recipe involves little practical work and instead it carries the torch for the expert designers in the product team at Qlik. Here is an example of how and why you should make use of a screen

and not just an object.

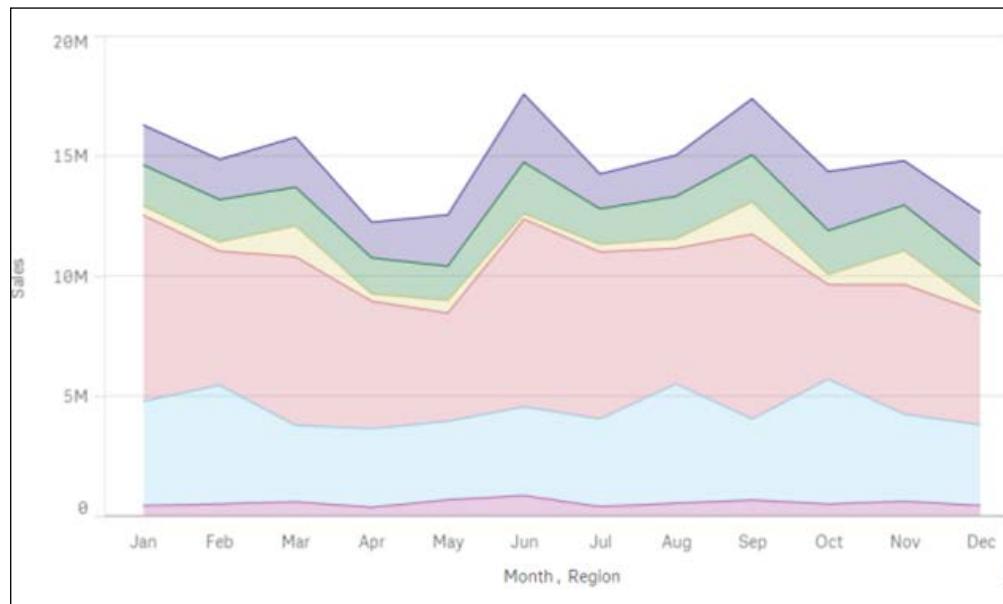
Getting ready

We will make use of the same Chapter 2 – Sales.qvf application used in the "Distribution" recipe.

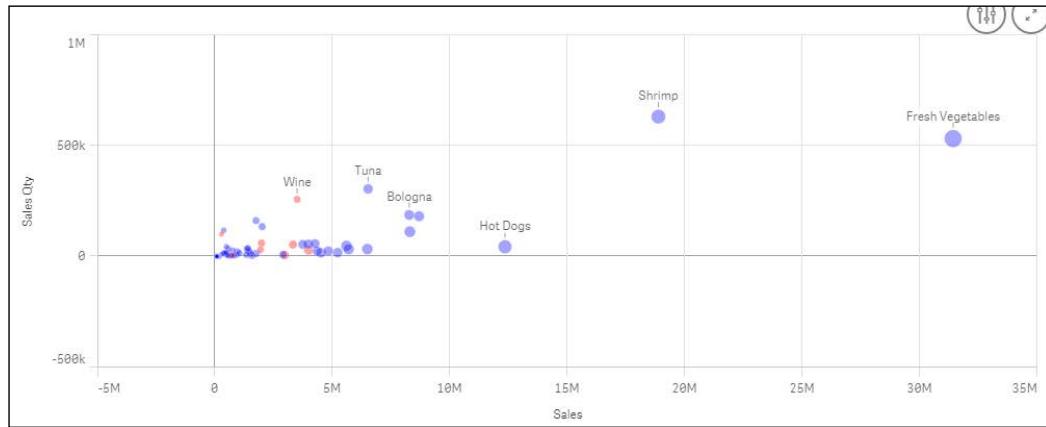
How to do it...

The charts that were built in the previous visualization category recipe are available as master visualizations in the Chapter 2 – Sales.qvf application. As mentioned during the introduction, the reason behind the structure of the previous four recipes was to think of the business question first. If you can answer someone's question about business with each page (tab), you have a book of gold.

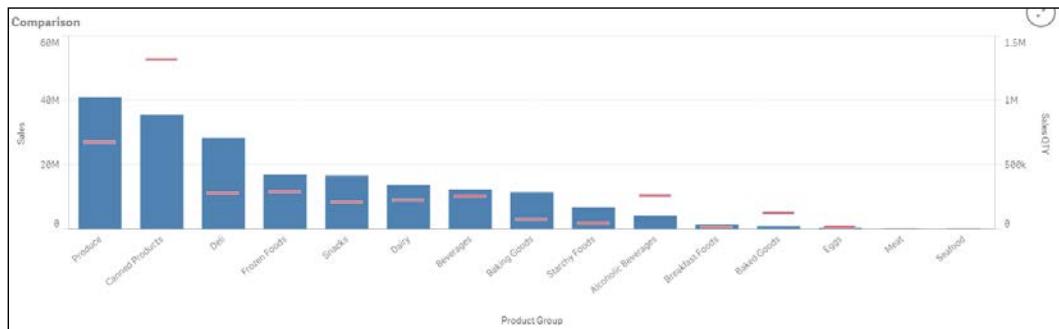
1. From the application overview, click the button in the top right-hand corner in order to create a new sheet. Name this sheet as Structuring Visualization.
2. Open the master items pane.
3. Under visualizations drag the **Composition** chart into the top left-hand corner:



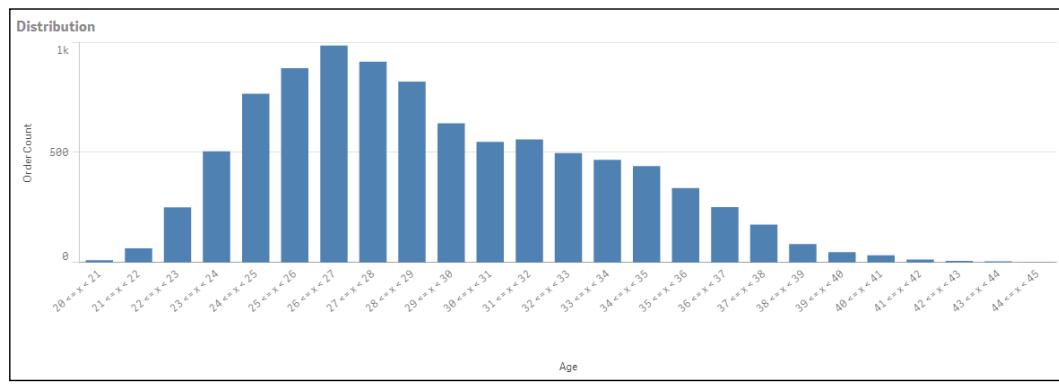
4. Next, drag the **Relationship** chart into the top right-hand corner:



5. Drag the **Comparison** chart into the bottom left-hand corner:

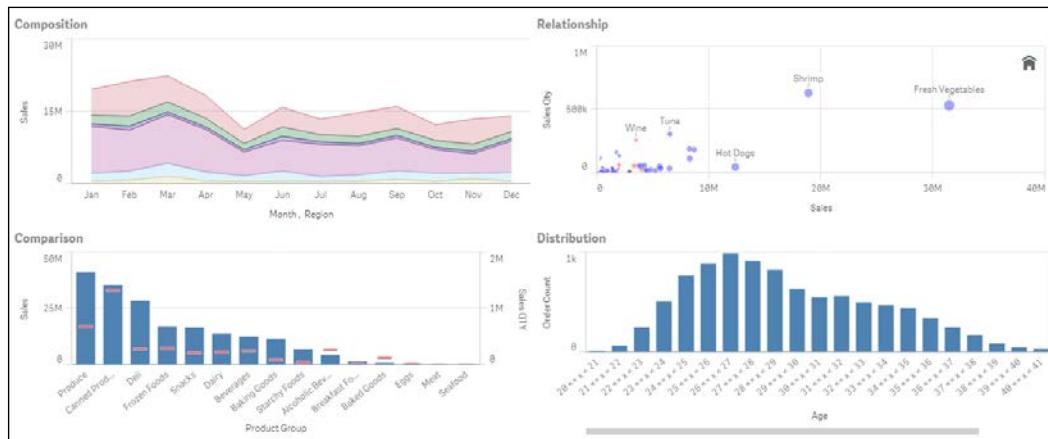


6. Finally, drag the **Distribution** chart into the bottom right-hand corner:



How it works...

Each of the four preceding recipes has a question to answer for a shared or similar goal. Placing complementary visualizations near each other is good page design. Each chart adds context to the others and helps to build up a clearer analysis picture. The final result should look like the following image. Selecting a point of interest in any of the four charts will show you that the data set from other angles gives you a greater insight with a single-click.



3

Scripting

In this chapter, we will discuss the creation of optimized and well-structured scripts for a Qlik Sense application. We are going to cover the following topics:

- ▶ Structuring the script
- ▶ Efficiently debugging the script
- ▶ Packaging the code in script files
- ▶ How to use subroutines in Qlik Sense®
- ▶ Optimizing the UI calculation speed
- ▶ Optimizing the reload time of the application
- ▶ Using a `For Each` loop to load data from multiple files
- ▶ Using the `Concat` function to store multiple field values in a single cell

Introduction

What is a script in Qlik Sense? In lay man's language, a script instructs the Qlik Sense engine on how to extract the data from the data source and what to do with it.

It forms an essential component of the ETL process. Hence, it is important to have a well-structured script in order to load the data efficiently. A good understanding of how to optimize an ETL process, leads to a better data model. A good data model is one of the core components along side well written expressions to realize a good user interface performance.

Structuring the script

The techniques for adding structure to the script is something that comes naturally to experienced developers. This is because they have often learned it the hard way, through other people's work and spent additional time to understand the script that can be made easier with a couple of simple additions. Again, this is something that won't be covered in user guides but it is a very important skill for new developers to have under their belt.

Getting ready

In this example, we will generate the required data automatically in the script.

How to do it...

1. Create a new Qlik Sense application and name it Structuring Scripts.
2. Create a new section in the data load editor called Change Log.
3. Add the following code:

```
/*
This application demonstrates the importance of adding
structure to the back end script of your applications
```

Change Log:

```
[10/06/2015] Philip Hand: Initial build
```

```
*/
```

4. Create another section called Calendar and add the following script:

```
/*=====
Section: Calendar Tab;
DESCRIPTION: Generates every date between the periods
    vMinDate & vMaxDate;
DEVELOPERS: Philip Hand, Neeraj Kharpate;
//=====
TRACE START:~~~Loading Calendar Tab~~~;

Let vMinDate=DATE(Floor(MakeDate(2009,1,1)),'DD/MM/YYYY');
Let vMaxDate=DATE(Floor(Today()));
TRACE Calendar date range set to $(vMinDate) & $(vMaxDate);

Let vDiff=vMaxDate-vMinDate+1;
```

```
Calendar:  
Load  
DateID,  
Year(DateID) As Year,  
Month(DateID) As Month,  
Date(DateID) As Date,  
Day(DateID) As Day,  
Week(DateID) As Week;  
Load  
RecNo() -1+$ (vMinDate) As DateID  
AutoGenerate($ (vDiff)) ;
```

```
TRACE END:~~~~Loading Calendar Tab~~~~;
```

5. Finally, save the data and load the script.

How it works...

The first tab gives an overview of the application and calls out any key information that a new developer who is seeing the script for the first time will find useful. It also includes a change log to track the changes.

The second tab has a high-level description of the contained code and the developers who have worked on it. Finally, we can make use of the TRACE statements to write information into the execution window. This allows you to see each action being performed during script execution and is a useful tool to debug errors.

Efficiently debugging the script

It is a good script practice to debug the script in your data load editor before its full execution. This way the developer minimizes the risk of script failures and also saves on valuable time. The process of debugging makes it possible to monitor every script statement and examine the variable values while the script is being executed. The following recipe explains how to debug the Qlik Sense script efficiently.

Getting ready

Load the following script, which gives information about the Products and Customers in the Qlik Sense data load editor. The sample code is available for download from the Packt Publishing website:

```
Products Temp:  
LOAD * INLINE [
```

Scripting

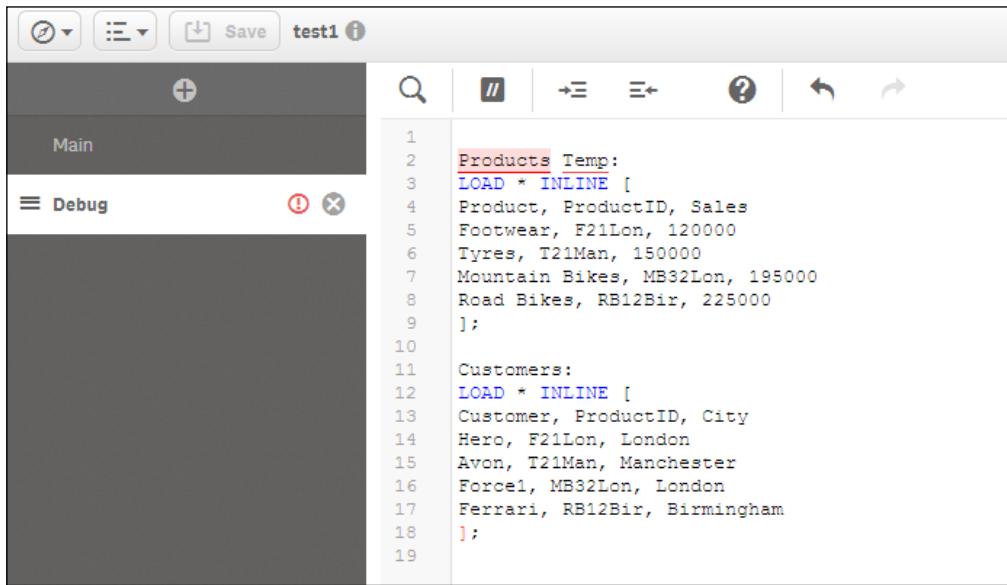
```
Product, ProductID, Sales
Footwear, F21Lon, 120000
Tyres, T21Man, 150000
Mountain Bikes, MB32Lon, 195000
Road Bikes, RB12Bir, 225000
];
```

Customers:

```
LOAD * INLINE [
Customer, ProductID, City
Hero, F21Lon, London
Avon, T21Man, Manchester
Force1, MB32Lon, London
Ferrari, RB12Bir, Birmingham
];
```

How to do it...

1. Save the preceding script.
2. When you save the script Qlik Sense automatically detects syntax issues present in the script, if any. The syntax issues are highlighted in red as shown in the following screenshot. Also make a note of the  mark beside the section name in the Section panel. This indicates that there is an issue with the script on the tab.

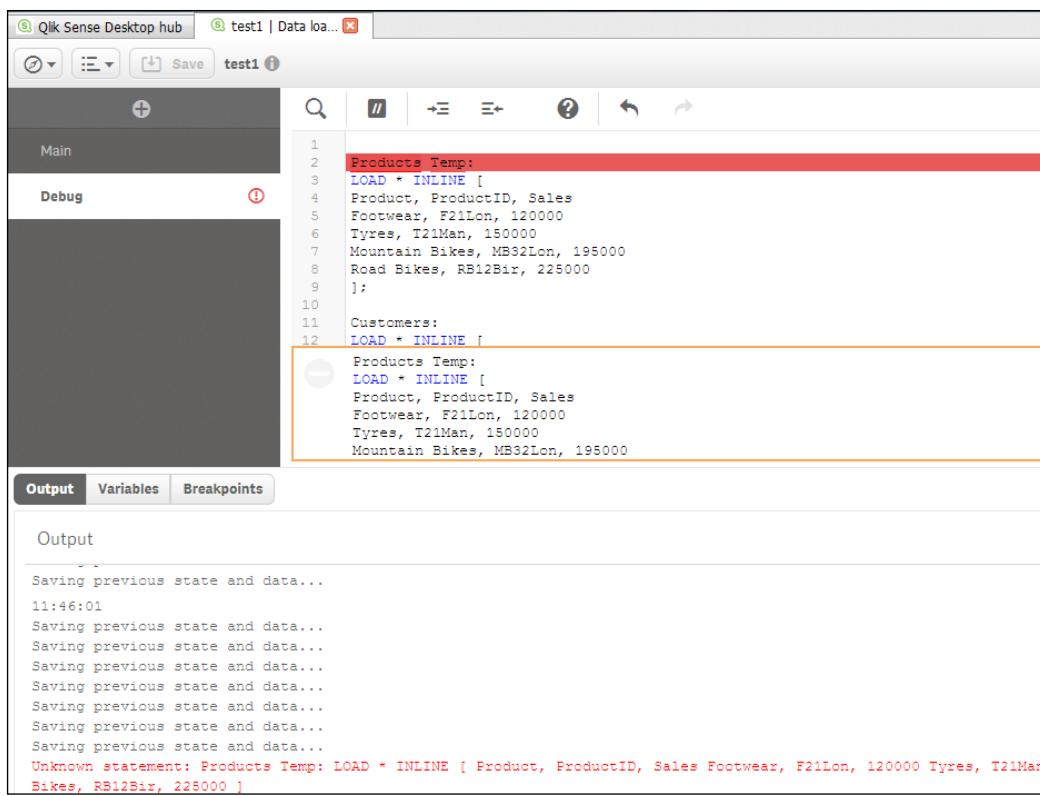


The screenshot shows the Qlik Sense Script Editor interface. The title bar says "test1". The left sidebar has tabs for "Main" and "Debug", with the "Debug" tab selected. A red exclamation mark icon is visible next to the "Debug" tab. The main editor area contains the following Qlik Sense script:

```
1 Products Temp:
2 LOAD * INLINE [
3 Product, ProductID, Sales
4 Footwear, F21Lon, 120000
5 Tyres, T21Man, 150000
6 Mountain Bikes, MB32Lon, 195000
7 Road Bikes, RB12Bir, 225000
8 ];
9
10 Customers:
11 LOAD * INLINE [
12 Customer, ProductID, City
13 Hero, F21Lon, London
14 Avon, T21Man, Manchester
15 Force1, MB32Lon, London
16 Ferrari, RB12Bir, Birmingham
17 ];
18
19
```

The word "Products" in line 1 is highlighted in red, indicating a syntax error. The "Customers" section at the bottom is also highlighted in red.

3. Next, click on the Show debug panel  button on the top right corner.
4. A debug panel pops-up from the bottom of the screen with 3 toggles, Output, Variables and Breakpoints.
5. In order to debug the script, load only the limited records, as this will speed up the process of debugging. Keep in mind that when you load limited records from a number of tables in your Qlik Sense script, there may be no records that associate the tables. However, you don't need to worry, as the main concern here is checking the accuracy of the script. Once it is confirmed, the script will run through without errors and you may go ahead and do a full reload.
6. Take a limited load of 10 records by ticking the box for **Limited Load** and entering 10 in the input box. Click on the run  button.
7. On running the debugger, Qlik Sense checks the entire script line by line and looks for any errors. If the script has an error, the execution stops at that point and the issue is highlighted in amber colored box, as shown. The line at which the execution has stopped is highlighted in red:



```

1 Products Temp:
2 LOAD * INLINE [
3 Product, ProductID, Sales
4 Footwear, F21Lon, 120000
5 Tyres, T21Man, 150000
6 Mountain Bikes, MB32Lon, 195000
7 Road Bikes, RB12Bir, 225000
8 ];
9
10
11 Customers:
12 LOAD * INLINE [
Products Temp:
LOAD * INLINE [
Product, ProductID, Sales
Footwear, F21Lon, 120000
Tyres, T21Man, 150000
Mountain Bikes, MB32Lon, 195000
]
```

Output

Saving previous state and data...
11:46:01
Saving previous state and data...
Unknown statement: Products Temp: LOAD * INLINE [Product, ProductID, Sales Footwear, F21Lon, 120000 Tyres, T21Man Bikes, RB12Bir, 225000]

8. The output window gives us the details of the encountered error. Click on the run ➤ button again to complete the script execution. Once the script execution is complete, you will notice that the `Customers` table is loaded fine but the `Products` temp table is not loaded at all. We can verify the same by checking the data model viewer.
9. Check the **Variables** tab. The **ScriptErrorCount** variable gives the count of errors and the **ScriptErrorList** shows the type of error, which in our case is a **Syntax Error**:

| Variables | |
|----------------------|----------------|
| ▼ Favorites | |
| ▼ All variables | |
| ★ ScriptError | |
| ★ ScriptErrorCount | 1 ← |
| ★ ScriptErrorDetails | <NULL> |
| ★ ScriptErrorList | Syntax Error ← |

10. At this point, the user can remove the space between the words `Products` and `temp` in the label for the `Products` table to rectify the script error.
11. We can also define breakpoints in our script by clicking on the area besides the line number in the script window. The breakpoints are denoted by  .
12. The script execution stops at the breakpoint.
13. The breakpoints can be enabled, disabled, and deleted at will by selecting and deselecting them under the **Breakpoints** list or by re-clicking on the  icon on the number line. The **Breakpoints** are ignored at the blank lines and inside the `LOAD` statement in the middle of the field list:

The screenshot shows the Qlik Sense Debug panel. At the top is a script editor window containing a Qlik Sense script. Below the editor are two panels: 'Variables' on the left and 'Breakpoints' on the right. The 'Variables' panel has sections for 'Favorites' and 'All variables'. The 'Breakpoints' panel lists two breakpoints: 'Debug: 1' and 'Debug: 12', both of which are checked.

```

1
2 ProductsTemp:
3 LOAD * INLINE [
4 Product, ProductID, Sales
5 Footwear, F21Lon, 120000
6 Tyres, T21Man, 150000
7 Mountain Bikes, MB32Lon, 195000
8 Road Bikes, RB12Bir, 225000
9 ];
10
11 Customers:
12 |
13 LOAD * INLINE [
14 Customer, ProductID, City
15 Hero, F21Lon, London
16 Avon, T21Man, Manchester
17 Force1, MB32Lon, London
18 Ferrari, RB12Bir, Birmingham
19 ];

```

14. Alternatively, the user can step through each statement of code in the script by clicking on the Step button.

How it works...

The Debug panel in Qlik Sense checks through the entire script for errors and makes sure that it is accurate. One major benefit of using a debugger is that the user can load only a few records into the data model for the test. The debugger also allows the user to check the output of the executed script and make sure that it is as desired.

There's more...

The Debug panel also helps you to identify issues related to variables and fields in the files defined under the `$ (include)` statement. We can also inspect the variables during the script execution. The variables can be accessed by clicking on the **Variables** toggle. One can set any of the variables as favorites by clicking on the next to the variable.

See also

- ▶ *Packaging the code in script files*

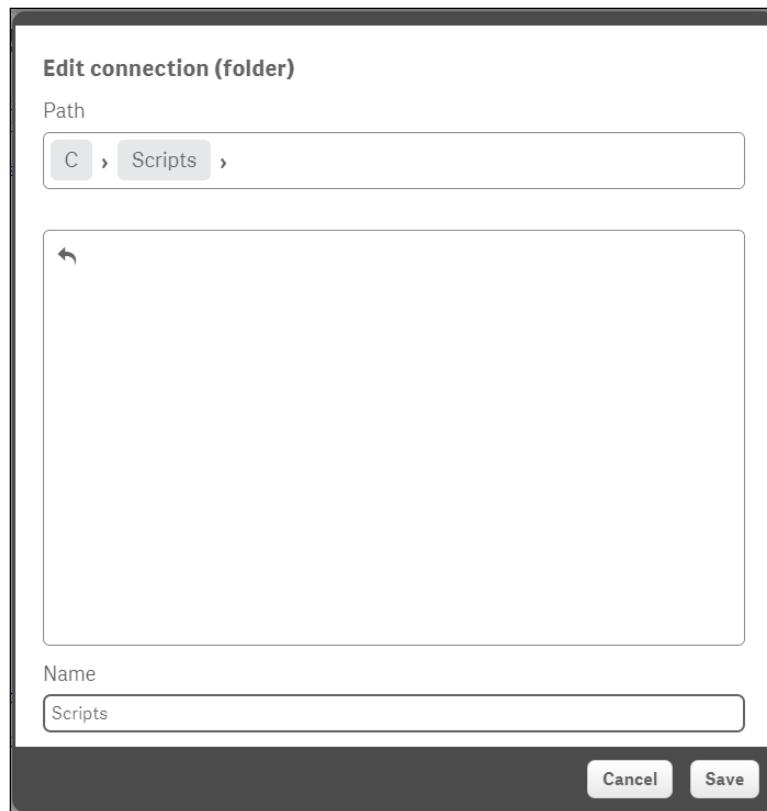
Packaging the code in script files

Script files are complete blocks of code that are stored in external files such as .qvs or .txt and they can be included in your application with a single reference. They are conceptually similar to the subroutines that are covered in another recipe in this chapter. However, there is a subtle difference in the usage. **QVS** simply stands for **QlikView Script File**.

Everything from data sources, expressions, and visualizations can be governed centrally and the script files can be leveraged in a similar way to help build standards in backend data preparation across multiple applications.

Getting ready

Open a new QlikSense application and create a data connection into a folder where you want to store your script files. As shown in the following example



How to do it...

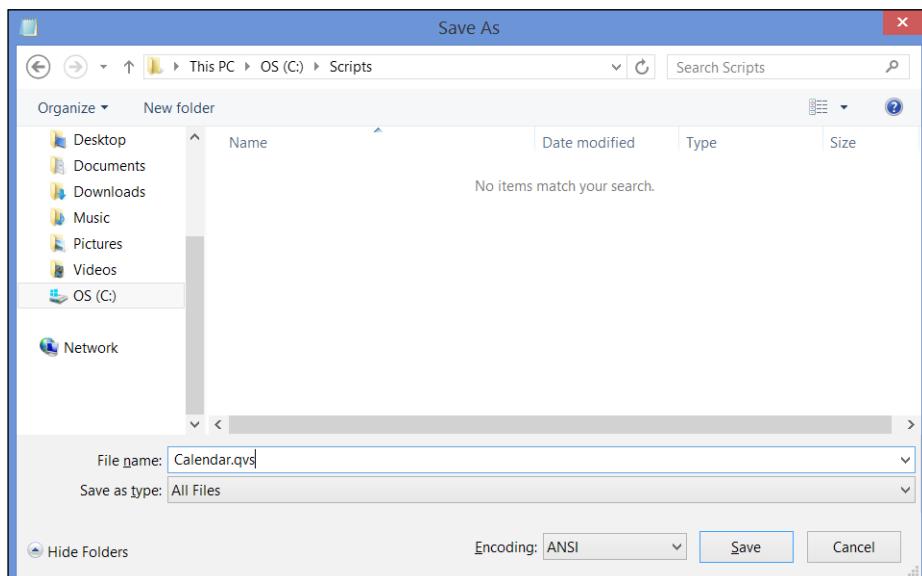
1. Open a Notepad document.
2. Copy the following subroutine script (a simplified version of the calendar code from the previous recipe) into the Notepad document:

```
SUB Calendar(vMinDate, vMaxDate)

Let vDiff=vMaxDate-vMinDate+1;
Calendar:
Load
DateID,
Year(DateID) As Year,
Month(DateID) As Month,
Date(DateID) As Date,
Day(DateID) As Day,
Week(DateID) As Week;
Load
RecNo()-1+$vMinDate) As DateID
AutoGenerate($vDiff);

END SUB
```

3. Save the Notepad file as `Calendar.qvs` into the folder for which you created the data connection. Remember to change the **Save as type** to **All Files**, as shown:



4. Add the following line of code to your application:

```
$ (Include=[lib://Scripts/calendar.qvs]);
```

5. You can now call the same subroutine without seeing the code, as in the previous example. In order to generate a calendar, use the following CALL statement in the script:

```
CALL Calendar(01/01/2010,Floor(Today())); ;
```

How it works...

We have replaced a page of code with just one line using a script file. If you have a code that can be packaged and reused across applications, it makes sense to store this code into a script file for others to use. Doing this reduces the complexity and keeps the focus of new developers on the backend code and on the matter that is relevant to that application.

It's worth pointing out that you could copy the code in the preceding point directly into the script editor and it will still get created and be ready for use. We save this code as a QVS file so that we can load the code using the `$ (Include...)` statement. Loading the script from external files using the `$ (Include...)` statement is a good method of reusing blocks of script across applications or using them for source control.

See also

- ▶ [How to use sub routines in Qlik Sense®](#)

How to use sub routines in Qlik Sense®

At times, it is mandatory to use the same set of code at different places in the script. To achieve this, developers will sometimes use a copy and paste approach. However, this makes it difficult to maintain and read the script. It is always advised to create subroutines and then simply call them as and when needed.

In the following recipe, we use subroutines to create QVDs and store them in a specific folder. We also generate fields using various functions within the subroutines, which also helps in auditing the QVD files.

Getting ready

1. This recipe makes use of certain functions such as `QVDTblename`, `QVDNoFields` and `QVDNoOfRecords`, which don't work in normal script mode in Qlik Sense. Hence, we need to activate the legacy mode by following the steps given in the recipe titled *How to activate Legacy mode in Qlik sense* in Chapter 1, *Getting Started with the Data*.

2. Once the legacy mode is activated, open Qlik Sense desktop and create a new application called Subroutines in Qlik Sense.
3. Create a folder called QVD at a desired location on the hard drive. For the sake of this recipe, we are creating the QVD folder at the following location:
C:\QlikSense_Cookbook\Chapters\3\QVD
4. This folder will store the QVDs generated in the subroutines.

How to do it...

1. Open the data load editor.
2. Create a new data connection called as QVDFolder. This data connection should create a folder connection to the QVD folder created in step 1.
3. In the data load editor, create a new section variable Setting, and add the following code to it:

```
LET vFileName = subfield(DocumentName(), '.', 1);

SET vTable1 =1; //Product

SET vTable2 =1; //Customer

LET vQVD='C:\QlikSense_Cookbook\Chapters\3';
```

4. Create a new Section Data, and add the following code to it:

```
SUB Create_T_Product

$(vTable):
LOAD * INLINE [
Product, ProductID, Sales
Footwear, F21Lon, 120000
Tyres, T21Man, 150000
Mountain Bikes, MB32Lon, 195000
Road Bikes, RB12Bir, 225000
];
END SUB
```

```
SUB Create_T_Customer

$(vTable):
LOAD * INLINE [
Customer, ProductID, City
Hero, F21Lon, London
```

```
    Avon, T21Man, Manchester  
    Force1, MB32Lon, London  
    Ferrari, RB12Bir, Birmingham  
];  
END SUB
```

5. Create a new section called `Store_Drop` and add the following code to it:

```
SUB Create_QVD_Standard(vTable,vSub)  
  
    LET vQVDStartTime = num(now());  
    CALL $(vSub)  
  
    STORE '$(vTable)' INTO $(vQVD)\QVD\$(vTable).qvd(qvd);  
  
    DROP TABLE $(vTable);  
  
    LET vFieldType = 'QVD_Standard';  
  
    LET vQVDEndTime = num(now());  
  
    LET vQVDTIMETaken = $(vQVDEndTime) - $(vQVDStartTime);  
  
    LET vTableFullPath = DocumentPath();  
  
    TablesLoaded:  
    LOAD  
        QVDTableName('$(vQVD)\QVD\$(vTable).qvd') AS [STDQVD  
        Name],  
        Timestamp($(vQVDStartTime), 'DD MMM YY hh:mm') AS [STDQVD  
        Start Time],  
        Timestamp($(vQVDEndTime), 'DD MMM YY hh:mm') AS [STDQVD  
        End Time],  
        Interval($(vQVDTIMETaken), 'hh mm ss') AS [STDQVD Time  
        Taken (hh mm ss)],  
        QVDNoOfFields('$(vQVD)\QVD\$(vTable).qvd') AS [STDQVD No  
        of Fields],  
        QVDNoOfRecords('$(vQVD)\QVD\$(vTable).qvd') AS [STDQVD No  
        of Records]  
        AUTOGENERATE (1);  
  
    END SUB
```

6. Create a new section called `Create_qvd` and add the following code to it:

```
LET vRunStart = timestamp(now(), 'DD MMM YYYY hh:mm:ss');

If $(vTable1) = 1 Then
CALL Create_QVD_Standard('T_Product', 'Create_T_Product')
ENDIF;

If $(vTable2) = 1 Then
CALL Create_QVD_Standard('T_Customer', 'Create_T_Customer')
ENDIF;

LET vRunFinish = timestamp(now(), 'DD MMM YYYY hh:mm:ss');

LET vRunTime = Interval(num(timestamp#('$(vRunFinish)', 'DD
    MMM YY hh:mm:ss'))
    -num(timestamp#('$(vRunStart)', 'DD MMM YY
        hh:mm:ss')), 'hh:mm:ss');
```

7. Save and reload the document.
8. On the front end, click on edit at the top right hand corner and create a new Table object by dragging it across the sheet from the left hand side panel.
9. Add all the available dimensions in the table to get the following output:

| QVD Audit | | | | | |
|-------------|---------------------|----------------------|-------------------|-----------------|------------------------------|
| STDQVD Name | STDQVD No of Fields | STDQVD No of Records | STDQVD Start Time | STDQVD End Time | STDQVD Time Taken (hh mm:ss) |
| T_Customer | 2 | 4 | 09 Jun 15 10:53 | 09 Jun 15 10:53 | 00 00 00 |
| T_Product | 3 | 4 | 09 Jun 15 10:53 | 09 Jun 15 10:53 | 00 00 00 |

How it works...

The first two subroutines named `SUB Create_T_Product` and `SUB Create_T_Customer` create the tables called `Product` and `Customer` and then store the data in these tables.

The third subroutine `SUB Create_QVD_Standard(vTable, vSub)` passes the values of the respective table names and the subroutines. Within this sub routine we also create a number of fields using the load script functions, which are used for our QVD audit purposes.

Further, the `CALL` statements call the subroutines and create QVDs to store them in specified folders.

Along with creating and storing the QVDs, we also get valuable information, such as the number of fields in each QVD, the time it takes to create the QVDs, and so on. It is especially helpful while loading a large dataset.

There's more...

The subroutines can be stored in an external file and further used in the script using an include statement.

See also

- ▶ [Packaging the code in script files](#)

Optimizing the UI calculation speed

The following recipe discusses the creation of Flags in the script and the use of these flags in the Chart expressions to optimize the calculation speeds.

A flag can be described as a binary status indicator that is used to indicate certain states of data. For example, creating a new field in the table called MonthToDate Flag. This field can be used to flag records with the number 1 if the record was created in the last month, else we mark the record with a 0.

Using this approach, we can now count the number of records in the table that were created in the last month using the expression `SUM([Month To Date Flag])`.

A flag is often used to code complex decision logic into the load script so that the binary "yes" or "no" decisions can be quickly identified from the calculations.

Getting ready

For this recipe we will generate a sales data in the script as defined in the following script. Load the following script into the data load editor:

```
Calendar:  
Load  
    DateID,  
    RowNo() AS ID,  
    Year(DateID) AS Year,  
    Year(DateID) & '' & NUM(Month(DateID), '00') AS YearMonth,  
    Month(DateID) AS Month,  
    Date(DateID) AS Date,  
    Day(DateID) AS Day,  
    Week(DateID) AS Week,  
    Floor(2000 * rand()) AS Sales;  
Load  
    RecNo() -1+makedate(2014) AS DateID  
    AutoGenerate(730);
```

How to do it...

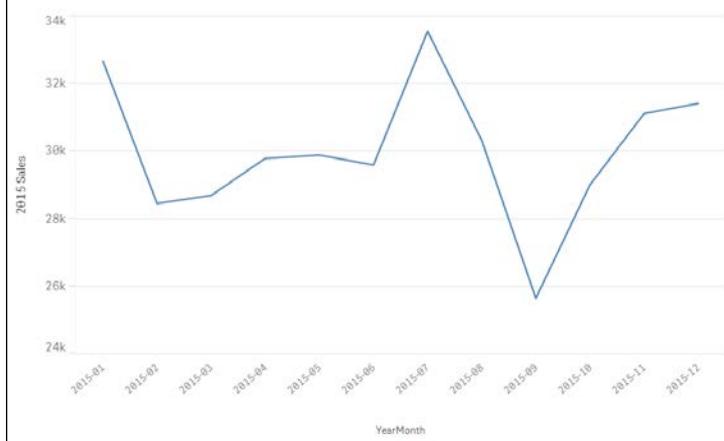
1. In the preceding Load statement, add the following line of code just below the DateID field:

```
if(Year(DateID)=2015,1,0) AS YearFlag_2015,
```

2. Reload the script.
3. Create a line chart with YearMonth as the dimension.
4. Add the following measure and label it as Sales:

```
sum({<YearFlag_2015={1}>}Sales)
```

5. Make sure that the sort-order for the months is maintained as Numeric and Ascending for the field YearMonth.
6. The graph should look like the following:



How it works...

The flag field that we set up simply adds an indicator against the records that fall within the rules we established. Now that we have identified the records from 2015, we can use this flag in the expression to calculate across those records.

The real world difference between using a flag to identify the records in 2015 and just using set analysis to identify the record directly will be almost nothing. The preceding code is a very simplified example of turning business logic into a flag indicator to be used later on. Once you have a grip on the concept and implementation, the same method can be used to add a level of complex and detailed business logic to a binary yes or no flag. Instead of writing the complex If-then-Else logic in the chart expressions, it is always advised to move it to the back end script and create flags. The flags are then used in the front end expressions thus making them more efficient.

Adding as much of the business logic to the script as possible makes everything much quicker and simpler to read. This way you don't have to look at each chart before making a change, you can make it in one place only and the change will propagate through the whole application.

Optimizing the reload time of the application

There are two methods of loading data from QVDs: optimized and non-optimized. The key point here is that the optimized loads can be up to 100 times quicker than the non-optimized loads.

This speed increase is a result of the data passing directly from the disk (QVD) into the memory (RAM) without being unpacked from its compressed QVD format.

As you may have guessed the reason every load is not optimized is because we often want to change the data coming out of the QVD. This requires it to be uncompressed before going into memory; hence, it is significantly slower. Just about any change to the data will cause the load to be non-optimized; however, there are a few things that we can do.

Getting ready

1. Open a new QlikSense application and go straight to the data load editor.
2. Create a folder library connection to any location where you want to save example data files and call that connection QVDs.

How to do it...

1. Copy the following code into the data load editor. Please note that if you are using a very low spec machine you can reduce the 20 million number on the third line to something smaller like 1 million:

ExampleData:

```
Load RecNo() AS TransID  
Autogenerate 20000000;
```

```
Store ExampleData into [lib://QVDs/Data1.qvd] (qvd);  
Drop Table ExampleData;
```

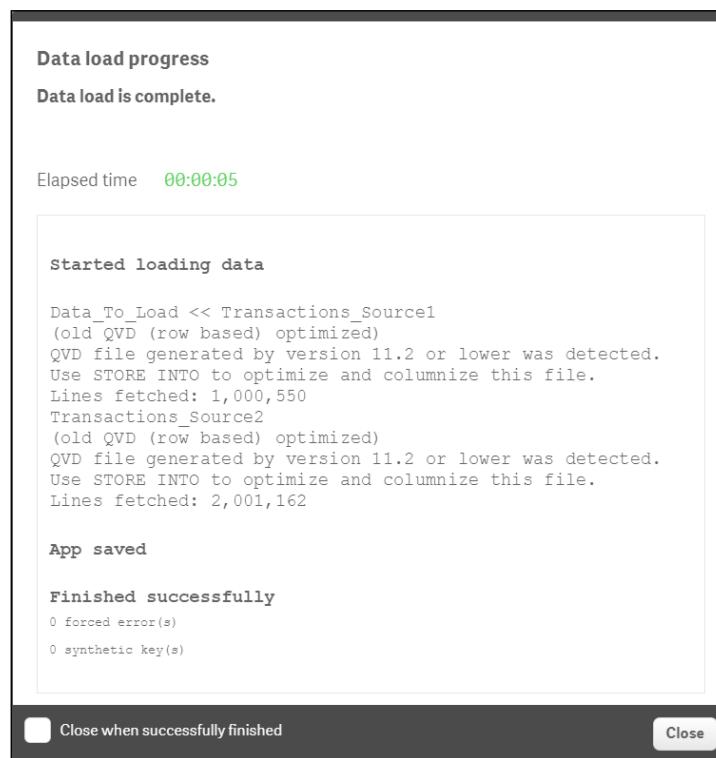
OptimizedLoad:

```
LOAD
TransID
FROM [lib://QVDs/Data1.qvd] (qvd);

Store OptimizedLoad Into [lib://QVDs/Data2.qvd] (qvd);
Drop Table OptimizedLoad;

UnoptimizedLoad:
LOAD
    'Example Text' AS NewField,
    TransID
FROM [lib://QVDs/Data2.qvd] (qvd)
Where Not IsNull(TransID);
```

2. Reload the application and make a note of the time it takes to load the records in each table:



How it works...

The first 20 million records loaded are simply auto-generated data records that we store in a `Data1.qvd` file to use later on. Now, we have a QVD available to read from, which we can use to demonstrate the difference between an optimized load and an un-optimized load. As a rule of thumb any data transformations on the QVD data in the script will cause the load to be un-optimized.

The second load of 20 million records simply reads the data from the `Data1.qvd` file (created in the preceding step) directly into memory and no further transformations take place. As no transformations take place in the `Load` statement, the load is an optimized load as stated in the Data progress window. We store the data loaded from this step into another QVD file called as `Data2.qvd`.

The third load is from the `Data2.qvd` file, the difference being that this time the script adds a `Where` clause and a new calculated field. Either of these transformations will cause Qlik Sense to use the unoptimized load method. Notice that the **Data progress** window does not specify "optimized load" even though we are loading the data from a QVD file.

You can think of optimized versus un-optimized loads as data being directly loaded into RAM for reading versus the unpacked data that is read line by line. A good exception to a `Where` clause that breaks the optimization rule is the `Exists()` function. Using `Where Exists(<Field>)` at the end of a load is a good method of loading the data that's relevant to what has been loaded previously.

Using a For Each loop to load data from multiple files

Often in a Qlik Sense application we need to load data from a directory which contains an identical set of data files. For example; sales for each country come in different files for each month. In such a case, we use a wildcard load, in order to fetch the data for our application. The following recipe discusses the data modeling issues encountered when using the wildcard load and how we make use of the `For each` loop structure in the script to overcome this issue.

Getting ready

For this exercise we will make use of two sample XLSX files, namely, `Apr2015.xlsx` and `May2015.xlsx` that contain mock sales data for six countries. These files can be downloaded from the Packt Publishing website.

How to do it...

1. Once the source files are downloaded, store them in a folder called `ForEachLoadData`.
2. Create a folder connection as explained in *Chapter 1, Getting Started with the Data* that points to the `ForEachLoadData` folder. Name the connection as `QlikSenseCookBookForEachLoadData`.
3. Select any file from the folder and extract its contents in the Qlik Sense application.
4. Next modify the script as the following example, to get the data from all files that reside in the `ForEachLoadData` folder. Note that we are using a wildcard `*` in place of the filename in the `from` statement. The `Filebasename()` function gets the filename so that we can identify the origin of the data:

```
CountrySales:  
    Load  
        Filebasename () AS Source, Country, Sales  
    FROM [lib://QlikSenseCookBookForEachLoadData/*.xlsx]  
        (ooxml, embedded labels, table is Sheet1);
```

5. Add the preceding load to the script. (the preceding `Load` is placed directly above the `Load` statement of the `CountrySales` table.):

```
    LOAD*,  
        Left(Source,3) as Month;
```

6. Upon loading, we observe that a synthetic key has been created in the data model.
7. In order to avoid the synthetic key, we will make use of the `For each` loop along with the wildcard load.
8. Modify the block of code to start with a `For Each` loop statement and end with a `Next`, as shown in the following code.

```
For each vFile in FileList  
    ('lib://QlikSenseCookBookForEachLoadData/*.xlsx')  
        CountrySales:  
            LOAD *,Left(Source,3) as Month;  
  
            Load  
                Country,  
                Sales,  
                Filebasename() as Source  
            from [$(vFile)]  
                (ooxml, embedded labels, table is Sheet1);  
        Next vFile
```

9. Once the script is in place, save and reload the application again.
10. We observe that all the files from the folder have been reloaded properly and there is no synthetic key in the data model.

How it works...

The * wildcard character loads all the files from the `ForEachLoadData` folder into the Qlik Sense application. When we use a preceding `load` statement to generate the `Month` field, the load is only applied to the first file loaded from the folder; hence, the `Month` field is created only for the first file. This is the reason why a synthetic key is created between the two tables.

When we use the `For` loop, every file is sequentially loaded from the source folder and then a preceding `load` statement is applied; thus, creating a month field in each created table. The two tables are then auto-concatenated, as they contain the same number of fields with the same name. As a result, a synthetic key is avoided and we get a clean data model.

There's more...

We used iteration or the `For Each` loop in the above recipe outside the `Load` statement. We can also have iterations inside the `Load` statement using the `Where` clause or the `Subfield` function. Iterations are also possible using the `Peek()` function. A useful article from *Henric Cronstrom* on Iterations can be accessed using the following URL:

<https://community.qlik.com/blogs/qlikviewdesignblog/2013/09/02/loops-in-the-script>

Using the Concat function to store multiple field values in a single cell

The information in orders and invoices is typically stored at the header or line level in the database. However, when we display the sales value for a particular order on the UI, it is sometimes desired that all the products for an order are displayed in a single cell rather than on a separate line. The `Concat` function is helpful in such a case.

Getting ready

For this recipe we will make use of an inline data load which gives sales information for orders. Load the following order line information in Qlik Sense:

```
Orders:  
LOAD * INLINE [  
OrderID,Product, ProductID, Sales
```

```
101,Footwear, F21Lon, 120000
101,Tyres, T21Man, 150000
101,Mountain Bikes, MB32Lon, 195000
102,Road Bikes, RB12Bir, 225000
102,Chains, F21Lon, 140000
103,lubricant, T21Man, 56869
103,Mountain Bikes, MB32Lon, 195000
104,Road Bikes, RB12Bir, 65233
];
LEFT JOIN
LOAD OrderID, CONCAT(Product,',') as Products
Resident
Orders
GROUP BY OrderID;
```

How to do it...

1. Create a Table Chart.
2. Add OrderID as the first dimension.
3. Add Products as the second dimension.
4. Add Sum(Sales) as the measure. Label it Sales.
5. The resultant table should look like the following:

| Orders | | Products | Sales |
|--------|---------|-------------------------------|---------|
| | OrderID | Q | Q |
| Totals | | | 1147102 |
| | 101 | Footwear,Mountain Bikes,Tyres | 465000 |
| | 102 | Chains,Road Bikes | 365000 |
| | 103 | Mountain Bikes,lubricant | 251869 |
| | 104 | Road Bikes | 65233 |

How it works...

The CONCAT function in the script is used to string together multiple product values in one single string separated by a specified delimiter. The CONCAT function is an aggregation function and would require a Group By clause after the from statement.

There's more...

The CONCAT function can also be used in the frontend instead of the script. In this case, we will have to create a calculated dimension, as follows:

```
=AGGR(Concat(DISTINCT Product, ',') , OrderID)
```

Name it as Products. As mentioned earlier, being an aggregation function, CONCAT requires an AGGR that is a substitute of Group By used in the script.

See also

The *Chapter 5, Useful Functions* chapter discusses some cool utilization of functions within Qlik Sense.

4

Managing Apps and User Interface

In this chapter, we will be dealing with the User Interface in Qlik Sense.

- ▶ Publishing a Qlik Sense® application on Qlik Sense® desktop
- ▶ Creating private, approved and community sheets
- ▶ Publishing a Qlik Sense® application on Qlik Sense® cloud
- ▶ Creating geo maps in Qlik Sense®
- ▶ Effectively using the KPI object in Qlik Sense®
- ▶ Creating Tree Maps
- ▶ Creating a Sales versus Target gauge chart in Qlik Sense®
- ▶ Creating dimensionless bar charts in Qlik Sense®
- ▶ Adding Reference Lines to trendline charts
- ▶ Creating text and images
- ▶ Applying limitations to charts
- ▶ Adding thumbnails – a clear environment
- ▶ Navigating many data points in a scatter chart

Introduction

The information required for analysis and decision making within an organization is communicated via the user interface in Qlik Sense. We discussed the best design practices in *Chapter 2, Visualization*. We will take this discussion a step further and learn to implement a few key objects found in Qlik Sense in our applications, which will help the business convey the desired information to the end user in an effective manner. The chapter starts with important concepts in managing the Qlik Sense applications, such as publishing the apps on the server and on the Qlik Sense cloud. In later parts, the chapter deals with topics such as "Use of Reference Lines" and "Navigating Data Points in Scatter Chart" in a Qlik Sense application.

Publishing a Qlik Sense® application created in Qlik Sense® desktop

The licensing model of Qlik would not be very useful if everyone used Qlik Sense desktop only for themselves. In a published BI environment, simply creating an application in Qlik Sense Desktop will not suffice. It has to be made available to the end user. The application needs to be published via the Qlik Sense management console.

Getting ready

Install Qlik Sense Server 2.1.1. The steps to install the Sense Server can be found under the Deploy section in the left panel on the Qlik Sense help website:

<https://help.qlik.com/sense/2.1/>

The Qlik Sense Installer file can be obtained from www.qlik.com. You need to login using the customer account credentials to get access to the files under **Support | Customer Downloads**.

How to do it...

Any Qlik Sense application that is created in Qlik Sense desktop needs to be imported using the Qlik Sense management console prior to its publishing.

1. To do this, open the Qlik Sense QMC through the windows shortcut or use the following URL:
<https://<Qlik Sense Server Hostname>/QMC>
2. In the QMC, click on **Apps** in the left pane and go to the **Apps** section.
3. Click on the **Import** button in the top right hand corner.

4. Click on **Choose File** and select the required application to be uploaded and press **Import**. Once imported, select the app and click **Publish** in the action bar.
5. You will be prompted to specify a stream for the application. Choose a stream from the defined streams in the dropdown menu.

How it works...

Publishing an application is the first step towards sharing the application with a wider set of end users. Once published the layout of the application cannot be changed. Also, the publishing of the application cannot be undone and you will have to delete the application to remove it from the stream. A better approach to handle such a situation is to duplicate the application without publishing it and make the desired changes to the duplicate application. We can then use the option of **Replace existing app** to replace a published app.

There's more...

Sheets and bookmarks can be published and categorized as private, approved, or community. This will be discussed in the next recipe.

Creating private, approved, and community sheets

Sheets are the key components of a Qlik Sense application. They contain all the objects that carry information and provide framework for analysis. There are three types of sheet that can be defined in a Qlik Sense application. These are the private, approved, and community sheets.

- ▶ Approved sheets are all sheets that are defined by the author of the application. These cannot be changed by the user and are defined as read only.
- ▶ Private sheets can be viewed only by the author of the application. These are not yet published for access by the end user.
- ▶ Community sheets are also private sheets but are defined and published by a user other than the author who has been granted access to the application on the hub.

Getting ready

Sheets can be defined as private, approved or community once the application has been imported to the Qlik Sense Management Console to be published and made available to the end users.

How to do it...

1. Once the application is published, all the sheets in the application become "Approved" sheets; approved sheets are read only.
2. The "Approved" Sheet cannot be modified unless duplicated as a "Private" sheet. As the name suggests, Private sheets are private to the author. In order to duplicate a sheet, right click on the sheet and select **Duplicate Sheet** option.
3. Once the Sheet is made "Private", modifications can be made to the relevant sheet if required. The sheet can then be published by right clicking and selecting **Publish Sheet**.
4. "Private" sheets can also be created by "Creating a new Sheet" in the published application.
5. Sheets can be created by other users and published to the hub. Sheets published in such a way are categorized as "Community" sheets.

How it works...

Only the published sheets can be accessed by the end users. Sheets kept as private or approved are not shared until published; hence, they add a layer of security. The concept of community sheets brings in the collaborative feature, wherein the other users can contribute and share objects and reports created by them.

There's more...

On a similar note, one can create private, approved, and community bookmarks in Qlik Sense. The idea and approach remain similar.

See also

- ▶ *Publishing a Qlik Sense® application created in Qlik Sense® desktop*

Publishing a Qlik Sense® application to Qlik Sense® cloud

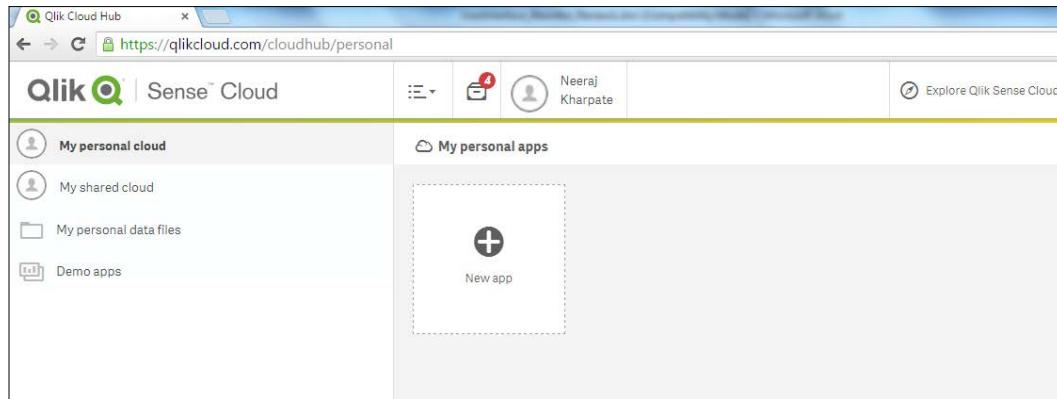
Qlik has come up with this wonderful concept of sharing Qlik Sense applications in the cloud. The author can share applications on the cloud with up to five people by sending an e-mail invitation. These applications can be viewed on any mobile device and on any web browser. In a small implementation, Qlik Sense cloud can be particularly helpful as one can share the applications over the web without installing the Sense Server.

Getting ready

Create a Qlik Sense Cloud account at <https://qlikcloud.com/login>.

How to do it...

1. The Qlik Sense Cloud web page appears like the following screenshot:

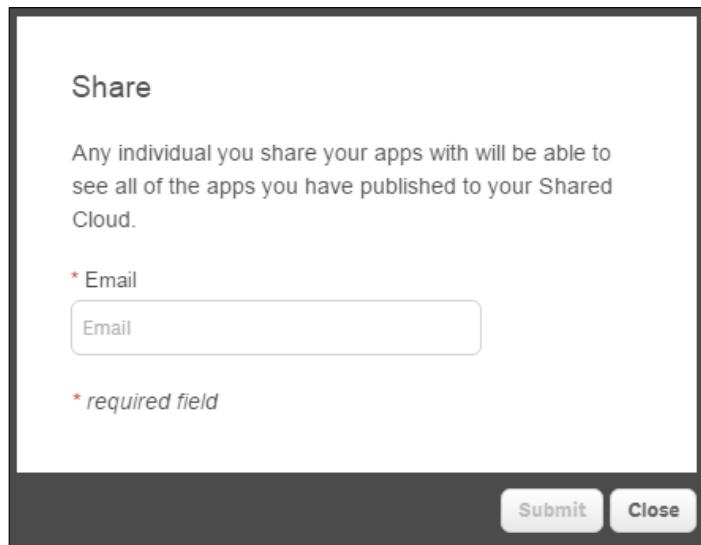


2. Under **My personal cloud**, Click on the button to import the desired Qlik Sense application to the cloud:

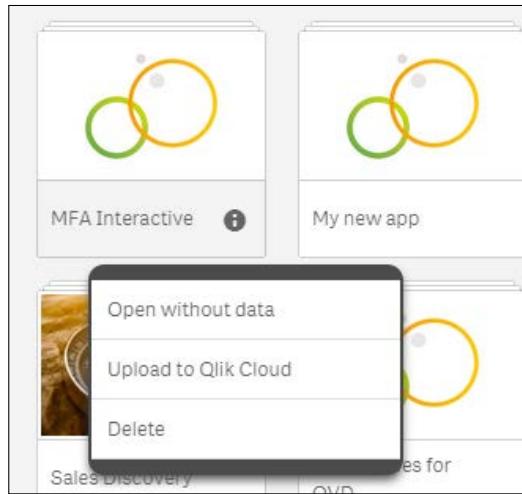
The dialog box is titled 'Create a new app in the cloud'. It contains a 'Title' input field and a 'Create app' button. A note below states: 'You must upload data files in the My personal data files section of the Qlik Sense Cloud hub to access your data within your apps'. There is a horizontal line with 'Or' in the center, followed by 'Upload an app'. Below this, it says 'Your apps can be found at the following location: C:\Users\<your_user_name>\Documents\Qlik\Sense\Apps\<file_name>'. It includes 'Choose file' and 'Import' buttons. At the bottom, it lists 'Upload requirements':

- You must close the app before importing it into Qlik Sense Cloud.
- An app may be up to 25 MB.
- Your total storage space is 250 MB.

3. Under **Upload an app**, click on the **Choose file** button to select the Qlik Sense application. The Qlik Sense applications are by default stored under:
`'C:\Users\<*your own user folder*>\Documents\Qlik\Sense\Apps'`
4. Select the desired application and click on the **Import** button.
5. At this stage, the application can be published to the shared stream by checking the **Publish this app to my shared stream** check box.
6. If the application is not published to the shared stream, it will appear in the **Personal** cloud on successful upload.
7. If the application is published on the shared stream, it will appear in the **My Shared Cloud** on a successful upload.
8. Click on **My Shared Cloud**. Here you will find the recently uploaded Qlik Sense applications.
9. Click on the **Share 5** button at the bottom of the window and the following window will pop up:



10. Enter the e-mail address of the recipient. The recipient will receive an e-mail with a link to create a Qlik Cloud account.
11. One can open Qlik Cloud while working on the Sense Desktop version by right-clicking on any application and clicking on **Upload to Qlik Cloud**. However, this link does not actually upload the document to the cloud, it only opens the Qlik Sense cloud with the correct dialog:



How it works...

The applications that are published to the cloud can be viewed by any recipient who has been provided with the shared link. Once the recipient registers on Qlik Sense, the author of the application will receive a notification that he has a new follower.

As the information is shared with different users within the organization and outside, it provides the user with a great collaboration feature. Any application in the **My Shared Cloud** area is considered to be shared and can be seen by all followers. It is not possible to share only one application with a particular user. The applications can be unpublished and can be moved to the personal cloud by right clicking and selecting the **Unpublish** option. As of now, there are certain restrictions in using the cloud:

- ▶ The maximum size of the application to be uploaded can be 25 MB.
- ▶ Images and extensions cannot be uploaded to the cloud.
- ▶ While creating new applications directly in the cloud, we can load data only through the files uploaded to the **My personal data files** section or through DataMarket.
- ▶ The followers are not notified if a new application is published to the shared stream.
- ▶ The applications on the cloud can be shared with a maximum of five followers.

There's more...

Qlik Cloud has the functionality of a personal cloud wherein the author can have his or her own private applications, which are not published to the outside recipients.

Personal data files can be added under the  My personal data files option. Once the files are added under personal data, they can be used to create a new Qlik Sense application in the cloud. The process to create a new Qlik Sense application directly in the cloud is as follows:



1. Select **My Personal cloud** and then click on the  button.
2. Add a title to the new application.
3. Click on  Create app. This will open the Qlik Sense application in a browser.
4. Open the data load editor to enter the script. If personal data files are uploaded on the cloud, you will notice that a data connection to these files is automatically created within the data load editor.
5. One can also make use of DataMarket to upload data to the application.
6. Add the required data to the script and reload the application.

Creating geo maps in Qlik Sense®

Geographical information can be plotted in Qlik Sense by making use of the Map object. In order to create geo maps in Qlik Sense, we need to load the location information also called point or area data. The location information can be loaded either from a **Keyhole Markup Language (KML)** file, if available, or a database, web service, or from a simple Excel file. Data can also be loaded inline, which is what we are going to do in our following recipe.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives us the location information for different countries in the form of latitudes and longitudes:

1. Create a new Qlik Sense file and name it Geolocations.
2. Add the following **Inline** table that contains the location information for countries:

Country:

```
Load RowNo() As CountryID, *,GeoMakePoint(Latitude,  
Longitude) As CountryGeoPoint Inline [  
Country, Latitude, Longitude  
Australia, -25.274398,133.775136  
Argentina, -38.416097,-63.616672  
India, 20.593684,78.962880
```

```
China, 35.861660,104.195397
Colombia, 4.570868,-74.297333 Great
Britain,55.378051,-3.435973
Switzerland ,46.8181887,8.227512
Netherlands,52.132633,5.291266
Salvador,13.794185,-88.896530
Italy,41.871940,12.567380
Peru,-9.189967,-75.015152
];
```

3. Next, add the following **Inline table** that contains the **Sales** information for each country:

Sales:

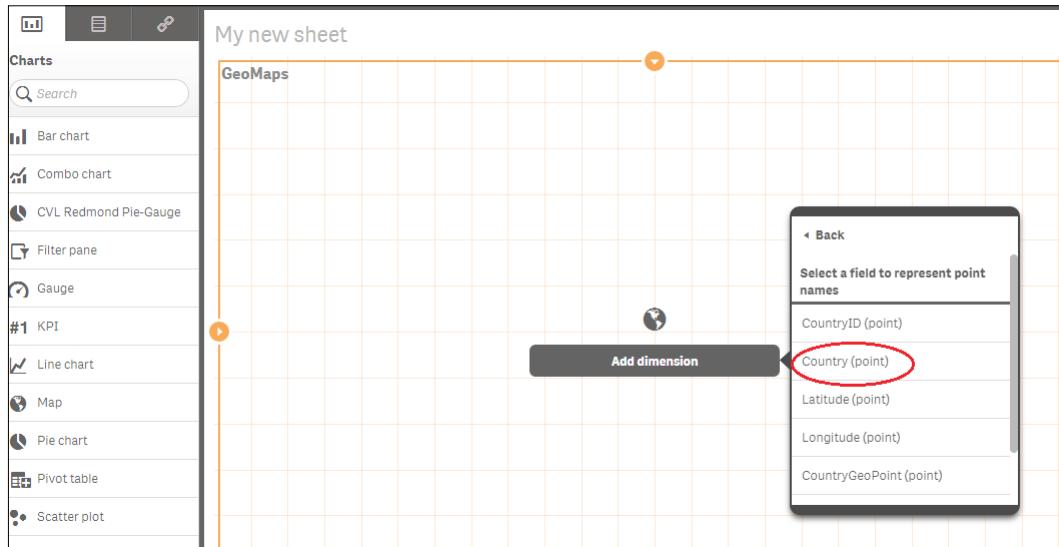
```
Load * Inline [
    Country, Region, Sales
    Australia,Australia,133775
    Argentina, Latam,6361672
    India, APAC,7896880
    China, APAC,10419397
    Colombia, Latam,742333
    Great Britain,EMEA,3590073
    Switzerland ,EMEA,8227512
    Netherlands,EMEA,521266
    Salvador,Latam,8886530
    Italy,EMEA,12567807
    Peru,Latam,750152
];
```

4. Load the data and save the file. Open the **App overview** by clicking on the Navigation dropdown  in the top left corner.

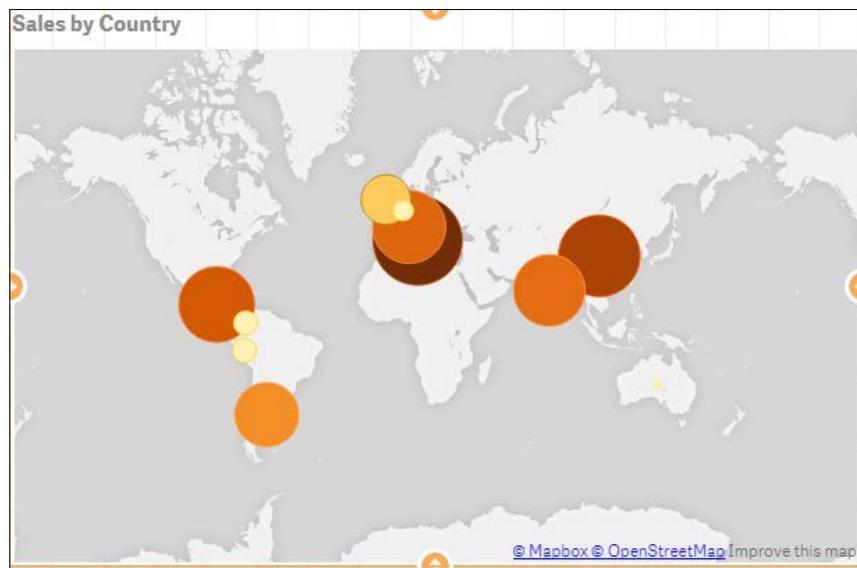
How to do it...

1. Create a new sheet in the Qlik Sense application.
2. Enter the Edit sheet mode and drag across the Map object from the left-hand side Assets panel on to the sheet. Name it **Sales by Country**.

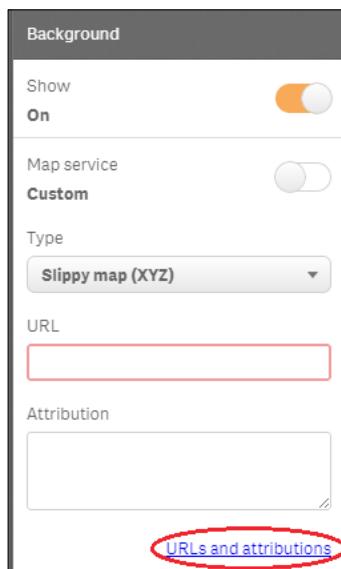
- Click on **Add Dimension** and then select **CountryGeoPoint**. Select **Country** to represent the point name:



- In the Properties panel to the right of your screen, add **Sum(Sales)** as your expression under data.
- The resulting map on the screen will look like the following. The map automatically picks up the Mapbox background in its point layer. The only available image type is slippy map:



6. Next, we will change the background of the map.
7. To do this, open the Properties panel and click on **Background**. The **Background** window's **Show** property is by default set to **On**. The **URL** and **Attribution** boxes get activated when you switch the **Map service** from **Auto** to **Custom**.
8. Click on the **URLs and attributions** hyperlink at the bottom of the **Background** window:



9. From the available list of slippy map servers, copy a URL and paste it into the **URL** text field.
10. Next, copy the attribution string for the chosen URL and paste the string into the **Attribution** text-field.
11. For the sake of our exercise, we will use the **URL** and **Attribution** string for MapQuest-OSM.

12. Click on  Done. The final map will look like the following screenshot:



13. When we hover over any bubble, the tool tip shows the relevant country and sales.

How it works...

While loading the script for this recipe, we made the use of the `GeoMakePoint()` function. This function creates and tags a point with its latitude and longitude information. Each country is thus linked to point data, which is plotted on the map. When we use a KML file as a source, Qlik Sense automatically detects the Geopoint field; therefore, there is no need to use a special function to define the same.

While changing the background for the map, we insert the required URL that connects to the tile server that we want to use. For copyright reasons, the attribution string should correspond to the desired URL.

There's more...

Point data can also be read from Excel files:

1. If the point data is stored in a single column called `Location`, that is, if each point is specified as an array of x and y coordinates and represented as `[x, y]`. Here `x=longitude` and `y=latitude` then:
 - The geospatial coordinates, namely, the latitudes and longitudes should be tagged with `$Geopoint` so that the field `Location` is recognized as a point data field.

- ❑ As an example, consider that the Location data is being extracted from a file called Country.xls having three columns Country, Location, and Sales; where Location contains the point data. The script in such a case will look like the following:

```
LOAD
Country,
Location,
Sales
FROM 'lib:///Country.xls' (biff, embedded labels, table
is (Sheet1$));

Tag Field Location with $Geopoint;
```

- ❑ Run the script and add the point dimension to the map.
2. If the point data is stored in two columns, that is, one for latitude and the other for longitude then:
 - ❑ The GeoMakePoint() function should be used to generate point based data
 3. Similarly, one can make use of the KML files, which contain point data, area data, or both in order to create maps in Qlik Sense. The following URL explains the process of generating maps in Qlik Sense using the KML files:
 - ❑ <https://community.qlik.com/docs/DOC-7354>

Reference lines in Sales versus Target gauge chart

Recently, while delivering a proof of concept, I was asked by a customer if we could create a "Stephen Few Bullet chart" in Sense. This is not possible out of the box because of the simple reason being that the bullet chart involves overlaying a bar chart on top of a gauge chart and overlaying objects in Sense is not allowed. So, I thought of delivering the same result using just the gauge chart and making use of reference lines.

Getting ready

Load the following script in the Qlik Sense data load editor, it gives information about the Sales and Target values for four countries:

```
LOAD * INLINE [
Country, Sales, Target
USA, 10000, 8500
UK, 7000, 9500
```

```
Germany, 5000, 4500  
Japan, 6000, 6000  
];
```

How to do it...

1. Drag across a gauge chart object onto the sheet from the Assets panel on the left.
2. Click on **Add measure** and type in the following expression. Label it as Sales vs Target:
 $=Sum(Target)$
3. Under **Add-Ons**, click on the **Reference lines** and add a reference line expression with the following definition:
 $=Sum(Sales)$.
4. Label the reference line expression as Sales. Change the color of the reference line to red by clicking on the color dropdown.
5. Under **Appearance**, click on **Presentation** and set the **Max range Limit** as:
 $=Sum(Target) * 1.2$
6. Select the representation as **Bar** and orientation as **horizontal**.
7. Check the **Use Segments** box.
8. Next, click on the **Add limit** button and add the following limit:
 - Segment 1:
 $=Sum(Target) * 0.30$
Click on the segment area to change the default color to Grey.
9. Again, click on the **Add limit** button to create a second segment with the following limit:
 - Segment 2:
 $=Sum(Target) * 0.60$
Click on the segment area to change the default color to Red.
10. Finally, click on the **Add limit** button to create a third-segment with the following limit:
 - Segment 3:
 $=Sum(Target)$
Click on the segment area to change the default color to Yellow.

11. Select the color of the last segment as Green. Check the **Gradient** box for the last segment.

12. Click on  Done when finished.

13. The resulting chart will look similar to the following screenshot:



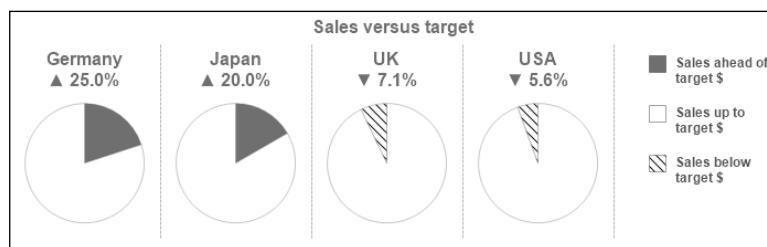
How it works...

The color segments signify how the sales of a particular country are performing, as compared to the target values. The red reference line indicates the sales value. The color red does not signify anything else and is used only to highlight sales. Hence, you can use any other color of your choice.

The sales can be more than the set target. Hence the **Max range Limit** is set to 1.2 times the target value. Due to this setting, the target value is represented by the black line at the end of the bar. Surely, sales can surpass the targets by more than 20 percent. So, the **Max range figure** can be altered by say to 1.5 times of the target value. One look at the graph and we can easily make out if we are in the red zone or are doing better than expected.

There's more...

A similar concept is explored in the Capventis Redmond Pie-Gauge Chart designed by Stephen Redmond. The gauge in this object is more of a modified bullet chart. If the sales are more than the target then good performance is shown as a shaded sector to the right of the vertical, while if the performance is below par it is shown as a shaded sector to the left of the vertical, as shown in the following figure:



The Capventis Redmond Pie-Gauge can be downloaded from Qlik Branch at <http://branch.qlik.com/projects/showthread.php?159-CapVentis-Redmond-Pie-Gauge-for-Qlik-Sense>

See also

- ▶ [Creating Tree Maps](#)

Effectively using the KPI object in Qlik Sense®

A visualization should provide the user with a careful and effective presentation of the data. Numbers have an impact value and they contain a message. Key performance indicators demonstrate the importance of numbers in business and also communicate the health of the business to the audience.

Getting ready

We will make use of the application from the preceding recipe. The application has the following script loaded, it gives information on the Sales and Target values for four countries:

```
LOAD * INLINE [  
    Country, Sales, Target  
    USA, 10000, 8500  
    UK, 7000, 9500  
    Germany, 5000, 4500  
    Japan, 6000, 6000  
];
```

How to do it...

1. Go to the **App overview** and create a new sheet.
2. Name the sheet as **KPI** and open it.
3. Go to the Edit mode by clicking on  .
4. Drag across the **#1 KPI** object from the Assets panel on to the sheet.
5. Next, add the following measure:
 $(\text{Sum}(\text{Sales}) - \text{Sum}(\text{Target})) / \text{Sum}(\text{Target})$

6. Name the label as Sales vs Target.
7. Once we add the measure, we can see a host of properties, such as number format, color, and so on, for the measure directly beneath the Expression editor box.
8. For the measure, change the number formatting to **Number** and select the percentage format (**12.3%**) from the available formats under the dropdown.
9. Next, add the limits to define colors. Switch on Conditional colors.
10. Click on **Add limit** and set the limit under function as **0**.
11. Click on the first segment of the color bar and select the color as red with a  symbol .Click on the second segment of the color bar and select the color as green with a  symbol.
12. The KPI object appears as the following:



13. Under **Appearance**, click on **General** and Switch on **Show titles** and name the title as Sales vs Target.
14. Name the subtitle as:
`IF(getselectedcount(Country)>0,Country,'')`
15. Next, go to the **Presentation** dropdown and uncheck **Show Title**.
16. Add the Filter pane object from the Assets panel on the sheet and select the dimension as **Country**. Select different countries to see how your organization is faring with respect to each country:



The screenshot shows a Power BI report. On the left, there is a KPI card with the title "Sales vs Target" and the value "11.1%" in green with an upward-pointing triangle symbol. On the right, there is a "Country" filter pane with a search icon. The list of countries includes Germany (which is selected, indicated by a green background and a checkmark), Japan, UK, and USA.

17. Next, we will link our KPI object to a sheet that shows detailed reports.

18. Create a new sheet called **Reports**.
19. Create a Table report on the reports sheet with **Country** as the dimension and the following measures:
 - Sum(Sales) : Label it **Sales**.
 - Sum(Target) : Label it **Target**.
 - (Sum(Sales) - Sum(Target)) / Sum(Target) : Label it **Sales vs Target**. For the measure, change the number formatting to number and select the percentage format (**12.3%**) from the available formats under dropdown.
20. Move back to the KPI sheet and enter the Edit mode by clicking on  **Edit**. Select the KPI object. This will activate the Properties panel on the right.
21. Under **Appearance**, go to **Presentation** and switch on **Link to Sheet**. Under **Select a sheet**, select the **Reports** sheet and click on  **Done**.
22. When we click on the KPI object on the user interface, it directs us to the reports sheet where you can analyze all the sales and target figures for each country:



How it works...

The KPI object is an important visualization object on any dashboard. The color segments we defined in the properties determine if the country is doing better than its set target value or not. If the sales are below the target values then the KPI figure is shown in red or else in green. Linking the KPI to the **Reports** sheets helps the user to dig deep into the data and see the more granular figures.

There's more...

The KPI object can also be represented using two measures. We can show a comparison between key figures in a single KPI object. For example, the absolute sales and target values can be shown adjacent to each other as separate figures. If the sales are greater than the target then the value is represented in a green color or else in a red color.

This can be achieved by following the steps mentioned in the following steps:

1. Create a new KPI object by following the steps given in the previous recipe. Label the object as Sales vs Target-1.
2. Add the following measures:
 - Sum(Sales): Label it Sales
 - Sum(Target): Label it Target
3. For Sales switch on the conditional colors.
4. Click on **Add limit**.
5. Set the limit under function as:
`=Sum(Target)`
6. Select the first color as red with a  symbol and the second as green with a  symbol.
7. For Target, select the font color as **Blue**
8. The resultant object will be similar to the following:



See also

- ▶ *Creating text and images*

Creating Tree Maps

The tree maps (previously called block charts in Qlikview) are a good way to show how different parts combine to form the whole. To add more depth to the visualization, you can easily highlight areas of importance by adding color codes.

Getting ready

For this recipe, we will make use on inline data load which gives the product sales information. Load the following code into the data load editor:

```
LOAD * INLINE [
    Product Line, Product Group, Product Sub Group, Year, Sales,
    Cost
    Drink, Beverages, Juice, 2015, 12000, 6000
    Drink, Beverages, Juice, 2014, 16000, 7000
    Drink, Beverages, Soda, 2015, 42000, 26000
    Drink, Beverages, Soda, 2014, 68000, 57000
    Drink, Beverages, Water, 2015, 18000, 8000
    Drink, Beverages, Water, 2014, 10000, 6000
    Drink, Dairy, Milk, 2015, 25000, 22000
    Drink, Dairy, Milk, 2014, 22000, 20000
    Food, Dairy, Cheese, 2015, 22000, 8000
    Food, Dairy, Cheese, 2014, 31000, 30000
    Food, Produce, Nuts, 2015, 50000, 30000
    Food, Produce, Nuts, 2014, 46000, 26000
    Food, Produce, Tofu, 2015, 26000, 21000
    Food, Produce, Tofu, 2014, 15000, 7000
    Food, Snacks, Chips, 2015, 31000, 6000
    Food, Snacks, Chips, 2014, 15000, 9000
    Food, Snacks, Dips, 2015, 10000, 6000
    Food, Snacks, Dips, 2014, 6000, 3000
];
;
```

How to do it...

1. Drag a Tree Map object onto the content page.
2. Add **Product line** as a dimension.
3. Add **Product Group** as a dimension.
4. Add **Product Sub Group** as a dimension.
5. Add Sum(Sales) as a measure and label it Sales.
6. From the Properties panel on the right-hand side of the screen under **Appearance | Colors**, toggle the option from **Auto** to **Custom**.
7. In the drop-down window, select **By Expression** and enter the following expression:
`If(Sum({<Year={2014}>}Sales)>Sum({<Year={2015}>}Sales),
 Red(),Green())`
8. Click on **Done**.

9. The finished result should resemble the following picture:



How it works...

The tree map object groups the data based on the order of the dimensions you added. By adding the color coding expression we can quickly see the products that are doing better this month compared to the previous month.

There's more...

The red and green indicators used in the preceding image can be useful to spot products that are not performing in-line with similar products. To get more value from these type of indicators, we can change the density of the color to reflect the magnitude of change.

Replace the color expression we used in step 7 of *How to do it...* with the following code:

```
If((Sum({<Year={2015}>}Sales) -  
    Sum({<Year={2014}>}Sales))/Sum({<Year={2015}>}Sales)>0,  
    ColorMix1((Sum({<Year={2015}>}Sales) -  
    Sum({<Year={2014}>}Sales))/Sum({<Year={2015}>}Sales),  
    white(),RGB(50,255,50)),  
    if((Sum({<Year={2015}>}Sales) -  
    Sum({<Year={2014}>}Sales))/Sum({<Year={2015}>}Sales)<0,  
    ColorMix1(fabs((Sum({<Year={2015}>}Sales) -  
    Sum({<Year={2014}>}Sales))/Sum({<Year={2015}>}Sales)),  
    white(),RGB(255,50,50))))
```

The chart should now resemble the following image:



Based on the values returned by the expression, the `ColorMix` function automatically assigns a range of colors. In the preceding example, we have set up two color ranges; the first `If` statement goes from white to green for the positive numbers and the second goes from white to red for the negative numbers. The `ColourMix` function only works with positive numbers, so we use the `Fabs` function to convert the negatives into positives once they are identified by the second `If` statement.

Creating dimensionless bar charts in Qlik Sense®

A bar chart is usually defined with one or two dimensions and a measure. However, we need to have dimensionless bar charts while designing KPIs on the dashboards and also in certain other scenarios. By default, Qlik Sense will not allow this. However, there is a workaround that is discussed in the following sections.

Getting ready

We will make use of the same application that we developed for the KPI recipe. The application has got the following script loaded, which gives information on the Sales and Target values for four countries. In addition, we will add a new column called as `Dummy`.

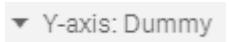
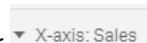
Make sure to save and load the script once the `Dummy` field is added:

```
LOAD * , 1 as Dummy INLINE [
Country, Sales, Target
USA, 10000, 8500
UK, 7000, 9500
```

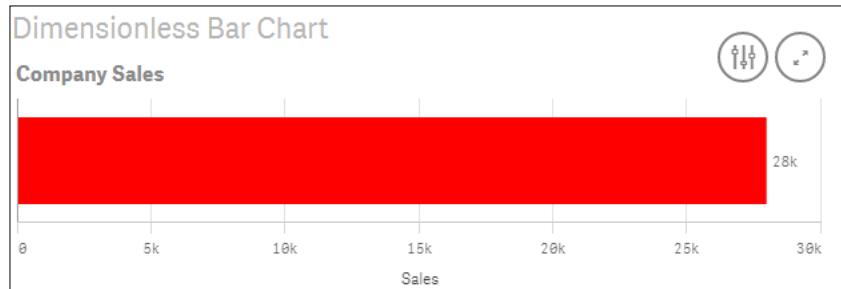
```
Germany, 5000, 4500  
Japan, 6000, 6000  
];
```

We want to display the overall sales for the company and change the color of the bar based on the threshold value.

How to do it...

1. Go to the **App overview** and create a new sheet.
2. Name the sheet as Dimensionless Bar Chart and open it.
3. Go to the Edit mode by clicking on  Edit .
4. From the Assets panel, drag across the  Bar chart object on the sheet.
5. Go to the **Master Items**  in the Assets panel and create a dimension with the name Dummy as shown:
`=ValueList('Dummy')`
6. In the chart under **Dimensions**, use the just created master dimension **Dummy** as the dimension.
7. Add the measure as **Sum(Sales)** and label it as Sales.
8. Under **Appearance**, click on **Presentation** and make the chart as **Horizontal** and switch on the **Value Labels**.
9. Under **Appearance**, click on **General** and add Company Sales as the **chart Title** under the **General** properties.
10. Under **Appearance**, click on **Colors and legend**. Switch off auto colors and select **By expression** under the drop-down menu.
11. Add the following color code expression:
`If (Sum(Sales) > Sum(Target), RGB(0, 255, 0), RGB(255, 0, 0))`
12. Make sure that **The expression is a color code** is checked.
13. Go to the labels and click on the **Title** option under  Y-axis: Dummy and select **None**.
14. Under  X-axis: Sales , switch off the **Auto** range and select **Min/max** under **custom**. Set the min value to **0** and the max value to **30000**.

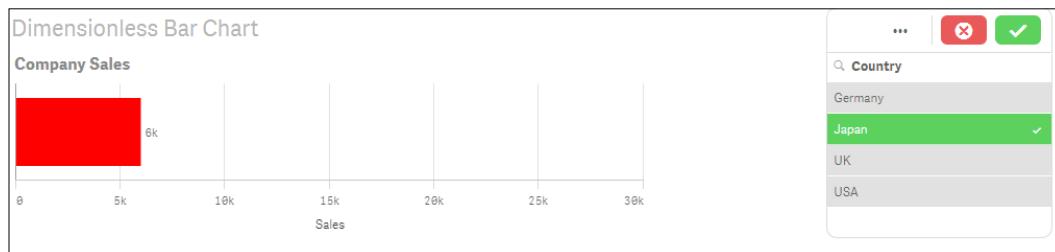
15. The final chart looks like the following:



16. Create a Filter pane object with **Country** as the dimension. Now select different countries and view the results.

How it works...

Qlik Sense doesn't allow dimensionless bar charts. So, we need to create a **Dummy** dimension that has only one single field value. Further when we select **none** under the **labels and title** option, it hides this field value from the axis, thus serving the purpose. The color code used for the bars will turn the bar red if the sales for a country are less than or equal to the target values:



There's more...

We can use the **Dummy** dimension directly from our source data instead of creating a master dimension in the frontend. Another approach is to use a calculated dimension =1 and name it **Dummy**. All the approaches will yield the same result. To make the chart more informative one can add reference lines for the target.

See also

- ▶ [Adding Reference Lines to trendline charts](#)

Adding Reference Lines to trendline charts

One cannot overstate the importance of adding context to analysis. Take the example of having the headline number **Average Call Time** displayed on a dashboard. While this might clearly be an important metric for a call center, but on its own it portrays very little. As shown in the *Dimensionless bar chart* recipe in the preceding section, we used reference lines to add the context required to make the number meaningful. Sticking to the example of **Average Call Time**, we may also want to see alongside; a previous point in times position, the national or a competitor's average, the internal target, and so on. This recipe extends the use of reference lines further.

Getting ready

For this recipe, we will make use on inline data load which gives us the call bounce rates for different periods. Add the following code into the data load editor and reload the Qlik Sense application:

```
WebStats:  
LOAD * INLINE [  
    Period, BounceRate  
    1, 0.26  
    2, 0.25  
    3, 0.24  
    4, 0.24  
    5, 0.27  
    6, 0.28  
    7, 0.21  
    8, 0.34  
    9, 0.24  
    10, 0.25  
];
```

How to do it...

1. Add a line chart object onto the content page.
2. Add **Period** as a dimension.
3. Add **AVG(BounceRate)** as a measure.
4. From the Properties panel | under **Add-ons** click on the **Reference lines** button and then on **Add reference line**

5. Set the **Label** as Upper Threshold and set the **Reference line** expression to the following:

```
=Avg(BounceRate)+Stdev(Total Aggr( Avg(BounceRate), Period))
```

6. Set the color to red.

7. Click **Add reference line** again, this time setting the label to Lower Threshold and the expression:

```
=Avg(BounceRate)-Stdev(Total Aggr( Avg(BounceRate), Period))
```

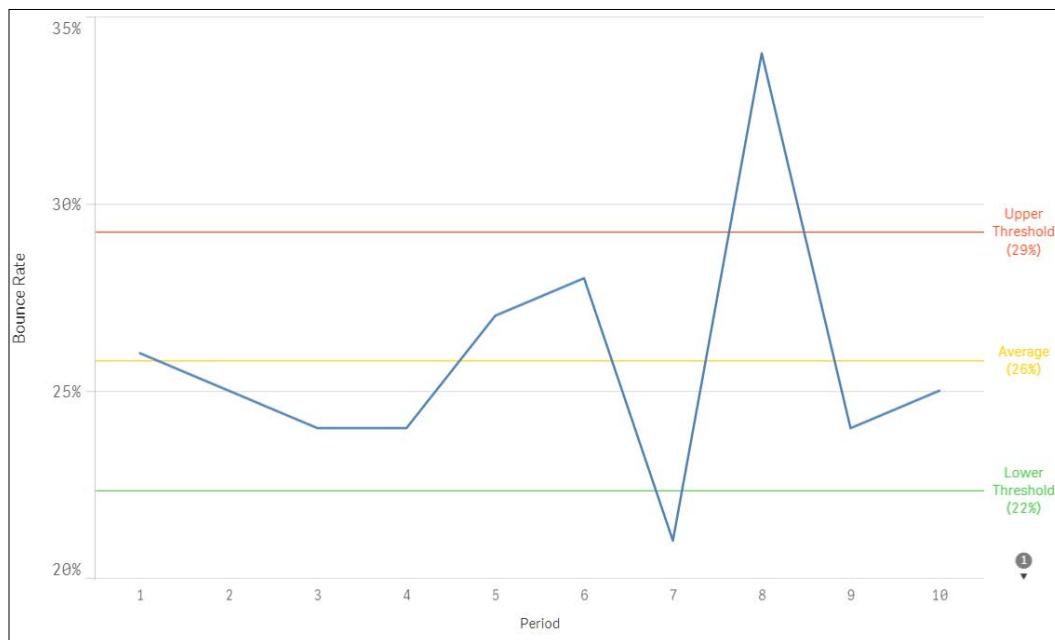
8. Set the color to yellow.

9. Click **Add reference line** for a third time and set the label to Average and the expression to:

```
=Avg(BounceRate)
```

10. Set the color to green.

11. The final visualization should resemble the following image:



How it works...

The preceding chart is often referred to as a **Statistical Process Control (SPC)** chart. The upper and lower threshold reference lines set a boundary of normal operation. Data points that fall outside of these reference lines differ from the norm and are highlighted as such. The upper and lower limits are simply the average plus or minus the standard deviation. We use the Aggr function to "pre-calculate" the average over the period dimension and then apply the Stdev function to this number.

Definition: Standard deviation (represented by the symbol sigma, σ) shows how much variation or "dispersion" exists from the average (mean) or expected value. A low standard deviation indicates that the data points tend to be very close to the mean; high standard deviation indicates that the data points are spread out over a large range of values:

http://en.wikipedia.org/wiki/Standard_deviation

Creating text and images

Images in Qlik Sense for desktop are stored in the following location by default, C:\Users\<*your own user folder*>\Documents\Qlik\Sense\Content\Default\

Once images have been added to the folder they are automatically made available in Qlik Sense. To add images to your dashboard follow these steps.

Getting ready

For this recipe, we will make use on inline data load which gives us sales information. Load the following code into your Qlik Sense application:

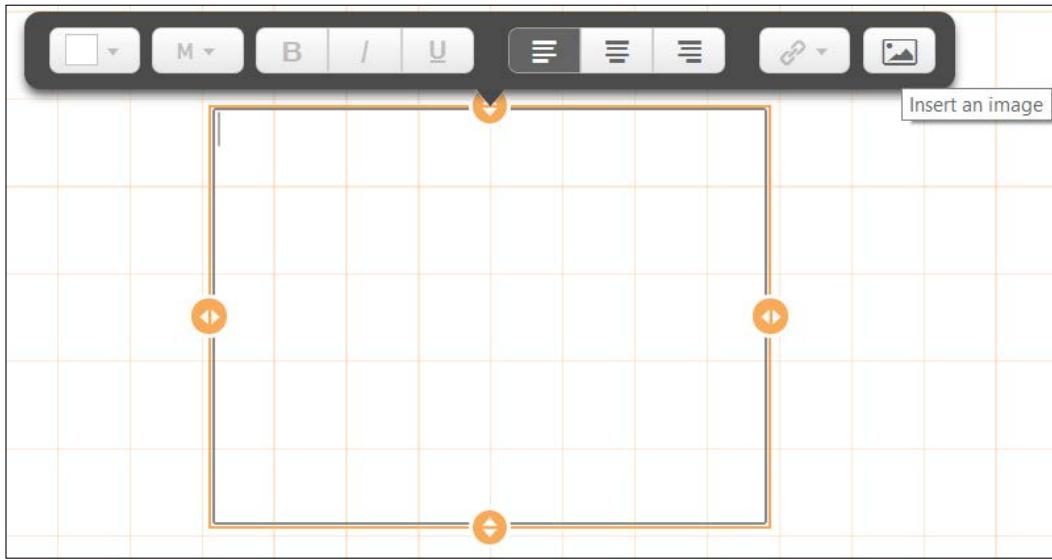
```
SalesData:  
LOAD * INLINE [  
    ID, Sales, Quantity, Cost  
    1, 15000, 50, 11000  
    2, 30000, 100, 25000  
];
```

How to do it...

Adding Images

1. Place your desired image file into the folder C:\Users\<*your own user folder*>\My Documents\Qlik\Sense\Content\Default\.
2. Add the Text & Image object from the Assets panel to the content area.

3. With the Text & Image object selected, click on  to add text and measures.
4. The design bar will appear, click on the **Insert an image** button in the far right, as shown in the following:



5. Select and insert the desired image from the default folder.
6. One can edit the sizing options of the image without clicking on the image button in the design bar as shown in the preceding step. To do this, go to the properties of the Text & Image object and set the same image as a background, using this method now gives you access to size options as shown in the following screenshot:



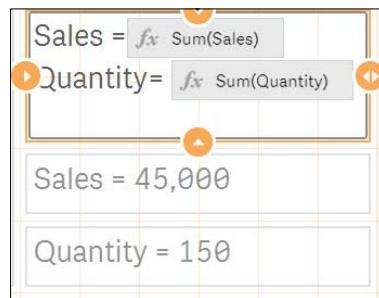
Adding Text

1. Add another Text & Image object from the Assets panel to the content area.
2. If you double click on the object in the content page you can immediately start typing the text. You will see some basic formatting options above the object, as seen in the following:

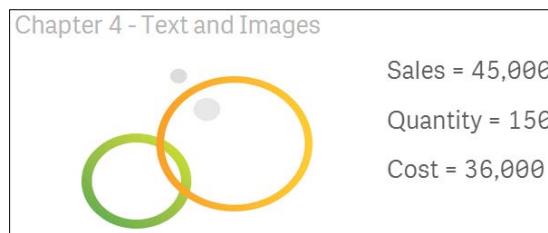


3. Type the following into the text box Sales =.
4. Next, from the properties pane under **Data** add the following measure:
`SUM(Sales)` and label it 'Sales'.
5. From the **Number formatting** drop-down menu select **Number** and from the next drop-down below select the top option with no decimal places (for example 1,000).

6. You can repeat the process using more text objects and different expressions if you like. Multiple measures can be added to the same object or they can be separated out as shown in the following examples:



7. An example of text boxes and images is in the following:



How it works...

Adding text or images to a dashboard can be the key to help users learn more about what they are looking at, not just the company branding.

In the preceding example, we have added a metric into the text box. Normally, we suggest using the KPI object in these instances.

However, text boxes are essential if you wish to add a narrative beyond a single number if all you are trying to do is show one number, using a text box does give the benefit of horizontal labeling. For example the "date of the last reload".

Applying limitations to charts

While outliers can reveal all kinds of useful intelligence such as issues in data capture or associated process patterns, they can cause problems when you are building data visualizations. The most common issue is to do with scale, as you can see in the following example.

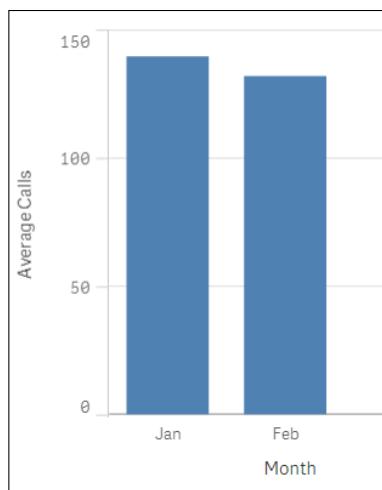
Getting ready

For this recipe, we will make use on inline data load which gives us the information on the number of call made for each month. Add the following code into the data load editor and reload the Qlik Sense application:

```
Data:  
LOAD * INLINE [  
    Month, Date, Calls  
    Jan, 27/01/15, 25  
    Jan, 28/01/15, 27  
    Jan, 29/01/15, 25  
    Jan, 30/01/15, 600  
    Jan, 31/01/15, 22  
    Feb, 01/02/15, 20  
    Feb, 02/02/15, 19  
    Feb, 03/02/15, 21  
    Feb, 04/02/15, 1  
    Feb, 05/02/15, 600  
];  
.
```

How to do it...

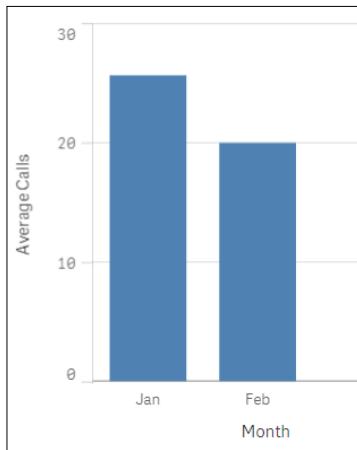
1. Add a Bar chart object onto the content page.
2. Add **Month** as a dimension.
3. Add **Avg(Calls)** as a measure. Label it **Average Calls**.
4. Click on **Done**. Notice that the values in both months are just below **150**:



5. Next, go back into the Edit mode and replace the measure with the following code:

```
Avg(If (Calls > Aggr(NODISTINCT Fractile(Calls, 0.1),  
Month) and Calls < Aggr(NODISTINCT Fractile(Calls, 0.9),  
Month),Calls))
```

6. Click on **Done**.
7. The chart should now resemble the following image. It not only has both bars significantly reduced down to below 30, but there is also a much bigger gap between January and February's average call volumes:



How it works...

If we look at the source data we loaded at the beginning of the recipe it is clear that there are some outliers present. To exclude these and get a real picture of the normal, average amount of calls, we remove the top and bottom 10 percent of the value. This is done using the `fractile` function. The `fractile` function calculates the cut-off point for 10 and 90 percent based on our data. The `Aggr` function is needed because `fractile` is an aggregation function being nested inside another aggregation.

There's more...

Another method of handling outliers is not to exclude them from the expression, but hide them from what is visualized. For example, if a data point far exceeds the norm, you can set the axis limit to the second largest value; this focuses the visualization on the points that are closely related. You can do this by going to the object properties:

- ▶ Under **Appearance**, click on **Y-axis**.
- ▶ Switch off **Auto Range** and set the **Max value** by using an expression such as the following:

```
=Max(aggr(avg(Calls), Date), 2)
```

Here, we simply work out what the second largest number is and set that as the axis limit. This way we can produce an all inclusive line graph by date, albeit one data point will be off the screen.

Adding thumbnails – a clear environment

It is easy to skip over minor features of a BI platform. Unlike Qlikview, which has a high number of options for chart customizations, Qlik Sense features are more universal. The majority of components will be relevant to you and as such should be given due consideration.

Here, our aim is to simplify the environment by adding thumbnails and metadata descriptions at a high level to the application and the sheets within.

Getting ready

Open the Qlik Sense Desktop hub and either open up an existing application or create a new one.

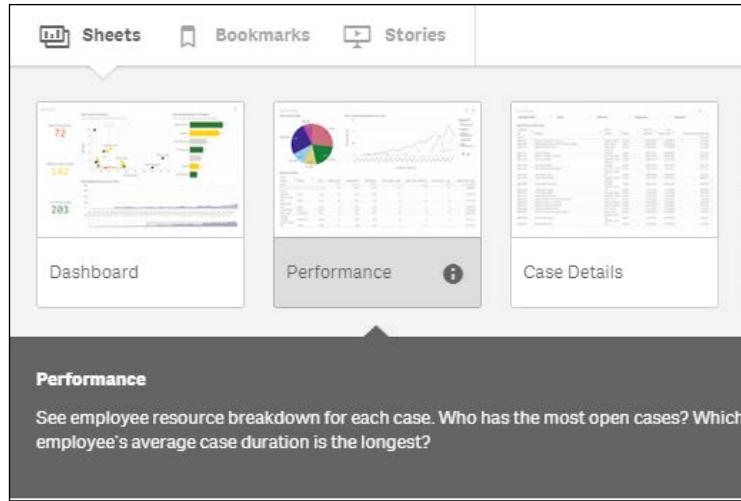
How to do it...

1. Find an image you want to use as the thumbnail for your application. Copy the image into the following folder C:\Users*your own user folder*\Documents\Qlik\Sense\Content\Default\
2. From the **App overview** screen click on the edit  button in the top right corner.
3. Give your application a **Title** and **Description**.
4. Adjacent to the **Title** and **Description** window is the area for the application thumbnail.
5. Click the change thumbnail image .
6. Select the image that you added to the Qlik folder in step one and then click **Insert**.
7. Finally, click the stop editing tick button in the top right hand corner:



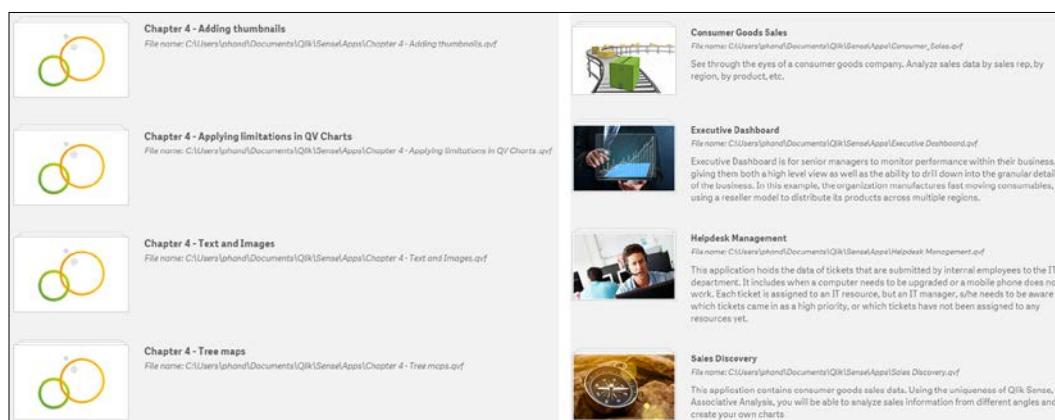
8. Depending on the image you have chosen, the color of the background will also change as shown in the preceding image.

9. You can repeat the process for sheets by clicking on the  button next to each sheet description.
10. You can see a great example of how this should be implemented in the default helpdesk management application that is available with each fresh install of Qlik Sense desktop. Take special note of the sheet descriptions that prompt questions you can answer:



How it works...

I have seen many unorganized BI environments before and it really has a negative impact on the user experience. When you first go into the Qlik Sense hub what do you prefer to be presented with:



1: Default image and text 2: With Thumbnails and descriptions

The second image looks more pleasing and professional to the eye. User experience is an important factor in adoption of the tool. If the first screen you see looks rushed or is confusing, it will start the user off on a bad foot.

The thumbnails and descriptions also apply to sheets within an application. By default, small thumbnails are displayed as an image showing objects by type and placement. These can be replaced with something clearer and more meaningful to the audience. This is hardly storyboarding, but you should know what each page is trying to achieve. Actually, asking questions about each screen can help you get a feel for the shape of the application and the flow of analysis. Are you asking these questions?

1. Who are the users of this screen?
2. What is the page showing?
3. What questions will the page answer?
4. What actions will that enable?

While not universal, asking questions such as these regularly will help keep your focus on the audience.

Navigating many data points in a scatter chart

The following recipe showcases an interesting concept in the use of scatter charts in Qlik Sense. This feature is available from version 2.0+ of Qlik Sense.

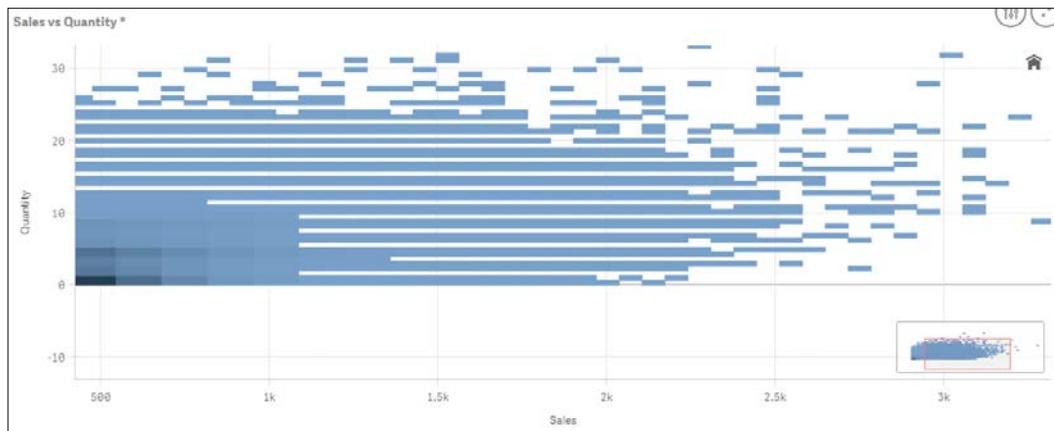
Getting ready

Load the following code into your Qlik Sense application:

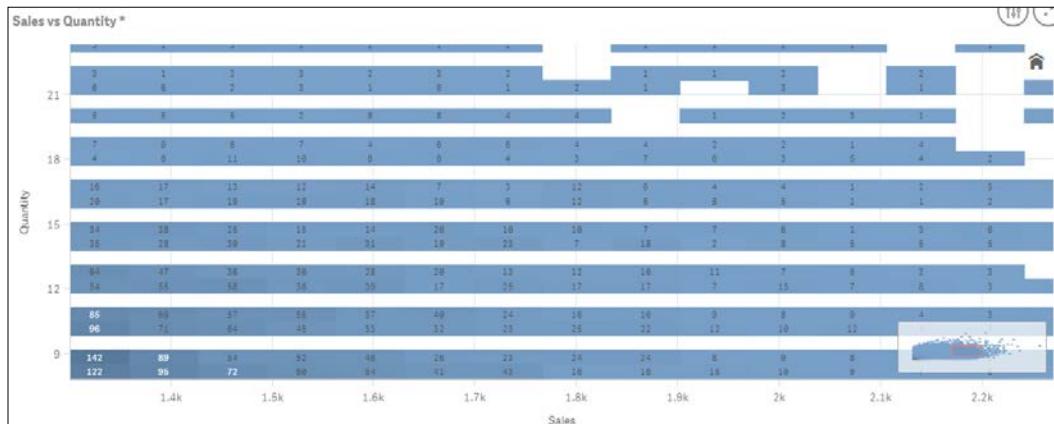
```
Transactions:  
Load  
Round(1000*Rand()*Rand()*Rand()) as Sales,  
Round(10*Rand()*Rand()*Rand()) as Quantity,  
RecNo() as TransID  
Autogenerate 1000000  
While rand()<0.5 or IterNo()=1;
```

How to do it...

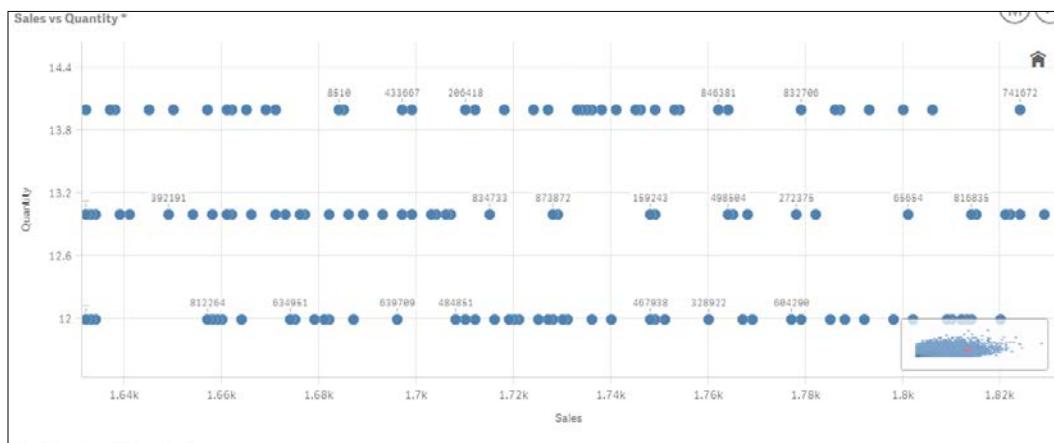
1. Create a new sheet and drag a scatter plot chart object onto the content page.
2. Add **TransID** as a dimension.
3. Add **Sum(Sales)** as the first measure. Label it **Sales**.
4. Add **Sum(Quantity)** as the second measure. Label it **Quantity**.
5. Click on **Done**.
6. Title the chart as **Sales vs Quantity**.
7. You will notice that you cannot select data inside the chart like you can with every other Sense visualization. To navigate the data you have to scroll in and out using the mouse wheel. The object will look like the following image; try scrolling in:



8. As you zoom further into the chart , the number of data points being displayed reduces at once. You will eventually see the details of each data point displayed in each block. The higher density blocks are also color coded as show in the following:



9. Eventually, you can zoom in enough and reduce the number of data points to where they are identifiable individually. At this point the graphic will revert its display to a more standard scatter chart look. Those with enough space around them actually have the value displayed as shown in the following:



How it works...

This is a very intelligent addition by Qlik to the normal scatter chart display. This chart plots over a million data points effortlessly. Doing this the traditional way is very process intensive. When you want to extract data volumes of this size, you normally tend to look at the pattern and not at the individual numbers. This achieves both by displaying the individual points at the point they would make sense and not before.

There's more...

While the color coding is fixed at the high level you can apply color coding expressions like normal. This only gets applied when you zoom in far enough to see the individual data points.

5

Useful Functions

In this chapter, we will focus on some interesting and useful functions available in Qlik Sense:

- ▶ Using an extended interval match to handle Slowly Changing Dimensions
- ▶ Using the `Previous()` function to identify the latest record for a dimensional value
- ▶ Using the `NetworkDays()` function to calculate the working days in a calendar month
- ▶ Using the `Concat()` function to display a string of field values as a dimension
- ▶ Using the `MinString()` function to calculate the age of the oldest case in a queue
- ▶ Using the `RangeSum()` function to plot cumulative figures in trendline charts
- ▶ Using the `Fractile()` function to generate quartiles
- ▶ Using the `FirstSortedValue()` function to identify the median in a quartile range
- ▶ Using the `Derive` and `Declare` functions to generate `Calendar` fields
- ▶ Setting up a moving annual total figure
- ▶ Using the `For Each` loop to extract files from a folder
- ▶ Using the `Peek()` function to create a currency Exchange Rate Calendar
- ▶ Using the `Peek()` function to create a Trial Balance sheet

Introduction

In this chapter, we will shift our focus to the functions available in Qlik Sense. In certain situations, the functions can be used in the script or they can be used in frontend expressions to get solutions for complex requirements. All the functions discussed in this chapter will find their way into most of the Qlik Sense implementations.

Using an extended interval match to handle Slowly Changing Dimensions

Sometimes while developing the Data model for a Business Intelligence application, one comes across dimensional values that tend to change with time. Such dimensions are known as *Slowly Changing Dimensions*. For example, an employee joins a company at a Junior Executive level and stays at the same position for 1 year. After one year, the designation changes to Senior Executive and then changes to Project Manager after 3 years. The position field in this case will be treated as a *Slowly Changing Dimension*.

Such Slowly Changing Dimensions can be represented in Qlik Sense, provided the historical data is stored at the source with a proper "Position Start Date" and "Position End Date".

In order to match the discrete date values to the date intervals, we will make use of the `intervalmatch` function. At the same time, we will match the values of the primary key. This will help us to build an optimized Data model and properly link the transactions to the Slowly Changing Dimensions.

Getting ready

The following recipe assumes a hypothetical situation wherein an HR department is trying to track the *Employee Journey* within an organization that is tracking the various positions the employee has held within his or her tenure with the company and the related compensation against each position. For this purpose, we will create the following `Inline` tables within Qlik Sense:

- ▶ **2 Dimension Tables:** Employee and Position
- ▶ **1 Date Intervals table to track changes in position for the employee:** Employment
- ▶ **1 Fact table:** EmpSalary

The steps to do so are as follows:

1. Create a new Qlik Sense application.
2. Load the following script in Qlik Sense:

```
// ===== Load the Employee table =====
Employee:
LOAD * INLINE [
    EmployeeID,EmployeeName
    11,Susan Sayce
    22,Adam Holliaoak
    33,Rod Marsh
    44,Alex Gerard
    55,Pete Cox
```

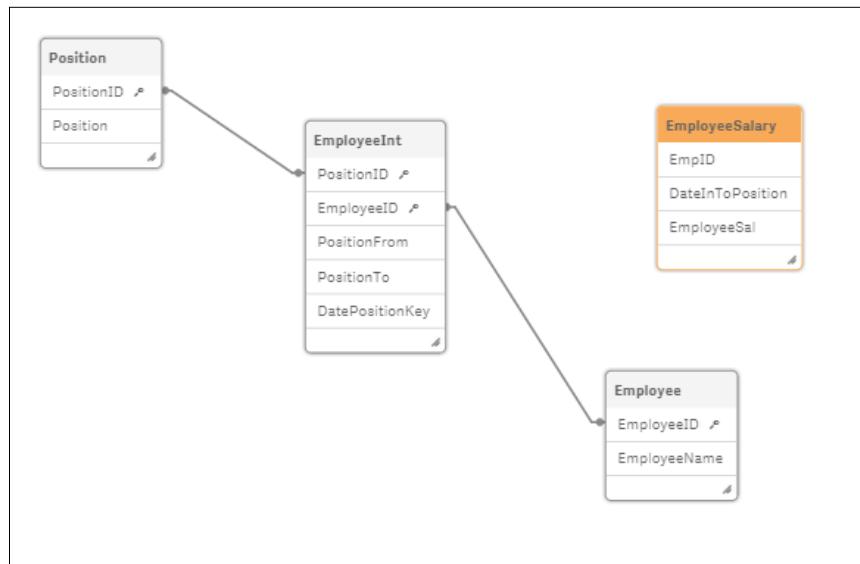
```
];
// ===== Load the Position table =====
Position:
LOAD * INLINE [
PositionID,Position
1,HR Analyst
2,HR Director
3,HR Executive
];

// === Load the Employee table with the Date Intervals ===
EmployeeInt:
LOAD *,
    Autonumber(EmployeeID & '-' & PositionFrom & '-' & PositionTo)
    as DatePositionKey;
LOAD DATE(Date#( PositionFrom,'DD/MM/YYYY')) as PositionFrom,
DATE(Date#( PositionTo,'DD/MM/YYYY')) as PositionTo, PositionID,
EmployeeID
INLINE [
    PositionFrom, PositionTo,PositionID,EmployeeID
    01/09/2009, 31/10/2010,2,11
    01/08/2008, 31/08/2009,1,11
    10/08/2008, 15/03/2010,1,22
    03/03/2008, 08/12/2008,2,33
    15/02/2008, 15/03/2010,3,44
    01/06/2008, 08/12/2008,3,55
];
// ===== Load the Employee Salary table =====
EmployeeSalary:
LOAD EmpID ,DATE(Date#( DateInToPosition,'DD/MM/YYYY')) as
DateInToPosition, EmployeeSal INLINE [
EmpID,DateInToPosition,EmployeeSal
11,01/09/2009,90000
11,01/08/2008,50000
22,10/08/2008,45000
33,03/03/2008,100000
44,15/02/2008,60000
55,01/06/2008,55000
];

```

How to do it...

1. Open the Data model viewer. The Data model is shown in the following figure. We can see that the `EmpSalary` table is not linked to the Data model. If we try to link the table through the `EmpID` field, then the employees who have changed their positions would reflect the same salaries for each position, which is not correct.



2. Open the **App overview** and create a new sheet. Drag a Table object onto the content area.
3. Add the following dimensions to the table: **EmployeeID**, **EmployeeName**, **Position**, **PositionFrom**, and **PositionTo**.
4. Under **Sorting**, promote **EmployeeName** to the top. Promote **PositionFrom** to the second position and set the sort order as numeric and ascending.

| Extended interval match | | | | | |
|-------------------------|----------------|--------------|--------------|------------|--|
| EmployeeID | EmployeeName | Position | PositionFrom | PositionTo | |
| 11 | Susan Sayce | HR Analyst | 01/08/2008 | 31/08/2009 | |
| 11 | Susan Sayce | HR Director | 01/09/2009 | 31/10/2010 | |
| 22 | Adam Holliaoak | HR Analyst | 10/08/2008 | 15/03/2010 | |
| 33 | Rod Marsh | HR Director | 03/03/2008 | 08/12/2008 | |
| 44 | Alex Gerard | HR Executive | 15/02/2008 | 15/03/2010 | |
| 55 | Pete Cox | HR Executive | 01/06/2008 | 08/12/2008 | |

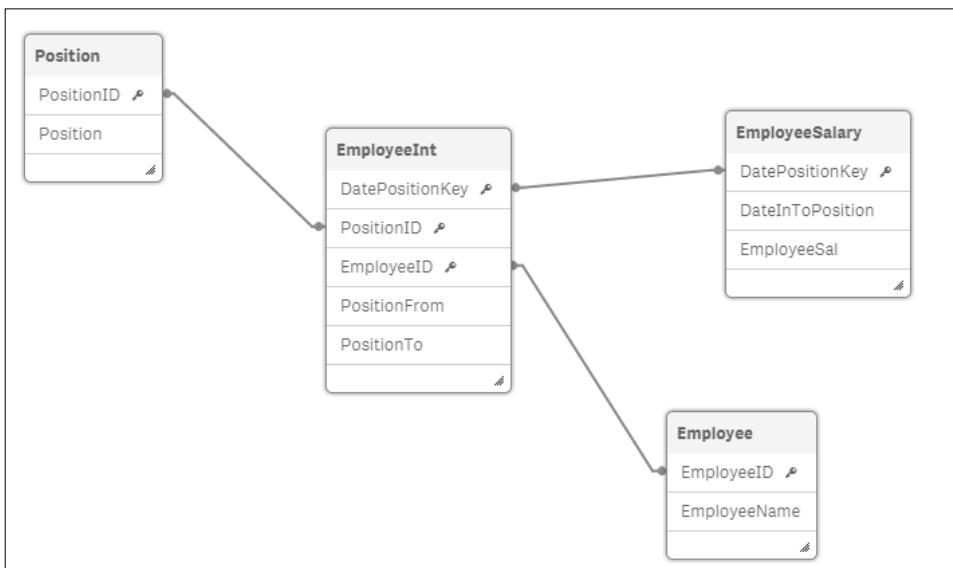
5. In the preceding script, **Susan Sayce** has changed her position from **HR Analyst** to **HR Director**. There is a DateInToPosition value associated with each position, which comes from the EmployeeSalary table.
6. We will make use of the IntervalMatch function, which will match the DateInToPosition to the date interval of PositionFrom and PositionTo.
7. Load the following script on a separate section:

```
// === Link Table using the IntervalMatch prefix ===
LinkTable:
IntervalMatch (DateInToPosition,EmpID)
Load distinct PositionFrom, PositionTo, EmployeeID AS EmpID
Resident EmployeeInt;

Left Join (EmployeeSalary)
Load
EmpID,
DateInToPosition,
Autonumber(EmpID & '-' & PositionFrom & '-' & PositionTo)
AS DatePositionKey
Resident LinkTable;

// ===== Cleanup =====
Drop Table LinkTable;
Drop Field EmpID;
```

8. On the final load, the Data model should look like this:



Useful Functions

9. Open the **App overview** via the navigation  dropdown on the top-left corner. Go back to the sheet created in step 2.
10. In the Table object, add the following measure and label it **Salary**:
`Sum(EmployeeSal)`
11. Make sure that the sorting order remains same as mentioned in step 4, that is, to promote `EmployeeName` to the top. Promote `PositionFrom` to the second position and set the sort order as numeric and ascending.
12. The resultant table would look like this:

| Extended interval match | | | | | | |
|-------------------------|----------------|--------------|--------------|------------|---------------|--|
| EmployeeID | EmployeeName | Position | PositionFrom | PositionTo | Salary | |
| Totals | | | | | 400000 | |
| 11 | Susan Sayce | HR Analyst | 01/08/2008 | 31/08/2009 | 50000 | |
| 11 | Susan Sayce | HR Director | 01/09/2009 | 31/10/2010 | 90000 | |
| 22 | Adam Holliaoak | HR Analyst | 10/08/2008 | 15/03/2010 | 45000 | |
| 33 | Rod Marsh | HR Director | 03/03/2008 | 08/12/2008 | 100000 | |
| 44 | Alex Gerard | HR Executive | 15/02/2008 | 15/03/2010 | 60000 | |
| 55 | Pete Cox | HR Executive | 01/06/2008 | 08/12/2008 | 55000 | |

13. Select a particular employee to see all the associated positions, start dates, end dates, and salaries.

How it works...

The dimension tables are loaded first. A composite key comprising `EmployeeID`, `PositionFrom`, and `PositionTo` is created in the `EmployeeInt` table.

The fact table `EmployeeSalary` is loaded with the `EmployeeID` value represented as `EmpId`.

Under `LinkTable`, an interval is assigned to each combination of `EmpID` and `DateInToPosition` using the `intervalmatch` function.

Finally, a key is created in `LinkTable` with the same combination of `EmployeeID`, `PositionFrom`, and `PositionTo`. The `LinkTable` is joined back to the `EmployeeSalary` table.

The problem of slowly changing dimensions can be solved using the extended intervalmatch syntax explained in the preceding steps. The employee, positions, and salaries will all be properly linked.

There's more...

In the preceding example we have joined LinkTable to the EmployeeSalary table. However, one should bear in mind that this can only be done if there is a *Many-One* relationship between the **Employee** and **Position**. If this doesn't hold true, that is, if an employee knowingly or unknowingly has more than one position for the same start and end dates in the source data, then the join between the link and the EmployeeSalary table will result in an increase in the number of records. In such a situation, the left join should be avoided.

Instead LinkTable must simply be linked through the DatePositionKey composite key to the EmployeeInt table. Another composite key comprising DateInToPosition and EmpID must be created which should link back to the same key in EmployeeSalary.

The resident load for the Link table would be as follows:

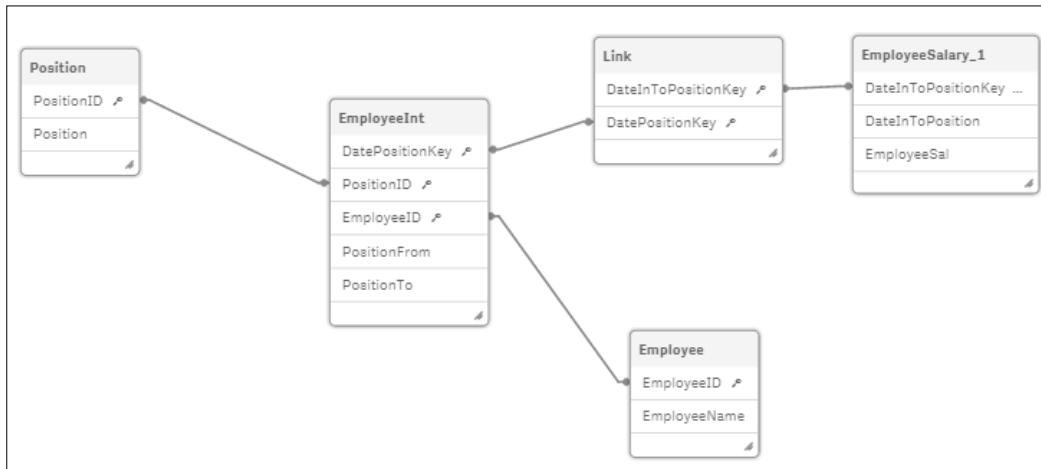
```
Link:  
Load  
Autonumber(EmpID & '-' & DateInToPosition) AS DateInToPositionKey,  
Autonumber(EmpID & '-' & PositionFrom & '-' & PositionTo)  
AS DatePositionKey  
Resident LinkTable;
```

The resident load for the Employee table would be as follows:

```
EmployeeSalary_1:  
Load  
*,  
Autonumber(EmpID & '-' & DateInToPosition) AS DateInToPositionKey  
Resident  
EmployeeSalary;  
DROP TABLE EmployeeSalary;
```

Useful Functions

On loading the script, the resulting Data model would be like this:



See also

- ▶ *Using the Previous() function to identify the latest record read for a dimensional value*

Using the Previous() function to identify the latest record for a dimensional value

In a line-level table, there are multiple records stored for a single dimensional value. For example, an Order Line table will have multiple lines for the same OrderID. Business requirements may warrant us to only consider the first or the latest line for each order. This can be done using the Previous() function available in Qlik Sense.

Getting ready

For the sake of continuity, we will make use of the same script and application as in the previous recipe. We will determine the most recent position for any employee during his or her tenure within the organization.

How to do it...

1. Open the data load editor. Change the name of the EmployeeInt table in the script to EmployeeIntTemp.

2. Insert the following lines of code after the EmployeeIntTemp table. If you are copying and pasting the code in the data load editor, make sure that the single quotes are copied in a proper format:

```
EmployeeInt:
LOAD *,
if([EmployeeID] = previous([EmployeeID]), 'No', 'Yes') AS
    LatestRecordFlag
RESIDENT EmployeeIntTemp
ORDER BY [EmployeeID] ASC, PositionFrom DESC;

DROP TABLE EmployeeIntTemp;
```

3. Save and load the script.
4. Add the field LatestRecordFlag in the Table object we created in the previous recipe.
5. Under **Sorting**, make sure that PositionFrom is promoted to the top. Switch off the **Auto sorting** feature for PositionFrom. No sorting options should be selected as this will then show the PositionFrom date in the load order.
6. The table would look like this:

| Extended interval match | | | | | | |
|-------------------------|---------------|--------------|--------------|------------|--------|------------------|
| EmployeeID | EmployeeName | Position | PositionFrom | PositionTo | Salary | LatestRecordFlag |
| Totals | | | | | 400000 | |
| 11 | Susan Sayce | HR Director | 01/09/2009 | 31/10/2010 | 90000 | Yes |
| 11 | Susan Sayce | HR Analyst | 01/08/2008 | 31/08/2009 | 50000 | No |
| 22 | Adam Holliaok | HR Analyst | 10/08/2008 | 15/03/2010 | 45000 | Yes |
| 33 | Rod Marsh | HR Director | 03/03/2008 | 08/12/2008 | 100000 | Yes |
| 44 | Alex Gerard | HR Executive | 15/02/2008 | 15/03/2010 | 60000 | Yes |
| 55 | Pete Cox | HR Executive | 01/06/2008 | 08/12/2008 | 55000 | Yes |

7. Select employee **Susan Sayce**. We can see that there are two positions associated with Susan. If we select the LatestRecordFlag value as **Yes**, it will only show the latest position for Susan: **HR Director**.

How it works...

The LatestRecordFlag can be used in calculations to determine the most recent position of any employee. In our script, we create the LatestRecordFlag using the Previous() function. The Previous() function basically parses the EmployeeID column. If the current record that is being read has the same EmployeeID value as the previous record, then it is flagged as No or else Yes.

The *ordering* of the fields plays an important role here. Because I wanted to determine the latest position for the employee, the field `PositionFrom` is arranged in descending order.

There's more...

We can also make use of the `Peek()` function in the preceding script. In our example, both `Peek()` and `Previous()` would yield the same result. However, `Peek()` is more effective when the user is targeting a field which has not previously loaded in the table or if the user wants to target a specific row. The `Previous()` function is more effective when the user wants to compare the current value with the previous value for the field in the input table.

See also

- ▶ *Using the Peek() function to create a Trial Balance sheet*

Using the NetworkDays() function to calculate the working days in a calendar month

One of the KPIs that companies often concentrate on is the average sales in a month. The average sales are calculated by dividing the total sales by the number of working days in a month.

Every month has a different number of working days. While calculating the number of working days for the current month, we only need to consider the days passed in the month and not the total days of the month in order to calculate the actual working days.

In order to arrive at the exact number of working days in a month, we need to exclude all the Fridays and Saturdays as well as the public and bank holidays from our calculations. The `Networkdays()` function helps us to achieve this.

Getting ready

For this exercise, we first need to prepare a list of all public holidays either in Excel or inline in the Qlik Sense script.

How to do it...

1. Copy and paste the following part of the script in the data load editor. This is a list of public holidays for 2014 and 2015:

```
HolidayTmp:  
LOAD DATE(Date#( Date,'DD/MM/YYYY')) as Date INLINE [  
Date  
01/01/2015  
03/04/2015  
06/04/2015  
04/05/2015  
25/05/2015  
31/08/2015  
25/12/2015  
28/12/2015  
01/01/2014  
18/04/2014  
21/04/2014  
05/05/2014  
26/05/2014  
25/08/2014  
25/12/2014  
26/12/2014  
];
```

2. Next, we will store the list of public holidays in a variable inside the script:

```
ConcatTmp:  
LOAD concat(chr(39) & Date & chr(39),',') AS HolidayDates  
RESIDENT HolidayTmp;  
LET vPublicHolidays = FieldValue('HolidayDates',1);  
  
LET vCurMonth=month(today());
```

3. Copy and paste the following fact table. Insert the last of the PostingDates in your table as today's date and put a sales figure against it. This is to demonstrate the use of today() in the WorkingDays calculation:

```
SalesTmp:  
LOAD DATE(Date#( PostingDate,'DD/MM/YYYY')) as PostingDate,  
Sales INLINE [  
PostingDate, Sales  
05/08/2014, 5000  
04/09/2014,522
```

```
24/10/2014,400  
15/11/2014,5000  
24/12/2014, 822  
29/12/2014, 633  
02/01/2015, 1000  
02/02/2015, 2000  
25/03/2015,2200  
25/04/2015,266  
09/05/2015, 3000  
18/05/2015, 4000  
15/06/2015,5000  
22/07/2015,456  
08/09/2015,4200  
26/10/2015,1875  
];
```

4. Next, calculate the number of working days:

```
Sales:  
LOAD *,  
Month(PostingDate) as Month,  
MonthName(PostingDate) AS MonthYear,  
IF(Year(PostingDate)=Year(TODAY()) AND Month(PostingDate)=MONTH(TODAY()),  
    NETWORKDAYS(MONTHSTART(today()),(Today()),  
    $(vPublicHolidays)), NETWORKDAYS(MONTHSTART(PostingDate),  
    MonthEnd(PostingDate),  
    $(vPublicHolidays))) AS WorkingDays RESIDENT  
SalesTmp;  
DROP table SalesTmp;  
DROP table HolidayTmp;
```

5. Load the script.
6. On the Qlik Sense sheet, create a table object and name it **Average Monthly Sales**.
7. Add **MonthYear** and **WorkingDays** as dimensions.
8. Add the following measure and label it as **Avg Sales**:
 $\text{Sum}(Sales) / \text{WorkingDays}$
9. Set the number formatting for **Avg Sales** to **Money**.
10. Under **Sorting**, make sure that the **MonthYear** field is promoted to the top.
11. Go to **Appearance | Presentation** and switch off **Totals**.

12. The final table object should look like this:

| Average Monthly Sales | | | |
|-----------------------|-------------|-----------|--|
| MonthYear | WorkingDays | Avg Sales | |
| Aug 2014 | 20 | £250.00 | |
| Sep 2014 | 22 | £23.73 | |
| Oct 2014 | 23 | £22.22 | |
| Nov 2014 | 20 | £250.00 | |
| Dec 2014 | 21 | £69.29 | |
| Jan 2015 | 21 | £47.62 | |
| Feb 2015 | 20 | £100.00 | |
| Mar 2015 | 22 | £100.00 | |
| Apr 2015 | 20 | £13.30 | |
| May 2015 | 19 | £368.42 | |
| Jun 2015 | 22 | £227.27 | |
| Jul 2015 | 23 | £19.83 | |
| Sep 2015 | 22 | £190.91 | |
| Oct 2015 | 18 | £104.17 | |

How it works...

The `Concat` function stores the aggregated string concatenation of all the holiday dates. These holiday dates are stored in a variable `vPublicHolidays`, which is further used in the `Networkdays()` function.

The `Networkdays()` function has three parameters. The first two parameters define the range of dates to consider. If the `PostingDate` date lies in the current month, the range of dates is defined by the first day of the month and today. From this range, we exclude the non-working days Saturdays, Sundays, and public holidays.

If the posting date is in a month prior to the current month, the first and the last day of said month determine the range of the days for calculating the working days.

See also

- ▶ *Using the `Concat()` function to display a string of field values as a dimension*

Using the Concat() function to display a string of field values as a dimension

A line-level table is normally the most granular data in a Data model. For example, consider an Order Line table. The orders for each customer are stored one row per product line, and we have corresponding costs for each product on each line. When we generate a Table report for such data, we will have a separate line for each product which in itself is not wrong. But recently a customer asked me to initiate an export for the Table report in such a way that all the products for a particular order are contained in a single cell and the sales column should show the aggregated figure for all the products under OrderID. To tackle this requirement, I created a calculated dimension using the Concat function. The process is explained in the following recipe.

Getting ready

1. Create a new Qlik Sense application.
2. Add the following INLINE table that contains the Order Line table details:

```
Orders:  
LOAD * INLINE [  
    Customer,OrderID,Product,Cost  
    1,201,Chain,20  
    1,201,Seat,40  
    1,201,Mudguard,50  
    2,202,Gloves,15  
    2,202,Basket,60  
    3,203,Helmet,70  
];
```

3. Load the data and save the file. Open **App overview** by clicking on the Navigation dropdown  in the top-left corner.

How to do it...

1. Create a new sheet.
2. Drag the Table object from the left-hand side Assets panel on to the sheet. Name it Sales by Order.
3. Add **OrderID** and **Customer** as dimensions.
4. Add the following as a third, calculated dimension and label it Products:

```
=AGGR(Concat(DISTINCT Product,','),OrderID)
```

5. Add the following expression as the measure. Label it **Total Sales**:

`Sum(Cost)`

6. Click on **Save** and click on  **Done**.

7. The resulting table on the screen will look like this:

| Sales by Orders | | | |
|-----------------|----------|---------------------|-------------|
| OrderID | Customer | Products | Total Sales |
| Totals | | | 255 |
| 201 | 1 | Chain,Mudguard,Seat | 110 |
| 202 | 2 | Basket,Gloves | 75 |
| 203 | 3 | Helmet | 70 |

8. As you can see, all the products for a particular OrderID value are stringed together in a single cell and the sales figures are the aggregated figures for each OrderID value.

How it works...

The `Concat()` function gives us the aggregated string concatenation of all the product values separated by the `,` delimiter. The `Concat()` function is an aggregation function and hence needs to be used with `AGGR` in order to be used as a dimension. For the sake of our dimension, the products are grouped by the `OrderID`.

The same functionality could have been achieved by defining products within a calculation in a measure as follows:

```
Concat(DISTINCT Product, ',')
```

But by doing so, we won't be able to select the products for a particular `OrderID` value inside the table.

When we use the calculated dimension, we get the advantage of selecting all the products for the `OrderID` value in a single go by selecting a cell in the products column.

There's more...

The `Concat()` function can also be used in the script along with the `Group By` clause.

See also

- ▶ Using the *Fractile()* function to generate quartiles

Using the Minstring() function to calculate the age of the oldest case in a queue

Support centers for any organization, log several customer cases during the day. These cases are sometimes tagged with a specific status such as *contact*, *review*, and so on. Each case goes through different statuses in the workflow until it reaches *closed* or *sign off* in the queue. The following example calculates the number of cases in each status of the workflow and then makes use of the *Minstring()* function to calculate the number of days passed since the oldest case logged for a particular status.

Getting ready

Load the following script which gives information on the cases logged at a debt collection agency:

```
LET vToday=num(today());  
Case:  
LOAD CaseID ,DATE(Date#( DateLogged, 'DD/MM/YYYY')) as DateLogged,  
Status INLINE [  
CaseID,DateLogged,Status  
101,01/01/2002,Advice  
101,25/04/2002,Contact  
101,21/06/2003,Creditors Meeting  
101,24/06/2003,Draft Allocation  
101,30/06/2003,Sign off  
102,18/10/2009,Contact  
102,28/10/2009,Advice  
102,11/02/2010,Creditors Meeting  
102,20/03/2010,Draft Allocation  
102,30/06/2010,Review  
103,11/02/2013>New Business  
103,19/06/2013,Draft Allocation  
104,30/06/2010>New Business  
105,30/06/2010>Contact  
105,11/02/2013>New Business  
106,19/06/2013,Drafting  
106,30/06/2010,Advice  
];
```

How to do it...

1. Drag the Table object onto the sheet from the Assets panel on the left. Name it **Oldest case in Queue (in days)**.
2. Add **Status** as the dimension.
3. Next, add the following expression as the first measure and label it **Case Volume**:
Count (CaseID)
4. Add the following expression as the second measure and label it **oldest item in Queue (in Days)**:
`Num(${vToday} - (MinString({$<DateLogged=>} [DateLogged])) , '#,##0')`
5. Under **Sorting**, promote **Status** to the top.
6. Under **Appearance**, click on **Presentation** and uncheck **Totals**.
7. Click on  when finished.
8. The resulting table should look like the following screenshot. The figures you get for the **Oldest Item in Queue** table may be different, as the calculation is based on today's date, which will be different in your case:

| Oldest case in Queue (in days)* | | | |
|---------------------------------|-------------|------------------------------|--|
| Status | Case Volume | Oldest Item in Queue in Days | |
| Advice | 3 | 4,951 | |
| Contact | 3 | 4,837 | |
| Creditors Meeting | 2 | 4,415 | |
| Draft Allocation | 3 | 4,412 | |
| Drafting | 1 | 764 | |
| New Business | 3 | 1,849 | |
| Review | 1 | 1,849 | |
| Sign off | 1 | 4,406 | |

How it works...

Today's date is stored in a number format in the variable `vToday()`. The `MinString()` function finds the oldest value in the `DateLogged` field from the total number of cases for each status. Next, we take a difference between `Today()` and the minimum date for each status to get the number of days for the oldest case.

There's more...

By making use of the `Peek()` and `Previous()` functions and using the correct sort order during load, we can determine the case volume for each change of status. For example, count of cases that went from *advice* to *contact*, *contact* to *creditors meeting*, and so on.

See also

- ▶ *Using the RangeSum() function to plot cumulative figures in trendline charts*

Using the Rangesum() function to plot cumulative figures in trendline charts

The charts in Qlik Sense don't provide the user with the in-built functionality to calculate the cumulative totals, as is the case with QlikView. In order to achieve the cumulative totals in a trendline chart, we make use of the `RangeSum()` function.

Getting ready

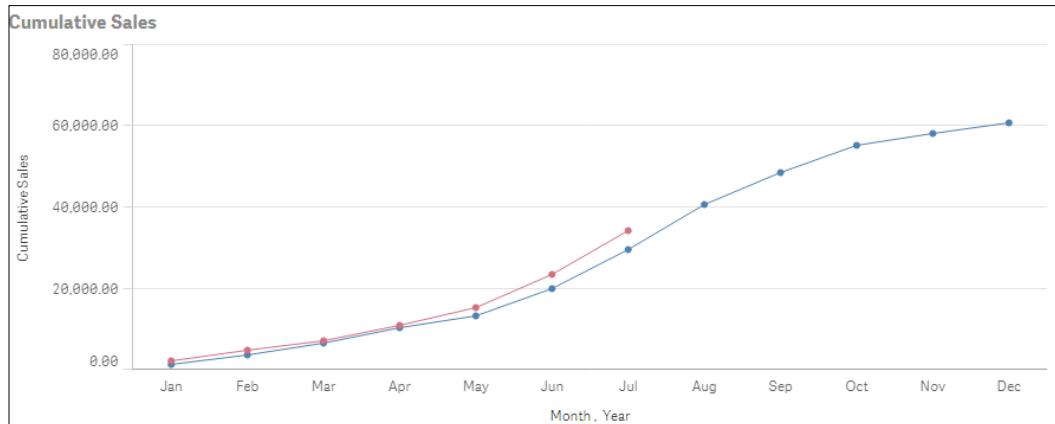
Load the following script that gives information of monthly sales figures for 2 years:

```
Sales:  
LOAD  
Month(Date#(Month, 'MMM')) as Month,  
Year,  
Sales  
INLINE [  
Month,Year,Sales  
Jan,2014,1000  
Feb,2014,1520  
Mar,2014,1600  
Apr,2014,3000  
May,2014,2500  
Jun,2014,4500  
Jul,2014,6000  
Aug,2014,6500  
Sep,2014,7800  
Oct,2014,6800  
Nov,2014,3000  
Dec,2014,2500  
Jan,2015,750  
Feb,2015,1200
```

```
Mar,2015,800  
Apr,2015,600  
May,2015,2100  
Jun,2015,3500  
Jul,2015,4700  
];
```

How to do it...

1. Click on **App overview** under the Navigation dropdown and create a new sheet.
2. Drag across the Line chart object from the Assets panel on the sheet and name it Cumulative Sales.
3. Add **Year** and **Month** as the dimensions.
4. Next, add the following measure and label it Cumulative Sales:
`RANGESUM(ABOVE(TOTAL Sum(Sales), 0, ROWNO(TOTAL)))`
5. Save the application and click on Done.
6. The final trendline chart should look like the following:



How it works...

There are three arguments defined in the syntax used for the `Above()` function:

- ▶ `Expression = Sum(Sales)`
- ▶ `Offset = '0'`

Since this is zero, the function evaluates the expression on the current row.

- ▶ Count = RowNo(Total)

The third argument tells the `Above()` function to evaluate the expression over a range of values. In this case, because we are specifying a total inside the `Rowno()` function, the result would be the number of the row the user is currently on.

The `Above()` function will return a range of values. Hence we will use the `RangeSum()` function to sum up all the values.

See also

- ▶ *Using the `FirstSortValue()` function to identify the median in a quartile range*

Using the `Fractile()` function to generate quartiles

Qlik Sense provides a host of statistical functions that can be put to effective use based on requirements in user reports. At a recent implementation, one of the requirements that popped out was to divide the data values into four quartiles. Quartiles are equivalent to percentiles which divide the data into four groups.

The first quartile is determined by every value which is equal to and less than the 25th percentile. The second quartile is determined by every value which is between the 25th and the 50th percentile. The third quartile is determined by every value which is between the 50th and the 75th percentile. The fourth quartile will be all the data values above and beyond the value of 75th percentile.

In order to generate quartiles in Qlik Sense, we make use of the `Fractile()` function. The following recipe explains the process.

Getting ready

For the sake of this recipe we create a hypothetical situation and make use of an inline data load which gives a case level information for an insurance company. Load the following script in Qlik Sense:

```
Case:  
LOAD * INLINE [  
CaseID,Value,Status  
101,1500,Active  
102,1800,Active  
103,800,Closed  
104,2590,Closed  
105,3500,Closed
```

```
106,1200,Active  
107,5600,Active  
108,8000,Closed  
109,5960,Closed  
110,5000,Active  
111,4000,Active  
112,2500,Active  
];
```

How to do it...

1. Click on **App overview** under the Navigation dropdown and create a new sheet.
2. Enter the Edit mode by clicking on  Edit.
3. Drag the Table object on to the sheet.
4. Add the following calculated dimension and label it Quartile:

```
=If (Value <= Fractile (TOTAL Value, 0.25), 'Quartile 1',  
    If (Value <= Fractile (TOTAL Value, 0.50), 'Quartile 2',  
        If (Value <= Fractile (TOTAL Value, 0.75), 'Quartile 3',  
            'Quartile 4')))
```
5. Add second dimension CaseID.
6. Add the following measure and label it Value:

```
Sum(Value)
```
7. Under **Sorting**, promote value to the top and sort it as numeric descending.
8. The resultant table would be as follows:

| Quartiles | CaseID | Value |
|---------------|--------|--------------|
| Totals | | 42450 |
| Quartile 4 | 108 | 8000 |
| Quartile 4 | 109 | 5960 |
| Quartile 4 | 107 | 5600 |
| Quartile 3 | 110 | 5000 |
| Quartile 3 | 111 | 4000 |
| Quartile 3 | 105 | 3500 |
| Quartile 2 | 104 | 2590 |
| Quartile 2 | 112 | 2500 |
| Quartile 1 | 102 | 1800 |
| Quartile 1 | 101 | 1500 |
| Quartile 1 | 106 | 1200 |
| Quartile 1 | 103 | 800 |

9. As seen in the preceding screenshot, each CaseID value is now grouped under the Quartile.

How it works...

The Fractile () function finds the value corresponding to the stated quartile in the range of the data values given by the expression. For example, a Fractile (TOTAL Value, 0.25) works in the following way.

A value corresponding to the 25th percentile is calculated. The total qualifier disregards the chart dimensions.

In our calculated dimension, every CaseID having the value below the 25th percentile mark is tagged as *Quartile 1*, between 25th and 50th as *Quartile 2*, and so on.

There's more...

We can make use of a distinct qualifier inside the Fractile() function. In such a case, only the unique values of the field Value are evaluated.

See also

- ▶ Using the FirstSortedValue() function to identify the median in a quartile range

Using the FirstSortedValue() function to identify the median in a quartile range

Our next task is to find a claim corresponding to the median value in each quartile. A median is nothing but a value corresponding to the 50th percentile. We can achieve this using the FirstSortedValue() and median() functions.

Getting ready

Continue with the same application as in the preceding recipe.

How to do it...

1. Go to the Edit mode by clicking on  Edit.
2. Select the table we created just now in the preceding recipe.

3. Edit the **CaseID** dimension and put in the following calculation:

```
=if(Match(CaseID,
'$(=FirstSortedValue(distinct{<Value={"<=$(=Median({<Value=
{'>=$(=fractile(Value, 0))<=$(=Fractile(Value, 0.25))'}>
Value)}"}>} CaseID, -Value))',
'$(=FirstSortedValue(distinct{<Value={"<=$(=Median({<Value=
{'>$(=fractile(Value, 0.25))<=$(=fractile(Value,
0.5))'}>} Value)}"}>} CaseID, -Value))',
'$(=FirstSortedValue(distinct{<Value={"<=$(=Median({<Value=
{'>$(=fractile(Value, 0.5))<=$(=fractile(Value,
0.75))'}>} Value)}"}>} CaseID, -Value))',
'$(=FirstSortedValue(distinct{<Value={"<=$(=Median({<Value=
{'>$(=fractile(Value, 0.75))<=$(=fractile(Value, 1))'}>
Value)}"}>} CaseID, -Value))'
),
CaseID,
Null()
)
```

4. Uncheck **Show Null Values for CaseID**.

5. The resultant table will look like this:

| Quartiles | CaseID | Value |
|---------------|--------|--------------|
| Totals | | 13660 |
| Quartile 4 | 109 | 5960 |
| Quartile 3 | 111 | 4000 |
| Quartile 2 | 112 | 2500 |
| Quartile 1 | 106 | 1200 |

6. As you can see, every quartile is now showing only the claim corresponding to the median value in each quartile.

How it works...

The calculated dimension for CaseID gives us the claims corresponding to the median values in each quartile. As you can see, a Match() function is being used to match the CaseID with each of the four expressions within.

Let's decipher the first expression inside the `Match()` function:

```
'$ (=FirstSortedValue(distinct  
{<Value={"<=$ (=Median({<Value={'}>=$ (=fractile(Value,  
0))<=$ (=fractile(Value, 0.25))'}>} Value))"}>} CaseID, -Value))'
```

The details of the expressions are as follows:

- ▶ The innermost set gives us the range of values which are between the 0th quartile value and the 25th Quartile value
- ▶ The `Median()` function then gives us the value which lies at the median of this range
- ▶ The `FirstSortedvalue()` returns the value of the output field (`CaseID`) based on the sorted values of the value field

In situations where the number of claims in any quartile is an even number, there will be two claims which will correspond to the median values. In such a scenario, we want to select only the claim which is higher in the sorting order. Hence, we use a `-Value` as the sort weight.

There's more...

Similar to medians, we can derive the quartiles within quartiles using the `Fractile()` function.

See also

- ▶ [Using the Fractile\(\) function to generate quartiles](#)

Using the Declare and Derive functions to generate Calendar fields

Defining a Master Calendar in Qlik Sense is a common requirement and can be done using the `Time` and `Date` functions. With Sense, Qlik has introduced the `Declare` and `Derive` functions, which make it easier to create the `Calendar` definition. This is still not commonly used, as most Qlik Sense developers stick to their old `Calendar` scripts, and there is nothing wrong with that. However, these functions are worth exploring.

Getting ready

Load the following part of the script that gives information on organization sales into the Qlik Sense application:

```
OrgSales:  
LOAD Product, OrderNo ,DATE(Date#( InvoiceDate,'DD/MM/YYYY')) as  
InvoiceDate,  
Sales INLINE [  
InvoiceDate,Product,OrderNo,Sales  
1/1/2013,Chains,101,5500  
8/2/2014,Seats,101,4800  
3/3/2014,Brake Oil,102,6500  
9/5/2015,Helmets,104,4500  
];
```

How to do it...

Using the `INLINE` table specified in the preceding code, we will generate a Master Calendar. We will generate the fields and Group definition using the `Declare` function.

1. In the data load editor, type in the following script:

```
Calendar:  
Declare Field Definition Tagged '$date'  
Parameters  
    first_month_of_year=1  
Fields  
    Year($1) as Year Tagged '$year',  
    Month($1) as Month Tagged '$month',  
    Date($1) as Date Tagged '$date',  
    Week($1,first_month_of_year) as Week Tagged '$week'  
  
Groups  
Year,Month,Date type collection as YearMonthDate;
```

2. Once the `Calendar` definition is created, it needs to be linked back to the date field using the `Derive` function. Insert the following statement in the script and reload the application:

```
Derive Fields from Fields InvoiceDate using Calendar;
```

3. On a new sheet, click on edit and then on the Fields tab  on the Assets panel to the left. At the bottom of the panel you will see there is a new tab for the time and date functions. Once you expand this, you should be able to see all the fields we created under the `Declare` statement.

How it works...

The Declare function is used to create the Calendar definition and tag it to \$date. The Calendar definition is then used to derive related dimensions such as **Year**, **Month**, **Week**, and so on.

The parameter `first_month_of_year` indicates what the first month of the year should be. It contains comma-separated values, but it is optional and can be skipped if needed.

Next, we define the fields we want to generate in the Calendar table. The `$1` represents the data field from which the date field will be generated, which is `InvoiceDate` in our case.

When the field definition is used, a comma-separated list of fields is generated. The Derive function is used in order to generate the derived fields such as **Year**, **Month**, and so on from the `InvoiceDate` field. The groups are defined at the end of the script that creates a drilldown group for **Year**, **Month**, and **Date**.

There's more...

The Derive function can be used to link back the Calendar to multiple dates separated by a comma. For example, "derive fields from fields `InvoiceDate, ShippingDate` using `Calendar`".

Similar to the resident load, a Calendar table can be loaded again in the script. We can change the parameter value of the first month of the year to 3. The earlier value of the parameter is overridden by doing this. This is achieved with the following commands:

```
MyCalendar:  
DECLARE FIELD DEFINITION USING Calendar WITH  
    first_month_of_year=3;  
DERIVE FIELDS FROM FIELDS InvoiceDate USING MyCalendar;
```

See also

- ▶ *Using the Peek() function to create a currency Exchange Rate Calendar*

Setting up a moving annual total figure

A **moving annual total (MAT)** is the total value of a variable, such as sales figures for a product, over the course of the previous 12 months. This is a rolling yearly sum, so it changes at the end of each month with data from the new month added to the total and data from the first month of the period taken away read more about moving annual total (MAT), at http://www.pmlive.com/intelligence/healthcare_glossary_211509/Terms/m/moving_annual_total_mat.

Getting ready

We are going to make use of variables in this recipe. We will define three variables in the script: vMonthFormat, vRolling12Months, and vMaxMonth.

Load the following script into your Qlik Sense application:

```
LET vMonthFormat = 'MMM-YYYY';
LET v12MonthsBack = 'Date(AddMonths(max([MonthYear]), -
12),$(vMonthFormat))';
LET vMaxMonth='Date(max([MonthYear]),$(vMonthFormat))';

Sales:
LOAD
Date(Date#(MonthYear, 'MMYYYY'), 'MMM-YYYY') as MonthYear,
Month(Date#(MonthYear, 'MMYYYY')) as Month,
Year(Date#(MonthYear, 'MMYYYY')) as Year,
Sales INLINE [
MonthYear, Sales
Jan2014, 1000
Feb2014, 1520
Mar2014, 1600
Apr2014, 3000
May2014, 2500
Jun2014, 4500
Jul2014, 6000
Aug2014, 6500
Sep2014, 7800
Oct2014, 6800
Nov2014, 3000
Dec2014, 2500
Jan2015, 750
Feb2015, 1200
Mar2015, 800
Apr2015, 600
May2015, 2100
Jun2015, 3500
Jul2015, 4700
Aug2015, 2100
Sep2015, 3500
Oct2015, 4700
];
```

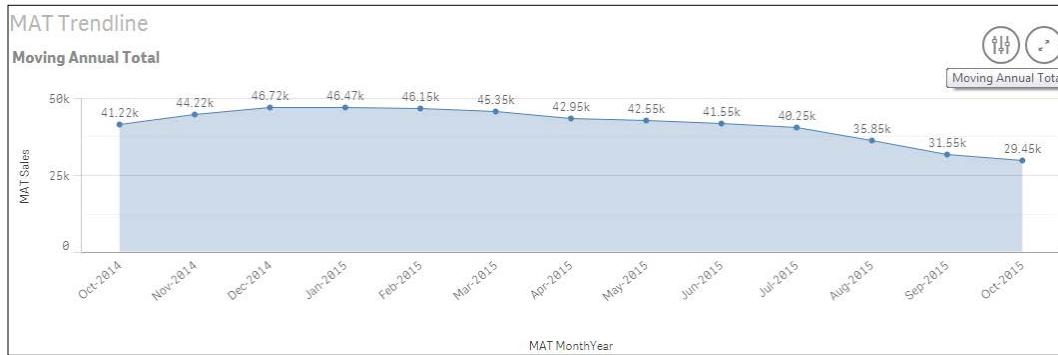
Useful Functions

```
FOR vMonth = 0 to 11
  MATMonthYear:
    LOAD
      [MonthYear],
      Date(AddMonths([MonthYear], $(vMonth)), '$(vMonthFormat)') as [MAT
      MonthYear]
    RESIDENT Sales
    WHERE AddMonths([MonthYear], $(vMonth)) < today()
    ;
    next
```

How to do it...

1. Once the data is loaded, open the **App overview** window and create a new sheet.
2. Enter the Edit mode by clicking on  **Edit**.
3. Drag across the  **Line chart** object from the Assets panel on the sheet.
4. Name it Moving Annual Total.
5. Add **[MAT MonthYear]** as the dimension.
6. Next, add the following measure and label it **MAT Sales**:

```
SUM({<[MAT
MonthYear]={">=$(vRolling12Months)<=
$(vMaxMonth)">}Sales)
```
7. Save the application and click on  **Done**.
8. Under **Appearance**, select the chart style as  **Area**.
9. Check the **Show Data** points.
10. Switch on the **Value Labels** options to show values on each data point.
11. The final trendline chart should look like this:



How it works...

The **Moving Annual Total** curve helps in smoothing out the spikes that occur in a single month by making use of the annual totals. This is achieved by calculating the rolling 12 months accumulated sales data for each data point.

We create a **MATMonthYear** field. You will notice that when we select any month and year value in this field, it associates the field value to the current MonthYear and the 11 MonthYears prior to the current, in the MonthYear field.

In the **MAT Sales** expression, we make sure that the rolling 12 months are always shown in the chart. This is achieved by restricting the **MATMonthYear** values shown in the chart between the `vRolling12Months` and the `vMaxMonth` variables.

Selecting any **MATMonthYear** will result in the trendline chart populating the MAT figures for the selected month and 11 months prior to that.

There's more...

There is a similar concept known as cumulative sums, which we discussed in the recipe *Using the Rangesum() function to plot cumulative figures in trendline charts*. However, there is a glaring difference between the two. While cumulative takes into consideration all the previous months and years to the current, a Moving Annual Total will always consider the previous 12 months. In a way it is a rolling 12 month sum at any given point of time.

See also

- ▶ *Using the Rangesum() function to plot cumulative figures in trendline charts*

Using the For Each loop to extract files from a folder

Picture a scenario where the month end sales data in an organization is stored in a folder on the network from where it needs to be picked up for reporting purposes.

Control statements such as *For Each next* can be used in Qlik Sense as an approach towards script iteration. The following recipe deals with extracting files in Qlik Sense from a folder, processing them to create QVD files and then transferring the source files to another folder. In the process, we will also deal with the incremental update of the QVD.

Getting ready

This recipe requires the Legacy mode to be activated. The steps are as follows:

1. To activate the Legacy mode, open the `Settings.ini` file under `C:\Users\<username>\Documents\Qlik\Sense`.
2. Change the value of the `StandardReload` parameter from 1 to 0.
3. For this recipe, we make use of four Excel files: `January.xlsx`, `February.xlsx`, `March.xlsx`, and `April.xlsx`. These files are provided with the chapter and can be downloaded from the Packt Publishing website.
4. Save the file `January.xlsx` under `c:\QlikSense`. If you are not able to write to this location, then you may change the storage location for the file. Note that in this case you will have to make relevant changes in the file location paths for the load script discussed in the *How to do it...* section for this recipe.
5. Create another folder named `Processed` inside the `QlikSense` folder we created in step 1. The path for the folder would be `c:\QlikSense\Processed`.
6. Create a third folder named `QVD` inside the `QlikSense` folder created in step 1. The path for the folder would be `c:\QlikSense\QVD`.

How to do it...

1. Create a new Qlik Sense application.
2. Open the data load editor.
3. Load the following script:

```
For each File in filelist ('C:\QlikSense\*.xlsx')
```

```
ProdSales:  
LOAD
```

```
left(FileBaseName(),18) AS ProdSalesFileName,
filename() as FileName,
[Product],
[Sales]
FROM [$File]
(ooxml, embedded labels, table is Sheet1)
WHERE Sales >250000;

Execute cmd.exe /C move "$(File)" "C:\QlikSense\Processed";

next File

SET rowCount = 0;
LET rowCount = NoOfRows('ProdSales');

IF rowCount > 0 AND Alt(FileSize('C:\ QlikSense
\QVD\ProdSales.QVD'),0) > 0 THEN
Concatenate(ProdSales)

LOAD * FROM C:\\QlikSense\\QVD\\ProdSales.QVD (qvd);

STORE ProdSales INTO C:\\QlikSense\\QVD\\ProdSales.QVD;

ELSE

STORE ProdSales INTO C:\\QlikSense\\QVD\\ProdSales.QVD;

END IF

DROP TABLE ProdSales;

LOAD * FROM C:\\QlikSense\\QVD\\ProdSales.QVD (qvd);
```

4. Now, add the remaining three Excel files, that is, February.xlsx, March.xlsx, and April.xlsx, to the source location; in the case of this recipe, it is c:\\QlikSense.
5. Load the script again. You will notice that all the files have been processed and moved to the processed folder. At the same time, the new data is appended to the ProdSales.QVD file.
6. In order to test the data loaded into the QVD, go to **App overview** and create a new sheet.
7. Drag a table object onto the sheet.

Useful Functions

8. Add **ProdSalesFileName** as the first dimension and label it Month.
9. Add **Product** as second dimension.
10. Add the following expression and label it as Sales:
Sum (Sales)
11. The resultant table would look like the following, with each month showing records only with Sales >250000:

| Month | Product | Sales |
|---------------|---------|----------------|
| Totals | | 6882967 |
| April | 2589 | 589686 |
| April | 4545 | 526352 |
| April | 7852 | 323256 |
| February | 2563 | 658968 |
| February | 2589 | 489868 |
| February | 7852 | 451185 |
| January | 4545 | 658936 |
| January | 7852 | 458788 |
| January | 2563 | 456698 |
| January | 4568 | 452658 |
| January | 7856 | 452563 |

How it works...

The for each next loop iterates through each file in the Source folder and processes it to pick up records with sales greater than 250,000. Once processed, the files are transferred to the processed folder using the command prompt.

The if condition checks for the row count of the processed file. If it is greater than zero then the file is concatenated to the existing ProdSales.QVD file. The LOAD statement inside the if condition has a WHERE not exists clause which makes sure to append only new files to the QVD.

Using the Peek() function to create a currency Exchange Rate Calendar

Organizations dealing in multiple currencies may use a web service to extract the exchange rates. They may even store the currency exchange rates in Excel files or sometimes in a database table. The exchange rates for any currency may be stored only for each RateStartDate that is for the day when the rate changes its value. However, for our reporting purposes we need exchange rates for each day and not just for the day when the rate changes. For this purpose, it is beneficial to create an Exchange Rate Calendar.

Getting ready

Create a new Qlik Sense application and load the following script into your Qlik Sense application:

```
ExchangeRateTemp:  
LOAD FromCurrency, ExchangeRate,  
DATE(Date#(RateStartDate, 'DD/MM/YYYY')) as RateStartDate INLINE [  
FromCurrency, ExchangeRate, RateStartDate  
EUR, 0.687, 01/08/2012  
EUR, 0.757, 02/09/2012  
EUR, 0.74, 08/09/2013  
EUR, 1.10, 24/10/2014  
SGD, 0.52, 01/08/2012  
SGD, 0.68, 27/02/2014  
SGD, 0.88, 28/03/2015  
USD, 0.75, 14/12/2013  
USD, 0.77, 16/01/2014  
USD, 0.85, 26/06/2015  
];
```

How to do it...

We will now generate the end dates for each currency exchange rate:

1. Load the following script to generate the RateEndDate for each exchange rate:

```
ExchangeRate:  
LOAD  
FromCurrency,  
ExchangeRate,  
Date (RateStartDate) AS RateStartDate,  
If (FromCurrency=Peek (FromCurrency), Date (Peek  
(RateStartDate)-1), Today ()) AS RateEndDate  
RESIDENT  
ExchangeRateTemp  
ORDER BY FromCurrency, RateStartDate DESC;  
  
DROP TABLE ExchangeRateTemp;
```

2. Go to the **App overview** window and open a new sheet.
3. Enter the Edit mode by clicking on .

Useful Functions

4. Drag the Table object onto the screen and add all the four dimensions to it. Promote **RateStartDate** to the top of the sorting order and set the sort order as numeric ascending.
5. The result would be as follows:

| FromCurrency | ExchangeRate | RateStartDate | RateEndDate |
|--------------|--------------|---------------|-------------|
| EUR | 0.687 | 01/08/2012 | 01/09/2012 |
| EUR | 0.757 | 02/09/2012 | 07/09/2013 |
| EUR | 0.74 | 08/09/2013 | 23/10/2014 |
| EUR | 1.10 | 24/10/2014 | 27/07/2015 |
| SGD | 0.52 | 01/08/2012 | 26/02/2014 |
| SGD | 0.68 | 27/02/2014 | 27/03/2015 |
| SGD | 0.88 | 28/03/2015 | 27/07/2015 |
| USD | 0.75 | 14/12/2013 | 15/01/2014 |
| USD | 0.77 | 16/01/2014 | 25/06/2015 |
| USD | 0.85 | 26/06/2015 | 27/07/2015 |

6. As we can see, every record for a currency now has a rate end date.
7. We will now use the RateStartDate and RateEndDate fields as our base dates for the Exchange Rate Calendar.
8. Now, copy and paste the following script after the `DROP TABLE ExchangeRateTemp` statement:

```
-----  
// Generate calendar dates  
-----  
  
LET ExStartDate = Num(Peek('RateStartDate', -1,  
    ExchangeRate));  
LET ExEndDate = Num(Peek('RateEndDate', 0, ExchangeRate));  
  
ExchangeRateCalendar:  
LOAD  
Date($(ExStartDate) + RecNo() - 1) AS ExchangeRateDate  
AUTOGENERATE  
($(ExEndDate) - $(ExStartDate) + 1);  
  
-----  
// INTERVAL MATCH JOIN the month records to the calendar  
// table  
-----  
  
LEFT JOIN (ExchangeRateCalendar)
```

```
INTERVALMATCH (ExchangeRateDate)
LOAD
RateStartDate,
RateEndDate
RESIDENT
ExchangeRate;

LEFT JOIN (ExchangeRateCalendar)
LOAD * RESIDENT ExchangeRate;

DROP TABLE ExchangeRate;

ExchangeRate:
LOAD
FromCurrency,
ExchangeRateDate,
ExchangeRate
RESIDENT
ExchangeRateCalendar;

DROP TABLE ExchangeRateCalendar;
```

9. Again create a Table object on the sheet and get all the dimensions from the ExchangeRate table.
10. We will have exchange rates for each of the missing dates as well as shown in the following screenshot:

| FromCurrency | ExchangeRate | ExchangeRateDate |
|--------------|--------------|------------------|
| EUR | 0.74 | 14/12/2013 |
| EUR | 0.74 | 15/12/2013 |
| EUR | 0.74 | 16/12/2013 |
| EUR | 0.74 | 17/12/2013 |
| EUR | 0.74 | 18/12/2013 |
| EUR | 0.74 | 19/12/2013 |
| EUR | 0.74 | 20/12/2013 |
| EUR | 0.74 | 21/12/2013 |
| EUR | 0.74 | 22/12/2013 |
| EUR | 0.74 | 23/12/2013 |
| EUR | 0.74 | 24/12/2013 |
| EUR | 0.74 | 25/12/2013 |
| EUR | 0.74 | 26/12/2013 |
| EUR | 0.74 | 27/12/2013 |

How it works...

The main purpose of creating this exchange rate calendar is to tag the exchange rates to every missing date in the range.

The initial data only comes with the rate start dates. So we create a rate end date for each exchange rate using the `Peek()` function. The `Peek()` function checks for the last read record for `FromCurrency` and if it matches, it generates a rate end date of `current RateStartDate -1`. If `FromCurrency` doesn't match, then the rate end date is set to today's date.

Using these start and end dates, the calendar is generated.

There's more...

The exchange rate calendar generated in the preceding recipe can be set for a daily update and stored in a QVD file that can then be used in any Qlik Sense application involving monetary analysis.

See also

- ▶ *Using the Peek() function to create a Trial Balance sheet*

Using the Peek() function to create a Trial Balance sheet

A Trial Balance sheet captures the activity across different accounts of a company with regards to the opening and closing balances. The following recipe focuses on creation of a trial balance sheet in Qlik Sense.

Getting ready

The recipe will make use of the `TrialBalance.xlsx` file, which can be downloaded from the Packt Publishing website.

Store the file on your system at the following location `C:/QlikSense`.

How to do it...

1. Create a folder connection to the `Trial Balance.xlsx` file. Name it `QlikSenseCookBook _TB`.

2. Load the data from the TrialBalance.xlsx file in the Qlik Sense file. We need to make use of the cross table functionality to load the data in a proper format:

```
Let vMaxMonth=Max(Month);  
  
TrialBalancetemp:  
CrossTable(Month, Amount, 4)  
LOAD [Company Number],  
    [Account Number],  
    [Year],  
    Forwarded,  
    [January],  
    [February],  
    [March],  
    [April],  
    [May],  
    [June],  
    [July],  
    [August],  
    [September],  
    [October],  
    [November],  
    [December]  
FROM [lib://QlikSenseCookBook_TB/Trial Balance.xlsx]  
(ooxml, embedded labels, table is Sheet1);
```

3. Next, we will generate the Month and the MonthYear field in a resident load. Copy and paste the following script:

```
TrialBalancetemp1:  
NoConcatenate LOAD  
[Company Number],  
[Account Number],  
Forwarded,  
Year,  
Month(Date#(Month, 'MMM')) as Month,  
Date(MakeDate(Year, Month(Date#(Month, 'MMM'))), 'MMM YYYY')  
as MonthYear,  
Amount  
Resident TrialBalancetemp;  
DROP Table TrialBalancetemp;
```

Useful Functions

4. The final step is to create the **Opening Balance** and **Closing Balance** fields using the **Peek ()** function. Copy and paste the following script in the editor:

```
TrialBalance:  
NoConcatenate LOAD  
CompanyAccountKey,  
[Company Number],  
[Account Number],  
MonthYear,  
Year,  
Month,  
Amount,  
if(Rowno() = 1 OR CompanyAccountKey <>  
Peek(CompanyAccountKey), Forwarded, Peek(Closing)) as  
Opening,  
    if(Rowno() = 1 OR CompanyAccountKey <>  
    Peek(CompanyAccountKey), Forwarded + Amount,  
    Peek(Closing) + Amount) as Closing  
;  
NoConcatenate LOAD  
[Company Number] & '_' & [Account Number] as  
CompanyAccountKey,  
[Company Number],  
[Account Number],  
Year,  
Month,  
MonthYear,  
Forwarded,  
Amount  
Resident TrialBalancetemp1  
Order By [Company Number], [Account Number], MonthYear;  
DROP Table TrialBalancetemp1;
```

5. Load the data and save the file. Open **App overview** by clicking on the Navigation dropdown  at the top-left corner.
6. Add the Table object to the sheet.
7. Add **MonthYear**, **Company Number**, and **Account Number** as dimensions.

8. Next, we will add the expressions for measures. We specify a range of months in the set analysis expression. When we define the range, it is enclosed within double quotes (" "). If you try to copy this expression and paste it in the Qlik Sense expression editor, sometimes the double quotes are not copied in the correct format. If the format for the quotes is incorrect, the vMaxMonth variable is highlighted in purple. In this case, the user must make sure that a proper format of double quotes is in place.
9. Add the first expression to the table and label it Opening:
 $\text{Sum}(\{\text{Month}=\{"<=\$ (\text{vMaxMonth})"\}\} \text{ Opening})$
10. Add the second expression to the table and label it Amount:
 $\text{Sum}(\{\text{Month}=\{"<=\$ (\text{vMaxMonth})"\}\} \text{ Amount})$
11. Add the third expression to the table and label it Closing:
 $\text{Sum}(\{\text{Month}=\{"<=\$ (\text{vMaxMonth})"\}\} \text{ Closing})$
12. Under **Sorting**, promote **Account Number** to the top and set the sort order as numerically ascending.
13. Promote **Company Number** to the second position in sorting and set the sort order as numerically ascending.
14. The final table report will look like this:

| TrialBalance | | | | | |
|---------------|----------------|----------------|---------------|-------------|---------------|
| MonthYear | Company Number | Account Number | Opening | Amount | Closing |
| Totals | | | 151000 | 9000 | 160000 |
| Jan 2015 | 1 | 1001 | 10000 | 1000 | 11000 |
| Feb 2015 | 1 | 1001 | 11000 | 1000 | 12000 |
| Mar 2015 | 1 | 1001 | 12000 | 1000 | 13000 |
| Apr 2015 | 1 | 1001 | 13000 | -1000 | 12000 |
| May 2015 | 1 | 1001 | 12000 | 1000 | 13000 |
| Jun 2015 | 1 | 1001 | 13000 | 1000 | 14000 |
| Jul 2015 | 1 | 1001 | 14000 | 1000 | 15000 |
| Aug 2015 | 1 | 1001 | 15000 | 1000 | 16000 |
| Sep 2015 | 1 | 1001 | 16000 | 1000 | 17000 |
| Oct 2015 | 1 | 1001 | 17000 | 1000 | 18000 |
| Nov 2015 | 1 | 1001 | 18000 | 1000 | 19000 |

How it works...

The script uses a `rowno()` function and a `Peek()` function to calculate the **Opening** and **Closing** balances.

The `rowno()` function determines the position of the current row. If we are at the first row, then the **Forwarded Amount** is taken as the opening balance. If the company and account have changed, then we use the `Peek()` function to determine the previous closing balance, which is taken as the opening balance.

Similarly, if we are at the first row, then the **Forwarded Amount + Amount** added for the particular month, is taken as the closing balance. If the company and account have changed, then we use the `Peek()` function to determine the previous closing balance and add this value to the amount to get the final closing balance.

See also

- ▶ *Using the Peek() function to create a currency Exchange Rate Calendar*

6

Set Analysis

In this chapter, we will focus on the concept of Set Analysis and its use in Qlik Sense. We will cover the following topics:

- ▶ Cracking the syntax for Set Analysis
- ▶ Using flags in Set Analysis
- ▶ Using the = sign with variables in Set Analysis
- ▶ Point in time using Set Analysis
- ▶ Using comparison sets in Set Analysis
- ▶ Using embedded functions in Set Analysis
- ▶ Creating a multi-measure expression in Set Analysis
- ▶ Using search strings inside a set modifier
- ▶ Capturing a list of field values using a `concat()` function in Set Analysis
- ▶ Using the element functions `P()` and `E()` in Set Analysis

Introduction

I will say it outright that Set Analysis is one of the most important technical features of Qlik solutions. It allows you to do things dynamically that just won't be possible with the default selections you have made. Set analysis can be termed as **Selection analysis**. The user tells Qlik Sense what set of records need to be picked for calculation which is similar to making a selection from a Filter pane or active objects. The only difference is that you define the selection inside the calculation. So that the expression can still look at the records you specified inside the Set Analysis expression even if you clear all the default selections.

Cracking the syntax for Set Analysis

Set Analysis is a very powerful concept in Qlik Sense. In very simple terms, each Set contains a "group" of selected dimensional values. The sets allow the users to create independent selections, other than the one being used in the active Qlik Sense objects. The aggregations inside the Set are compared with current selections to get the desired results.



Any set that has been created in Qlik Sense only alters the context of the expression that uses it. Unless they are referencing label names inside the same visualization, all expressions using the set syntax are independent of each other. As such, basic expressions not using the Set Analysis will react to the normal selections made inside the Qlik Sense document.

A Set Analysis expression consists of three main parts:

1. Set identifiers for example \$, 1, 1-\$, and so on
2. Set operators
3. Set modifiers (optional)

The set expression is defined inside curly brackets { }. The set identifiers are separated from the modifiers by angular (< >) brackets.

Set identifiers define the relationship between the set expression and the field values or the expression that is being evaluated (Qlik, help).

The set modifier is made up of one or several field names, each followed by a selection that should be made on the field (Qlik, help).

For example, to compare the current year versus last year sales for three countries, we can write the following Set Analysis expression:

```
Sum({$<Year={2014,2015},Country={'USA','UK','GERMANY'}>}Sales)
```

Getting ready

Load the following script that gives information on the Sales values for four customers:

```
Sales:
LOAD * INLINE [
Customer,Month,Volume,Sales
ABC,Jan,100,7500
DEF,Feb,200,8500
GHI,Mar,400,12000
JKL,Apr,100,4500
];
```

The following recipe will explain the basics of set expression. The aim is to retrieve customers with volumes greater than or equal to 200.

How to do it...

1. Drag across the Table object onto the Sheet from the Assets panel. Name it Set Analysis.
2. Add **Customer** as Dimension.
3. Add **Sum(Sales)** as the measure and label it as Sales.
4. In order to define the set, open the Expression editor window by clicking on the button.
5. Start constructing the set expression by first inserting the curly brackets {} just before the word Sales.
6. Inside the curly bracket, put in the set identifier \$.
7. Finally, we will define the set modifier. As mentioned earlier, the modifiers are separated from the identifier using angular brackets < >. Insert < > after the \$ sign. Type in Volume = {} inside the angular brackets.
8. Inside the angular brackets after Volume, type in >=200. Note the double quotes.
9. The final Set Analysis expression will look similar to the following:
`Sum({$<Volume = {">=200"}>} Sales)`
10. Click when finished.
11. The table should look similar to the following:

| Set Analysis | | |
|---------------|--|--------------|
| Customer | | Sales |
| Totals | | 20500 |
| DEF | | 8500 |
| GHI | | 12000 |

How it works...

The set identifier \$ represents the records for the current selection.

The set modifier retrieves the records for customers **DEF** and **GHI** who have the volume of 200 and 400 respectively. It is mandatory to use double quotes while specifying a range of values in the set modifier. Hence, in our case the value `>=200` is in double quotes.

There's more...

A set modifier can consist of multiple field names with selections made on them. We can also exclude selections in a particular field by specifying the field name followed by an = sign. For example, if we want to exclude the month selection then our expression will become:

```
Sum({$<Month=,Volume ={">=200"}>} Sales)
```

See also

- ▶ *Creating a multi-measure expression in Set Analysis*

Using flags in Set Analysis

Set Analysis expressions tend to become overly complex when there are too many comparison sets and conditions put in place. In order to reduce the complexity, one can make use of the flags created in the script in the Set Analysis expression. The flags can be set up to be simple binary values, 0 and 1. Use of flags optimizes the performance of frontend calculations. The following recipe explores this possibility by creating flags in the script to identify the "On-time" and "Late" shipments.

Getting ready

For the purpose of this recipe, we will be using an inline data load which contains shipment details for each customer. Load the following script within the Qlik Sense data load editor:

```
SalesTemp:  
LOAD DATE(Date#(DeliveryDate,'DD/MM/YYYY')) AS DeliveryDate,  
DATE(Date#(ShipmentDate,'DD/MM/YYYY')) AS ShipmentDate,  
Invoiceno.,Customer,Month,Sales INLINE [  
Invoiceno.,Customer,Month,DeliveryDate,ShipmentDate,Sales  
101,ABC,Jan,01/01/2015,29/12/2014,10000  
102,ABC,Feb,02/02/2015,25/01/2015,10000  
103,ABC,Mar,03/03/2015,02/03/2015,12000  
104,ABC,Apr,04/04/2015,24/01/2015,10000  
105,DEF,Feb,03/02/2015,03/02/2015,25000  
106,DEF,Mar,25/03/2015,21/03/2015,25000  
107,DEF,Apr,18/04/2015,14/04/2015,25000  
108,GHI,Jan,24/01/2015,18/01/2015,8500  
109,GHI,Mar,14/03/2015,09/03/2015,7000  
110,GHI,Jun,06/08/2015,07/06/2015,5000  
];
```

```
Sales:  
LOAD * ,  
IF(num(DeliveryDate) - num(ShipmentDate) >=0 AND  
Num(DeliveryDate) - num(ShipmentDate) < 5 , 1,  
IF(num(DeliveryDate) - num(ShipmentDate) >=5 AND  
Num(DeliveryDate) - num(ShipmentDate) < 25 , 2, 3)) AS  
OntimeLateFlag  
RESIDENT SalesTemp;  
DROP TABLE SalesTemp;
```

How to do it...

1. Drag across the Table object from the left-hand side Assets panel on to the sheet.
Name it **Invoiced Sales**.
2. Add the following dimensions:
 - InvoiceNo.
 - DeliveryDate
 - ShipmentDate
3. Add the following expression under data and label it as **Sales**:
`Sum({$<OntimeLateFlag={1}>}Sales)`
4. Under **Sorting**, promote **Sales** to the top.
5. Click on **Save** and .
6. The resulting table on the screen should look similar to the following:

| Invoiced Sales | | | | |
|----------------|--------------|--------------|--------------|--|
| Invoiceno. | DeliveryDate | ShipmentDate | Sales | |
| Totals | | | 97000 | |
| 105 | 03/02/2015 | 03/02/2015 | 25000 | |
| 106 | 25/03/2015 | 21/03/2015 | 25000 | |
| 107 | 18/04/2015 | 14/04/2015 | 25000 | |
| 103 | 03/03/2015 | 02/03/2015 | 12000 | |
| 101 | 01/01/2015 | 29/12/2014 | 10000 | |

Set Analysis

7. Note that only the invoices with a delivery time of less than 5 days are shown in the preceding table.

How it Works...

The calculation to identify the "On time" and "late" shipments is done in the script and it is executed only once. Every OnTime shipment is flagged as 1 and a slight delay as 2 and late as 3. Use of these flags in the frontend objects will filter the data in the table accordingly.

There's more...

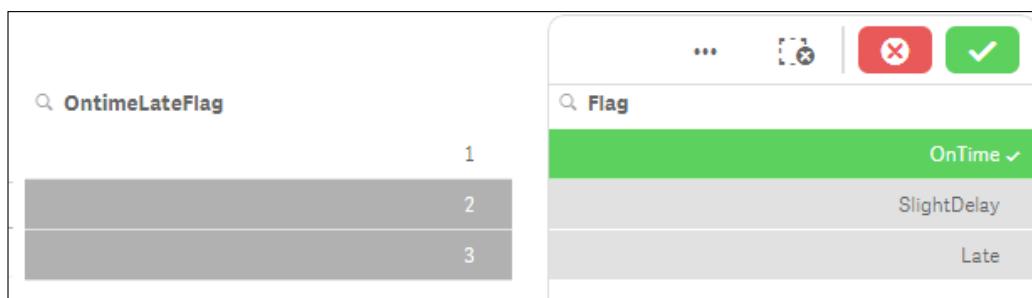
In order to give a more meaningful representation to the flags in the frontend, we may use the Dual function. A Dual() function combines a number and a string into a single record. The number representation of the record can be used to sort and also for calculation purposes, while the string value can be used for display purposes.

In order to do this:

1. Rename the Sales table to SalesTemp1.
2. Add the following Resident load:

```
Sales:  
LOAD *,  
IF(OnTimeLateFlag =1, Dual('OnTime',1),  
IF(OnTimeLateFlag =2, Dual('SlightDelay',2),  
Dual('Late',3))) As Flag  
RESIDENT SalesTemp1;  
DROP Table SalesTemp1;
```

3. Save and reload the application.
4. In the frontend, drag across the Filter pane object and add the **OnTimeLateFlag** and **Flag** dimension to it.
5. Note that every **OnTimeLateFlag** value is now associated with text:



Using flags with a string format in Set Analysis expressions may not always be the most efficient way of optimizing the performance of the Qlik Sense objects. With a big data set, using a flag with a string representation in the expression does not offer a massive advantage as far as the performance standpoint is concerned. However, if we have binary flags 0 and 1 then multiplying these flags by the measures results in a faster performance in the user interface.

Hence, we conclude the following:

- ▶ To make selections in the application, use the String representation of flags in the Filter pane objects
- ▶ To calculate a condition inside a Set Analysis expression, use the numeric representation of flags

See also

- ▶ *Using embedded functions in Set Analysis*

Using the = sign with variables in Set Analysis

We can make use of variables and calculations in the set modifiers. The following recipe explains the syntax to use variables for comparison in sets and how to effectively use the = sign in the dollar sign expansions.

Getting ready

For the purpose of this recipe, we will be using an inline data load which contain shipment details for each customer. Load the following script in the Qlik Sense data load editor. Make sure that the last record in this script has the Month set to today's month and the DeliveryDate set to today's date:

```
Let vToday=Today();  
  
Sales:  
LOAD DATE(Date#(DeliveryDate,'DD/MM/YYYY')) AS DeliveryDate,  
DATE(Date#(ShipmentDate,'DD/MM/YYYY')) AS ShipmentDate,  
Customer,Month,Volume,Sales,Supplier  
INLINE [
```

Set Analysis

```
Customer,Month,DeliveryDate,ShipmentDate,Volume,Sales,Supplier  
ABC,Jan,01/01/2015,29/12/2014,100,10000,DEF  
ABC,Feb,02/02/2015,25/01/2015,100,10000,DEF  
ABC,Mar,03/03/2015,02/03/2015,400,12000,DEF  
ABC,Apr,04/04/2015,24/01/2015,100,10000,GHI  
DEF,Feb,03/02/2015,03/02/2015,200,25000,GHI  
DEF,Mar,25/03/2015,21/03/2015,300,25000,GHI  
DEF,Apr,18/04/2015,14/04/2015,200,25000,ABC  
GHI,Jan,24/01/2015,18/01/2015,200,8500,ABC  
GHI,Mar,14/03/2015,09/03/2015,200,7000,ABC  
GHI,Jun,06/08/2015,07/06/2015,200,5000,ABC  
];
```

How to do it...

1. Drag across a Table object from the Assets panel onto the sheet.
2. Add **Customer** as dimension.
3. Now add the following calculation as the measure and label it Sales:
`Sum({$<DeliveryDate={$vToday}>}Sales)`
4. Click on **Save** and then **Done**.
5. The resultant table is similar to the following figure with only one record for customer **GHI**:

| Sales | | |
|---------------|--|-------------|
| Customer | | Sales |
| Totals | | 5000 |
| GHI | | 5000 |

6. Next, update the Sales calculation as shown:

```
Sum({$<DeliveryDate={$TODAY()}>}Sales)
```

7. When you save this calculation, Qlik Sense won't be able to interpret the result and we will get the following output:

| Sales | |
|----------|-------|
| Customer | Sales |
| Totals | 0 |
| - | 0 |

8. Tweak the sales calculation by adding a = sign in front of TODAY():

```
Sum({$<DeliveryDate={$(=TODAY())}>}Sales)
```

9. The result will be as seen earlier with one record for the customer **GHI**.

How it works...

We have defined the vToday variable in the script. This variable stores the values for today's date. When we use this variable inside the set modifier, we just use a simple \$ sign expansion.

The vToday variable is calculated before the script is executed. However, Qlik Sense fails to interpret the result when we use the TODAY() function inside the set modifier instead of vToday. The reason being that the \$ sign expansion needs to perform a calculation in the form of TODAY() and without the preceding = sign the date for today won't be calculated.

Hence, we proceed to TODAY() with the = sign. Once the = sign is in place, the sales for customers with today's delivery date are calculated.

If we are not using any calculation inside the set modifier then the variable can be defined with or without the = sign.

See also

- ▶ *Point in time using Set Analysis*

Point in time using Set Analysis

How is this month looking compared to the last? This is one of the most common questions asked in BI solutions. In this recipe, we will build two charts and both will compare one year to the other. The first chart expression will limit the range of data and make use of the **Year** dimension. The second chart will not use the **Year** dimension but will build the year comparison directly into the expression itself.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives yearly sales information for different fruits. Load the following data into the data load editor:

```
Data:  
LOAD * INLINE [  
    Fruit, Year, Sales  
    Apple, 2013, 63  
    Apple, 2014, 4  
    Cherry, 2014, 1150  
    Cherry, 2013, 1180  
    Fig, 2013, 467  
    Fig, 2013, 374  
    Fig, 2014, 162  
    Fig, 2014, 267  
    Fig, 2014, 586  
    Orange, 2013, 10  
    Orange, 2013, 50  
    Orange, 2013, 62  
    Orange, 2013, 131  
    Orange, 2013, 145  
    Orange, 2014, 93  
    Orange, 2014, 102  
    Pear, 2013, 27  
    Pear, 2013, 157  
    Pear, 2013, 384  
    Pear, 2014, 489  
    Pear, 2014, 782  
    Plum, 2013, 148  
    Plum, 2014, 36  
    Plum, 2014, 412  
    Plum, 2012, 700  
];
```

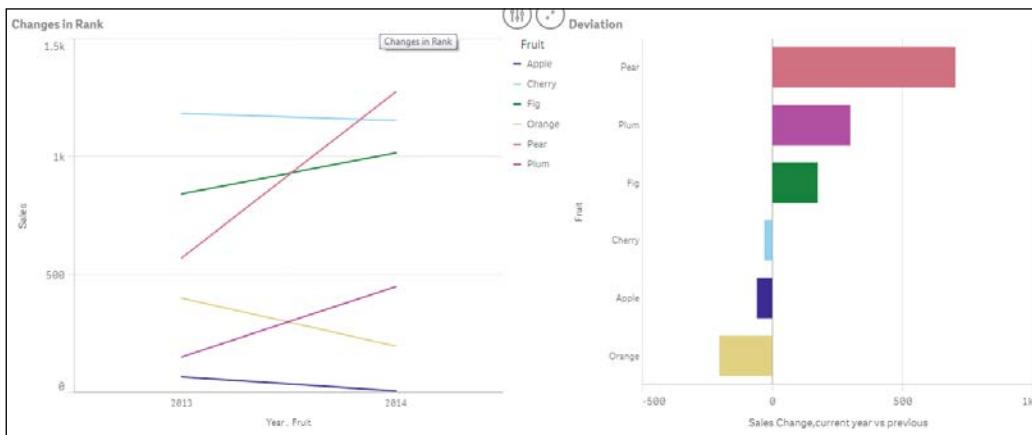
How to do it...

1. Drag a line chart object from the Assets panel onto the content area. Title it Changes in Rank.
2. Add **Year** as a dimension.
3. Add **Fruit** as a dimension.
4. Add the following expression and label it as Sales:

```
sum({<Year={">=$ (=MAX(Year) -1)<=$ (=MAX(Year))"}>}Sales)
```

5. Under **Appearance | Colors and legend**, switch on **Show Legend** and click on **Save**.
6. Next, drag a bar chart onto the content area. and title it as Deviation.
7. Add **Fruit** as a dimension.
8. Add the following expression and label it as Sales Change, current year vs previous.

$$\text{sum}(\{\text{Year}=\$\{=\text{MAX}(\text{Year})\}\})\text{Sales}) - \text{sum}(\{\text{Year}=\$\{=\text{MAX}(\text{Year})-1\}\})\text{Sales})$$
9. Select **Horizontal** under **Appearance | Presentation**.
10. Under **Appearance | Colors and legend**, toggle the colors button to uncheck **Auto** colors and switch on custom colors. Select **By dimension** and check the **Persistent** colors button.
11. Your graphs will look similar to the following image:



How it works...

The first Set Analysis expression makes use of a search string; thus, defining the set of records we want to calculate across. A pseudo code will read like this.

```
Sum where the Year = {"Search for records that fulfill a
particular requirement "}
```

Using the double quotes denotes that we will be doing a search starting with < or >. Only values that fulfill the numeric requirement will be matched.

Set Analysis

In our example, we define the numeric requirement of the search string dynamically using the following code:

```
={ ">=$(=MAX(Year)-1) <=$(=MAX(Year)) }
```

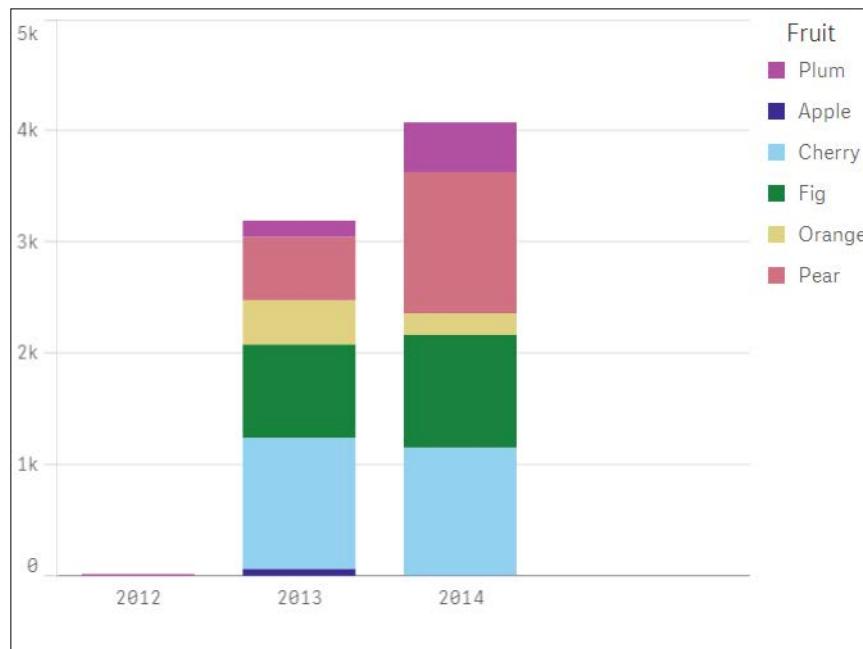
This code evaluates the max year and the year previous to that. If we changed the `-1` in the preceding code to `-2` the calculation will cover three years and not just two; this is the benefit of using search strings in Set Analysis. For the second chart, we have not used a search string but specified literals. We have kept the dynamic part of the expression as:

```
{$(=MAX(Year))}
```

Now, the max year available will be picked up automatically as opposed to saying `Year={2015}` and updating the expression next year.

Using comparison sets in Set Analysis

The following figure is of a stacked bar chart, a standard way of comparing separate entities. Each value that you select is displayed as a segment in each bar by year:



Using a comparative analysis lets you group the separate selections dynamically, so that you can compare them against each other. In the preceding example we can group together **Plum** and **Apple** versus **Fig** and **Orange**.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives yearly sales information for different fruits. Load the following script in the Qlik Sense data load editor:

```
Data:  
LOAD * INLINE [  
    Fruit, Year, Sales  
    Apple, 2013, 63  
    Apple, 2014, 4  
    Cherry, 2014, 1150  
    Cherry, 2013, 1180  
    Fig, 2013, 467  
    Fig, 2013, 374  
    Fig, 2014, 162  
    Orange, 2013, 131  
    Orange, 2013, 145  
    Orange, 2014, 102  
    Pear, 2014, 489  
    Pear, 2014, 782  
    Plum, 2013, 148  
    Plum, 2014, 412  
];  
  
DataIslandFruit:  
LOAD * INLINE [  
    FruitAlt  
    Apple  
    Cherry  
    Fig  
    Orange  
    Pear  
    Plum  
];
```

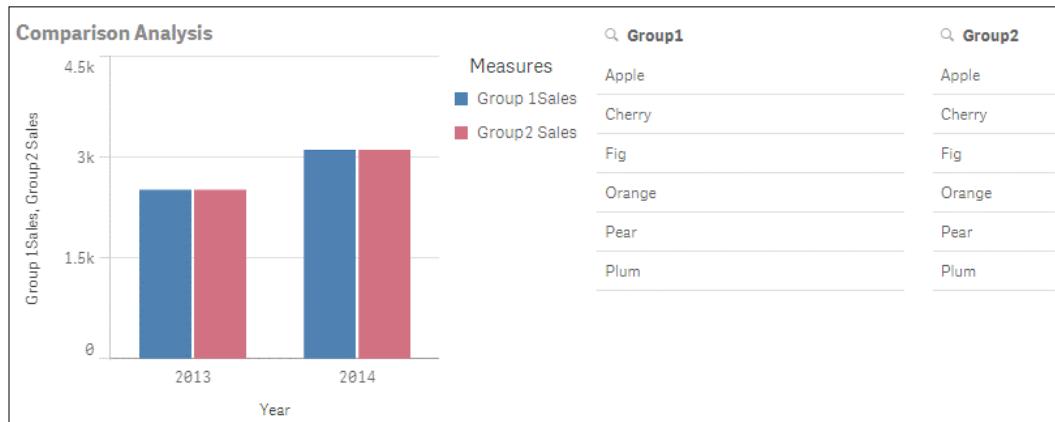
How to do it...

1. Drag a bar chart onto the content area and title it Comparison Analysis.
2. Add **Year** as a dimension.
3. Add the following expression and label it as Group 1 Sales:
`Sum(Sales)`

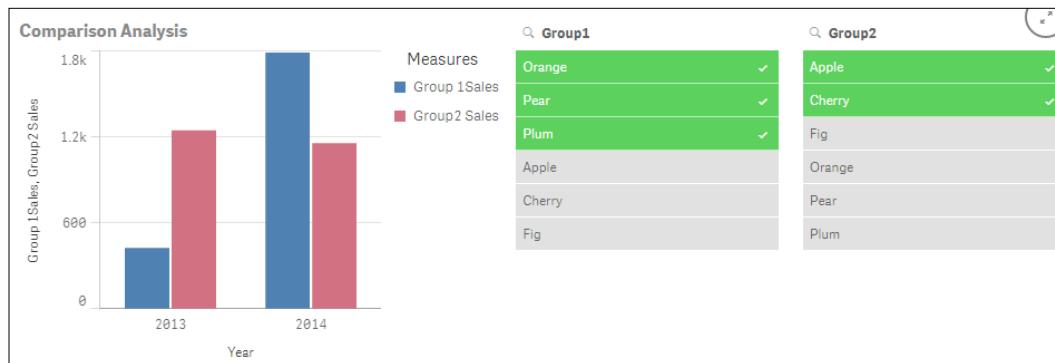
Set Analysis

4. Add the following expression and label it as **Group 2 Sales**:
`Sum({<Fruit={$ (=GetFieldSelections(FruitAlt)) }>}Sales)`
5. Under **Appearance | Colors and legend** switch on the **Show legend** option.
6. Create a Filter pane object and add the first dimension as **Fruit**. Label the dimension as **Group 1**.
7. Add **FruitAlt** as the second dimension to the Filter pane and label the dimension as **Group 2**.
8. The final chart should resemble one of the following images if you have already made the selections to test the comparative analysis.

The following is an example where no selections are made:



The following is an example where selections are made:



How it works...

The second table we loaded is what's known as a data island, this table is not connected to the rest of the data model in any way. However, we can use its contents in our Set Analysis expression to compare different groups of the same field.

The first expression is completely standard. The second expression gives the total sales where the **Fruit** field (part of the complete dataset) matches the values selected in the **FruitAlt** field (part of the disconnected data island). This method allows us to select groups of data for aggregation in our graph that we can not do normally by effectively breaking the association (green, white, and grey) using Set Analysis.

Using embedded functions in Set Analysis

As you have seen in the previous recipes, we have used functions, such as `Max()` and `GetFieldSelections()` inside our Set Analysis. Embedding the functions inside a Set Analysis expression, specifically in the rules area that defines the set of records we want to calculate across is known as dollar sign expansion.

Defining a set of records in the simplest literal form is as follows `Year= {2015}`.

The expression needs to know the year you want to use, dollar sign expansion allows us to generate the text dynamically. Understanding how to use dollar sign expansion in your Set Analysis expressions enriches the amount of analysis you can perform. Sometimes just using the function alone or specifying literals in Set Analysis is either too time consuming or adds unnecessary maintenance to the application.

Getting ready

For the purpose of this recipe, we make use product sales data as defined in the following script. Load the following data into the Qlik Sense data editor:

```
Transactions:  
Load  
    date(today()-IterNo()) AS Date,  
    Pick(Ceil(3*Rand()), 'Standard', 'Premium', 'Discount') AS  
        ProductType,  
    floor(date(today()-IterNo())) AS DateNum,  
    Round(1000*Rand()*Rand()*Rand()) AS Sales  
Autogenerate 1000  
While Rand()<=0.9 or IterNo()=1;
```

How to do it...

1. Create a new sheet and double click on the table object to add it to the main content area.
2. Add **Product Type** as a dimension.
3. Add the following measure as an expression, label it **Total Sales**:
 $\text{sum}(\text{Sales})$
4. Add the following measure as the second expression, label it **WTD**:
 $\text{sum}(\{\text{DateNum}=\text{"}>=\$ (\text{Today}() - 7)\}\text{Sales})$
5. Add the following measure as the third expression, label it **Previous WTD**:
 $\text{sum}(\{\text{DateNum}=\text{"}>=\$ (\text{Today}() - 14) < \$ (\text{Today}() - 7)\}\text{Sales})$
6. Add the following as the fourth expression, label it **Weekly Variance**:
 $(\text{COLUMN}(2) - \text{COLUMN}(3)) / \text{COLUMN}(2)$
7. For the expression in step 6, change the number formatting to **Number** and then select the percentage format from the drop-down list.
8. You should have a table that looks similar to the following image. The figures may not be similar to the following image as we are using the Rand function to generate the initial set of data in the script:

| ProductType | | Total Sales | WTD | Previous WTD | Weekly Variance |
|---------------|--|-------------------|-----------------|-----------------|-----------------|
| Totals | | £1,213,426 | £640,889 | £295,293 | 53.9% |
| Premium | | £410,761 | £218,494 | £97,393 | 55.4% |
| Standard | | £407,902 | £209,730 | £103,621 | 50.6% |
| Discount | | £394,763 | £212,665 | £94,279 | 55.7% |

How it works...

When calculating something like week to date sales, the set of records you identify in your Set Analysis expression will change every day. When you use functions such as `Today()` inside the Set Analysis expression, the literal text values that the expression uses change automatically. Ultimately using dollar sign expansion is just a replacement for the text strings that you could use.

If the date today is 06/08/2015 then. The user see the set condition as:

```
DateNum= { "}>=$ (\text{Today}() - 7)"}
```

While Qlik Sense sees the set condition as:

```
DateNum={ " >=31/07/2015" }
```

This is because the function inside the dollar sign is evaluated first and it simply expands into the text/field values that we want to calculate across.

There's more...

The fourth expression is written as:

(COLUMN(2) - COLUMN(3)) / COLUMN(2). Here we pick up the column numbers instead of the actual field names for our calculation.

We can also write the expression in the following manner:

```
([WTD] - [Previous WTD]) / [WTD] .
```

We will get a bad field name: ([WTD] - [Previous WTD]) / [WTD] at the bottom of the Expression editor window. But don't worry, as Qlik Sense will still interpret the results correctly. This chink may be ironed out in future releases of Qlik Sense.

The expression does not make use of the fields that we have loaded into the applications data model. It instead uses the expression labels we have already created for the previous calculations. This is always a best practice option if you need to use the same calculation in the same table more than once. It makes things simpler, you only have to change something once and best of all it is optimized and the calculation is already cached in RAM.

Creating a multi-measure expression in Set Analysis

Sometimes you may have groups of expressions you want to view that either they don't need to be viewed at once or you don't have the room to display them all. In these cases you do not have to go and create another sheet, you can add a control to let users select what is calculated.

The output of this recipe is similar to the preceding recipe, only with slightly different expressions to add depth of analysis in the same object.

Getting ready

For the purpose of this recipe, we make use product sales and margin data as defined in the following script. Load the following data into the data editor:

```

Transactions:
Load
Date(today()-IterNo()) AS Date,
Pick(Ceil(3*Rand()),'Standard','Premium','Discount') AS
    ProductType,
Floor(date(today()-IterNo())) AS DateNum,
Round(1000*Rand()*Rand()*Rand()) AS Sales,
Round(10*Rand()*Rand()*Rand()) AS Quantity,
Round(Rand()*Rand(),0.00001) AS Margin
Autogenerate 10000
While Rand()<=0.9 or IterNo()=1;

Measures:
LOAD * INLINE [
    Measures
    Sales
    Quantity
    Margin
];

```

How to do it...

1. Create a Filter pane object and add **Measures** as the dimension.
2. Next, drag across a table object onto the main content area.
3. Add **Product Type** as a dimension
4. Add the following expression as the first measure and label it **Total Sales**:

$$\text{sum}(\$(\text{GetFieldSelections}(\text{Measures}))$$
5. Add the following expression as the second measure and label it **WTD**:

$$\text{sum}(\{\langle \text{DateNum}=\{ " \gt;= \$(\text{Today}() - 7) " \} \gt; \} \$ (\text{GetFieldSelections}(\text{Measures}))$$
6. Add the following measure as the third expression and label it **Previous WTD**:

$$\text{sum}(\{\langle \text{DateNum}=\{ " \gt;= \$(\text{Today}() - 14) < \$(\text{Today}() - 7) " \} \gt; \} \$ (\text{GetFieldSelections}(\text{Measures}))$$
7. Add the following as the fourth expression and label it **Weekly Variance**:

$$(\text{COLUMN}(2) - \text{COLUMN}(3)) / \text{COLUMN}(2)$$

8. For the expression in step 7 change the number formatting to **Number** and then select the percentage format from the drop-down list.
9. If you come out of the Edit mode and select one value from the Filter pane object, you can see the calculation changing.
10. You should have a table that looks similar to the following image. The figures may not be exactly similar to the following image since we are using the Rand function to generate the initial set of data in the script:

Select one option below to view the figures

| ProductType | Total | WTD | Previous WTD | Weekly Variance |
|-------------|-------------|-----------|--------------|-----------------|
| Totals | £12,739,229 | 6,609,537 | £3,243,454 | 50.9% |
| Discount | £4,278,988 | 2,218,842 | £1,088,209 | 51.0% |
| Premium | £4,237,879 | 2,203,479 | £1,080,869 | 50.9% |
| Standard | £4,222,362 | 2,187,216 | £1,074,376 | 50.9% |

How it works...

Here we capture the field values that we want to calculate using a data island. When we use a data island, we simply pick an option from the measures box without filtering the data in any way. But this approach allows us to control what calculations are being returned.

The `GetFieldSelections (Measures)` function simply returns **Sales**, **Margin**, or **Quantity** depending on what you have selected. As such, writing the expression `Sum (GetFieldSelections (Measures))` means we can have any of the three options displayed just by selecting the value from the Filter pane.

As mentioned in the previous recipe, we can write the `Weekly Variance` expression using the expression labels previously defined in the table as follows:

```
( [WTD] - [Previous WTD] ) / [WTD]
```

We will get a warning for "Bad field" at the bottom of the Expression editor window. Ignore it, as this chink may be ironed out in future releases of Qlik Sense.

Using search strings inside a set modifier

A set modifier contains one or several field names that make up the set expression. We can define a *range of values* within the selection made in the set modifier. The following recipe makes use of search strings to calculate the sales for customers within a specified date range.

Getting ready

For the purpose of this recipe, we will be using an inline data load which contain shipment details for each customer. Load the following script within the Qlik Sense data load editor:

```
Sales:  
LOAD DATE(Date#(DeliveryDate,'DD/MM/YYYY')) AS DeliveryDate,  
DATE(Date#(ShipmentDate,'DD/MM/YYYY')) AS ShipmentDate,  
Customer,Month,Volume,Sales,Supplier INLINE [  
Customer,Month,DeliveryDate,ShipmentDate,Volume,Sales,Supplier  
ABC,Jan,01/01/2015,29/12/2014,100,10000,DEF  
ABC,Feb,02/02/2015,25/01/2015,100,10000,DEF  
ABC,Mar,03/03/2015,02/03/2015,400,12000,DEF  
ABC,Apr,04/04/2015,24/01/2015,100,10000,GHI  
DEF,Feb,03/02/2015,03/02/2015,200,25000,GHI  
DEF,Mar,25/03/2015,21/03/2015,300,25000,GHI  
DEF,Apr,18/04/2015,14/04/2015,200,25000,ABC  
GHI,Jan,24/01/2015,18/01/2015,200,8500,ABC  
GHI,Mar,14/03/2015,09/03/2015,200,7000,ABC  
GHI,Jun,11/06/2015,07/06/2015,200,5000,ABC  
];
```

How to do it...

1. Drag across the Table object from the Assets panel on to the sheet.
2. Add **Customer** as dimension.
3. Add the following measure, which calculates the sales for delivery dates ranging between 14/01/2015 and 14/04/2015. Label the measure as Sales:

```
Sum({< DeliveryDate = { '>=$(=DATE(Date#('14/01/2015',  
'DD/MM/YYYY')))<=$(=DATE(Date#('14/04/2015',  
'DD/MM/YYYY')))' } >} Sales)
```

4. Click on **Save** and .

5. The resultant table will be as following. Note that we get a subset of the **Sales** value based on the date range specified in the set modifier:

| Sales | | |
|---------------|--|--------------|
| Customer | | Sales |
| Totals | | 97500 |
| ABC | | 32000 |
| DEF | | 50000 |
| GHI | | 15500 |

6. Drag across the Filter pane object onto the sheet and add the **DeliveryDate** as dimension.
 7. Select any random delivery dates. Observe that the **Sales** figure for each customer remains unchanged.

How it works...

In the set modifier we specify two dates enclosed within single quotes (' '). The first date is the start date of the range and it is preceded by a >= sign, while the second date is an end date of the range and is preceded by a <= sign. We use a date function in order to interpret the strings as date. The \$ sign expansion evaluates the expression inside the bracket.

There's more...

The preceding recipe considers two static dates for the date range. We can also make the date range dynamic by tweaking our Set Analysis expression the following way:

```
Sum({<DeliveryDate =
{ ">=$(=min(ShipmentDate)) <=$(=max(ShipmentDate)) " } >} Sales )
```

Here we are comparing the delivery date to the shipment date and calculating sales for the delivery dates lying between the range of shipment dates.

For example:

1. Add **ShipmentDate** as a new dimension in the Filter pane object.
2. Select the shipment dates from **18/01/2015** to **25/01/2015**.

Set Analysis

3. The resultant table shows the sales value only for the delivery date, as **24/01/2015**:

| Sales | ShipmentDate | DeliveryDate |
|----------|--------------|--------------|
| Customer | 29/12/2014 | 24/01/2015 |
| Totals | 18/01/2015 ✓ | 02/02/2015 |
| GHI | 24/01/2015 ✓ | 04/04/2015 |
| | 25/01/2015 ✓ | 01/01/2015 |

See also

- ▶ *Using the = sign with variables in Set Analysis*

Capturing a list of field values using a concat() function in Set Analysis

While we have used the search strings in previous recipes to do numeric search, we can also do text searches by using the wild card character *. However, sometimes you might want to compare the values in one field to the values stored in another. We can also achieve this using Set Analysis and the concat() function.

Getting ready

For the purpose of this recipe, we make use product sales data as defined in the following script. Load the following script into the data load editor:

```
Transactions:  
Load *,  
    If(Len(TmpSubCategory)=0,Null(),TmpSubCategory) AS  
    SubCategory;  
Load * INLINE [  
    ProductType, Category, TmpSubCategory, Sales  
    Premium,A4,A4,300  
    Standard,A4,A4,100  
    Premium,A5,A5,500  
    Standard,A5,A5,200  
    Premium,A6,A6,1000  
    Standard,A6,A6,600  
    Premium,A1,,700  
    Standard,A1,,300  
    Premium,A2,,300  
    Premium,A3,,200  
    Standard,A3,,60  
];
```

How to do it...

1. Drag a table object onto the content area and label it as Product Sales.
2. Add **Product Type** as a dimension
3. Add the following expression as the first measure and label it as Total Sales:
`Sum (Sales)`
4. Add the following expression as the second measure and label it as Sub Category Sales:
`Sum ({<Category = ${=concat (distinct [SubCategory], ', ') } >} Sales)`
5. You should have a table that looks similar to the following image:

| Product Sales | | | |
|---------------|--|-------------|--------------------|
| ProductType | | Total Sales | Sub category Sales |
| Totals | | 4260 | 2700 |
| Premium | | 3000 | 1800 |
| Standard | | 1260 | 900 |

How it works...

The concat () function wraps around a field name; when expressed it lists every field value separated by a delimiter. As such, the function concat (Distinct Subcategory, ', ') returns A4, A5, A6, which are all the values in the sub-category field with no selections made.

Using the concat () function means you can avoid having to write out large lists of text strings in your Set Analysis expression. Even better, if these lists come from a source system where they are automatically updated with data.

Using the element functions P() and E() in Set Analysis

So far we have seen how the sets can be used to manipulate the result of an expression. To take the concept a bit further, we will now see how to use the P() and E() functions inside a Set Analysis expression. In the previous Set Analysis expressions, all field values were explicitly defined in the sets or variables or in certain cases through defined searches. The P() and E() functions make use of nested set definitions.

Set Analysis

A `P()` function returns a set of all possible values while an `E()` function returns a set of all excluded values.

Getting ready

For the purpose of this recipe, we make use customer sales data as defined in the following inline data load. Load the following script in Qlik Sense data load editor:

```
P_E:  
LOAD * INLINE [  
Customer,Month,Volume,Sales,Supplier  
ABC,Jan,100,10000,DEF  
ABC,Feb,100,10000,DEF  
ABC,Mar,400,12000,DEF  
ABC,Apr,100,10000,GHI  
DEF,Feb,200,25000,GHI  
DEF,Mar,300,25000,GHI  
DEF,Apr,200,25000,ABC  
GHI,Jan,200,8500,ABC  
GHI,Mar,200,7000,ABC  
GHI,Jun,200,5000,ABC  
];
```

How to do it...

1. On a new sheet, drag and drop the table object from the Assets panel on the left-hand side of the screen. Name the table as Possible Sales.
2. Add **Customer** and **Month** as dimensions.
3. Add the following expression for Sales:
`Sum({$<Customer=P({1<Month={'Jan'}})>}>)Sales)`
4. Click on **Save** and  **Done**.
5. The resultant table will look similar to the following. Note that it only shows all the records for customers **ABC** and **GHI**:

| Possible Sales | | | |
|-----------------------|-------|--------------|--|
| Customer | Month | Sales | |
| Totals | | 62500 | |
| ABC | Mar | 12000 | |
| ABC | Apr | 10000 | |
| ABC | Feb | 10000 | |
| ABC | Jan | 10000 | |
| GHI | Jan | 8500 | |
| GHI | Mar | 7000 | |
| GHI | Jun | 5000 | |

6. Next, create another table with the same dimensions, such as **Customer** and **Month** and name it **Excluded Sales**.
7. Add the Sales expression as follows:
 $\text{Sum}(\{\$<\text{Customer}=\text{E}(\{1<\text{Month}=\{'Jan'\}>\})>\}\text{Sales})$
8. The resultant table will look similar to the following screen shot. Note that we only have one customer **DEF** in the table:

| Excluded Sales | | | |
|-----------------------|-------|--------------|--|
| Customer | Month | Sales | |
| Totals | | 75000 | |
| DEF | Apr | 25000 | |
| DEF | Feb | 25000 | |
| DEF | Mar | 25000 | |

How it works...

1. The **P()** function selects all the possible values from the set. In the first expression:

$\text{Sum}(\{\$<\text{Customer}=\text{P}(\{1<\text{Month}=\{'Jan'\}>\})>\}\text{Sales})$

We select the customers who have made sales in the month of January.

Set Analysis

2. However, the `E()` function selects all the excluded values from the set. In the second expression:

```
Sum({$<Customer=E({1<Month= {'Jan'}})>}Sales)
```

We select the customers who have made sales in all months except January.

There's more...

The concept of `P()` and `E()` can also be used with two fields for comparison inside the nested sets.

For example: if one needs to find out all those customers where the suppliers had a volume of 300, the set expression will be defined in the following way:

```
Sum({$<Customer=p({1<Volume={300}>}Supplier)>}Sales)
```

Here, the element function `P()` returns a list of possible suppliers who had a volume of 300. The list of suppliers is then matched to the customers to make the relevant selections.

The resultant table will look similar to the following:

| Possible Sales | | | | | |
|----------------|-------|--------|--------------|--|--|
| Customer | Month | Volume | Sales | | |
| Totals | | | 20500 | | |
| GHI | Jan | 200 | 8500 | | |
| GHI | Mar | 200 | 7000 | | |
| GHI | Jun | 200 | 5000 | | |

An `E()` function in place of `P()` will result in all the customers whose suppliers never had a volume of 300:

| Excluded Sales | | | |
|----------------|-------|---------------|--|
| Customer | Month | Sales | |
| Totals | | 117000 | |
| ABC | Mar | 12000 | |
| ABC | Apr | 10000 | |
| ABC | Feb | 10000 | |
| ABC | Jan | 10000 | |
| DEF | Apr | 25000 | |
| DEF | Feb | 25000 | |
| DEF | Mar | 25000 | |

See also

- ▶ *Using embedded functions in Set Analysis*

7

Extensions in Qlik Sense®

In this chapter, we will focus on some of the advanced visualization techniques in Qlik Sense and discuss the following topics:

- ▶ Creating an HTML visualization extension for Qlik Sense®
- ▶ Defining a Properties panel in Qlik Sense® visualizations
- ▶ Creating custom components within Qlik Sense® visualizations
- ▶ Using data with extensions

Introduction

Before we jump the gun, it is expected of the user to have an intermediate level of JavaScript and HTML knowledge to develop extensions in Qlik Sense.

Qlik Sense has an extensive library of chart objects to display data. However, of late there has been an increase in the demand for custom visualizations from business users and such visualizations are used in specific circumstances. Similar to Qlikview, we can also develop visualization extension objects in Qlik Sense using open standards, such as HTML5, CSS, and JavaScript.

However, the method to create these extensions differs in Qlik Sense. Qlik Sense visualizations are compatible with an AJAX interface or any other web browser. We can also use JavaScript code from external visual libraries, such as D3 to make intuitive and user friendly extension objects.

The following recipes discuss different concepts in advanced visualizations, such as HTML extensions, custom components, and use of data with extensions.

Creating an HTML visualization extension for Qlik Sense®

To begin with, let us discuss a recipe to create a simple HTML extension in Qlik Sense. The two files that are mandatory to create any Qlik Sense extension are:

- ▶ **.JS** file: This file contains the JavaScript required to implement the extension and is built around the RequireJS framework
- ▶ **.QEXT** file: This is an extension metadata file, which contains the JSON description to be used within the desktop client

In addition to the preceding mandatory files, one can also make use of additional files, such as:

- ▶ Script files from external libraries such as D3 or Raphael
- ▶ CSS files: To add styles to the extensions
- ▶ Images, Fonts, Icons, and so on

The default directory for Qlik Sense Desktop extensions is C:\Users\[UserName]\Documents\Qlik\Sense\Extensions\.

In this example we will print the words "Hello World" on the screen using our first Qlik Sense extension. This is a common first task used when we learn various programming languages. The idea is to keep the code as simple as possible while providing information on the structure of the code and anatomy of the extension environment.



A little slice of history: Using the "Hello, World" example dates back as far 1974. The first known version in computer literature was taken from a 1974 Bell Laboratories internal memorandum on programming in C.

Getting ready

This recipe is built entirely from the *How to do it...* section and does not require data to be loaded first. We use notepad to write the code in the following examples. A suggested alternative is Notepad++, which the user can download separately. Notepad++ is a free tool that improves the readability of the code by highlighting methods, functions, and so on.

Taking into consideration the two mandatory files, let's start creating a simple extension using HTML:

1. Create a folder called as QlikSense Cookbook - Hello World to store the .JS and .QEXT files.
2. The folder should be created under C:\Users\[UserName]\Documents\Qlik\Sense\Extensions\.

How to do it...

1. In the QlikSense Cookbook - Hello World folder, create a new notepad document and add the following code:

```
{  
    "name" : "QlikSense Cookbook - Hello World",  
    "description" : "QlikSense Cookbook - Chapter 7, Recipe  
1: Hello World.",  
    "icon" : "extension",  
    "type" : "visualization",  
    "version": "1",  
    "preview" : "bar",  
    "author": "Your Name"  
}
```

2. Click on **Save As** in the notepad document and change the **Save as type** to **All Files (*.*)**.
3. Call the file name QlikSense-Cookbook-C7-R1-HelloWorld.qext.
4. Create another blank notepad file in the same location and add the following code:

```
define( [  
    'jquery'  
,  
    function ( $ ) {  
        'use strict';  
        return {  
            paint:function ( $element, layout ) {  
                $element.empty();  
                var $helloWorld = $(  
                    document.createElement( 'div' ) );  
                $helloWorld.html('Hello World from the  
extension "QlikSense Cookbook -  
HelloWorld"<br/>');  
                $element.append( $helloWorld );  
            }  
        };  
    } );
```

5. Click on **Save As** in the notepad document and change the **Save as type** to **All Files (*.*)**. Call the file named `QlikSense-Cookbook-C7-R1-HelloWorld.js`.
6. Create a new application in Qlik Sense Desktop and name it `QlikSenseCookBook_Extensions`.
7. Open the data load editor and load the following code:

```
LOAD 1 as Dummy AUTOGENERATE(1);
```
8. Once the script is successfully run, open the **App overview** from the Navigation dropdown at the top. Create a new sheet and go to the Edit mode.
9. In the visualization area, we will be able to see the extension alongside the normal default visualizations with the name **QlikSense Cookbook-Hello world**.
10. Drag across the object onto the sheet.
11. Add the **Title** as `Qlik Sense Extension`.
12. The resultant object on the screen should look similar to the following:

Qlik Sense Extension

Hello World from the extension "QlikSense Cookbook - HelloWorld"

How it works...

The `QlikSense-Cookbook-C7-R1-HelloWorld.js` contains the JavaScript to build what the extension will actually do. It is formed of two main parts **Define** and **Paint**:

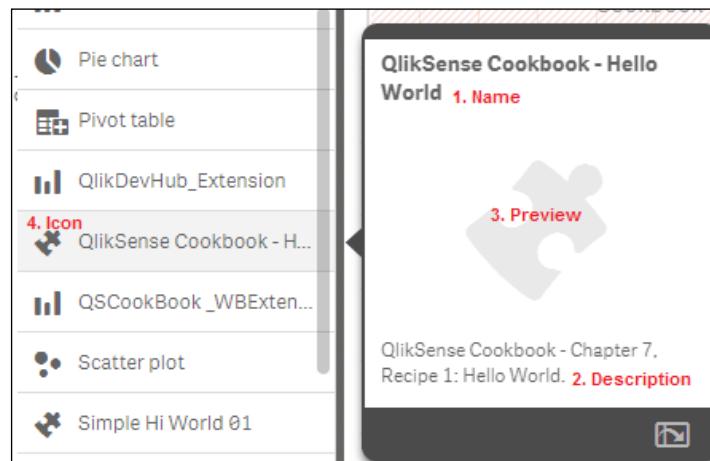
- ▶ **Define:** This is used to define the dependencies in the JavaScript file. It follows the concept specified in the RequireJS framework. In our recipe we have not loaded any external dependency. However, if the need arises, this can be loaded prior to the execution of the main script.
- ▶ **Paint:** This is the main part of the script which basically renders the visualization. It is formed of two parts `$element` and `layout`:
 - `$element` contains the HTML content
 - `layout` contains the data and properties of the extension

The `QlikSense-Cookbook-C7-R1-HelloWorld.qext` file contains the metadata about the extension, such as the name, description, icon, type, version, and author. Out of these, the name and type properties are mandatory.

The basic structure of a .qext file is shown in the following code:

```
{  
    "name" : "QlikSense Cookbook - Hello World",  
    "description" : " QlikSense Cookbook, Simple Hello World.",  
    "preview" : "bar",  
    "icon" : "extension",  
    "type" : "visualization",  
    "version": "1",  
    "author": "Your Name"  
}
```

The first four lines control what is displayed in the following image:



The type should always be "visualization". The default value for an icon is "extension" but it can be changed to a predefined list of icon names, such as the "Line chart", "Bar chart", and so on. This specifies the icon displayed in the Assets panel besides the extension object:

- ▶ **version:** Specifies the version of the extension
- ▶ **author:** Specifies the author of the extension

One can also define a preview image for the extension object under the `preview` property in the .qext file. For example "`Preview": "QSExtension.png`". The .PNG file must be stored in the same folder as the extension.

If we don't define the preview image, then the Icon definition will supersede.

There's more...

The extension discussed in the preceding recipe displays static text **Hello World** from the extension of "QlikSense Cookbook - HelloWorld". However, we can make it dynamic by making some simple additions to the code, as discussed in the following steps:

1. Open the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file.
2. Inside `Define` add another object called `definition` which describes the basic design of the property panel for the extension object:

```
definition: {
    type: "items",
    component: "accordion",
    items: {
        appearancePanel: {
            uses: "settings",
            items: {
                QSPROPERTYPanel: {
                    ref: "QSDYNAMICExtension",
                    type: "string",
                    label: "QlikSense extension Text"
                }
            }
        }
    },
},
```

3. In order to render what we enter in the text box defined in step 1, we need to enter the `console.log(layout);` statement at the start of the `paint` block.
4. Finally, to make the result dynamic, modify the output statement to:

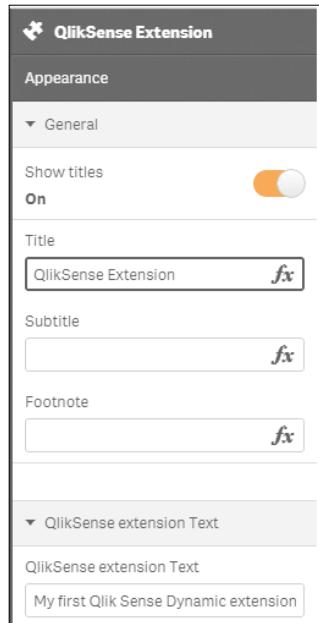
```
'$helloWorld.html(layout.QSDYNAMICExtension);'
```

5. The final code in the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file should look like the following code:

```
define([
    'jquery'
],
function ($) {
    'use strict';
    return {
        definition: {
            type: "items",
            component: "accordion",
            items: {
                appearancePanel: {
                    uses: "settings",
                    items: {
                        QSPROPERTYPanel: {
```

```
        ref: "QSDynamicExtension",
        type: "string",
        label: "QlikSense extension Text"
    }
}
},
),
paint: function ( $element, layout ) {
console.log(layout);
$element.empty();
var $helloWorld =
$(document.createElement('div'));
$helloWorld.html(layout.QSDynamicExtension);
$element.append($helloWorld);
}
);
}
);
);
```

6. Refresh your Qlik Sense document by pressing *F5* before implementing the new dynamic extension.
7. The Properties panel of the resultant extension object will look like the following image:



8. Put any desired text in the **QlikSense extension Text** box and check results.

See also

Creating a Qlik Sense® visualization using Qlik Dev Hub in *Chapter 7, Extensions in Qlik Sense®*.

Defining a Properties panel in Qlik Sense® visualizations

Typically, the most common properties used by any Qlik Sense visualization are Dimensions, Measures, and Appearance. The appearance section is included by default when we create any Qlik Sense visualization, even if the Properties panel is not defined.

We can extend the definition of these properties in the JavaScript code to reuse the built-in sections. The following recipe demonstrates how to define and extend the definitions of properties in Qlik Sense visualization and further reference these properties in our code.

Getting ready

This recipe is a continuation of the previous recipe. So, we will be using the same `QlikSense-Cookbook-C7-R1-HelloWorld.js` file, which contains the code for the dynamic extension as discussed in the *There's more...* section.

How to do it...

1. Open the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file located at `C:\Users\<username>\Documents\Qlik\Sense\Extensions\ QlikSense Cookbook - Hello World\`.
2. We have reused the settings section while creating the dynamic extension, which is nothing but the internal name for the **Appearance** section.
3. Next, we will extend the definition to reuse other built-in sections. In this example we will reuse the sorting sections.
4. Add the following piece of code after the `QSpropertyPanel` under `items`:

```
sorting: {  
    uses: "sorting"  
}
```
5. Save the JavaScript file and refresh your Qlik Sense application at this stage so that the "QlikSense Cookbook – Hello World" gets updated. The Properties panel for the extension will look similar to the following image:



6. We will output some values to the paint method by inserting the code as shown in the following code inside the `paint` function:

```
console.info('paint >> layout >> ', layout);
$element.empty();
var $helloWorld = 
$(document.createElement('div'));
// Variable holding the output
var html = '<b>Property values:</b><br/>';
html += 'Title: ' + layout.title + '<br/>';
html += 'SubTitle: ' + layout.subtitle +
'<br/>';
// Assigning the variable to our output
container
$helloWorld.html( html );
//$helloWorld.html(layout.QSDynamicExtension);
$element.append($helloWorld);
```

7. The final script for the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file will look like the following code:

```
define([
    'jquery'
],
function ( $ ) {
    'use strict';
    return {
        definition: {
            type: "items",
            component: "accordion",
            items: {
```

```

        appearancePanel: {
            uses: "settings",
            items: {
                QSPropertyPanel: {
                    ref: "QSDynamicExtension",
                    type: "string",
                    label: "QlikSense extension Text"
                },
            }
        },
        sorting: {
            uses: "sorting"
        }
    }
},
paint: function ( $element, layout ) {
//console.log(layout);
console.info('paint >> layout >> ', layout);
$element.empty();
var $helloWorld =
$(document.createElement('div'));
// Variable holding the output
var html = '<b>Property values:</b><br/>';
html += 'Title: ' + layout.title + '<br/>';
html += 'SubTitle: ' + layout.subtitle + '<br/>';
// Assigning the variable to our output
//container
$helloWorld.html( html );
$element.append($helloWorld);
}
);
}
);

```

8. The changes we made in the JavaScript file will introduce the, **Property Values**, section to the Properties panel.

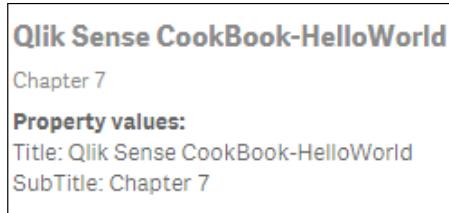
9. Save the file and return back to the Qlik Sense application.

10. Create a new sheet.

11. Go to the  mode.

12. On the left hand side pane, you will notice the extension object "QlikSense Cookbook -Hello World". Drag the extension object onto the sheet.

13. Next, go to the General section under Properties and add **Title** as Qlik Sense CookBook-Hello World and **SubTitle** as Chapter 7.
14. The resulting output will be as follows:



How it works...

We reference the defined properties in our JavaScript file under the `paint` section. We use a `layout` parameter that includes the current scope of the visualization extension together with the properties; this parameter is passed to the `paint` method.

There's more...

We can use other native Qlik Sense properties in our Properties panel definitions such as Dimension, Measure, Data Handling, Reference lines, and so on. For more information on the Add-Ons go to the following address:

https://help.qlik.com/sense/2.0/en-US/developer/#../Subsystems/Extensions/Content/extensions-reusing-properties.htm#3FTocPath%3DBuilding%2520visualization%2520extensions%7CGetting%2520started%2520building%2520visualization%2520extensions%7CBuilding%2520a%25-20properties%2520panel%7C_____1

Creating custom components within Qlik Sense® visualizations

Other than the predefined properties, the user may want to create custom components to alter the properties of the extension object. This is done in the appearance according to the main JavaScript file. The use of custom components provides the user with more options to customize the extension objects from the Properties panel.

The list of different UI components that you can use in the custom properties is as follows:

- ▶ Check box
- ▶ Input box/Text box
- ▶ Drop down list

- ▶ Radio button
- ▶ Button group
- ▶ Switch
- ▶ Slider
- ▶ Range-slider

Getting ready

This recipe is a continuation from the previous recipe. So we will be using the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file for this recipe as well, which we created for dynamic extensions in the *There's more...* section.

The recipe explains the procedure to create a check box in the Properties panel.

How to do it...

1. Open the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file located at `C:\Users\<username>\Documents\Qlik\Sense\Extensions\ QlikSense Cookbook - Hello World\`.
2. Next, we will add the property definition of the custom Check box as a new accordion item. This will be put inside the return block under items.
3. The definition of check box will be as follows:

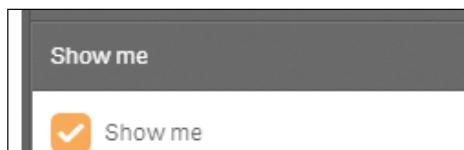
```
MyAccordion: {  
    type: "boolean",  
    label: "Show me",  
    ref: "myproperties.show",  
    defaultValue: true  
},
```

4. The final script for the `QlikSense-Cookbook-C7-R1-HelloWorld.js` file will look like the following code:

```
define( [  
    'jquery'  
,  
    function ( $ ) {  
        'use strict';  
        return {  
            definition: {  
                type: "items",  
                component: "accordion",  
                items: {  
                    MyAccordion: {
```

```
        type: "boolean",
        label: "Show me",
        ref: "myproperties.show",
        defaultValue: true
    },
    appearancePanel: {
        uses: "settings",
        items: {
            QSPROPERTYPanel: {
                ref: "QSDYNAMICExtension",
                type: "string",
                label: "QlikSense extension Text"
            }
        }
    }
},
paint: function ( $element, layout ) {
    console.log(layout);
    $element.empty();
    var $helloWorld =
        $(document.createElement('div'));
    $helloWorld.html(layout.QSDYNAMICExtension);
    $element.append($helloWorld);
}
);
}
);
});
```

5. Save the file and return back to the Qlik Sense application.
6. Create a new sheet.
7. Go to the  mode.
8. On the left-hand side pane, you will find the extension object "QlikSense CookBook – Hello World". Drag the extension object onto the sheet.
9. The Properties panel to the right displays the **Show me** checkbox as the following image:



How it works...

The definition for each of the custom properties is stated in the definition block of the code in the main JavaScript file of the extension.

In our example, the definition for the check box contains four fields namely `type`, `ref`, `label`, and `defaultvalue`. The field `type` is mandatory and should be assigned a Boolean value for a check box property definition.

- ▶ `label` is used to label the check box with a header in the Properties panel
- ▶ `ref` is an Id to refer to the check box property
- ▶ `defaultvalue` defines the default value for the check box

There's more...

In a way similar to the one described in the preceding recipe, you can also define properties to create sliders, radio buttons, description boxes, and so on. The following URL reflects upon the procedure to create all these custom components:

<https://help.qlik.com/sense/2.0/en-US/developer/#..../Subsystems/Extensions/Content/Howtos/working-with-custom-properties.htm>

Using data with extensions

In the previous recipes in this chapter we created our first Hello World extension, added properties and added custom components to the properties. Now, it's time to get your hands on the data inside your application.

We will go a step further in this recipe and define "Dimensions" and "Measures" in our JavaScript code block. This way we can extract the data from the tables and display it in a chart on our Qlik Sense sheet.

Getting ready

1. Open Qlik Sense hub and create a new Qlik Sense application.
2. Load the following script in order to auto-generate some example data:

```
Transactions:  
Load *,  
    mod(TransID,26)+1 AS Period,  
    Pick(Ceil(3*Rand1), 'Standard', 'Premium', 'Discount') AS  
    ProductType,
```

```
Pick(Ceil(6*Rand1), 'Apple', 'Orange', 'Cherry', 'Plum',
    'Fig', 'Pear') AS Category,
Round(1000*Rand()*Rand()*Rand1) AS Sales,
Round(Rand()*Rand1, 0.00001) AS Margin;
Load
    date(41275+IterNo()-1) AS Date,
    Rand() AS Rand1,
    RecNo() AS TransID
Autogenerate 1000 While Rand()<=0.5 or IterNo()=1;
```

3. Save the application.

How to do it...

1. Create a new folder in the extension directory and call it Qlik Sense Cookbook-C7-R3 - Hello Data.
2. Create a new .qext file, as we did in the first recipe of this chapter and name it Qlik Sense Cookbook-C7-R3 - Hello Data.qext. In the .qext file use the following code:

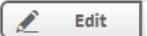
```
{
    "name" : "Hello Data",
    "description" : "Examples how to use data in
visualization extensions.",
    "icon" : "extension",
    "type" : "visualization",
    "version": "0.1.0",
    "author": "Your Name"
}
```

3. Next, create the JavaScript (.js) file, as we did previously in the same folder location and name it Qlik Sense Cookbook-C7-R3 - Hello Data.js. Enter the following code:

```
define( [],
    function ( ) {
        'use strict';
        return {
            definition: {
                type: "items",
                component: "accordion",
                items: {
                    dimensions: {uses: "dimensions"},
                    measures: {uses: "measures"},
                    sorting: {uses: "sorting"},
                    appearance: {uses: "settings"}
                }
            }
        }
    }
}
```

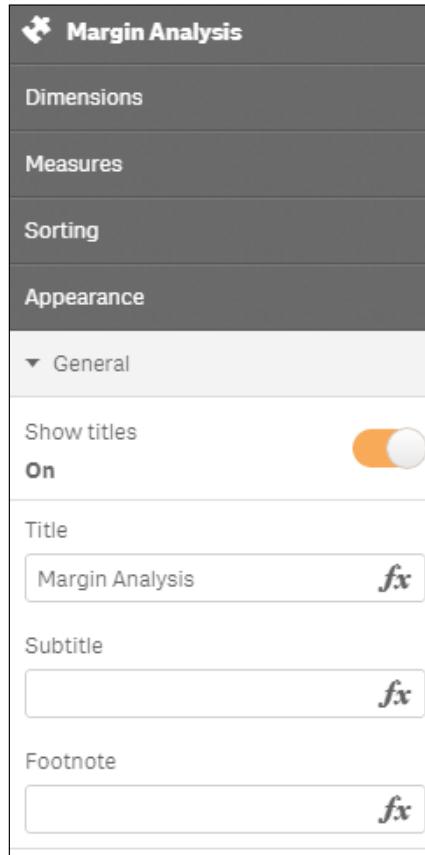
```
        }
    },
    initialProperties: {
        qHyperCubeDef: {
            qDimensions: [],
            qMeasures: [],
            qInitialDataFetch: [
                {
                    qWidth: 10,
                    qHeight: 100
                }
            ]
        }
    },
    paint: function ( $element, layout ) {
        var hc = layout.qHyperCube;
        //console.log( 'Data returned: ', hc );
        // Default rendering with HTML injection
        $element.empty();
        var table = '<table border="1">';
        table += '<thead>';
        table += '<tr>';
        for (var i = 0; i < hc.qDimensionInfo.length; i++) {
            table += '<th>' + hc.qDimensionInfo[i].qFallbackTitle +
            '</th>';
        }
        for (var i = 0; i < hc.qMeasureInfo.length; i++) {
            table += '<th>' + hc.qMeasureInfo[i].qFallbackTitle +
            '</th>';
            table += '</tr>';
        }
        table += '</thead>';
        table += '<tbody>';
        for (var r = 0; r < hc.qDataPages[0].qMatrix.length; r++) {
            table += '<tr>';
            for (var c = 0; c < hc.qDataPages[0].qMatrix[r].length;
            c++) {
                table += '<td>';
                table += hc.qDataPages[0].qMatrix[r][c].qText;
                table += '</td>';
            }
        }
    }
}
```

```
        table += '</tr>';
    }
    table += '</tbody>';
    table += '</table>';
    $element.append( table );
}
);
} );
```

4. Go to the **App overview** and create a new sheet in Qlik Sense.
5. Go to the  mode.
6. From the left-hand side Assets panel, drag across the "Hello Data" extension, which we created just now.
7. Title the **Object** as Margin Analysis.
8. Add **Category** as a dimension.
9. Add the following measure and label it as Margin:
Sum(Margin)
10. The final result will be similar to the following image:

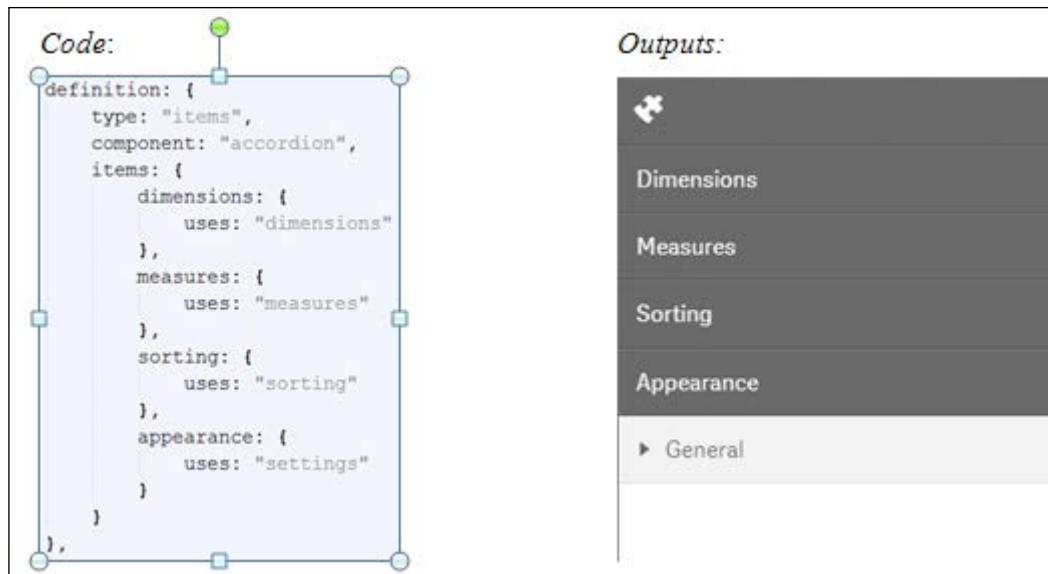
| Margin Analysis | |
|-----------------|-----------|
| Category | Margin |
| Apple | 15.37643 |
| Cherry | 71.21623 |
| Fig | 115.43042 |
| Orange | 41.11345 |
| Pear | 144.92144 |
| Plum | 101.98290 |

11. The resultant table will look like the following image. Also observe that while in the Edit mode for the object, the property panel to the right shows properties, such as **Dimensions**, **Measures**, **Sorting**, and **Appearance**; similar to a normal Qlik Sense object.

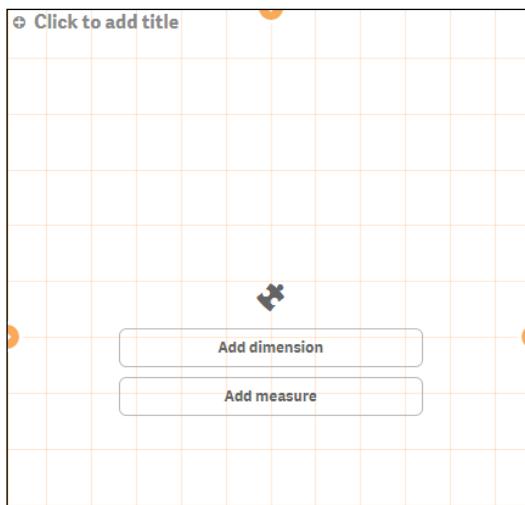


How it works...

In the previous recipes, we have already looked at the code to define the properties pane. This section of the code allows us to reuse the built in capability to define dimensions and measures just like the other objects in Qlik Sense:



By setting the default items in the properties window to include standard dimensions and measures options, we can now add data to the visualization. The data fields can be added directly in the object as shown in the following image:



As soon as you add dimensions and measures to a visualization extension, the Qlik Engine will return what is known as a **HyperCube**. Although it is a huge simplification, you can simply think of it as a table returned from the engine.

For a full breakdown of the HyperCube structure, see the QlikSense development toolkit on qlik.com. For now, we are interested in three objects of the HyperCube and they are:

- ▶ layout.qHyperCube.qDimensionInfo: used dimensions
- ▶ layout.qHyperCube.qMeasureInfo: used measures
- ▶ layout.qHyperCube.qDataPages: the result

In the print statement, we create a basic HTML table with a header for labels and body for the data. The skeleton of a basic HTML table is shown in the following code:

```
var table = '<table border="1">';  
table += '<thead>';  
table += '</thead>';  
  
table += '<tbody>';  
table += '</tbody>';  
table += '</table>';
```

The first two bullet points in the preceding section are used for qDimensionInfo and qMeasureInfo. The third bullet point is for qDataPages:

- ▶ qDataPages is an array
- ▶ The data is held with qDataPages[0].qDataPages.qMatrix
- ▶ It is also an array of objects (the rows)
- ▶ Each holds an array of other objects (the cells)

The preceding skeleton has been completed in our example and is shown in the following image:

The following image is the result of the header:

```
table += '<thead>';  
table += '<tr>';  
for (var i = 0; i < hc.qDimensionInfo.length; i++) {  
    table += '<th>' + hc.qDimensionInfo[i].qFallbackTitle + '</th>';  
}  
for (var i = 0; i < hc.qMeasureInfo.length; i++) {  
    table += '<th>' + hc.qMeasureInfo[i].qFallbackTitle + '</th>';  
}  
table += '</tr>';  
table += '</thead>';
```

The following image is the result of the body:

```
table += '<tbody>';
  for (var r = 0; r < hc.qDataPages[0].qMatrix.length; r++) {
    table += '<tr>';
    for (var c = 0; c < hc.qDataPages[0].qMatrix[r].length; c++) {
      table += '<td>';
      table += hc.qDataPages[0].qMatrix[r][c].qText;
      table += '</td>';
    }
    table += '</tr>';
  }
table += '</tbody>';
```

See also

- ▶ [Creating an HTML visualization extension for Qlik Sense®](#)

8

What's New in Version 2.1.1?

In this chapter, we will focus on some of the latest features that have been released in Qlik Sense Version 2.1.1:

- ▶ Using the visual exploration capability in Qlik Sense®
- ▶ Defining Variables in Qlik Sense®
- ▶ Exporting stories to MS PowerPoint
- ▶ Using the Qlik Dev Hub in Qlik Sense® 2.1.1
- ▶ Using Extension editor in Qlik Dev Hub
- ▶ Using Qlik Dev Hub to generate mashups
- ▶ Embedding Qlik Sense® application on a website using a single configurator
- ▶ Using the Qlik DataMarket
- ▶ Using Smart Search
- ▶ Creating dynamic charts in Qlik Sense®
- ▶ Using smart data load profiling

Introduction

This chapter deals with some new functionalities introduced in the latest release of Qlik Sense 2.1.1. While the core essence of Qlik Sense remains same, the new functionalities bring out a more sophisticated and convenient approach to interact and build engaging applications. The new features also boost the self-service functionality of Qlik Sense by providing options such as *Exporting the Stories to PowerPoint* and *Defining variables outside the script using the new Variable interface*.

Using the visual exploration capability in Qlik Sense® 2.1.1

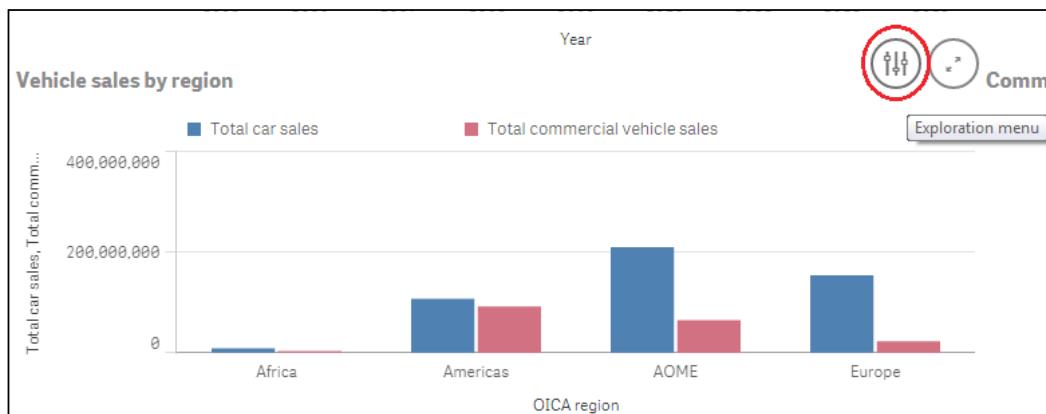
The visual exploration capability introduced in Qlik Sense 2.1 strengthens the concept of self-service Business Intelligence. It puts more power in the hands of business users or users who are not the original authors of the application. It allows users to change properties of certain objects such as bar charts, scatter charts, and trendline charts without entering the Edit mode or changing the underlying content of the application. The feature works in all versions of Qlik Sense, namely Qlik Sense Desktop, Qlik Sense Enterprise, and Qlik Sense Cloud.

Getting ready

This recipe will make use of the `Automotive.qvf` application available on the Qlik Sense hub. This application usually comes with the default installation of Qlik Sense Desktop. If you don't get the application with the installation, you can download the same from the source material for this chapter available on the Packt Publishing website.

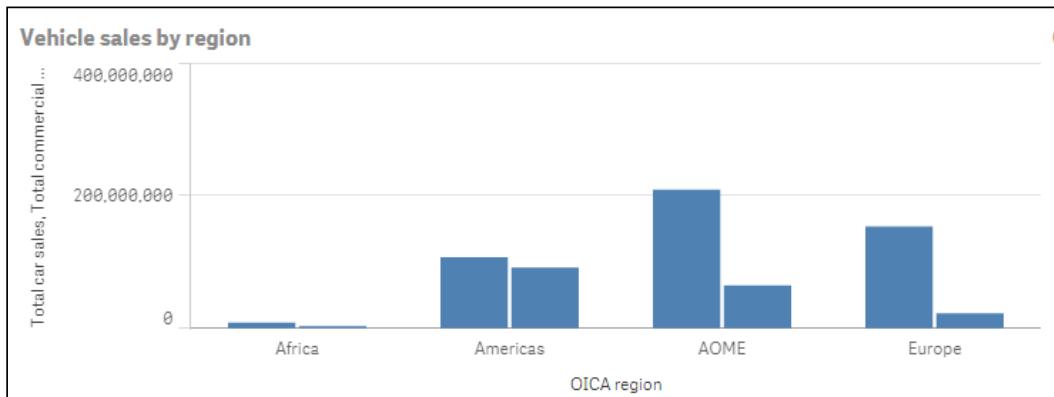
How to do it...

1. Open the `Automotive.qvf` application from the hub.
2. Open the Sales overview sheet.
3. Hover over the **Vehicle sales by region** bar chart and just beside the fullscreen icon, you will find the **Exploration menu** icon. Click on it.



4. On clicking, the object goes fullscreen and while still in the analysis mode, the Properties panel gets activated.

5. In this mode, we can change a subset of properties such as sorting, colors, presentation, and so on.
6. Change the color scheme of the bar chart to Single color.
7. Once done, confirm the changes.
8. Exit the fullscreen mode.
9. The object with the changed properties look like this:



How it works...

The visual exploration feature allows the business user to change the properties of the onscreen objects without entering the Edit mode. If the user himself is the author of the application, then they can keep the changes and make them a part of the original application.

An end user who is accessing a published application on Qlik Sense Cloud or Qlik Sense Enterprise can make the changes to the properties using visual exploration techniques but can't keep them. In other words, the changes remain only in the users' session.

There's more...

The visual exploration capability can be used only with certain Qlik Sense objects. It cannot be used with gauge charts, table objects, or pivot charts. It cannot be used with extension objects either.

See also

- ▶ *Exporting stories to MS PowerPoint*

Defining variables in Qlik Sense®

For versions prior to v2.1.1, Qlik Sense does not provide the option to define variables outside the script as you could in the **Variable Overview** window in Qlikview. With v2.1.1, Qlik has introduced a new variable interface that enlists the existing variables created in the script and also provides the user with the option to create new variables.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives the sales information for four countries:

1. Create a new Qlik Sense application and call it QS_Variables.
2. Load the following script in the application:

```
Sales:  
LOAD * INLINE [  
Country, Sales,COS  
USA, 1000,500  
UK, 2000,1000  
France, 3000,2500  
Germany, 4000,4700  
];  
  
Let vRedColor=RGB (255, 0, 0);  
Let vSales= 'Sum(Sales)';
```

How to do it...

1. Open the QS_Variables application.
2. Create a new sheet called as Sales and go to the Edit mode for the sheet.
3. While in the Edit mode, notice that we have a new icon  on the lower-left corner. Click on the icon to open the **Variables** interface window.
4. The **Variables** interface window lists all the variables that we have defined in the script:

| Variables | | Create new |
|------------|------------------|-------------------|
| Name | Definition | |
| vBlueColor | RGB(164,216,255) | |
| vSales | Sum(Sales) | |

5. At the same time, it gives us the option to create new variables outside the script directly in the interface using the **Create new** button at the top-right corner.
6. Click on the **Create new** button and define a new variable, as shown here:

| Variables | | Create new |
|--------------|--|-------------------|
| Name | Definition | |
| New variable | | |
| Name: | <input type="text" value="vCOS"/> | |
| Definition: | <input type="text" value="SUM(COS)"/> | |
| Description: | <input type="text" value="Cost of Sales"/> | |
| Tags: | <input type="text"/> | |

What's New in Version 2.1.1? ---

7. Click on the **Save** button and close the variable interface. Reopen to see the list of variables.
8. While still in the Edit mode, drag a table object onto the sheet.
9. Use **Country** as a dimension.
10. Create a measure with the following expression and label it **Sales**:
 $= \$ (vSales)$
11. Create a second measure with the following expression and label it **Cost of Sales**:
 $=\$ (vCOS)$
12. Define the background color expression for **Cost of Sales** as follows:
`=if([Cost of Sales]>Sales,vRedColor,White())`
13. The resulting table would look like this:

| Sales | | | |
|---------------|--|--------------|---------------|
| Country | | Sales | Cost of Sales |
| Totals | | 10000 | 8700 |
| France | | 3000 | 2500 |
| Germany | | 4000 | 4700 |
| UK | | 2000 | 1000 |
| USA | | 1000 | 500 |

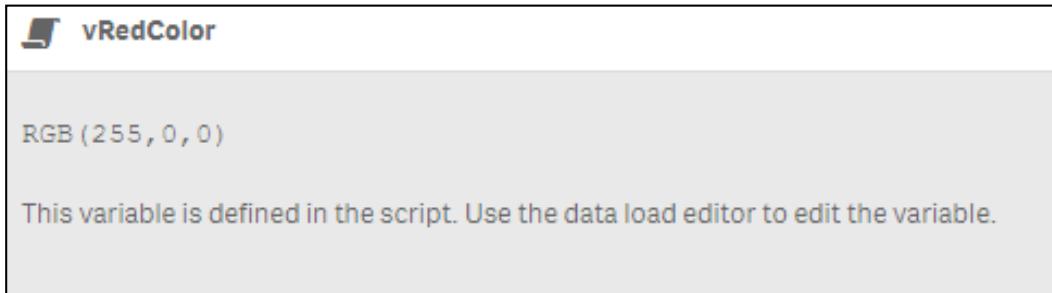
How it works...

The variables can be put to an effective use in the application to define expressions as well as to store certain field values. If numeric values are stored in the variables, then we don't need to use the \$ sign expansion while calling the variables. It is however a good practice to always use the \$ sign, as it is needed in case of expression syntax, tests or literals.

A point to be noted in our recipe is regarding the background color expression defined in step no 12. Cost of Sales and Sales are expression labels we defined earlier and not fields from the Data model.

The background color expression simply references the label of the expressions containing the numbers we need. Referencing an existing expression label instead of repeating the same code can also benefit overall chart performance. This is because Qlik Sense only has to aggregate the values at a base Data model level once; thereafter, the output can be reused from the cached memory where needed.

The variables that are defined in the script are denoted by a  symbol in the variable interface and cannot be edited only through the data load editor, as shown in the following screenshot:



There's more...

The variables can also be defined in external files such as a text file and then loaded into the application through the data load editor.

In order to try this, complete the following steps:

1. Download the `Variables.xlsx` file from the Packt Publishing website and set up a library connection to the file location called `QlikSenseCookBook_SourceFiles` (to resemble the `FROM...` code used in the following code).
2. Copy and load the following code:

```
VariableDefinitions:  
LOAD  
    Variable,  
    Expression  
FROM [lib:// QlikSenseCookBook_SourceFiles/Variables.xlsx]  
(ooxml, embedded labels, table is Variables);  
  
Let vNumberOfRows = NoOfRows('VariableDefinitions');  
For vI = 0 to (vNumberOfRows - 1)  
Let vVariable_Name = Peek('Variable',vI,'Expression');  
Let [$ (vVariable_Name)] =  
    Peek('Expression',vI,'Expression');  
Next
```

What's New in Version 2.1.1?

If you now go back to the Variable list from the edit sheet window, you will see a variable has been created for each row in the Excel file attached. The code below the FROM statement simply loops through each row of the Excel file, creating a new variable each time. The values in column A become the variable names and the corresponding values in column B are used as the variable definitions.

See also

- ▶ [Using smart data load profiling](#)

Exporting stories to MS PowerPoint

Stories in Qlik Sense are a great feature that can help create insights within data and share those insights with the users in the form of slideshow. These slideshows are native to Qlikview, and hence only users having access to the Qlik Sense application can view the storyline.

With Version 2.1.1, Qlik has introduced the concept of exporting the stories to PowerPoint presentations in order to provide access to users outside the system. Once the storyboard is exported, the users can edit the presentation using the standard formatting functions in MS PowerPoint. One can also print and share the presentation with a larger audience base.

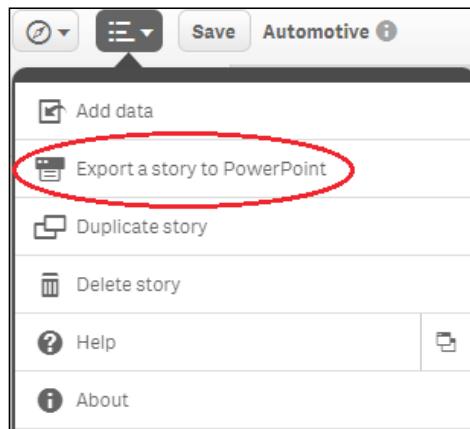
Getting ready

In *Chapter 2, Visualizations* we created a storyboard using the `Automotive.qvf` application. We will be using the same storyboard to analyze the export function. If you don't have the storyboard saved in the application, you can create one again and then proceed with the *How to do it...* section.

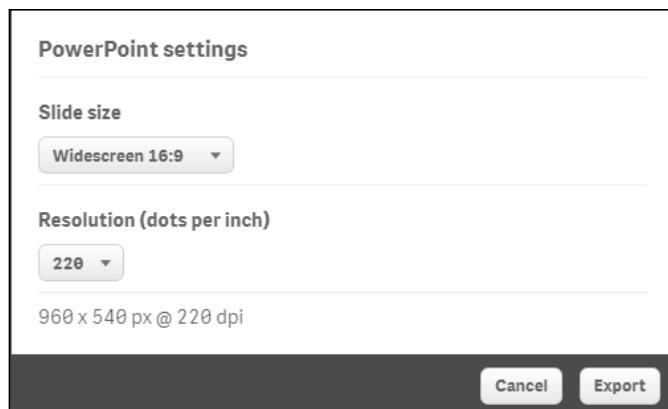
How to do it...

1. Open the `Automotive.qvf` application from Qlik Sense hub.
2. Click on the  Stories button to display the available stories.
3. Open the Sales Overview story.

4. From the action menu at the top, select **Export a story to PowerPoint**.



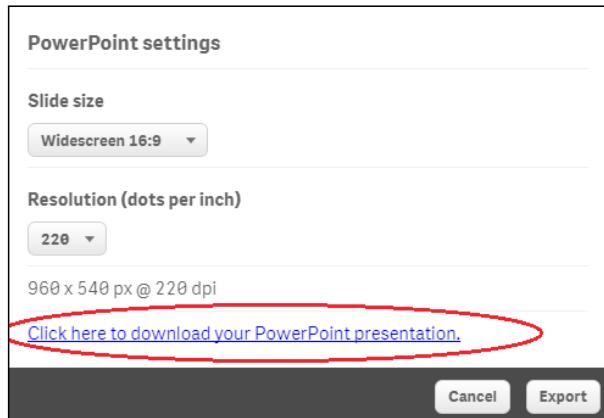
5. On clicking, the following **PowerPoint Settings** window appears:



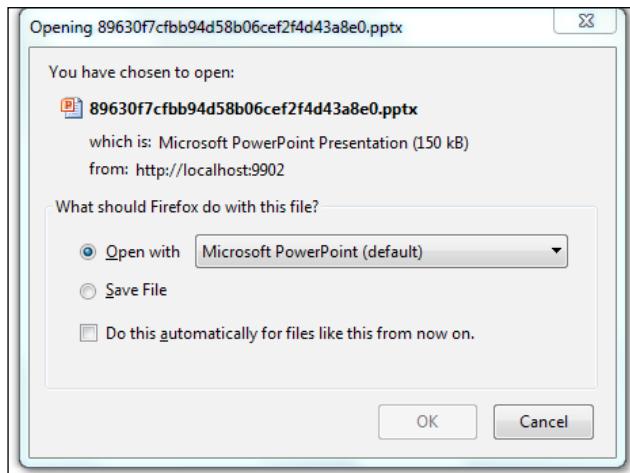
6. Select the appropriate values for **Slide size** and **Resolution** and click on **Export**.

What's New in Version 2.1.1? —

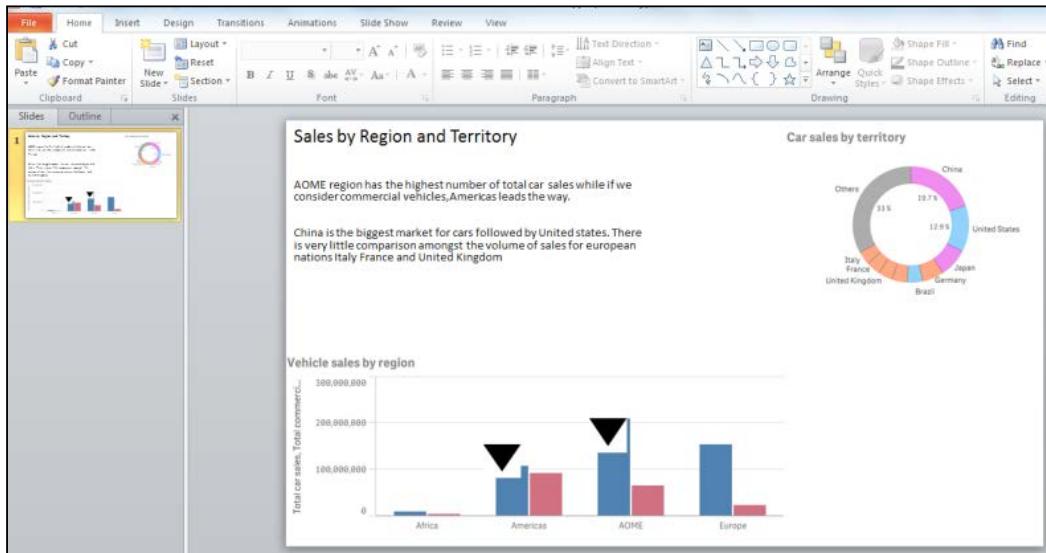
7. The export process creates a hyperlink that needs to be used for downloading the PowerPoint presentation:



8. Click on the hyperlink to open the storyboard in a PowerPoint file:



9. Click on **OK** to open the PowerPoint file, which looks like this:



How it works...

When we click on the hyperlink in the PowerPoint Settings window, Qlik Sense uses the default browser on the system to download and open the story in a PowerPoint file. The export functionality is designed purposefully for the users who don't have access to the Qlik Sense application. As with any other PowerPoint file, you can edit the look and feel of the objects using the formatting functions in PowerPoint.

There's more...

Currently, the export functionality does not work with the extension objects or the embedded sheets. But this will be possible in future versions of the software.

It has also been noticed that sometimes the wizard takes a long time to generate the hyperlink for the PowerPoint export. In such a case, clear out the cache in your default browser and test again.

See also

- ▶ Defining variables in Qlik Sense®

Using the Qlik Dev Hub in Qlik Sense® 2.1.1

Qlik Sense Version 2.1.1 combines the Single configurator, Mashup editor, and the Extension editor under a common single platform called as the **Qlik Dev Hub**. It formally replaces Qlik Sense Workbench used in versions of Qlik Sense prior to 2.1.1 and provides a nice, easy interface to work on these different tools and utilities.

Users who have knowledge of the Workbench editor with prior versions of Qlik Sense must keep in mind that the basic principles of using the Single configurator or creating mashups and extensions remain the same. However, there are subtle changes in the individual interfaces of each of the editors which we will discuss in this recipe.

This recipe will not extensively explain how to create extension objects and mashups or how to deal with Single configurator. However, it will introduce the user to the change in concept when working with Qlik Sense Version 2.1.1.

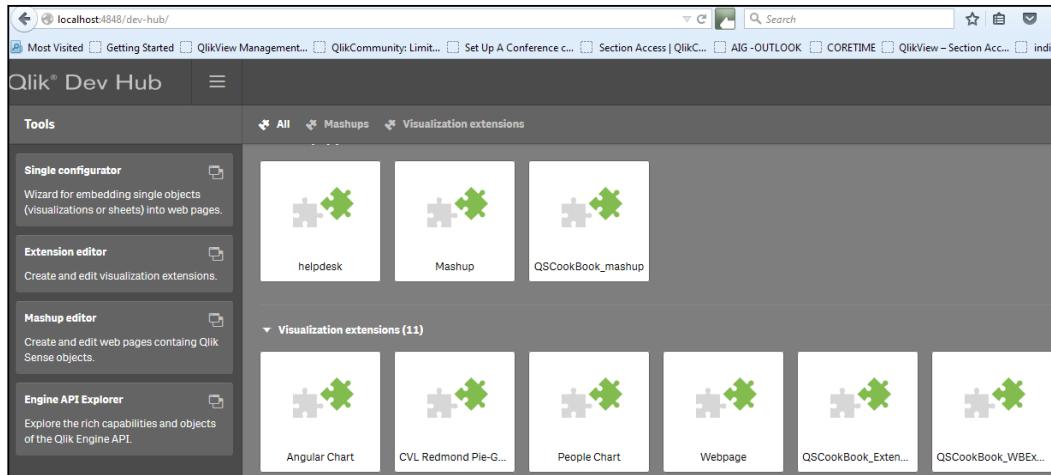
The next few recipes will be more descriptive in nature and explain the step-by-step process of generating the extensions and mashups using the Qlik Dev Hub. Also, we will discuss the process of embedding Qlik Sense content on a web page using a Single configurator.

Getting ready

Before starting Qlik Dev Hub, make sure you have Qlik Sense Desktop running in the background.

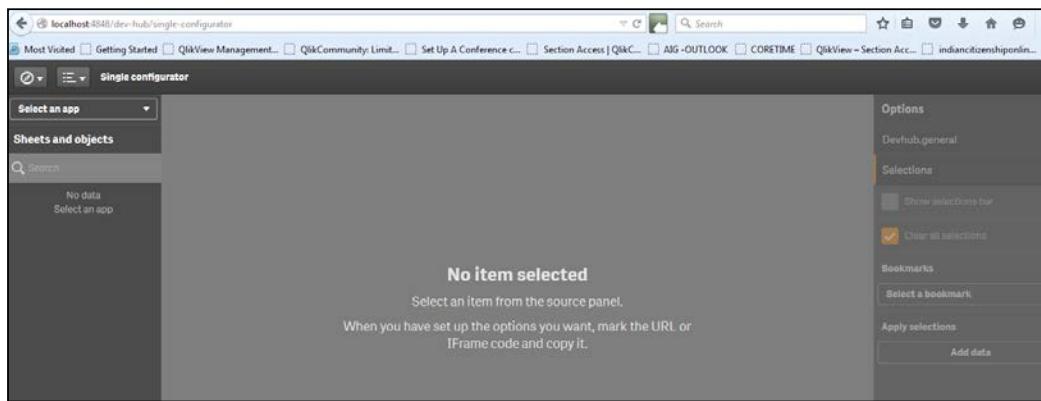
How to do it...

1. Open Qlik Dev Hub using the following URL:
`http://localhost:4848/dev-hub`
2. As with the Qlik Sense workbench, we will see that the interface shows all the available mashups and visualization extensions in the form of tiles. Along with this, the left-hand side pane gives access to tools such as **Single configurator**, **Mashup editor**, **Extension editor**, and **Engine API Explorer**:



Single configurator

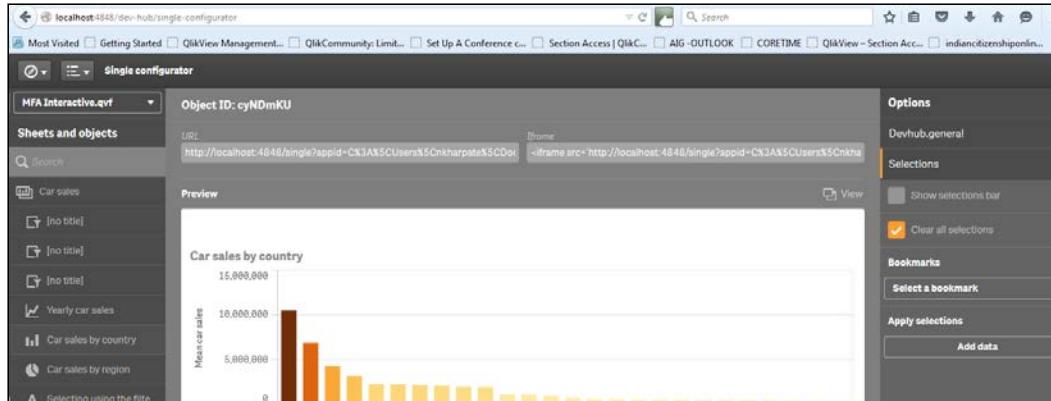
1. Click on **Single configurator**. The following screen appears:



2. The Source panel on the left-hand side lists all the available applications on Qlik Sense hub in the dropdown.

What's New in Version 2.1.1? —

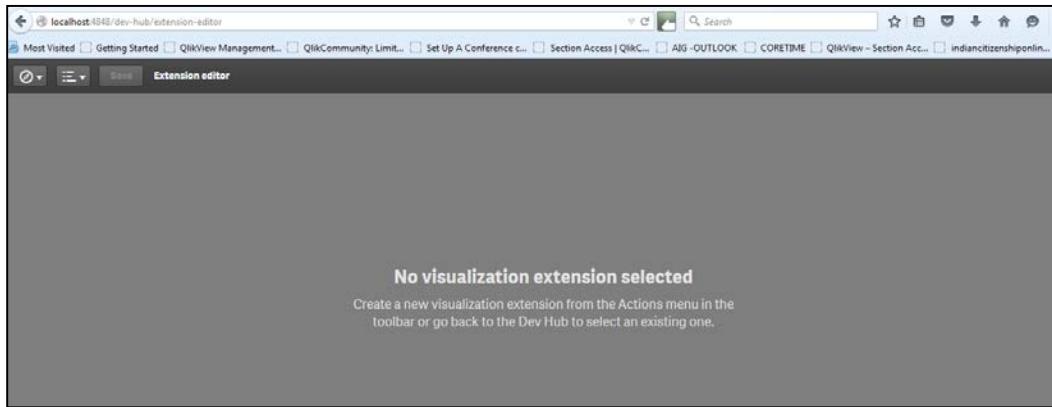
3. Select MFA interactive.qvf and then click on the **Car sales by country** chart:



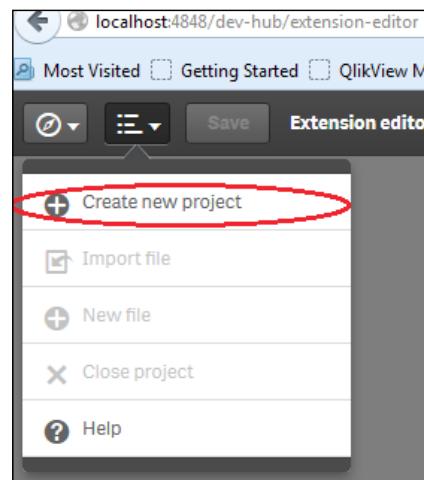
4. As seen, the properties for the chart get activated.
5. The **URL** and **iframe** link are located at the top of the **Preview** window. The single configurator allows the user to access all the Qlik Sense applications and visualization objects and it references them via a URL. The URL can be copied and used in any web browser. The **iframe** link can be used to embed the chart in a website.
6. The **Options** for chart interaction, selections, and bookmarks are in the right-hand side panel. The **URL** and **iframe** links are modified based on the options selected by the user.

Extensions editor

1. In order to access the **Extension editor**, return to the Dev Hub page by clicking on  under the navigation dropdown.
2. Click on **Extension editor** in the left-hand side pane. Contrary to the workbench wherein the extension can be created via the main screen, **Extension editor** opens up a new window:

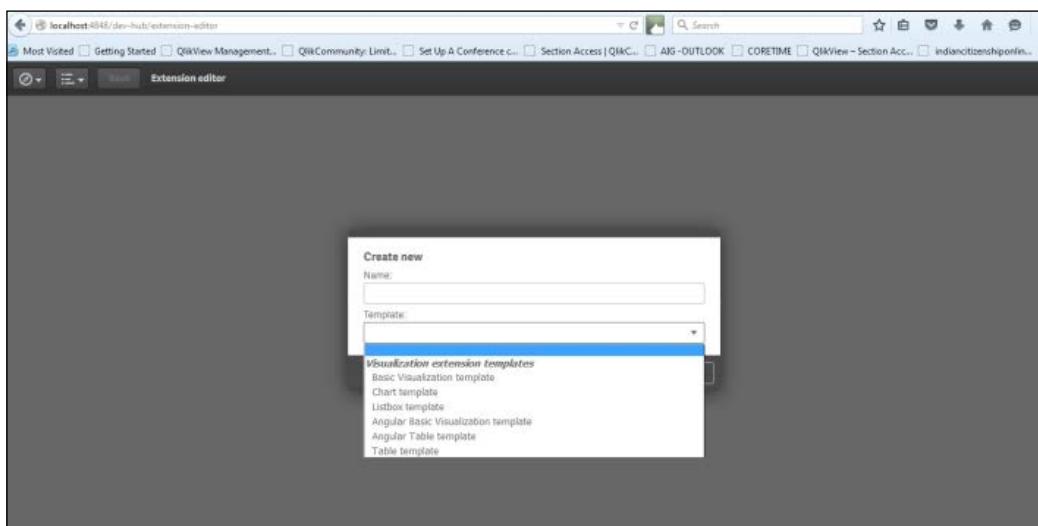


3. In order to create a new extension, click on **Create new project** under the menu on the toolbar:



What's New in Version 2.1.1?

4. On clicking, the following window appears which is exactly similar to the previous workbench version. Here, we can give a name to our new extension and use the basic default templates available with the editor:

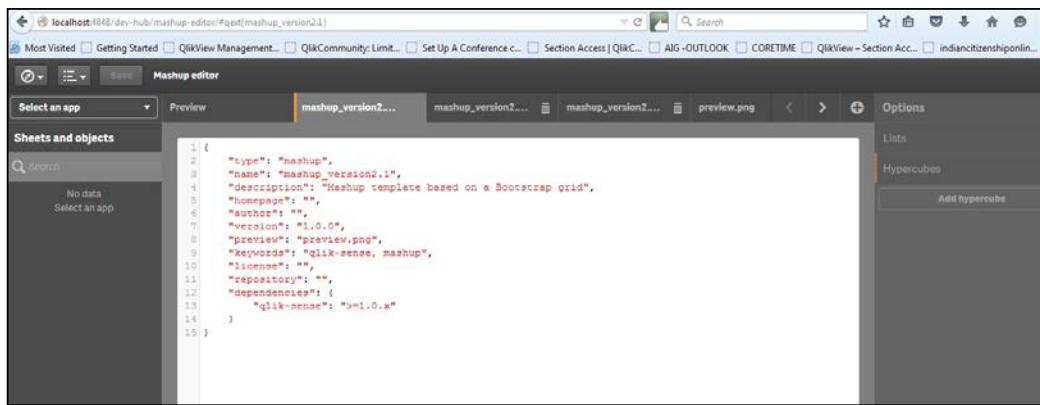


5. Once we define the name and template for our extension object, click on the **Create & edit** button. This opens up the tabs for the JavaScript, .qext, and other related files for the extension. If we want to get added functionalities in the extension, we can modify the JavaScript code here as per the requirement.
6. As we develop the extension object in **Extension editor**, the Dev Hub makes it live in the Qlik Sense environment.

Mashup editor

1. In order to access the **Mashup editor**, return to the Dev Hub page by clicking on  **Qlik Dev Hub** under the navigation dropdown.

2. Click on **Mashup editor** in the left pane.
3. The **Mashup editor** interface is very much similar to the **Extension editor** interface.
4. Once we define the name and template for the mashup and click on **Create & edit**, the following window appears:



The screenshot shows the Qlik Sense developer hub's Mashup editor. The top navigation bar includes 'localhost:4040/dev-hub/mashup-editor#qext/mashup_version2.1'. The main interface has tabs for 'Preview' and 'mashup_version2...'. On the left, there's a 'Select an app' dropdown and a 'Sheets and objects' panel with a search bar and a note 'No data'. The central area displays a large block of JSON code:

```
1: {
2:   "type": "mashup",
3:   "name": "mashup_version2.1",
4:   "description": "Mashup template based on a Bootstrap grid",
5:   "homepage": "",
6:   "author": "",
7:   "version": "1.0.0",
8:   "preview": "preview.png",
9:   "keywords": "qlik-sense, mashup",
10:  "license": "",
11:  "repository": "",
12:  "dependencies": [
13:    "qlik-sense": ">=1.0.x"
14:  ]
15: }
```

The right side features an 'Options' panel with sections for 'Lists' and 'Hypercubes', and a button 'Add hypercube'.

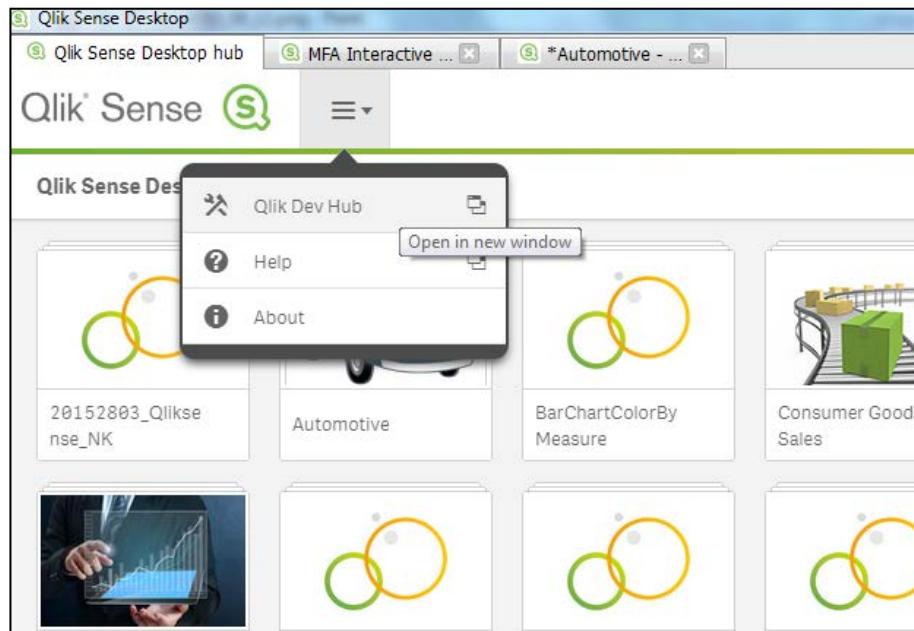
5. We can select the application and visualization objects from the Source panel on the left. The visualization objects can be laid out in the grid by the simple process of drag and drop. There is a menu bar at the top that gives access to the preview window as well as the underlying files for the mashup.

How it works...

The developer hub is a single platform for creating the extensions and mashups in Qlik Sense. It gives access to JavaScript APIs, which consist of a number of methods and properties to build the custom visualizations. The editors autogenerated the mandatory files required for the extension and mashups to work.

There's more...

Qlik Dev Hub can also be opened by opening **Qlik Sense Desktop** and then clicking on the dropdown option under the menu item at the top:



Single configurator

While working with the Single configurator, if we use multiple charts in our web page, then all the charts will continue to interact with each other similar to an out of the box Qlik Sense environment.

Extension editor

For non-programmers, Qlik provides an easy way to amend the extension script files. In order to do that, one can access the example codes at http://help.qlik.com/sense/2.1/en-us/developer/#.../Subsystems/Dev-Hub/Content/Examples/dev-hub-code-examples.htm%3FTocPath%3Dqlik%2520Dev%2520Hub|Examples|_____0.

Select a particular extension under **Examples** and then grab the JavaScript code and the QEXT code for your extension. Make sure you also copy and paste the code for other related files the JavaScript is referring to, for example, the .CSS files and the .PNG files.

See also

- ▶ *Using the Qlik DataMarket*

Using Extension editor in Qlik Dev Hub

In the previous chapter, we discussed a recipe to develop a visualization extension in Qlik Sense by writing the code manually in the .js and .qext files. One of the easier methods to develop extension visualization in Qlik Sense is using **Extension editor** in Qlik Dev Hub.

A Qlik Dev Hub is an integrated development toolbox used for building visualizations and mashup websites. It does not come with a separate installation package, but the editor and the API libraries are provided with Qlik Sense Desktop.

The previous recipe has explained all the basics of the Dev Hub. This recipe will introduce the user to the process of building a basic chart extension using **Extension editor** in Qlik Dev Hub. The extension workbench supports the .js, .qext, .css, and .html formats.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives the sales information for four countries:

1. Create a new Qlik Sense application and call it QS_DevHub_Extensions.
2. Load the following script in the application:

```
Sales:  
LOAD * INLINE [  
Country, Sales  
USA, 1000  
UK, 2000  
France, 3000  
Germany, 4000  
];
```

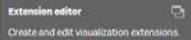
3. Save the application at this point and move on to build the visualization using **Extension editor**.
4. Before starting **Extension editor**, make sure you have Qlik Sense Desktop running in the background.

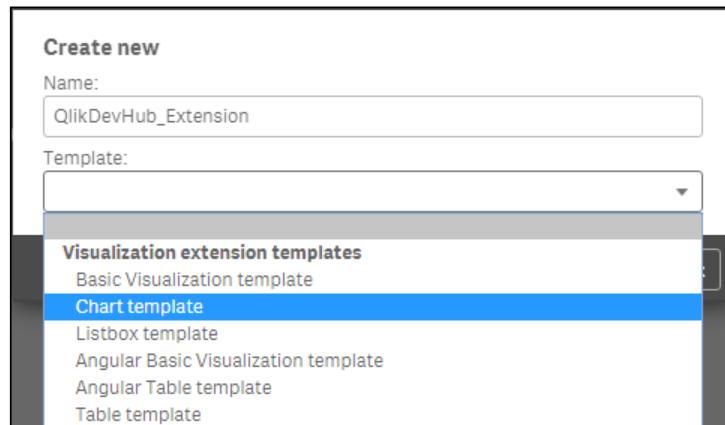
How to do it...

1. Open Qlik Dev Hub using the following URL:

<http://localhost:4848/dev-hub>

What's New in Version 2.1.1? —

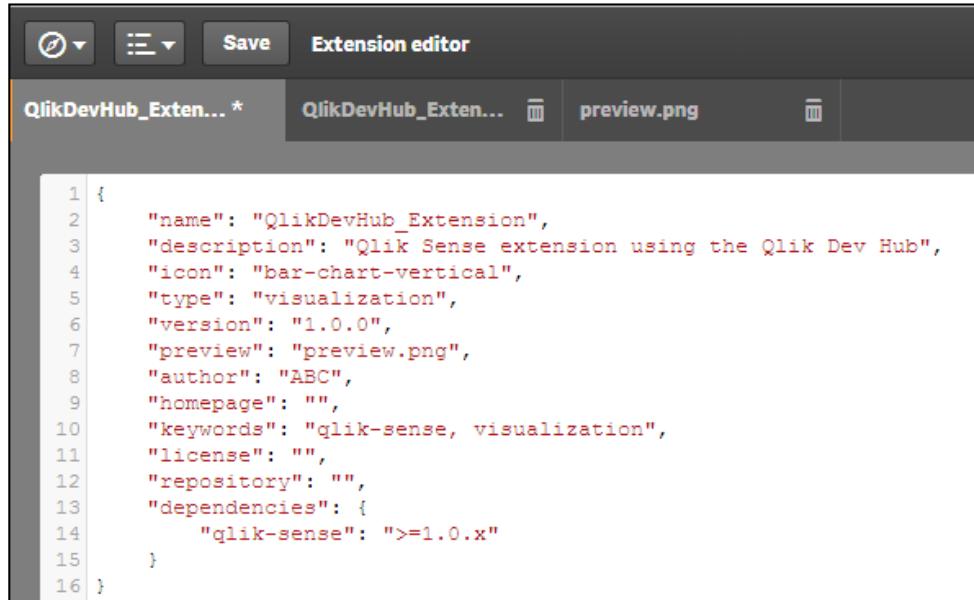
2. Click on the  button available in the left-hand side pane of Qlik Dev Hub.
3. A new **Extension editor** window opens. Click on the menu dropdown  on the toolbar.
4. Now, click on  to create a new project.
5. Name the new visualization as `QlikDevHub_Extension`.
6. Under **Template**, select **Chart template** and click on .



7. This opens up the tabs for the JavaScript, `.qext`, and other related files for the extension. If we want to get added functionalities in the extension, we can modify the JavaScript code here as per requirements.
8. As seen, the editor has automatically created the mandatory files and the script required for the extension.
9. Edit the script on the `QlikDevHub_Extension.qext` tab as follows:

```
"description": "Qlik Sense extension using the Qlik Dev  
Hub",  
"author": "<User Name>"
```

10. The result of the preceding code will look like this:



The screenshot shows a software interface titled "Extension editor". The window has a dark header bar with icons for refresh, save, and a dropdown menu. Below the header, there are two tabs: "QlikDevHub_Extension.js" and "preview.png". The "QlikDevHub_Extension.js" tab is active and contains the following code:

```
1 {
2     "name": "QlikDevHub_Extension",
3     "description": "Qlik Sense extension using the Qlik Dev Hub",
4     "icon": "bar-chart-vertical",
5     "type": "visualization",
6     "version": "1.0.0",
7     "preview": "preview.png",
8     "author": "ABC",
9     "homepage": "",
10    "keywords": "qlik-sense, visualization",
11    "license": "",
12    "repository": "",
13    "dependencies": {
14        "qlik-sense": ">=1.0.x"
15    }
16 }
```

11. The `QlikDevHub_Extension.js` tab contains the JavaScript required for rendering the visualization. One can edit this tab if required to do so.
12. Save and close the window.
13. Open the `QS_DevHub_Extensions` application we created in step 1 of the *Getting ready* section in Qlik Sense Desktop.
14. Create a new sheet and enter the Edit mode by clicking on .
15. The `QlikDevHub_Extension` extension object is now available in the Assets panel on the left. Drag the extension onto the sheet.
16. Add **Country** as a dimension.
17. Add the following measure:
`Sum (Sales)`
18. Name the chart as `Sales by Country`.

19. The resultant chart would be as follows:



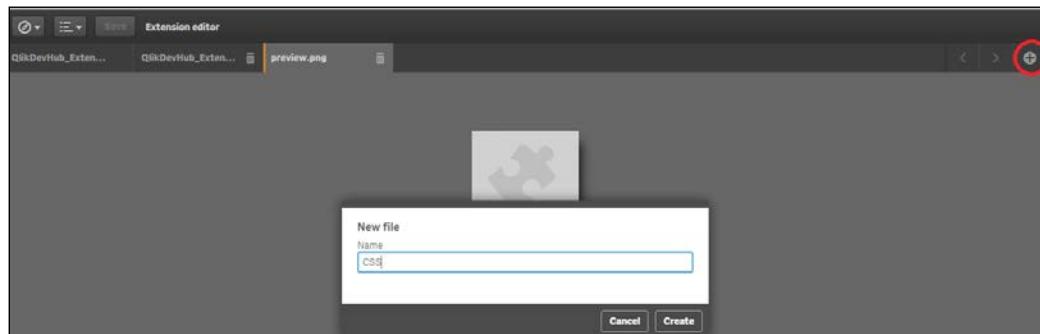
How it works...

The Qlik Sense **Extension editor** provides JavaScript APIs, which consist of a number of methods and properties to build the custom visualizations. The editor autogenerated the mandatory files required for the extension to work.

There's more...

If we want to define custom styles for our visualization, we can do so by using one or more CSS files. The content for the CSS files first need to be loaded to the document's header or alternatively added as a link to a style sheet to the document's header. Styles can also be defined using the RequireJS CSS plugin.

Additional files such as .css, .js, and .html can be added using the button located at the top-right hand corner of the **Extension editor** workspace:



See also

- ▶ *Using Qlik Dev Hub to generate mashups*

Using Qlik Dev Hub to generate mashups

A mashup is a web page consisting of content from more than one source displayed in a single user interface. When we design mashups in Qlik Sense, we integrate multiple random objects from a Qlik Sense application into a predefined layout. In doing so, we use the active content from the Qlik Sense application. Hence, the visualizations get updated automatically when the state of the object changes.

The **Mashup editor** in Qlik Dev Hub allows us to build mashups using the Mashups API. These Mashup APIs are used to display Qlik Sense objects on a website or web application where one can interact with the Qlik Sense datasets.

Getting ready

We make use of the `Automotive .qvf` application for this recipe. This application comes with the default installation of Qlik Sense. If not, it is available for download from the Packt Publishing website. Before starting the Qlik Dev Hub editor, make sure you have the Qlik Sense Desktop running in the background.

How to do it...

1. Open Qlik Dev Hub using the following URL:

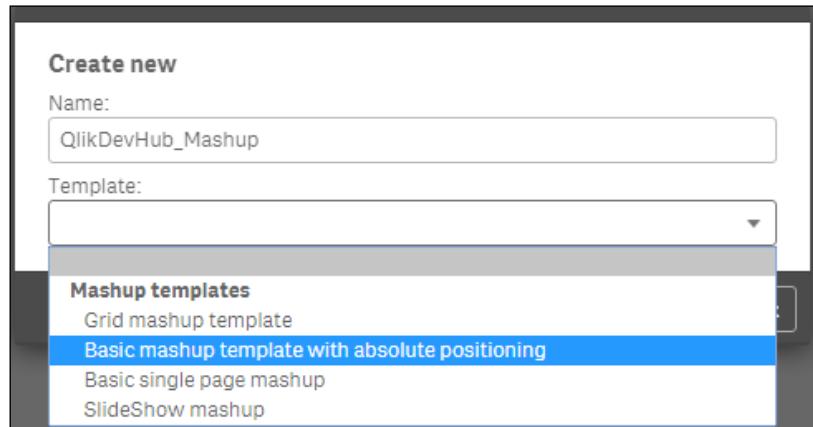
`http://localhost:4848/dev-hub`



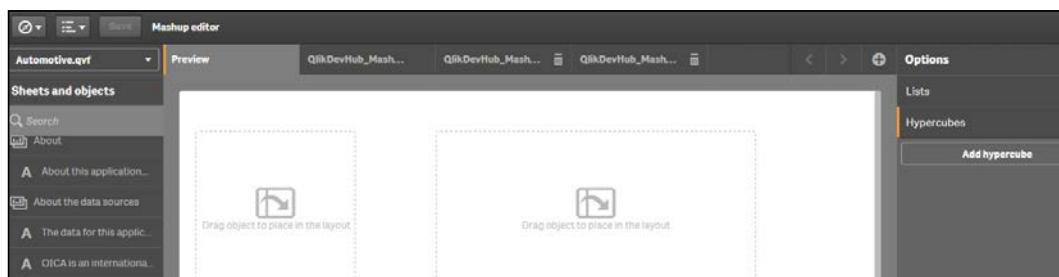
2. Click on the `Single configurator` button available in the left-hand side panel of the Qlik Dev Hub.
3. A new **Mashup editor** window opens.
4. Now click on `+ Create new project` to create a new project.
5. Name the new mashup object `qlikDevHub_Mashup`.

What's New in Version 2.1.1? —————

6. Select the template as **Basic mashup template with absolute positioning**:



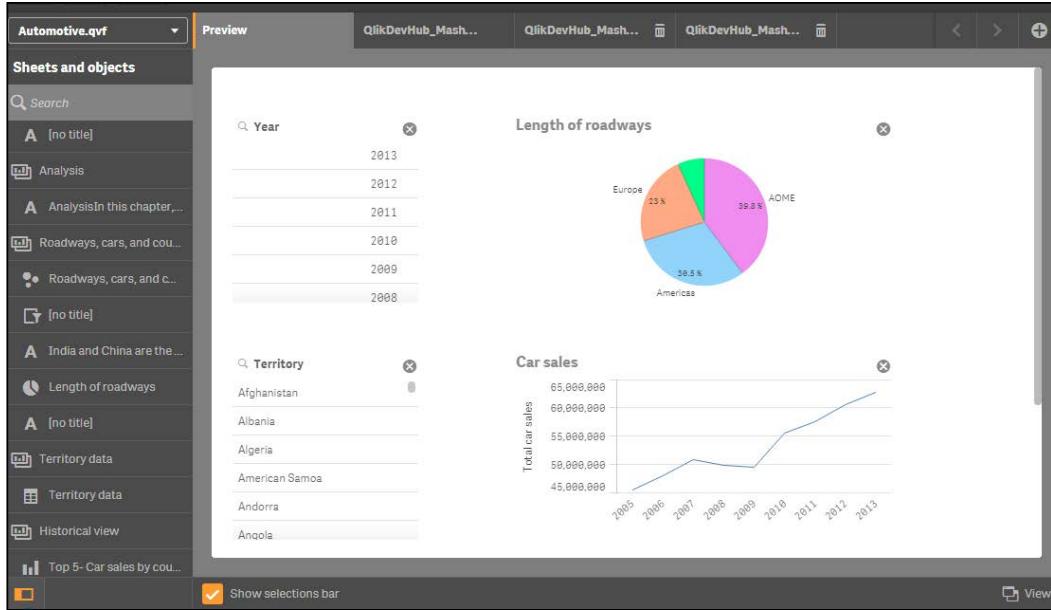
7. Click on **Create & edit**.
8. Once we create the mashup, the **Mashup editor** window reopens. It consists of three panes. The one on the left-hand side gives an option to select a Qlik Sense application on the hub and the objects it may contain. The central pane consists of the **Preview** window for the Mashup Layout and the two main files that help to generate that layout: **.html** and **.js**. The Qlik Sense content is stored in the **.html** file, while the **.js** script file contains the code for the mashups. The right-hand side pane gives options to add **Lists** and **Hypercubes** to the mashup:



9. In the left-hand side pane, select the **Automotive.qvf** application from the dropdown. Once selected, objects within the application will be displayed.

10. Check **Show selections bar** at the bottom of the central pane.
11. Scroll down to the **Roadways, Cars, and Countries** sheet and drag the Filter pane within that sheet onto the layout. This will display the **Year** selection on the layout.
12. Next, drag and drop the **Length of roadways** pie chart onto the layout.
13. Scroll down to the **Country car data** sheet and drag the Filter pane within that sheet onto the layout on the right-hand side. This will display the **Territory** selection on the layout.
14. Drag and drop the **Car sales** trend line chart onto the layout.

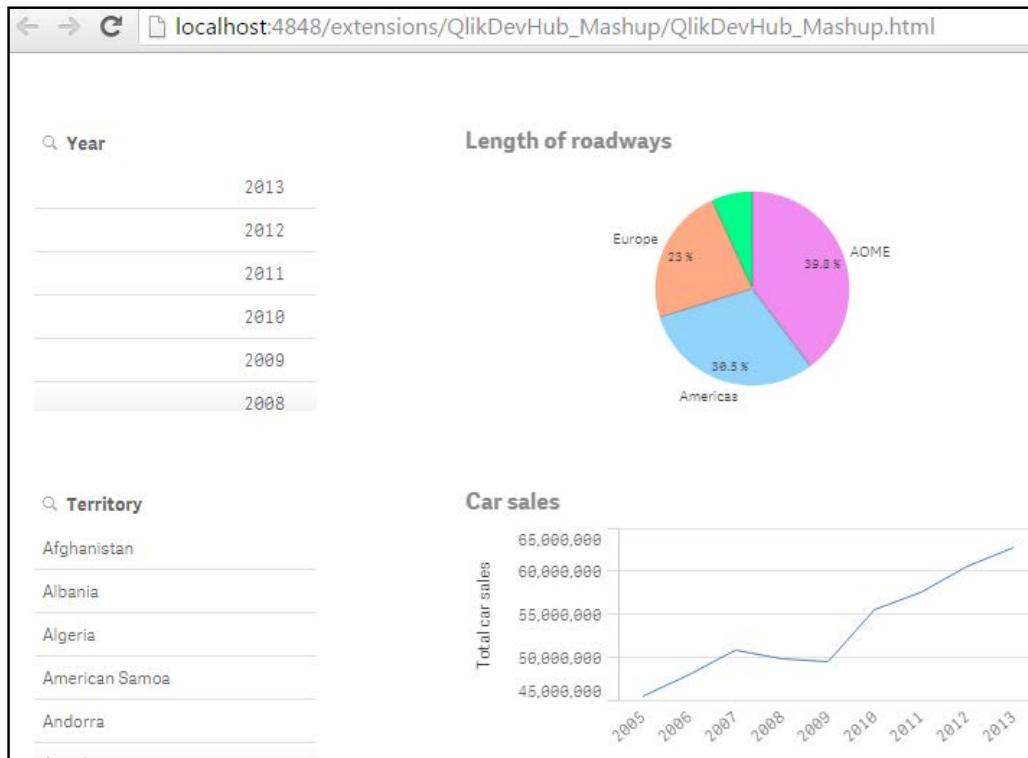
15. The layout should look like this:



16. In order to preview the mashup, click on the **View** button at the bottom of the central pane.
17. The user can make selections on the mashup page similar to Qlik Sense.

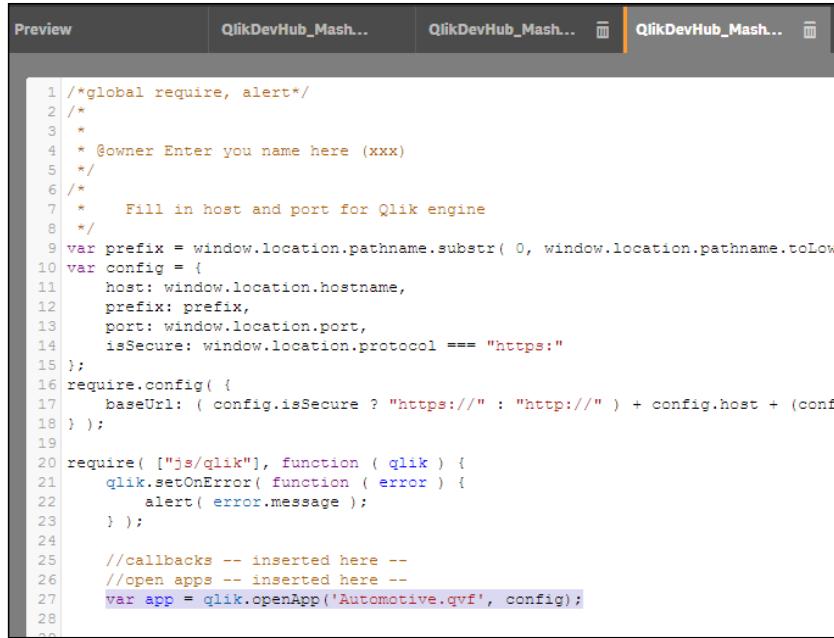
What's New in Version 2.1.1?

18. Click on **Save**. The mashup page can be launched using the Qlik Dev Hub link, selecting the mashup, and clicking on **View**. Alternatively, the mashup link can be shared amongst the users. For example, in our case, it would be like this:



How it works...

1. When we select the application, in our case, `Automotive.qvf`, the JavaScript in the `QlikDevHub_Mashup.js` file gets updated as the following. Every new application opened will add a new line in the open apps section:



```

1 /*global require, alert*/
2 /*
3 *
4 * @owner Enter your name here (xxx)
5 */
6 /*
7 *     Fill in host and port for Qlik engine
8 */
9 var prefix = window.location.pathname.substr( 0, window.location.pathname.toLowerCase().lastIndexOf( '/' ) );
10 var config = {
11     host: window.location.hostname,
12     prefix: prefix,
13     port: window.location.port,
14     isSecure: window.location.protocol === "https:"
15 };
16 require.config({
17     baseUrl: ( config.isSecure ? "https://" : "http://" ) + config.host + ( config.port ? ":" + config.port : "" )
18 });
19
20 require( ["js/qlik"], function ( qlik ) {
21     qlik.setOnError( function ( error ) {
22         alert( error.message );
23     });
24
25     //callbacks -- inserted here --
26     //open apps -- inserted here --
27     var app = qlik.openApp('Automotive.qvf', config);
28 }
29

```

2. When we select the visualizations to display on our mashup page, the `QlikDevHub_Mashup.html` page gets updated. By taking a look at the tab, we can see that the code looks like this:



```

6 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7 <meta name="HandheldFriendly" content="True">
8 <meta name="MobileOptimized" content="320">
9 <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
10 <meta name="apple-mobile-web-app-capable" content="yes">
11 <meta name="apple-mobile-web-app-status-bar-style" content="black">
12 <meta http-equiv="cleartype" content="on">
13 <!--Add host to these 3 links if the mashup is on another webserver than qlik sense-->
14 <link rel="stylesheet" href="../../resources/autogenerated/qlikui.css">
15 <link rel="stylesheet" href="../../resources/assets/client/client.css" media="all">
16 <script src="../../resources/assets/external/requirejs/require.js"></script>
17 <script src="QlikDevHub_Mashup.js"></script>
18
19 <style>
20     article.qvobject
21     {
22         position: absolute;
23         overflow: hidden;
24         padding: 10px;
25     }
26 </style>
27 </head>
28 <body style="overflow:auto">
29 <div id="QV01" style="position: absolute; top: 50px; left: 20px; width: 200px; height: 200px;">
30 <div id="QV02" style="position: absolute; top: 50px; left: 320px; width: 400px; height: 200px;">
31 <div id="QV03" style="position: absolute; top: 300px; left: 20px; width: 200px; height: 200px;">
32 <div id="QV04" style="position: absolute; top: 300px; left: 320px; width: 400px; height: 200px;">
33 <div id="QV05" style="position: absolute; top: 550px; left: 20px; width: 200px; height: 200px;">
34 <div id="QV06" style="position: absolute; top: 550px; left: 320px; width: 400px; height: 200px;">
35

```

There's more...

We can also add lists to the Qlik Sense application using the list builder. The lists are not a part of the Qlik Sense application our mashup is connected to. Along with the lists, we can also add a **Hypercube** with specified dimensions and measures to further enhance our mashups.

Both the options can be found on the right-hand side panel of the **Mashup editor** window.

See also

- ▶ *Embedding Qlik Sense® application on a website using a single configurator*

Embedding Qlik Sense® application on a website using a single configurator

Qlik defines a Single configurator as a tool that provides an easy way of creating simple mashup pages without having to write any code at all. It helps to create a URL that contains the embedded Qlik Sense visualization. A user can embed a sheet, an object, or even a snapshot from the Qlik Sense application. The URL can be embedded onto the desired web page using the `iframe` integration or the `Div` integration.

Getting ready

For this recipe, we will develop a simple HTML page and then embed a Qlik Sense sheet onto the page.

1. In order to generate a web page, copy and paste the following script in a text file:

```
<html>
<title>My Web-page</title>
<body bgcolor="beige">
Qlik Sense
<marquee>Embedding Qlik Sense application in website using
single configurator!</marquee>
</body>
<html>
```

2. Save the file and name it `QlikDevHub_WebPage.html`.

3. The preceding steps create a simple website that displays a Qlik Sense icon and a rolling marquee displaying **Embedding Qlik Sense application in website using single configurator!**:



4. Our next step would be to embed Qlik Sense visualization on this sheet.

How to do it...

1. Open Qlik Dev Hub using the following URL:

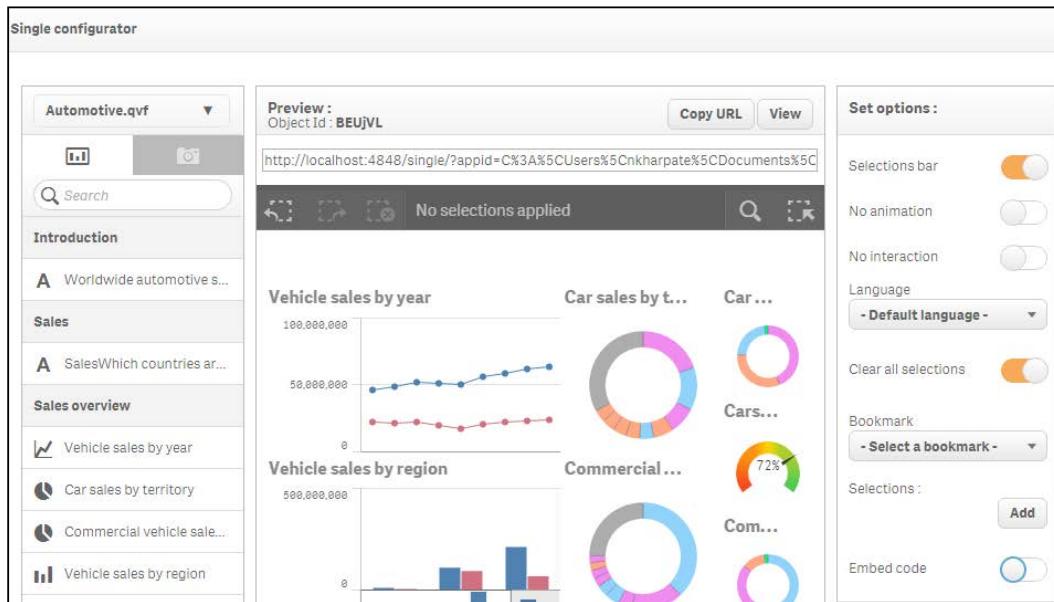
<http://localhost:4848/dev-hub>



2. Click on the **Single configurator** button available in the left-hand side panel of the Qlik Dev Hub. A new **Single configurator** editor window opens. The source panel on the left lists all the applications available on Qlik Sense hub.
3. From the dropdown, select the **Automotive.qvf** application. Once we select the application, observe that all the sheets and the objects within the application are listed underneath. Any snapshots within the application are also listed under the **Snapshot** tab.
4. Now, select the **Sales overview** sheet from the list. On selecting this sheet, two more panes get activated on the screen. The central pane is a **Preview** window that shows the selected sheet along with the **Preview Object ID** option at the top and a URL that contains the HTML information on the object.

What's New in Version 2.1.1? —

5. The right-hand side panel gives the user options to activate or deactivate certain properties in the mashup. For example, show the **Selections bar**, **Chart animations**, **Interaction**, **Bookmarks**, and so on:



6. Make the following changes from the right-hand side panel:
 - ❑ Under **Devhub.general**, ensure that **Disable interaction** is switched off.
 - ❑ Under **Selections**, ensure that the show **Selection bar** and **Clear all selections** are switched on.
7. The code to be inserted within an `iframe` tag in our HTML script for the web page is autogenerated in the Iframe box just above the **Preview** window.
8. The code will look something like this:

```
<iframe  
src="http://localhost:4848/single/?  
appid=C%3A%5CUsers%5Cnkharpate%5CDocuments%  
5CQlik%5CSense%5CApps%5CAutomotive.qvf&sheet=  
BEUjVL&opt=currsel&select=clearall"  
frameborder="0"></iframe>
```
9. Copy and paste it in between the `<marquee>` and `` lines in the HTML script for the web page.
10. We will slightly alter this code for changing the height, width, and the alignment of the frame:

```
<iframe src="http://localhost:4848/single/?appid=C%3A%5CUsers%5Cnkharpate%5CDocuments%5CQlik%5CSense%5CApps%5CAutomotive.qvf&sheet=BEUjVL&opt=currsel&select=clearall"></iframe>
```

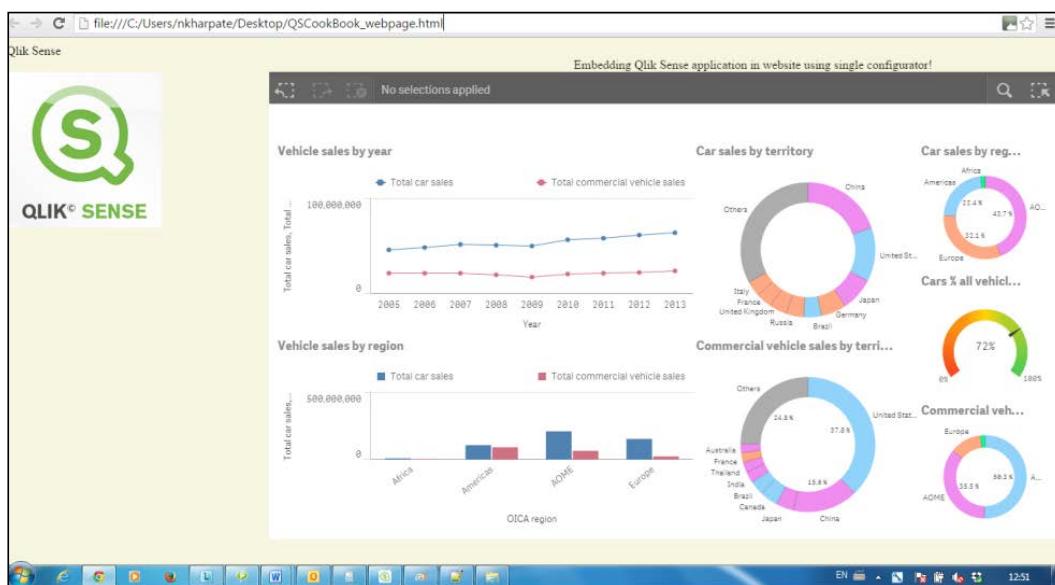
11. The final HTML code for the web page should look like this:

```
<html>
<title>My Web-page</title>
<body bgcolor="beige">
Qlik Sense
<marquee>Embedding Qlik Sense application in website using
single configurator!</marquee>
<iframe
src="http://localhost:4848/single/?appid=C%
3A%5CUsers%5Cnkharpate%5CDocuments%5CQlik%5CSense%5CApps%5CAutomotive.qvf&sheet=BEUjVL&opt=currsel&select=clearall"></iframe>

```

12. Save the changes made in the document.

13. Open the web page to see the embedded Qlik Sense sheet:



How it works...

The embedded sheet on the web page works exactly like a sheet within Qlik Sense. The user can make any selection in the charts and the data would be filtered accordingly.

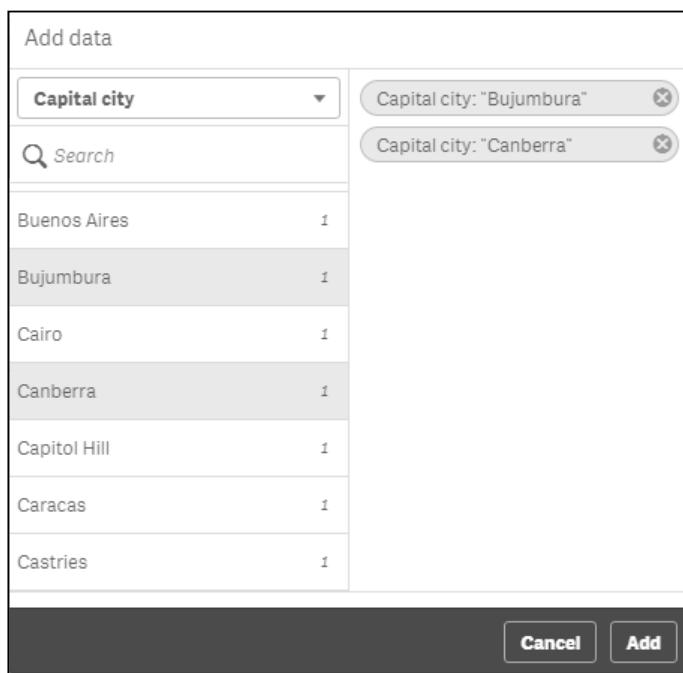
The current selections are displayed at the top. The charts contain active content, and hence they are always in sync with the actual Qlik Sense application.

There's more...

The `iframe` script inserted in the HTML page can further be modified as required. We can very well add multiple objects from different Qlik Sense applications on the same web page.

Another good option to explore in the configurator is adding data.

We can define explicit field value selections within the `iframe` code. The data on the sheet would always adhere to these selections:



See also

- ▶ [Using Qlik Dev Hub to generate mashups](#)

Using the Qlik DataMarket

The Qlik DataMarket allows you to source additional data externally. This data is provided by Qlik to enrich your current reporting data set. It is a "data as a service" cloud offering, which allows you to access a collection of different valuable, up-to-date and ready-to-use datasets.

Getting ready...

For the purpose of this recipe, we will make use of an inline data load which gives the information on sales and the base currency:

1. Create a new Qlik Sense application and call it QS_DataMarket.
2. Load the following data into the QlikSense data load editor:

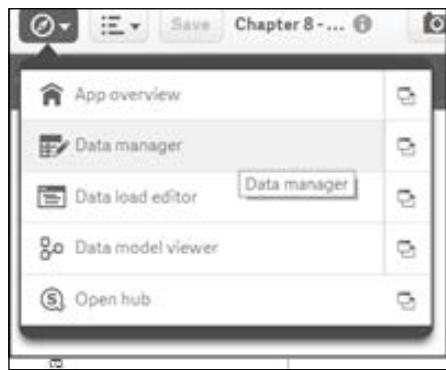
ExampleData:

```
LOAD * INLINE [
    Base currency, Sales
    US dollar, 6300
];
```

3. Save and reload the application.

How to do it...

1. Open the QS_DataMarket application.
2. Create a new sheet called Sales and go to the Edit mode for the sheet.
3. From Qlik Sense Desktop, open the **Data manager** as shown in the following screenshot. Please note that this feature is only available from Version 2.0 onwards.



What's New in Version 2.1.1? ---

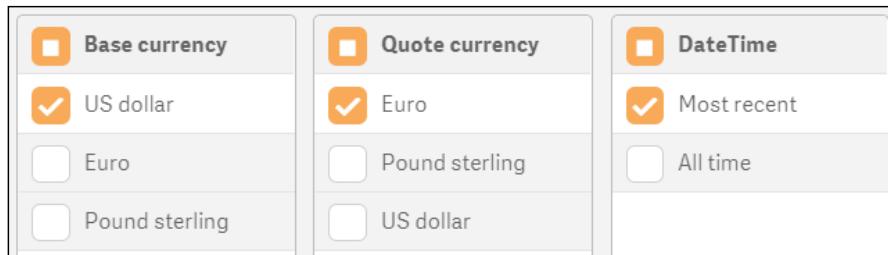
The data manager is a new feature of Qlik Sense. We will cover the different aspects in other recipes; for now, let's look at the Qlik DataMarket.

4. Click on the **Add data** button located on the left-hand side of the **Data manager** screen.

5. Next, click on the  **Qlik DataMarket** button.

6. For this example, we will click on the  **Currency** option.

7. Next, select the  option.
8. The next screen allows you to choose what data you want to load. Select the options as shown in the following screenshot:



9. Click on the **Load and finish** button at the bottom-right corner of the screen.
10. Once the load has finished, go to the application editor and drag a KPI object from the asset pane onto the main content pane.
11. Create a new master item Measure with the name **Sales (\$)** and the following expression:
`Sum(Sales)`

12. Create a second master item Measure with the name Sales (in selected quote currency) and the following expression:

```
Sum ({<[Quote currency]=${(='[' & GetFieldSelections([Quote currency]) & ']')}>}Sales * [Exchange rate])
```

13. Drag the first and second measures onto the KPI object. The final object should look like the following image:



How it works...

In this example, we enrich the existing internal data set with an external source. Here, we simply add a euro conversion rate from our dollar amounts but we could have added several other currency conversions. The limitations of how much data you can augment to your internal data set is based on the data Qlik provide in the DataMarket, and having the right fields in your internal data set to cross reference to the right DataMarket field. The ones shown in this example are free. To get access to even more data sources in the DataMarket, you have to pay a subscription fee. The price list for these subscriptions can be obtained from Qlik.

The current offerings in the Qlik DataMarket are broken down into six categories:

- ▶ Business
- ▶ Currency
- ▶ Demographics
- ▶ Society
- ▶ Weather
- ▶ Economy

What's New in Version 2.1.1? _____

These categories are shown in the following screenshot:

The screenshot shows the 'Select a data source' screen in Qlik Sense. At the top, there are three buttons: 'Connections' (with a person icon), 'Connect my data' (with a lightning bolt icon), and 'Qlik DataMarket' (with a green 'D' icon). Below these are six categories arranged in a 3x2 grid:

| Category | Description |
|--|---|
| Business Data on firms and establishments classified by industry, country and locality. | Currency Foreign exchange rates for major world currencies. Updated daily. |
| Demographics Detailed population data for North America, Europe and the rest of the world. | Society Major socioeconomic data on wages, unemployment and poverty. |
| Weather Current and historical weather conditions for major locations around the world. | Economy Major economic indicators featuring data on GDP, prices, education and environment. |

Being able to expand your own core data with that in the Qlik Data Market allows you to take an outside-in view of the business and its environment, helping you to explore and connect market trends to see opportunities and challenges. Qlik plans to keep adding to the DataMarket with more sources in future releases. The data sources are updated in Qlik as they are updated in the source systems. For example, exchange rate information will likely be updated daily, whereas country population data is more likely to be updated annually.

See also

- ▶ [Using the Qlik Dev Hub in Qlik Sense® 2.1.1](#)

Creating dynamic charts in Qlik Sense®

To increase the flexibility of a single chart object, you can set it up so that the dimension used is based on what the user wants to see. This is a much more efficient use of space for single sheets and makes the whole experience much more dynamic.

Getting ready

For the purpose of this recipe, we will make use of the sales information for different fruits as defined in the script:

1. Create a new Qlik Sense application and call it QS_DynamicCharts.
2. Load the following data into the data load editor:

```
Transactions:  
Load  
    Mod(IterNo(),26)+1 AS Period,  
    Pick(Ceil(3*Rand()),'Standard','Premium','Discount') AS  
        ProductType,  
    Pick(Ceil(6*Rand()),'Apple','Orange','Cherry','Plum','Fig',  
        'Pear') AS Category,  
    Pick(Ceil(3*Rand()),'Heavy','Medium','Light') AS Weight,  
    Pick(Ceil(2*Rand()),'2013','2014') AS Year,  
    Round(1000*Rand()*Rand()*Rand()) AS Sales  
Autogenerate 20  
While Rand()<=0.5 or IterNo()=1;  
  
SET vDimension = 'GetFieldSelections(Dimensions)';  
  
Dimensions:  
LOAD * INLINE [  
    Dimensions  
    Weight  
    ProductType  
    Category  
    Period  
];
```

How to do it...

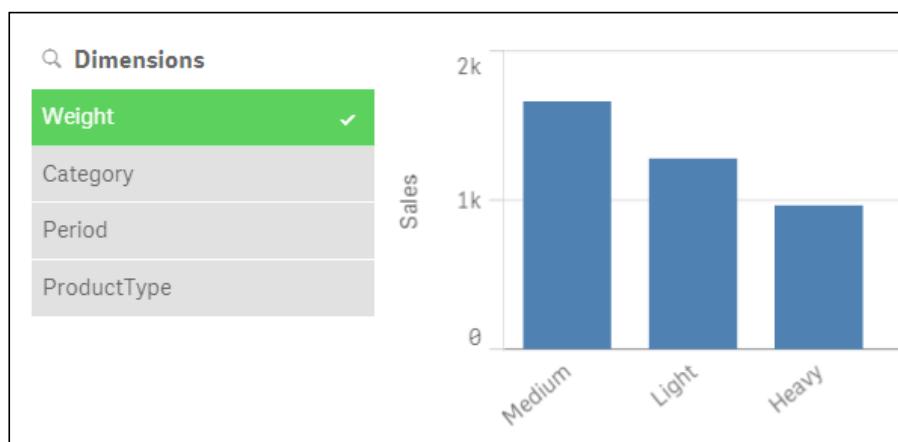
1. From the **App overview**, create a new sheet and enter the Edit mode.
2. Add the field **Dimensions** onto the main content pane.
3. Drag a bar chart object onto the content pane.
4. Add the following expression as a measure:

Sum(Sales)

5. Add a following calculated dimension by clicking on the  button:

```
= '[' &  
    Pick(Match(${vDimension}, 'Weight', 'ProductType',  
    'Category'), Weight, ProductType, Category) & ']'
```

6. Enter one click of the space bar as the label to make it appear as if there is no dimension label.
7. Exit the editor mode and select a value in the **Dimensions** field.
8. The final product should resemble the following screenshot:



How it works...

In the script loaded at the beginning of the recipe, we set a variable called `vDimension`. The `GetFieldSelections()` function will return the values selected in the field we specify inside the brackets `GetFieldSelections(Dimensions)`. The `Dimensions` field is simply a hardcoded list of specific fields in the Data model. The code we wrote in the dimension field of the chart uses this variable value to set the dimension dynamically to whatever value the user picks in the list box we created.

There's More....

If you are running Qlik Sense 2.1.1, then you can now enter expressions in the chart title area. If so, enter the following code:

```
= 'Showing Sales by: ' & ${vDimension}
```

This displays the selected dimension in the chart title dynamically. If you are running an earlier version of Qlik Sense, simply leave the label blank. After this, I would suggest creating a textbox explaining the chart, which will display whatever the value is, in the associated list box. You can also use the preceding expression in the explanation textbox.

Using Smart Search

Smart Search is a new feature of Qlik Sense. As with most search features, you type in what you are looking for and a list of possibilities are returned. These are based on the field values in your data. This recipe shows you how to tailor what is returned when performing a smart search.

Getting ready

For the purpose of this recipe, we will make use of an inline data load which gives the release information for different labels. Load the following data in the script load editor:

```
Data:  
LOAD * INLINE [  
    Label, DJ, Next Album Release, Release Year  
    Blunderbuss Records, Kevin Mullaney, The Chat, 2016  
    Weirdo Cats, Heather McKay, Unknown,  
    Dragon Disks, Rhys Hayward , Unknown,  
    Caped Capers, Simon Conyers, No Fashion, 2016  
    Shadow Giggles, Ski Mask, Electro Ski, 2015  
    Fiddle Pits, Isabel Franken, Boogie Fingers, 2015  
];
```

How to do it...

1. Open the main content pane and click on the  button in the top-right corner.
2. Type `ski` into the search box and note that two values are returned, as shown in the following image:



What's New in Version 2.1.1? ---

3. Now, go to the script editor and type in the following code:

```
Search Exclude "Next Album Release", "Release Year";
```
4. Save and reload the application.
5. Repeat step 2. This time, only a single value is returned in the smart search results.

How it works...

In this example, we used the Search Exclude function to restrict two fields from the Data model we don't want users to be able to search on. The fields are called Next Album Release and Release Year. There is also a Search Include function, where listing only the fields you want users to search on is simpler than listing those you don't want to include. Examples of fields you would normally exclude are Key fields that join the tables in your Data model. Removing unnecessary fields also helps with the performance of searches.

There's More....

Other than just controlling what fields are accessible via the smart search functionality, there are different ways to perform a search. This involves using special characters other than just typing in the literals you are looking for. Examples are given in the following table:

| Character | Example | Description |
|--------------|-----------------|--|
| " " | "Orange Juice" | Encapsulating the value in quotes makes Qlik Sense search for the whole word instead of Orange and Juice separately. |
| + | +Orange +Juice | Finds strings that include both words although they can be in any order. Such words could be <i>Orange and apple juice</i> or <i>Juice from oranges</i> . |
| ~ | ~Orange | This is Fuzzy Search, where the values are ranked and sorted according to the similarity to the search string. This search only works when selecting the search icon for an individual field and does not work in the global search at the top of the screen. |
| >, <, >=, <= | =Sum(Sales)>100 | Expression searches also only act on a single field similar to Fuzzy Search. The results are returned based on the aggregation. Numeric searches such as >01/01/2015 also work without an explicit aggregation function and can be used to narrow down the search. |

| Character | Example | Description |
|-----------|---------|--|
| * | Oran* | The * symbol is a wildcard character and will return any values that start with the letters <i>Oran</i> . The wildcard character replaces a single or block of characters in a search string. It can be used at the start and middle of a search string, for example, *Oran, Oran*, or *Oran*. |
| ? | Oran?e | The ? character is also a wildcard that represents a single character. |

QlikView also has other types of searches, such as Numeric, Fuzzy, Expression, and Compound Search. I believe these are on the road map, although they are not included in the product to date. To see if these features have been released in later versions of Qlik Sense, please refer to the Qlik Sense online help feature accessed via the hub.

See also

- ▶ *Using smart data load profiling*

Using smart data load profiling

As you know from the earlier chapters on accessing data, Qlik Sense makes associations between tables using similar field names. As of Qlik Sense Version 2.0, there is a data profiling tool that can be used to help you make the correct table associations.

Getting ready

1. Create a folder on your local drive called `TestData`. For this example, we will use the **C:** drive: `C:\TestData`.
2. Create a folder library connection to the directory above in the data load editor.
3. Load the following script into the application:

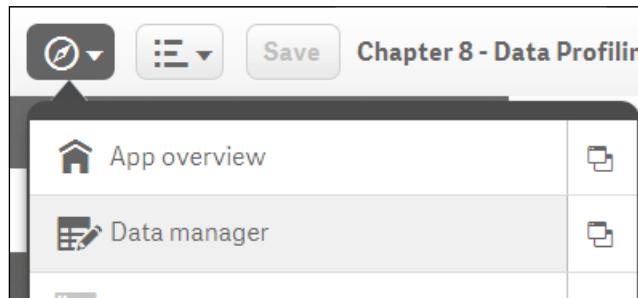
```
Transactions:
LOAD DATE(Date#( TransactionDate,'DD/MM/YYYY')) as
    TransactionDate ,Sales INLINE [
        TransactionDate, Sales
        01/01/2013, 1000
        02/01/2013, 3000
        03/01/2013, 500
        04/01/2013, 4000
        05/01/2013, 2000
```

What's New in Version 2.1.1? -----

```
];  
  
Calendar:  
LOAD DATE(Date#( TransactionDate,'DD/MM/YYYY')) as Date,  
    Month,Year INLINE [  
    Date, Month, Year  
    01/01/2013, Jan, 2013  
    02/01/2013, Jan, 2013  
    03/01/2013, Jan, 2013  
    04/01/2013, Jan, 2013  
    05/01/2013, Jan, 2013  
];  
  
STORE Transactions INTO  
[lib://TestData/Transactions.txt] (txt);  
Drop Table Transactions;  
  
STORE Calendar INTO [lib://TestData/Calendar.txt] (txt);  
Drop Table Calendar;
```

How to do it...

1. Click the Navigation dropdown button on the top-left and select the **Data manager**, as shown here:

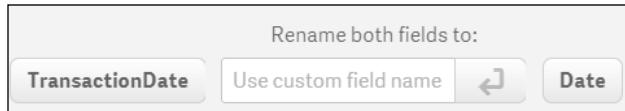


2. Select **Add data** from the menu bar on the left-hand side.
3. Click on the **TestData** library connection we established earlier.
4. Select the **Transaction.txt** file as shown here, and click on the next arrow at the bottom of the page:



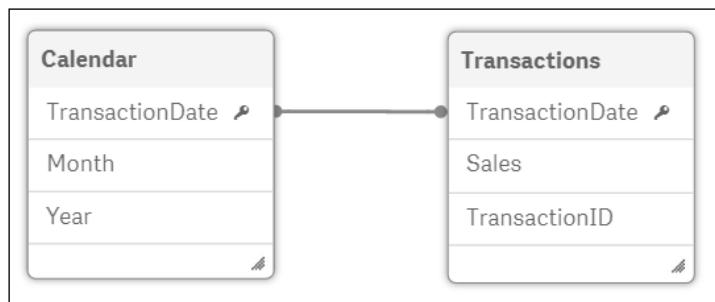
5. You will see a preview of the data to be loaded. Click on **Load and finish** at the bottom of the page.
6. Once loaded, you will return to the data manager and the transactions table will be listed on the left. Let's load the second table. Repeat steps 2, 3, and 4 only. Click on the **Calendar.txt** data this time.
7. Instead of clicking on the **Load and finish** button, click on the button.

-
- A screenshot of a 'Recommendations' dialog box. It shows a list of field mappings: 'TransactionDate' is mapped to 'Date' with a '100%' completion rate. The 'Date' entry is highlighted with a green border. Below the list is a button labeled 'Profile >'.
8. Click on the option.
 9. Under the Rename fields option, click on the button labeled **TransactionDate**.



10. Click on **Load and finish** in the bottom-right hand corner.
11. Close the execution window and click on **Save**.

12. If you open the Data model viewer from the main hub menu now, the two tables should now be joined, as shown here:



How it works...

The script we loaded at the beginning of the recipe simply generates the data we use in the data load editor. The profiler looks at these data files to make recommendations on fields you should use to join the two datasets together. In this example, there is only one suggestion made, which is for the **TransactionDate** field and the **Date** field.

If there are more, you can step through the various suggestions using the arrows in the load editor page. This is shown in the following screenshot:



Our example identified that there is a 100 percent match on the values contained between both of our data sets but the name of those fields are different. As such, one or both of the fields need to be renamed in order to associate the tables. The fields were renamed automatically when we clicked on the button labeled **TransactionDate**. We could have entered a new name for both fields by typing in the **Rename both fields to** box.

There are several other warnings the data profiler will make you aware of, depending on the data you load. One example is if there were fields with the same name but data that didn't match. Another example is if there are multiple possible connections you could make; in this example, the profiler will recommend you to either keep one of the fields as a key then rename the others, or break the table association.

There's More....

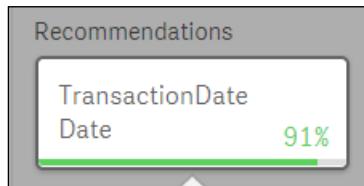
The Data manager allows you to bring in data and make associations from a number of sources without ever seeing a line of code. The code is still present, but it is generated automatically and saved in a system generated script tab. After completing this recipe, open the data editor to see the new tab, as show in the following image:



It is also worth making clear that the data profiler works without finding 100 percent matches on shared field values between tables. To test this, repeat the recipe with the following line of code added to the end of the transactions table in the *Getting ready* section of the recipe:

6, 06/01/2013, 100

The profiler works in exactly the same way as recommending the correct field link, but now it will inform you there is a **91%** match between the values contained in both tables:



Conclusion

With this we come to the end of this book. This book is a small effort to help tackle day-to-day issues faced by Qlik Sense developers. We have introduced the users to some of the key traits of Qlik Sense through recipes. The book travels through the basics of Qlik Sense to more advanced scripting, calculations, and extensions. Keeping in mind the importance of User Interface we have also dealt with a few recipes related to the best practices in design of a Qlik Sense application.

We don't undermine the fact that there may be many different ways to overcome the challenges discussed in some of the recipes but we have tried to present the best approach in this book.

The knowledge shared in this book is something that we have experienced and learnt over the course of many years of business intelligence implementations. We believe that learning never ends and having worked with Qlik product suite for so long, we are still learning new things every day. In fact, even while writing this book, both of us exchanged many ideas, which in a way were helpful in expanding our knowledge base.

What's New in Version 2.1.1? _____

Having said that, our learning is more valuable only when it is shared with others. This book is the source of sharing our thoughts with the wider world.

We would consider our efforts to be worthy if the aspiring as well as experienced Qlik Sense developers find our book helpful.

Wish you an enjoyable journey with Qlik Sense..!!

Appendix

Keyboard shortcuts in Qlik Sense® Desktop

The following keyboard shortcuts assume the use of MS Windows. For Mac OS, use *Cmd* instead of *Ctrl*:

| Shortcut | Action |
|--|--|
| <i>Ctrl + P</i> | This prints the current view or active sheet/story. |
| <i>Ctrl + C</i> | This copies the selected item to the clipboard. |
| <i>Ctrl + X</i> | This 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. |
| <i>Ctrl + V</i> | This pastes the most recently copied item from the clipboard. |
| <i>Ctrl + Z</i> | Using this combination, you can undo an action. You can repeat it to undo earlier actions. |
| <i>Ctrl + Y (Cmd + shift + Z for Mac OS)</i> | Using this combination, you can redo actions. |
| <i>Ctrl + H</i> | This opens the online help in the context of the current function, while in the data load editor or the expression editor. |
| <i>Ctrl + F</i> | This opens smart search. |
| <i>Ctrl + E</i> | In the sheet view, this opens and closes the editing of the selected sheet. |
| <i>Ctrl + S</i> | This saves changes to the app. |
| <i>Ctrl + O</i> | This opens an app copied to the clipboard using <i>Ctrl + C</i> . |
| <i>Ctrl + A</i> | This selects all the code in the data load editor. |

| Shortcut | Action |
|---------------------------|--|
| <i>Ctrl + D</i> | This deletes the content of the current line in the data load editor or in the expression editor. |
| <i>Ctrl + K</i> | This comments or uncomments the selected lines in the data load editor. |
| <i>Ctrl + O0</i> | This inserts a test script in the data load editor. |
| <i>Tab</i> | This indents the code in the data load editor. |
| <i>Shift + Tab</i> | This outdents the code in the data load editor. |
| Left arrow | This navigates to the previous slide in the storytelling view. |
| Right arrow | This navigates to the next slide in the storytelling view. |
| Up arrow | This scrolls up in a table. |
| Down arrow | This scrolls down in a table. |
| <i>Ctrl + left arrow</i> | This navigates to the previous sheet in the sheet view. |
| <i>Ctrl + right arrow</i> | This navigates to the next sheet in the sheet view. |
| <i>Ctrl + up arrow</i> | This navigates to the first sheet of the app in the sheet view. |
| <i>Ctrl + down arrow</i> | This navigates to the last sheet of the app in the sheet view. |
| <i>Esc</i> | This exits play mode in the storytelling view. This deselects a visualization when editing in the sheet view. This deselects an object. This undoes selections in a visualization. This closes a dialog or window. |
| <i>Delete</i> | This deletes the selected item. |
| <i>Backspace</i> | This deletes the selected item. |
| <i>Enter/Return</i> | This performs the actions for the active option or button (for example, in dialogs). |
| <i>Ctrl + +</i> | Using this combination, you can zoom in. |
| <i>Ctrl + -</i> | Using this combination, you can zoom out. |
| <i>Ctrl + O</i> | This resets zooming. |

Module 3

Predictive Analytics Using Rattle and Qlik Sense

Create comprehensive solutions for predictive analysis using Rattle and share them with Qlik Sense

1

Getting Ready with Predictive Analytics

Analytics, predictive analytics, and data visualization are trendy topics today. The reasons are:

- Today a lot of internal and external data is available
- Technology to use this data has evolved a lot
- It is commonly accepted that there is a lot of value that can be extracted from data

As in many trendy topics, there is a lot of confusion around it. In this chapter, we will cover the following concepts:

- Introducing the key concepts of the book and tools we're going to use
- Defining analytics, predictive analytics, and data visualization
- Explaining the purpose of this book and the methodology we'll follow
- Covering the installation of the environment we'll use to create examples of applications in each chapter

After this chapter, we'll learn how to use our data to make predictions that will add value to our organizations. Before starting a data project, you always need to understand how your project will add value to the organization. In an analytics project, the two main sources of value are cost reduction and revenue increase. When you're working on a fraud detection project, your objective is to reduce fraud; this will lead into a cost reduction that will improve the margin of the organization. Finally, to understand the value of your data solution, you need to evaluate the cost of your solution. The real value added to an organization is the difference between the provided value and the total cost.

Working with data to create predictive solutions sounds very glamorous, but before that we'll learn how to use Rattle to load data, to avoid some problems related to the quality of the data, and to explore it. Rattle is a tool for statisticians, and, sometimes, we need a tool that provides us with a business approach to data exploration. We'll learn how to use Qlik Sense Desktop to do this.

After learning how to explore and understand data, we'll now learn how to create predictive systems. We'll divide these systems into unsupervised learning methods and supervised learning methods. We'll explain the difference later in this book.

To achieve a better understanding, in this book we'll create three different solutions using the most common predictive techniques: Clustering, Decision Trees, and Linear Regression.

To present data to the user, we need to create an application that helps the user to understand the data and take decisions; for this reason we'll look at the basics of data visualization. Data Visualization, Predictive Analytics and most of the topics of this book are huge knowledge areas. In this book we'll introduce you to these topics and at the end of each chapter you will find a section called *Further learning* where you will find references to continue learning.

Analytics, predictive analytics, and data visualization

In January 2006, Thomas H. Davenport, a well-known American academic author, published an article in *Harvard Business Review* called *Competing on Analytics*. In this article, the author explains the need for analytics in this way:

"Organizations are competing on analytics not just because they can – business today is awash in data and data crunchers – but also because they should. At a time when firms in many industries offer similar products and use comparable technologies, business processes are among the last remaining points of differentiation. And analytics competitors wring every last drop of value from those processes."

After this article, companies in different industries started to learn how to use traditional and new data sources to gain competitive advantages; but what is analytics?

Today, the term analytics is used to describe different techniques and methods that extract new knowledge from data and communicate it. The term comprises statistics, data mining, machine learning, operations research, data visualization, and many other areas.

An important point is that analytics will not provide any new value or advantage by itself; it will help people to take better decisions. **Analytics** is about replacing decisions based on feelings and intuition with decisions based on data and evidence.

Predictive analytics is a subset of analytics whose objective is to extract knowledge from data and use it to predict something. Eric Siegel in his book *Predictive Analytics* describes the term as:

"Technology that learns from experience (data) to predict the future behavior of individuals in order to drive better decisions."

Generally, in real life, an accurate prediction is not possible, but we can extract a lot of value from predictions with low accuracy. Think of an insurance company, they have a lot of claims to review, but have just a few people to do it. They know that some claims are fraudulent, but they don't have enough people and time to review all claims. They can randomly choose some claims or they can develop a system that selects the claims with a higher probability of fraud. If their system predictions are better than just guessing, they will improve their fraud detecting efforts and they will save a lot of money in fraudulent claims.

As we've seen, everything is about helping people to take better decisions; for this reason we've got to communicate the insights we've discovered from data in an easy to understand and intuitive way, especially when we deal with complex problems.

Data visualization can help us to communicate our discoveries to our users. The term, data visualization, is used in many disciplines with many different meanings. We use this term to describe the visual representation of data; our main goal is to communicate information clearly and efficiently to business users.

In this introduction, we've used the term *value* many times and it's important to have an intuitive definition. We develop software solutions to obtain a business benefit; generally, we want to increase income or reduce cost. This business benefit has an economic value; the difference between this economic value and the cost of developing the solution is the value you will obtain.

Usually, a predictive analytics project follows some common steps that we call the predictive analytics process:

1. **Problem definition:** Before we start, we need to understand the business problem and the goals.
2. **Extract and load data:** An analytics application starts with raw data that is stored in a database, files, or other systems. We need to extract data from its original location and load it into our analytics tools.
3. **Prepare data:** Sometimes, the data needs transformation because of its format or because of poor quality.
4. **Create a model:** In this step, we will develop the predictive model.
5. **Performance evaluation:** After creating the model, we'll evaluate its performance.
6. **Deploy the model and create a visual application:** In the last step, we will deploy the predictive model and create the application for the business user.

The steps in this process don't have strict borders; sometimes, we go back and forth in the process.

Purpose of the book

This is not a technical guide about R and Qlik Sense integration, or a Rattle guide for software developers. This book is an introduction to the basic techniques of predictive analytics and data visualization. We've written this book for business analysts, and people with an IT background, but without analytics experience.

"Tell me and I forget, teach me and I may remember, involve me and I learn."

— Benjamin Franklin

We believe that the best way to learn is by practicing, and for this reason this book is organized around examples, which you can do with a simple Windows computer. Don't be afraid, we will use two software tools, Rattle and Qlik Sense Desktop, in order to avoid complex code. To create the predictive analysis, we'll use *Rattle* and for data visualization, we'll use *Qlik Sense Desktop*.

There are two ways of using Rattle, or R, and Qlik Sense Desktop together. These are listed as follows:

- In the first approach, it is possible to integrate Qlik Sense Desktop and R. The business users select some data. Qlik Sense Desktop sends this selected data to an R server, the server processes the data and performs a prediction. The R server returns the data to Qlik Sense Desktop, and this shows the data to the user. This model has a great advantage – the interactivity, but it also has a disadvantage; it requires additional software to integrate the two different environments.
- The second approach is based on two steps. In the first step, the R environment loads the data, performs the prediction, and stores the original data with the prediction. In the second step, Qlik Sense Desktop loads the data and the prediction, and shows it to the business user. This second approach has a great advantage which is simplicity, but also has a disadvantage which is the lack of interactivity.

In this book, we'll use the second approach because in predictive analytics choosing the appropriate model is the key. For this reason we want to focus on introducing you to different models, avoiding the technical stuff of integration. We'll use Rattle and Qlik Sense Desktop in a two-step process. We'll load data in Rattle to enrich it with a predictive model and then load it in Qlik Sense Desktop to share it by creating data visualizations. This process is illustrated in the following diagram:



Introducing R, Rattle, and Qlik Sense Desktop

In this section, we will introduce the tools we'll use in this book: R, Rattle, and Qlik Sense Desktop.

R is a free programming language for statistics and graphics available under the terms of the Free Software Foundation's **General Public License (GNU)**. The R language is widely accepted for statistical analysis and data mining. There is a big community of developers that develop new packages for R, such as Rattle.

R is a very powerful and flexible programming language, but to create predictive models with R you need to be a skilled programmer. For this reason, we will use Rattle.

Rattle is a **Graphical User Interface (GUI)** for data mining developed by Graham Williamson using R. Similar to R, Rattle is also licensed under the GNU. R and Rattle are the predictive analysis environments that we will be using in this book.

Using Rattle, we'll be able to load and prepare data, create a predictive model, and evaluate its performance without writing R code; Rattle will write the code for us.

In order to create a visual and intuitive application for the business user, we'll use Qlik Sense Desktop, the personal and free version of Qlik Sense. Qlik Sense is a self-service data visualization tool developed by Qlik.

We'll use Qlik Sense Desktop instead of Qlik Sense Enterprise because we want to build a free learning environment to develop the examples of this book. For the propose of this book, Qlik Sense Desktop and Qlik Sense are very similar. When you deploy your applications in Qlik Sense Enterprise, the platform provides you:

- Data governance.
- Security
- Scalability
- High availability

Qlik has two different tools for data analysis and data visualization: QlikView and Qlik Sense. Each tool is designed to solve a different problem:

- With QlikView, developers have a powerful tool to create guided analytic applications
- With Qlik Sense, business users can create their own analysis and visualizations with drag and drop simplicity

We will use Qlik Sense Desktop instead of QlikView because the book is written for business users and analysts, and Qlik Sense is designed to provide business users with the ability to create visualizations on their data.

- Qlik Sense has two different editions:
- Qlik Sense Enterprise, a sever based edition for use in organizations.
- Qlik Sense Desktop, a desktop edition for personal use.
- In this book we'll use Qlik Sense Desktop to complete the examples. This edition is free for personal utilization.

R and Rattle can be installed on Windows, Mac OS, and Linux, but Qlik Sense Desktop can only be installed on a Windows machine. For this reason, we will use a Windows-based computer for this book. Qlik Sense Desktop and R load all data into memory; we suggest that you use a 64-bit computer instead of a 32-bit computer.

In order to install R, Rattle, and Qlik Sense Desktop, you'll need administration rights, and an Internet connection to download the software.

Installing the environment

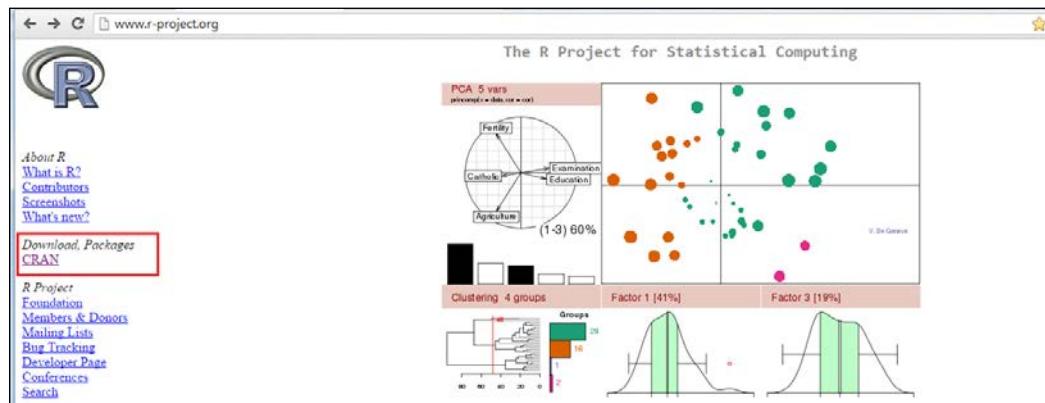
In the examples, we'll use Rattle and Qlik Sense Desktop, but, as we've explained, Rattle is an R package and we need to install R too. We will follow these steps:

1. Download and install R.
2. Download and install Rattle.
3. Download and install Qlik Sense Desktop.

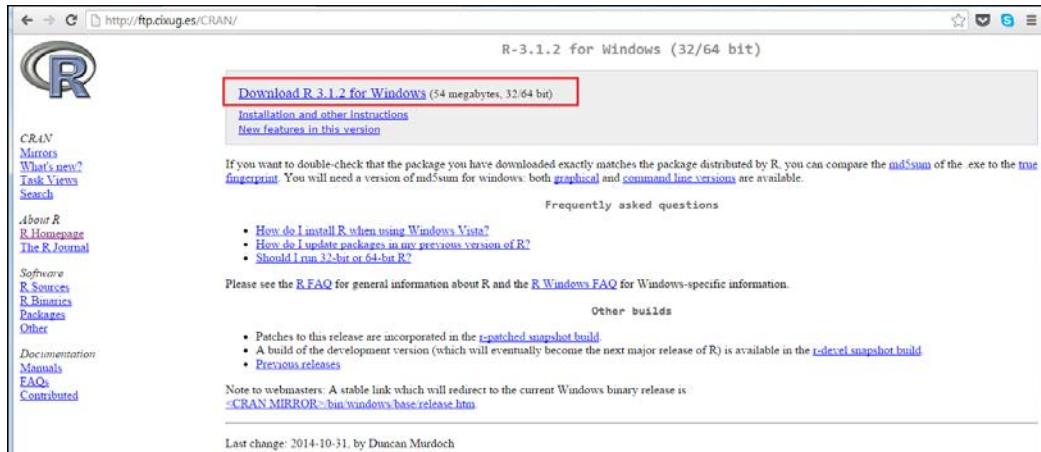
Downloading and installing R

These steps must be followed for installing R:

1. Go to the homepage of **R Project for Statistical Computing** at www.r-project.org.
2. In the navigation bar, click on **Comprehensive R Archive Network (CRAN)** and you will be redirected to a list of CRAN mirrors. Choose a download mirror that is the closest to your geographic location, as shown here:



3. You will reach a different page; choose **Download R for Windows**, and in the following page click on **install R for the first time**.
4. Finally, you will reach the download page. As of writing this book, the latest version for Windows was 3.1.2. Click on **Download R 3.1.2 for Windows** to download the installation program, as shown in this screenshot:



5. Run the installation program, R-3.1.2-win.exe, to start the process. Depending on the level of security of your system, it will ask you for permission to execute the program and to make modifications on your system. You have to allow this to start the process.
6. In the next step, you have to choose a language; choose **English**. For the rest of the installation process, leave the default options.
7. When the installation process finishes, you will have two new icons on your desktop – **R i386 3.1.2** and **R x64 3.1.2**; use the first one if you are using a 32-bit computer and the second one if you are using a 64-bit computer:

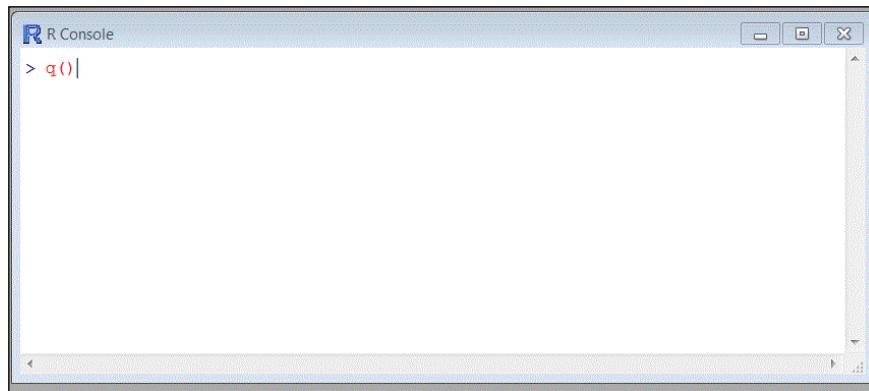


Starting the R Console to test your R installation

The **R Console** is a window used to interact with R language; you can type commands and functions here, and you will see the results in the same window. We will not focus on R, so we'll only learn the commands needed to work with Rattle.

The following steps are needed to start and close **R Console**:

1. Double-click the R icon to start the **R Console**.
2. To exit the **R Console**, type `q()` and press *Enter*, as shown here:



Downloading and installing Rattle

Rattle is an R package, which is a collection of functions and data someone else has developed, and we can use it in our programs. If you already have some hands-on experience with R, then this task should be a much lighter task.

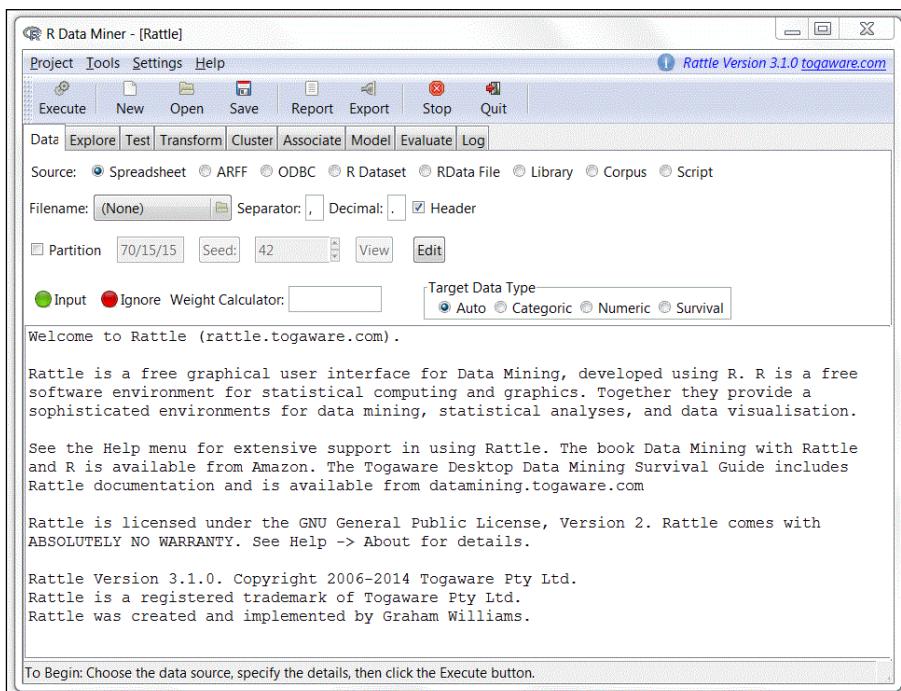
Before starting with the installation, remember that you need an active Internet connection. The following are the steps to install Rattle:

1. We will install Rattle from **R Console**; to open it double-click on the **R x64 3.1.1** desktop icon.

2. In **R Console**, type `install.packages("rattle")` and press *Enter*. The **R Console** will show you a list of **CRAN** mirrors; choose a download mirror that is the closest to your geographic location and R will download the Rattle package, as shown here:



3. After you have downloaded it, type `library(rattle)` and R will load the Rattle package into memory, and you will be able to use it. Use the `rattle()` command to start Rattle:



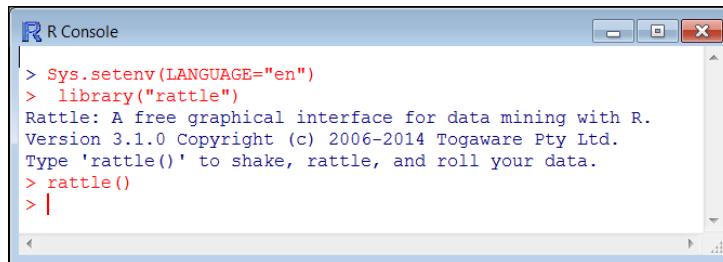


Rattle needs other R packages to work properly, the first time you open Rattle, the system will ask your permission to install some packages; in order to execute Rattle, you have to accept the installation of these packages.

4. To exit, click on the **Quit** icon from Rattle GUI and type `q()` in the **R Console**:



If you are from a non-English speaking country, you've probably installed everything in English, but Rattle's texts appear in your own language. Rattle will work fine in your language, but this book is written in English and it will refer to Rattle's functions and menus using English names. If you prefer to execute Rattle in English, quit Rattle and type `Sys.setenv(LANGUAGE="en")` in your **R Console** and start Rattle again.



Rattle's menu now appears in English.

Installing Qlik Sense Desktop

In order to install **Qlik Sense Desktop**, you need a 64-bit computer with the following specifications:

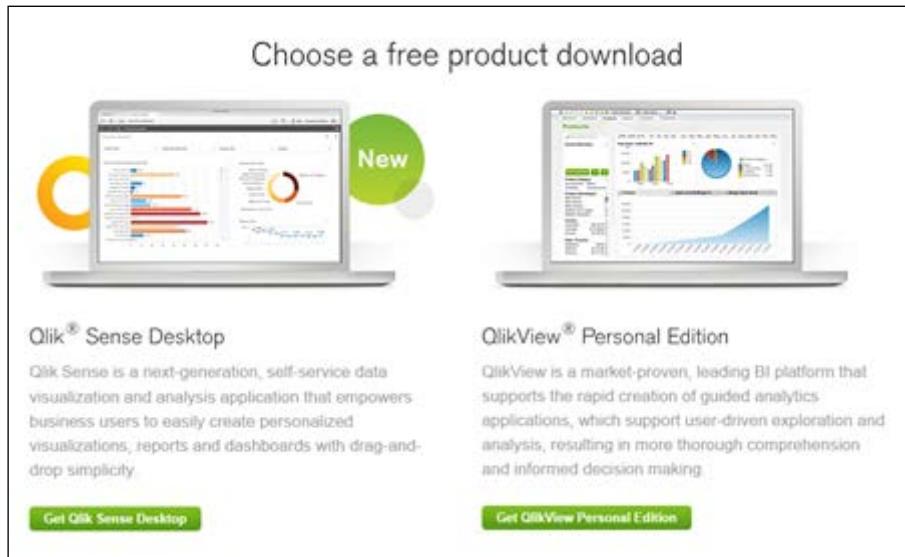
- Windows 7 or Windows 8.x
- Administrator privileges

- .NET Framework 4.0
- 4 GB of RAM memory
- 500 MB of disk space
- Intel Core 2 Duo processor or higher

Probably, you are not sure if you have .NET Framework on your computer; don't worry if you don't have it, the installer will offer to install it.

The following steps are used to install **Qlik Sense Desktop**:

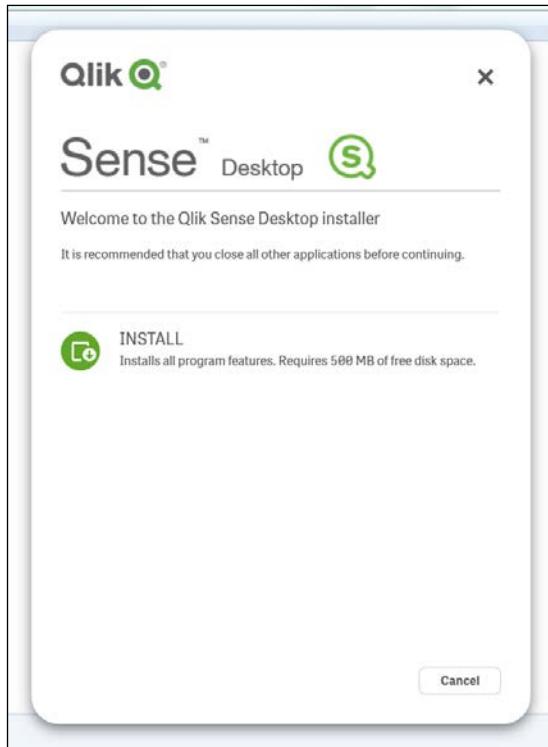
1. Go to the Qlik home page, <http://www.qlik.com>. Click on the **Free Downloads** link in the upper-right corner. The following page will open:



2. Click on the **Get Qlik Sense Desktop** button to download **Qlik Sense Desktop**.
3. When the download finishes, execute the installation program by double-clicking the file you've downloaded:

| Name | Date modified | Type | Size |
|--------------------------|------------------|-------------|------------|
| Qlik_Sense/Desktop_setup | 25/07/2014 15:16 | Application | 122,563 KB |

4. The installation process is very easy; you just need to click on **INSTALL** when **Qlik Sense Desktop installer** starts and accept the license agreement:



[ In case the installer prompts to install .NET Framework 4.0 (if you haven't already done so), then follow the onscreen instructions to install it.]

5. When the installer finishes, click on **Finish** to exit the installation program. You'll find a new **Qlik Sense Desktop** icon on your desktop.

[ Keep the installation program in a safe directory on your hard disk. You can use it to repair your installation if something happens and to uninstall **Qlik Sense Desktop**.]

Exploring Qlik Sense Desktop

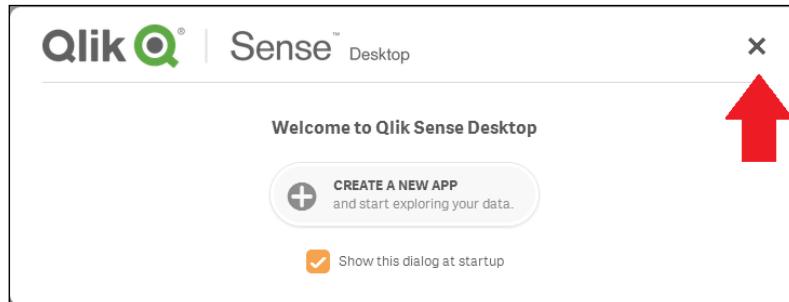
In this section, we will get a first taste of **Qlik Sense Desktop**. We will open it and do a quick exploration. After installing it, **Qlik Sense Desktop** has three example applications **Executive Dashboard**, **Helpdesk Management**, and **Sales Discovery**. We will explore the **Executive Dashboard** application.

Follow these steps to explore **Qlik Sense Desktop**:

1. Open **Qlik Sense Desktop** by double-clicking the **Qlik Sense Desktop** icon on the desktop:



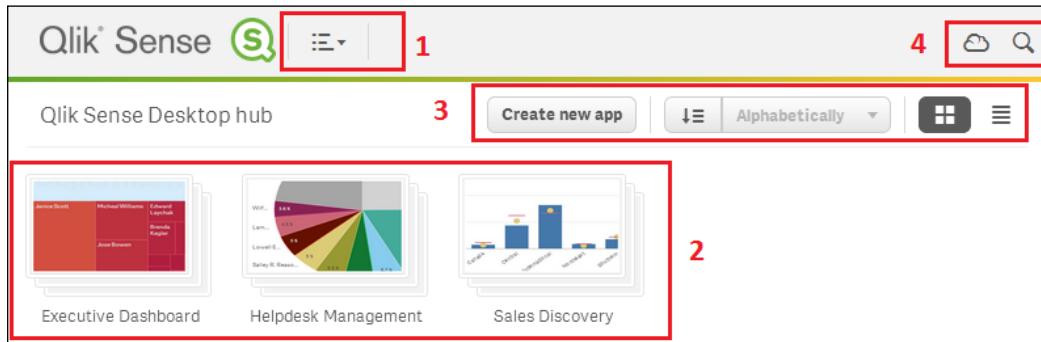
2. When **Qlik Sense Desktop** opens, click on the cross (highlighted in the following screenshot) in the central window to close the startup dialog:



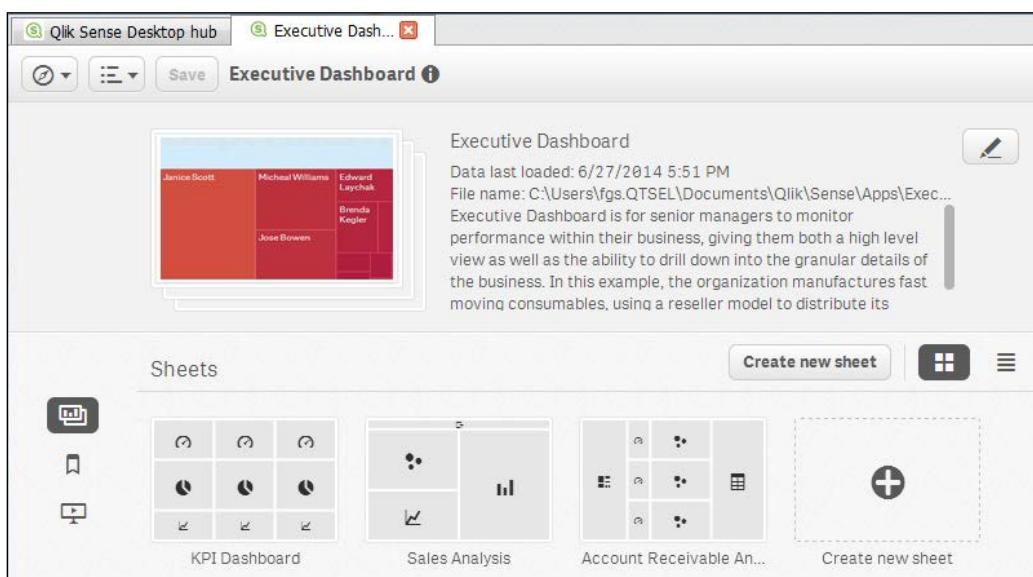
Now, you are in the **Qlik Sense Desktop** hub, the main screen of **Qlik Sense Desktop**. From this screen, the user can open, access, and manage his applications. We've highlighted four different areas in the hub's screen:

- In area **1**, you can find the help section.
- Area **2** is the main section of this screen. In this section, the users have their applications. As you already know, **Qlik Sense Desktop** comes with three demo applications.

- In area **3**, there is a button to create a new application and two buttons to manage the layout.
- In area **4**, you have a button to access QlikCloud and a search button, as shown here:



3. We'll explore an application. Click on the **Executive Dashboard** button to open it.
4. A Qlik Sense application is organized in different sheets, such as a spreadsheet. This application contains three sheets, and you can always create a new sheet by clicking in the **Create new sheet** button, which is visible in the following screenshot:

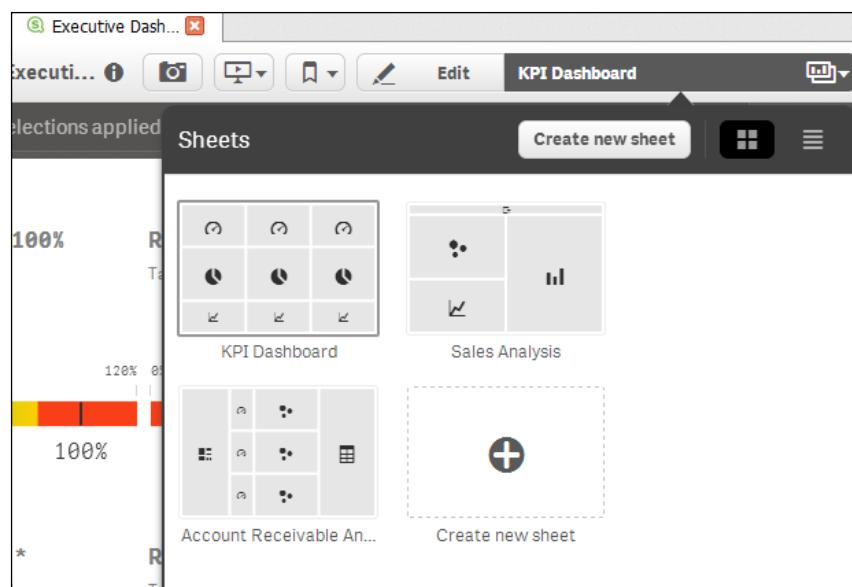


Getting Ready with Predictive Analytics

5. Click on the **KPI Dashboard** icon to open this sheet. This sheet shows three **Key Performance Indicators (KPI)** – **Expenses**, **Revenue vs Last year**, and **Account Receivables**. For each KPI, we see the current level, a distribution, and temporal evolution. At the left-hand side, there are three filters – **Product**, **Segment**, and **Customer**:



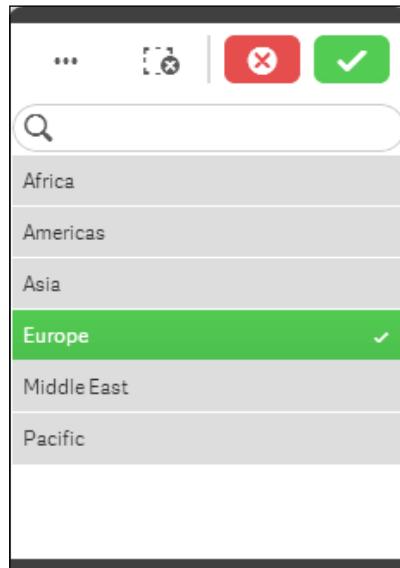
6. Using the button located in the top-right corner, you can toggle between the sheets. Go to the **Sales Analysis** sheet:



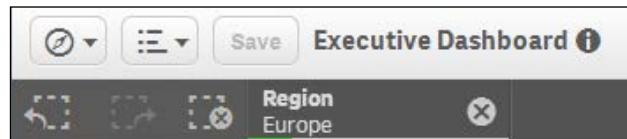
7. The **Sales Analysis** sheet has four filters in the top area: **Segment**, **Region**, **Sales Rep Name**, and **Product Group**:



8. Click on the **Region** filter and select **Europe**; the dashboard will react to show only the data related to **Europe**. Confirm your selection by clicking the highlighted tick icon:



9. Qlik Sense keeps your selection in the top-left side of the screen. You can delete these filters by clicking on the cross (highlighted in the following screenshot):



10. We'll come back to Qlik Sense in *Chapter 4, Creating Your First Qlik Sense Application*; now close the window to exit **Qlik Sense Desktop**.

Further learning

In August 2006, after his famous article, *Thomas H. Davenport* and *Jeanne G. Harris* published a book about the same idea:

Competing on Analytics: The New Science of Winning, by Thomas H. Davenport and Jeanne G. Harris, *Harvard Business School Press*.

A good place to understand all the power of predictive analytics is the book by Eric Siegel. This book contains 147 examples of predictive analytics in its central pages:

Predictive Analytics: the power to predict who will click, buy, lie or die, Eric Siegel, John Wiley & Sons, Inc.

We will not cover R Language in this book. If you want to learn how to program in R, I recommend that you read the following book:

Statistical Analysis with R, John M. Quick, Packt Publishing.

We'll come back to **Qlik Sense Desktop** in *Chapter 4, Creating Your First Qlik Sense Application*. If you want to be more familiar with Qlik Sense, you can start here wwwqlik.com/en-US/sense/gettingstarted.

Summary

In this chapter, we've introduced analytics as a process that starts with raw data and creates new knowledge to help people to take better decisions.

We also defined predictive analytics as a process that learns to create predictions from the data. Finally, we've defined data visualization as a technology that will help us to communicate data-based knowledge more efficiently.

After introducing the key concepts of the book, we've also described R, Rattle, and Qlik Sense, the tools we'll use to build the examples. And finally, we've installed the environment.

In *Chapter 2, Handling Docker Containers*, we'll explain how to load data into Rattle and how we can use Rattle to transform it.

2

Preparing Your Data

The French term *mise en place* is used in professional kitchens to describe the practice of chefs organizing and arranging the ingredients up to a point where it is ready to be used. It may be as simple as washing and picking herbs into individual leaves or chopping vegetables, or as complicated as caramelizing onions or slow cooking meats.

In the same way, before we start cooking the data or building a predictive model, we need to prepare the ingredients—the data. Our preparation covers three different tasks:

- Loading the data into the analytic tool
- Exploring the data to understand it and to find quality problems with it
- Transforming the data to fix the quality problems

We say that the quality of data is high when it's appropriate for a specific use. In this chapter, we'll describe characteristics of data related to its quality.

As we've seen, our *mise en place* has three steps. After loading the data, we need to explore it and transform it. Exploring and transforming is an iterative process, but in this book, we'll divide it in two different steps for clarity.

In this chapter, we'll discuss the following topics:

- Datasets and types of variables
- Data quality
- Loading data into Rattle
- Assigning roles to the variables
- Transforming variables to solve data quality problems and to improve data format of our predictive model

In this chapter, we'll cover how we explore the data to understand it and find quality problems.

Datasets, observations, and variables

A **dataset** is a collection of data that we're going to use to create new predictions. There are different kinds of datasets. When we use a dataset for predictive analytics, we can consider a dataset like a table with columns and rows.

In a real-life problem, our dataset would be related to the problem we want to solve. If we want to predict which customer is most likely to buy a product, our dataset would probably contain customer and historic sales data. When we're learning, we need to find an appropriate dataset for our learning purposes. You can find a lot of example datasets on the Internet; in this chapter, and in the following one, we're going to use the Titanic passenger list as a dataset that has been taken from Kaggle.

 **Kaggle** is the world's largest community of data scientists. On this website, you can even find data science competitions. We're not going to use the term **data science**, in this book, because there are a lot of new terms around analytics and we want to focus just on a few to avoid noise. Currently, we use this term to refer to an engineering area dedicated to collect, clean, and manipulate data to discover new knowledge. On www.kaggle.com, you can find different types of competitions; there are introductory competitions for beginners and competitions with monetary prizes. You can access a competition, download the data and the problem description, and create your own solutions. An example of an introductory Kaggle competition is *Titanic: Machine Learning from Disaster*. You can download this dataset at <https://www.kaggle.com/c/titanic-gettingstarted>. We're going to use this dataset in this chapter and in Chapter 3, *Exploring and Understanding Your Data*.

A **dataset** is a matrix where each row is an observation or member of the dataset. In the Titanic passenger list, each observation contains the data related to a passenger. In a dataset, each column is a particular variable. In the passenger list, the column **Sex** is a variable. You can see a part of the Titanic passenger list in the following screenshot:

| PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|-------------|----------|--------|---|--------|-----|-------|-------|-----------|---------|-------|----------|
| 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22 | 1 | 0 | A/5 21171 | 7.25 | S | |
| 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Thayer) | female | 38 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26 | 0 | 0 | STON/O2. | 7.925 | S | |
| 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35 | 1 | 0 | 113803 | 53.1 | C123 | S |
| 5 | 0 | 3 | Allen, Mr. William Henry | male | 35 | 0 | 0 | 373450 | 8.05 | S | |
| 6 | 0 | 3 | Moran, Mr. James | male | | 0 | 0 | 330877 | 8.4583 | Q | |
| 7 | 0 | 1 | McCarthy, Mr. Timothy J | male | 54 | 0 | 0 | 17463 | 51.8625 | E46 | S |
| 8 | 0 | 3 | Palsson, Master, Gosta Leonard | male | 2 | 3 | 1 | 349909 | 21.075 | S | |
| 9 | 1 | 3 | Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg) | female | 27 | 0 | 2 | 347742 | 11.1333 | S | |
| 10 | 1 | 2 | Nasser, Mrs. Nicholas (Adele Achem) | female | 14 | 1 | 0 | 237736 | 30.0708 | C | |
| 11 | 1 | 3 | Sandstrom, Miss. Marguerite Rut | female | 4 | 1 | 1 | PP 9549 | 16.7 | G6 | S |

Before we start, we need to understand our dataset. When we download a dataset from the Web, it usually has a variable description document.

The following is the variable description for our dataset:

- **Survived:** If the passenger survived, the value of this variable is set to 1, and if the passenger did not survive, it is set to 0.
- **Pclass:** This stands for the class the passenger was travelling by. This variable can have three possible values: 1, 2, and 3 (1 = first class; 2 = second class; 3 = third class).
- **Name:** This variable holds the name of the passenger.
- **Sex:** This variable has two possible values **male** or **female**.
- **Age:** This variable holds the age of the passenger.
- **SibSp:** This holds the number of siblings/spouses aboard.
- **Parch:** This holds the number of parents/children aboard.
- **Ticket:** This holds the ticket number.
- **Fare:** This variable holds the passenger's fare.
- **Cabin:** This variable holds the cabin number.
- **Embarked:** This is the port of embarkation. This variable has three possible values: **C**, **Q**, and **S** (**C** = Cherbourg; **Q** = Queenstown; **S** = Southampton).

For predictive purposes, there are two kinds of variables:

- **Output variables or target variables:** These are the variables we want to predict. In the passenger list, the variable **Survived** is an output variable. This means that we want to predict if a passenger will survive the sinking.
- **Input variables:** These are the variables we'll use to create a prediction. In the passenger list, the variable **sex** is an input variable.

Rattle refers to output variables as target variables. To avoid confusion, we're going to use the term target variable throughout this book. In this dataset, we've ten input variables (**Pclass**, **Name**, **Sex**, **Age**, **SibSp**, **Parch**, **Ticket**, **Fare**, **Cabin**, and **Embarked**) that we want to use to predict if this person is a potential customer or not. So in this example, our target variable is **Survived**.

In *Titanic: Machine Learning from Disaster*, the passenger list is divided into two CSV files: `train.csv` and `test.csv`. The file `train.csv` contains 891 observations or passengers; for each observation, we have a value for the variable **Survived**. It means that we know if the passenger survived or not. The second file, `test.csv`, contains only 418 customers, but in this file, we don't have the variable **Survived**. This means that we don't know if the passenger survived or not. The objective of the competition is to use the training file to create a model that predicts the value of the **Survived** variable in the test file. For this reason, the variable **Survived** is the target variable.

Rattle distinguishes two types of variables—numeric and categorical. A **numeric** variable describes a numerically measured value. In this dataset, **Age**, **SibSp**, **Parch**, and **Fare** are numeric variables.

A **categorical** variable is a variable that can be grouped into different categories. There are two types of categorical variables—ordinal and nominal. In an **ordinal categorical** variable the categories are represented by a number. In our dataset, **Pclass** is an ordinal categorical variable with three different categories or possible values 1, 2, and 3.

In a **nominal categorical** variable, the group is represented by a word label. In this dataset, **Sex** is an example of this type. This variable has only two possible values, and the values are the label, in this case, **male** and **female**.

Loading data

In Rattle, you have to explicitly declare the *role* of each variable. A variable can have five different roles:

- **Input:** The prediction process will use input variables to predict the value of the target variable.
- **Target:** The target variable is the output of our model.
- **Risk:** The risk variable is a measure of the target variable.
- **Ident or Identifier:** An identifier is a variable that identifies a unique occurrence of an object. In our preceding example, the variable **Person** is an identifier that identifies a unique person.
- **Ignore:** A variable marked Ignore will be ignored by the model. We'll come back to this role later—some variables can create noise and decrease the performance of your predictive model.

Rattle can load data from many data sources. Here are some options:

- Use the **Spreadsheet** option to load data from a **Comma Separated Value (CSV)** file.
- **Open Database Connectivity (ODBC)** is a standard to define database connectivity. Using this standard, you can load from most common databases. This will allow you to load data from ERP, CRM, data warehouse systems, and others.
- Use **Attribute-Relation File Format (ARFF)** to load data from Weka files. **Weka** is a machine learning software written in Java.

- You can also load R Datasets; these are tables loaded in memory using R. Currently, Rattle supports R data frames.
- The **RData file** option allows you to load an R Dataset that has been saved in a file, usually with the .Rdata extension.
- With the **Library** option, Rattle can load sample datasets provided by R packages.
- The **Corpus** option allows loading and processing a folder of documents.
- In the following screenshot, you can see a **Script** option, but this option is not implemented. It will be available in a future version.

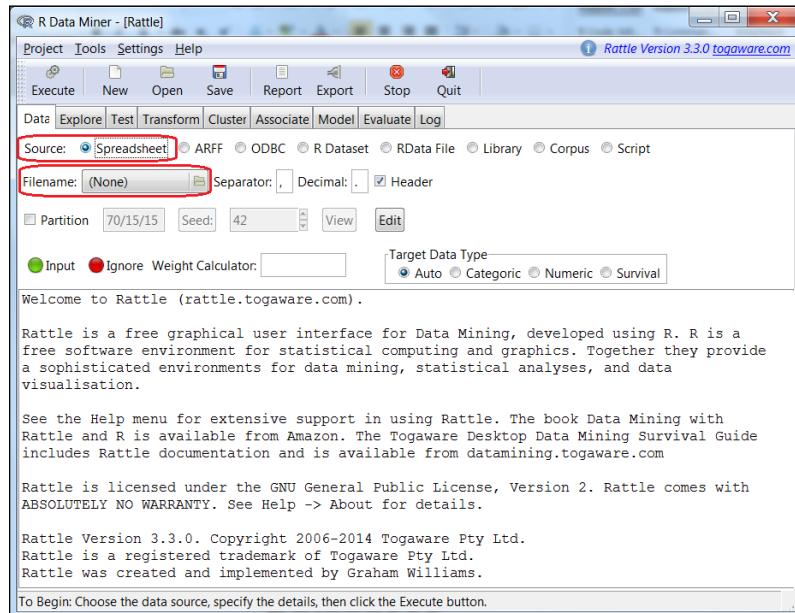
In this book, we're going to load data from the CSV files to explain Rattle's functionalities. CSV is widely used to load data, and we'll find example datasets on the Internet as CSV files.

Loading a CSV File

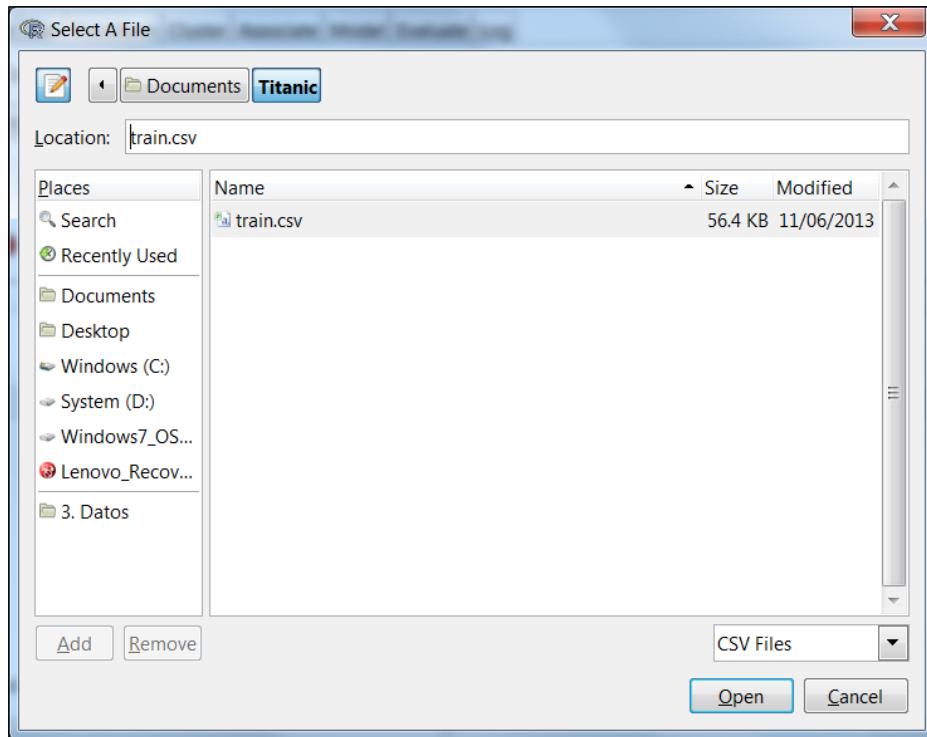
As we've seen before, we'll use a CSV file from Kaggle to learn how to load a dataset into Rattle. Download the file `train.csv` from the competition page at <http://www.kaggle.com/c/titanic-gettingStarted>.

The steps to load the `train.csv` file are as follows:

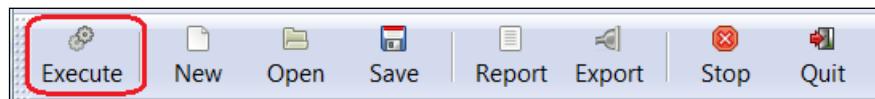
1. Open Rattle and go to the **Data** tab:



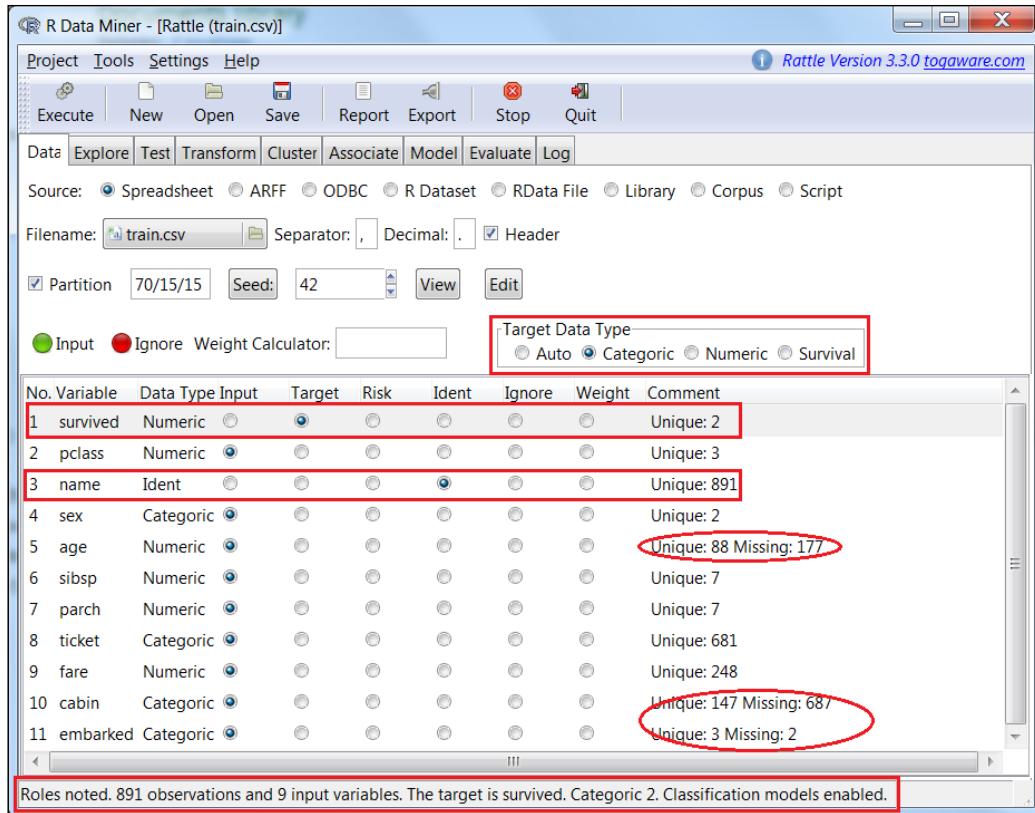
2. Select **Spreadsheet** as the data source and click on the **Filename** folder icon.
3. Select the file `train.csv` and click on **Open**:



4. Finally, click the **Execute** button to load the dataset:



Rattle loads the data from the file, analyzes it, and guesses the structure of the dataset. Now we can start exploring the structure of our data. In the Rattle window, we can see that the loaded dataset has 891 observations with nine input variables and **Survived** as the target variable. We can change the role of each variable with the radio buttons. Note that **Age**, **Cabin**, and **Embarked** have missing values:



We'll focus on these missing values in the next section of this chapter.

The objective of this dataset is to predict whether or not a passenger will survive the sinking of the Titanic. Our target variable is **survived** and has two possible values:

- 0 (not survived)
- 1 (survived)

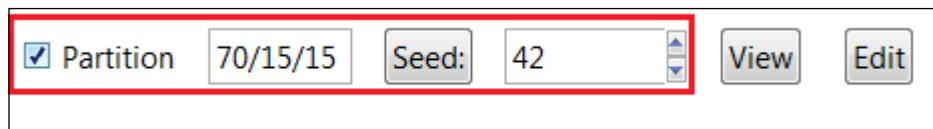
The variable **name** is an identifier that identifies a unique passenger. For this reason, it has 891 observations and 891 different values.

Make changes in the roles of the different variables and click on the **Execute** button to update the data. To save your work, click on the **Save** button and give it an appropriate file name.

The **Save** button will save our work, but it will not modify the data source (the CSV file).



In Rattle's **Data** tab, there are two useful buttons – **View** and **Edit**. With these buttons, you can edit and visualize your data. We also have a **Partition** check box, as you can see in the following screenshot:



Generally, we split the datasets into three subsets of data—a training dataset, a validation dataset, and a testing dataset. We're going to leave this option for now and we'll come back to partitioning in *Chapter 5, Clustering and Other Unsupervised Learning Methods*, and *Chapter 6, Decision Trees and Other Supervised Learning Methods*.

The last option in data loading is **Weight Calculator**. This option allows us to enter a formula to give more importance to some observations.

 You can assign roles to variables automatically by modifying their names in the data source. When you load a variable with a name that starts with **ID**, Rattle marks it automatically as having a role of ident. You can also mark a variable as target, risk, and ignore using **Target**, **Risk**, and **Ignore**.

Transforming data

Data transformation and exploratory data analysis are two iterative steps. The objective is to improve the data quality to create a more accurate model. In order to transform your data, you need to understand it first. So, in real life, you can explore and transform iteratively until you are fine with your data.

For simplicity, we'll cover data transformation in this chapter and data exploration in the next chapter.

Data mining experts usually spend a lot of time preparing data before they start modeling. Preparing data is not as glamorous as creating predictive models but it has a great impact in the model performance. So, be patient and spend time to create a good dataset.

When we execute a transformation in a variable, Rattle doesn't modify the original variable. Rattle creates a new variable with a prefix that indicates the performed transformation and the name of the original variable. An example can be seen in the following screenshot:

| No. Variable | Data Type | Input | Target | Risk | Ident | Ignore | Weight | Comment |
|--------------|-----------|----------------------------------|----------------------------------|-----------------------|----------------------------------|-----------------------|-----------------------|--------------------------|
| 1 survived | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 2 pclass | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 |
| 3 name | Categoric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 891 |
| 4 sex | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 5 age | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 88 Missing: 177 |
| 6 sibsp | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 7 |
| 7 parch | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 7 |
| 8 ticket | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 681 |
| 9 fare | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 248 |
| 10 cabin | Categoric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 147 Missing: 687 |
| 11 embarked | Categoric | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 Missing: 2 |
| 12 RRK_fare | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 248 |

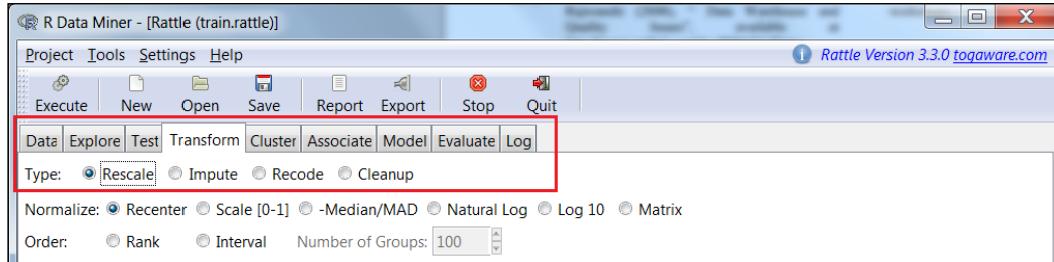
We see the list of variables contained in Rattle after applying a rank transformation to the variable **fare**.

Transforming data with Rattle

Rattle's **Transform** tab offers four different types of transformations:

- **Rescale**
- **Impute**
- **Recode**
- **Cleanup**

These transformation options are shown in the following screenshot:



Rescaling data

In real life, measures use different scales; for example, in the Titanic passenger list, the minimum value for the variable **Age** is 0.42 and the maximum value is 80. For the variable **Fare**, the minimum is 0 and the maximum is 512.3. For this reason, a difference of 10 is a big difference for the variable **Age** and a small difference for **Fare**. Some algorithms and techniques need all variables with the same scale, and we need to adjust values measured on different scales to a common scale. **Rescaling** is the process of adjusting the numeric values of a variable to a different scale.

In Rattle, the **Rescale** option has two sub-options – **Normalize** and **Order**. To **Normalize** variables means to modify the values of the different observations to fit into a scale. The most common normalization is **Scale [0-1]**. If we apply this option to a variable, Rattle will map its values between 0 and 1. In the following table, we've used five values of the variable **Age** to create an example. As we've seen, the minimum value is 0 and the maximum 80. Rescaling the variable from 0 to 1, the minimum value is mapped to 0 and the maximum to 1. The intermediate values are mapped in between 0 and 1, as shown in the following table:

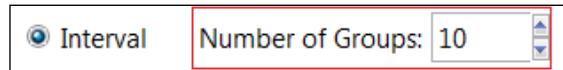
| Age | New Value |
|------|-------------|
| 0.42 | 0 |
| 5 | 0.057552149 |
| 19 | 0.233475748 |
| 54 | 0.673284745 |
| 80 | 1 |

Rattle provides two different **Order** transformations – **Rank** and **Interval**. With the **Rank** option, Rattle will convert variable values to a rank assigning 1 to the minimum value of the variable. We use this option when we're more interested in the relative position of value in the distribution than in the real value.

In our example, the first value of the variable age is 0.42 with the first position in our rank. The second position in the rank is for the value 0.67, and the third and fourth position in the rank has the same value, 0.75. In this case, Rattle doesn't use position three and four, it uses 3.5, as shown in the following table:

| Age | Rank |
|------|------|
| 0.42 | 1 |
| 0.67 | 2 |
| 0.75 | 3.5 |
| 0.75 | 3.5 |
| 0.83 | 5.5 |

Finally, **Interval** groups the values into different groups. Use the input box to choose the number of groups you want to create, as shown in the following screenshot:



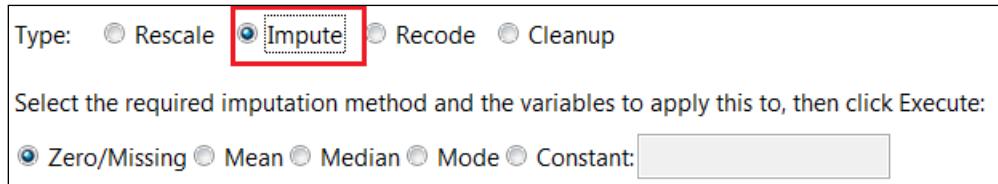
Using **Interval** and the value **10** for **Number of Groups**, Rattle will create 10 groups and will label the groups **0, 1, 2, 3, 4, 5, 6, 7, 8, and 9**. Depending on their value, Rattle will map each observation to a different group. The minimum value 0.4 will be in the group **0**; 80, the maximum value, will be mapped to group **9**.

Using the Impute option to deal with missing values

Sometimes, you will have incomplete observations with missing values for some variables. There are different reasons for missing values. Sometimes, data is manually collected, and not everybody collects it with the same accuracy. Sometimes data is collected from many sensors, and one of them could be temporarily out of order.

Detecting missing values could be difficult. In R, the value **NA**, which means Not Available, indicates a missing value, but there are a lot of data sources that codify a missing value with a concrete value. For numeric values, 0 or 99999 could identify a missing value. You'll need to explore your data carefully to find the real missing values. As we have seen, in the Titanic dataset, variables **Age**, **Cabin**, and **Embarked** have missing values.

With the **Impute** option, we can choose how we want to fill the missing values in our variables, as shown in this screenshot:



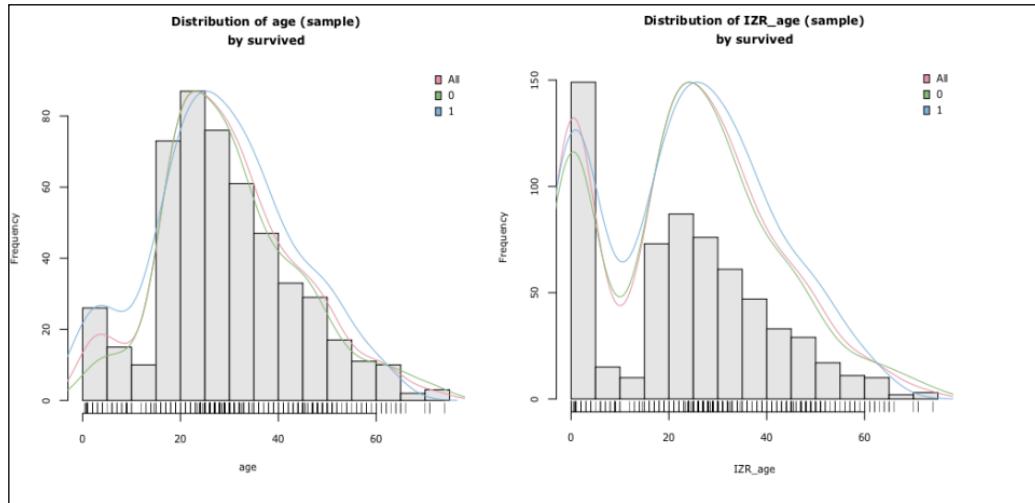
Rattle allows us to apply the following transformations to the missing values:

- **Zero/Missing:** Using this option, Rattle will replace all missing values in a numeric variable with 0 and missing values in a categorical variable with **Missing**.
- **Mean:** This option will use the mean to fill missing values in a numeric variable. Of course, we cannot use this option with a categorical variable.
- **Median:** With this option, we can replace missing values with the median. As with the **Mean** option, this option can only be used with numeric variables.
- **Mode:** Using the **Mode** option, Rattle will replace missing values with the most frequent value. This option can be used with both numeric as well as categorical variables.
- **Constant:** This option will allow us to enter a constant value to replace missing values. Like the **Mode** option, we can use it with both numeric as well as categorical variables.

Rattle has five different options, and if you need to use a different approach, you'll need to code in R, or fill the missing values before loading data into Rattle.

Now you probably must be thinking that the **Median**, **Mean**, and **Mode** options are very similar, and you don't know how to choose among the three different options. To choose one of these options, we need to see how values are distributed into the different observations. We'll see, in the next chapter, that the histogram is the best plot to see the value distribution in a variable, and you'll learn how to plot a histogram using Rattle.

To understand how to fill the missing values, you can analyze the histogram of the original variable, then apply a transformation and analyze the new histogram. With the example of the variable **Age**, we've created a histogram with the original variable (left-hand side). We've applied a **Zero** imputation and created a new histogram. When we apply a **Zero** imputation, we fill those values with all missing values. You will have something like the following graph:

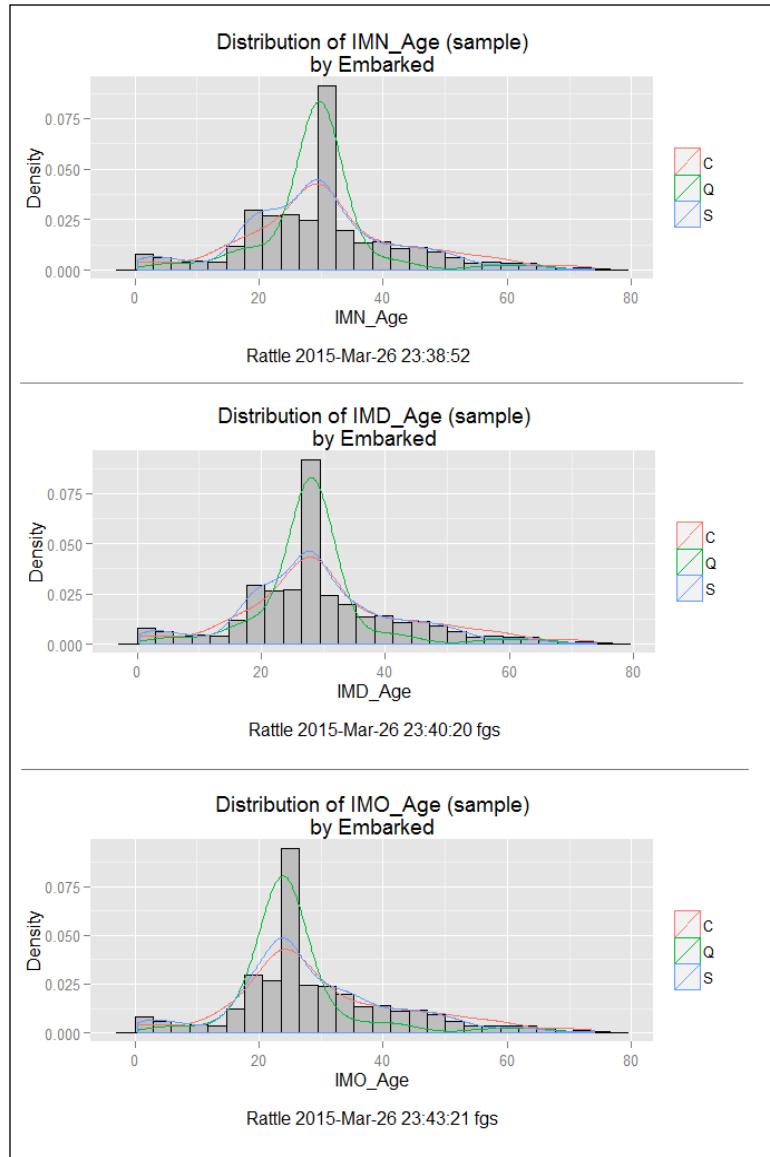


The histogram on the left shows the shape of the original variable **Age**; the mean is 29.7 years. In the Titanic dataset, the variable **Age** has 177 missing values. During the imputation, these 177 values are set to 0. This moves the mean of the distribution to 23.8. In this case, you can see a lot of people with 0 years. As we'll see, the performance of some techniques or algorithms can be affected by this change in the distribution shape.

Now, we can apply **Mean** imputation (fill the missing values with the mean), **Median** imputation (fill the missing values with the median), or **Mode** imputation (fill the missing values with mode).

Preparing Your Data

These three screenshots show the distribution of the **Age** variable histogram after applying a **Mean** imputation (upper), a **Median** imputation (middle), and a **Mode** imputation (lower):



Additionally, you have to consider deleting all observations with missing values in the variable **Age**. This variable has 177 missing values in 891 observations; filling the gaps with a fixed value will probably produce a bad performance.

Recoding variables

We use the **Recode** option to transform the values of variables by distributing the values into different bins or by changing the type of the variable.

Binning

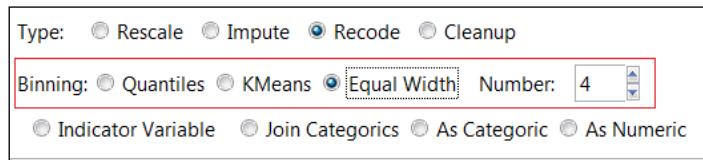
Some models and algorithms only work with categorical variables. **Binning** is an operation that can be useful to transform a numeric variable into a categorical variable. The original values that fall in a bin take the value that represents that bin.

This is how we bin a variable:

- Divide the range of values into a series of small intervals or bins
- Distribute each value into its interval or bin

To define the groups or bins, we have three options:

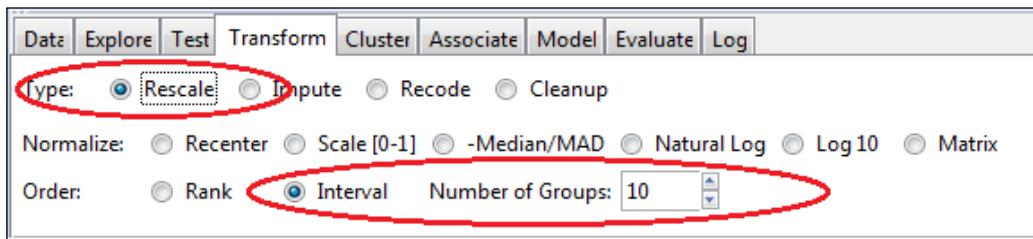
- Use **Quantiles** to create groups with the same number of observations
- Use **KMeans** to create groups of members based on the distance of the values
- Choose **Equal Width** to distribute the values of a variable into groups of the same width, as shown in this screenshot:



Like in this screenshot, try to apply an **Equal With** transformation (under **Binning**) to the variable **age**. Rattle will create **10** groups and will place each observation in a group.

Preparing Your Data

To distribute values into different groups, you can also select **Type** as **Rescale**, and then **Order** as **Interval** and set **Number of Groups** as **10**, as shown in the following screenshot:



What is the difference between the two options? The variable **Age** is a numeric variable; when you use **Recode**, the result is a numeric variable. If you use **Binning**, the new variable is a categorical variable, as shown in this screenshot:

| No. | Variable | Data Type and Number Missing |
|-----|-------------|--|
| 1 | PassengerId | Numeric [1 to 891; unique=891; mean=446; median=446]. |
| 2 | Survived | Numeric [0 to 1; unique=2; mean=0; median=0]. |
| 3 | Pclass | Numeric [1 to 3; unique=3; mean=2; median=3]. |
| 4 | Name | Categorical [891 levels]. |
| 5 | Sex | Categorical [2 levels]. |
| 6 | Age | Numeric [0.42 to 80.00; unique=88; mean=29.70; median=28.00; miss=177; ignored]. |
| 7 | SibSp | Numeric [0 to 8; unique=7; mean=0; median=0]. |
| 8 | Parch | Numeric [0 to 6; unique=7; mean=0; median=0]. |
| 9 | Ticket | Categorical [681 levels]. |
| 10 | Fare | Numeric [0.00 to 512.33; unique=248; mean=32.20; median=14.45]. |
| 11 | Cabin | Categorical [147 levels; miss=687]. |
| 12 | Embarked | Categorical [3 levels; miss=2]. |
| 13 | RIN_Age_10 | Numeric [0.00 to 9.00; unique=10; mean=3.29; median=3.00; miss=177]. |
| 14 | BE10_Age | Categorical [10 levels; miss=177]. |

In the previous screenshot, we created **RIN_Age_10** using rescale and **BE10_Age** using binning.

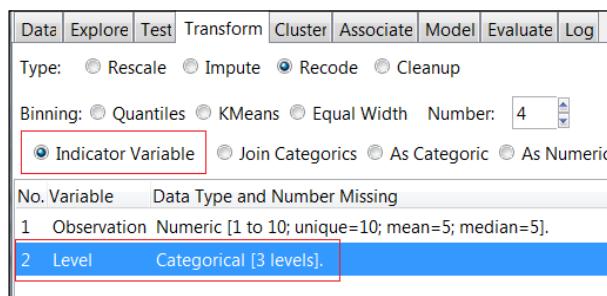
Binning could also be used to reduce small observation errors. By replacing the original value by a representative value of the group, you will reduce the effect of small observation errors.

Indicator variables

As opposed to the previous section some algorithms (like many clustering models) only work with numeric variables. A simple technique to convert categorical variables into numeric variables is **indicator variables**. Take a categorical variable like **Level** with three categories – **Beginner**, **Medium**, and **Advanced** – and create three new variables called **Beginner indicator**, **Medium indicator**, and **Advanced indicator**. If the value of **Level** is **Beginner**, set variable **Beginner indicator** to 1 and the rest to 0, as shown in this diagram:

| Observation | Level | Beginner indicator | Medium indicator | Advanced Indicator |
|-------------|----------|--------------------|------------------|--------------------|
| 1 | Beginner | 1 | 0 | 0 |
| 2 | Medium | 0 | 1 | 0 |
| 3 | Medium | 0 | 1 | 0 |
| 4 | Beginner | 1 | 0 | 0 |
| 5 | Advanced | 0 | 0 | 1 |
| 6 | Advanced | 0 | 0 | 1 |
| 7 | Medium | 0 | 1 | 0 |
| 8 | Beginner | 1 | 0 | 0 |
| 9 | Beginner | 1 | 0 | 0 |
| 10 | Advanced | 0 | 0 | 1 |

In Rattle, the **Transform** tab has an **Indicator Variable** option. In order to apply this transformation, select the variable (in this case, **Level**), select **Indicator Variable**, and click on **Execute**, as shown in the following screenshot. Rattle will create a variable for each category belonging to the categorical variable:



Join Categories

With the **Join Categories** option, Rattle will convert two categorical variables into a single one. In the following table, we've used Rattle to convert **Level** and **Sex** to a single variable:

| Observation | Level | Sex | TJN_Level_Sex |
|-------------|----------|--------|-----------------|
| 1 | Begginer | Male | Begginer_Male |
| 2 | Medium | Male | Medium_Male |
| 3 | Medium | Male | Medium_Male |
| 4 | Begginer | Male | Begginer_Male |
| 5 | Advanced | Female | Advanced_Female |
| 6 | Advanced | Female | Advanced_Female |
| 7 | Medium | Female | Medium_Female |
| 8 | Begginer | Female | Begginer_Female |
| 9 | Begginer | Female | Begginer_Female |
| 10 | Advanced | Male | Advanced_Male |

As Category

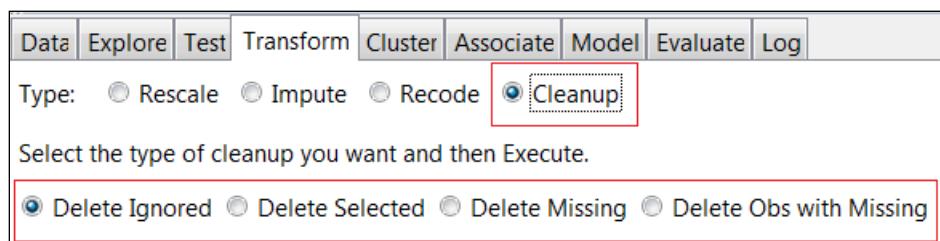
Using the **As Category** option, you can convert numeric variables into categorical.

As Numeric

Using the **As Numeric** option, Rattle will convert categorical variable into numeric.

Cleaning up

The **Cleanup** option in the **Transform** tab allows you to delete columns and observations from your dataset, as shown in this screenshot:



The following are the different available cleanup options:

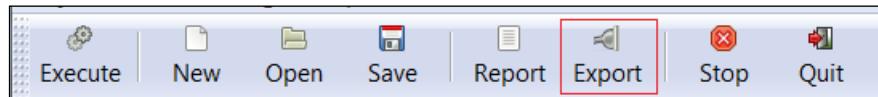
- **Delete Ignored:** This will delete variables marked as ignore
- **Delete Selected:** This will delete the selected variables
- **Delete Missing:** This will delete all variables with any missing values
- **Delete Obs with Missing:** This will delete observations with missing values in the selected variable

You've learned how to transform variables. When Rattle transforms a variable, it doesn't modify the original one. It creates a new variable with the corresponding modification. If you apply a transformation to the variable **Age**, you will have the variable **Age** and the new one. Your algorithms only need one variable, the original or the transformed, so you have to change the role of the one not to be used to **Ignore**. By default, after the transformation, Rattle sets the original variable to **Ignore**. In the following screenshot, you can see the original variable **Age** set to **Ignore** and the new transformed variable set to **Input**:

| No. | Variable | Data Type | Input | Target | Risk | Ident | Ignore | Weight | Comment |
|-----|-------------|-----------|----------------------------------|----------------------------------|-----------------------|----------------------------------|----------------------------------|-----------------------|--------------------------|
| 1 | PassengerId | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 891 |
| 2 | Survived | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 3 | Pclass | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 |
| 4 | Name | Categoric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 891 |
| 5 | Sex | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 6 | Age | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | Unique: 88 Missing: 177 |
| 7 | SibSp | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 7 |
| 8 | Parch | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 7 |
| 9 | Ticket | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 681 |
| 10 | Fare | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 248 |
| 11 | Cabin | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 147 Missing: 687 |
| 12 | Embarked | Categoric | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 Missing: 2 |
| 13 | R01_Age | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 88 Missing: 177 |

Exporting data

After data transformation, you have to export your new dataset, as shown in this screenshot:



In the main menu, press the **Export** icon; this will open a dialog window. Choose a directory and a filename and press **Save**. This book is the reference for Rattle.

Further learning

An extended explanation of data transformation in Rattle can be found in *Data Mining with Rattle and R*, by Graham Williams, Springer. Graham Williams is a well-known data scientist; he created and developed Rattle.

Summary

We started this chapter comparing the term *mise en place* used by professional chefs to the task of loading and preparing the data before we start creating predictive models.

During this chapter, we introduced the basic vocabulary to describe datasets, observations, and variables. We also saw how to load a CVS file into Rattle and described the most usual data transformations.

This chapter, as well as *Chapter 3, Exploring and Understanding Your Data*, covered the *mise en place* for our data. After going through these chapters, we'll be able to prepare our data to analyze it and discover hidden insights.

In the next chapter, we'll explore the dataset to have a better understanding and to find data quality problems. The next two chapters are tied because exploring the dataset and transforming it are complementary tasks.

When you are cooking, the quality of the ingredients has a great influence on the quality of your dish. Working with data is very similar; it's very hard to achieve good results if you use low quality data. For this reason, these two chapters are really important.

3

Exploring and Understanding Your Data

In the previous chapter, we've explained how to load data and how to transform it using Rattle. In this chapter, we're going to learn how to use Rattle to:

- Summarize dataset characteristics
- Identify missing values in the data
- Create charts to represent data point distributions

We have two main objectives when we explore data. We would like to understand the problem we want to solve and we want to understand the structure of the dataset in order to choose the most appropriate predictive technique.

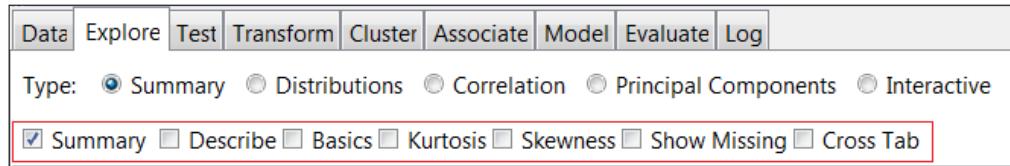
If you are a business analyst, Qlik Sense is a great tool to explore and understand your data. With Qlik Sense, you can find relationships between customers, products, and sales people in a very intuitive way. In the next chapter, we're going to learn how to use Qlik Sense to load and explore data.

As some predictive techniques are based on statistics, if you are preparing a dataset to apply a predictive technique, you would probably prefer a more formal or mathematical approach. We call this approach **Exploratory Data Analysis (EDA)**. This is a large subject developed in 1997 by John W. Tukey

y. In this chapter, we'll see some EDA methods and concepts and we'll use Rattle to explore data. Mainly, we're going to use the **Test** tab to explore the data, but don't forget to start the exploration by looking at the data in tabular form.

Text summaries

The **Summary** option in the **Explore** tab provides us with some descriptive statistics such as **Summary**, **Describe**, **Basics**, **Kurtosis**, and **Skewness** reports. Descriptive statistics covers methods to summarize data. The **Summary** option also provides a very useful **Show Missing** report:



Summary reports

Rattle provides us with these summary reports:

- **Summary**
- **Describe**
- **Basics**
- **Kurtosis**
- **Skewness**

These reports summarize variable distributions and help to give an initial understanding of our data. In order to understand these reports, you only need a basic understanding of descriptive statistics.

Measures of central tendency – mean, median, and mode

For a variable, a measure of central tendency describes the center of the distribution as follows:

- **Mean:** The mean is the average and is the best central tendency measure if the distribution is normal.
- **Median:** Half of the observations have a lower value than this variable and the other half have a higher value. This is a good measure if there are extreme values.
- **Mode:** The mode is the most repeated value. In a histogram, the peak is the mode.

Measures of dispersion – range, quartiles, variance, and standard deviation

Measures of dispersion help us to summarize how spread out these scores are. To describe dispersion, Rattle provides us with some statistics, including the range, quartiles, absolute deviation, variance, and standard deviation. Dispersion gives us an idea of how the values for individual observations are spread out around the measure of central tendency.

Range

Range is the difference between the maximum and minimum values. The range can be useful to detect outliers if you are measuring variables with a critical low or high.

Quartiles

In a ranked variable, **quartiles** explain the spread of a dataset by breaking the dataset into quarters, which are described as follows:

- 25 percent of the observations have equal or lower value than the first quartile, or Q1
- 25 percent of the observations have equal or higher value than the third quartile, or Q3
- Half of the observations have a lower value than the median or the second quartile (Q2), and the other half have a higher value

Finally, we call the difference between Q3 and Q1 the **interquartile range**.

In the following screenshot, you can see the output of the **Summary** report for a numeric variable, **Age**, and for a categorical variable, **Embarked**:

| Age | Embarked |
|---------------|----------|
| Min. : 0.42 | C : 117 |
| 1st Qu.:21.00 | Q : 56 |
| Median :28.00 | S : 449 |
| Mean :29.82 | NA's: 1 |
| 3rd Qu.:39.00 | |
| Max. :74.00 | |
| NA's :123 | |

For a numerical variable like **age**, this report tells us the minimum and maximum values (range), the quartiles, the mean, the median, and the number of missing values.

For a categorical variable like **embarked**, this report gives us the number of occurrences of each category and the number of missing values.



You can find a more accurate description of quartile at this wiki:
<http://en.wikipedia.org/wiki/Quartile>.



Variance

In a group of data, high **variance** means that the data points are widely spread; low variance means that the values are concentrated around the mean. Following is the formula for variance:

$$\text{Variance} = \sum (X - \text{Mean})^2 / (N - 1)$$

Here, **X** is the value of each observation and **N** is the total number of observations.

Standard deviation

The **standard deviation** is closely related to variance and it also measures the spread of values within a dataset. The formula is very simple – it is the square root of the variance. A low standard deviation indicates that the values are concentrated close to the mean; a high standard deviation indicates that the values are more spread out. The standard deviation is the most commonly used measure of spread because it is expressed in the same units as mean, whereas variance is expressed in square units.

Measures of the shape of the distribution – skewness and kurtosis

With **kurtosis** and **skewness**, we can have an intuition of the shape of the distribution of a variable. Kurtosis describes if a particular distribution is peaked and skewness measures the asymmetry of that distribution. A flatter distribution has a lower kurtosis.

In a negative skewed distribution, the left tail is longer than the right tail or the center of the data is moved to the right. For a positive skewed distribution, the longest tail is the right one or the center of the distribution is moved to the left.



As with quartiles, I suggest you take a look at the wiki for a more academic description:

- <http://en.wikipedia.org/wiki/Kurtosis>
- <http://en.wikipedia.org/wiki/Skewness>

In the following screenshot, you can see the **Basics** report for the variable **age**:

| Data | Explore | Test | Transform | Cluster | Associate | Model | Evaluate | Log | | | | | | | |
|-------------|--|------|-----------|---------|-----------|-------|----------|-----|--|--|--|--|--|--|--|
| Type: | <input checked="" type="radio"/> Summary <input type="radio"/> Distributions <input type="radio"/> Correlation <input type="radio"/> Principal Components <input type="radio"/> Interactive | | | | | | | | | | | | | | |
| | <input type="checkbox"/> Summary <input type="checkbox"/> Describe <input checked="" type="checkbox"/> Basics <input type="checkbox"/> Kurtosis <input type="checkbox"/> Skewness <input type="checkbox"/> Show Missing <input type="checkbox"/> Cross Tab | | | | | | | | | | | | | | |
| \$age | | | | | | | | | | | | | | | |
| | X....X..2 | | | | | | | | | | | | | | |
| nobs | 623.000000 | | | | | | | | | | | | | | |
| NAs | 123.000000 | | | | | | | | | | | | | | |
| Minimum | 0.420000 | | | | | | | | | | | | | | |
| Maximum | 74.000000 | | | | | | | | | | | | | | |
| 1. Quartile | 21.000000 | | | | | | | | | | | | | | |
| 3. Quartile | 39.000000 | | | | | | | | | | | | | | |
| Mean | 29.817180 | | | | | | | | | | | | | | |
| Median | 28.000000 | | | | | | | | | | | | | | |
| Sum | 14908.590000 | | | | | | | | | | | | | | |
| SE Mean | 0.634329 | | | | | | | | | | | | | | |
| LCL Mean | 28.570895 | | | | | | | | | | | | | | |
| UCL Mean | 31.063465 | | | | | | | | | | | | | | |
| Variance | 201.186596 | | | | | | | | | | | | | | |
| Stdev | 14.184026 | | | | | | | | | | | | | | |
| Skewness | 0.385387 | | | | | | | | | | | | | | |
| Kurtosis | 0.119261 | | | | | | | | | | | | | | |

This report gives us the measures we've seen in this chapter, such as maximum, minimum, mean, median, standard deviation, variance, kurtosis, and skewness. There are still some strange measures we don't understand.

What are **SE Mean**, **LCL Mean**, and **UCL Mean**? They mean the following:

- **SE Mean:** This stands for the **Standard Error** of the mean
- **LCL Mean:** This stands for the **Lower Confidence Level** mean
- **UCL Mean:** This stands for the **Upper Confidence Level** mean

These measures explain the error margin of the mean and the confidence interval of the mean.

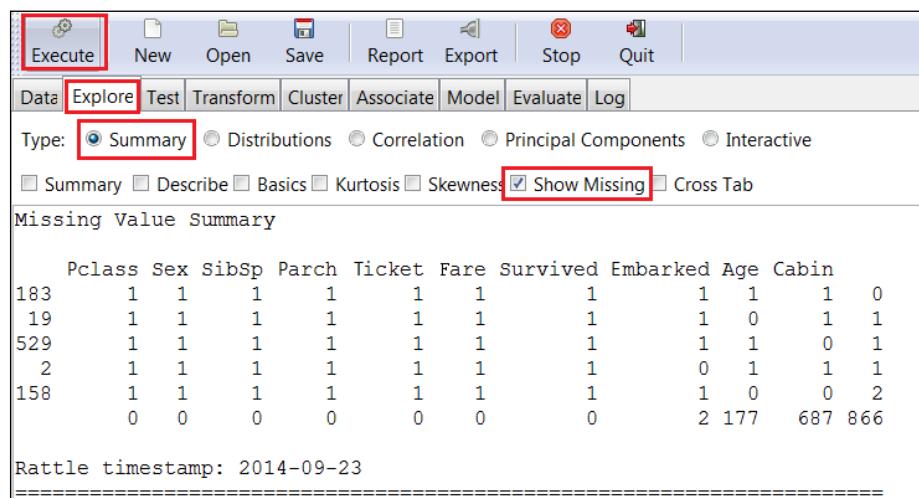
Showing missing values

Missing values is an important problem. For this reason, this report deserves special attention. We saw in the previous chapter how important missing values are and how we can fill these values; now we're going to see how to understand the impact of missing values in our dataset.

We're going to use the Titanic passenger list dataset. Perform the following steps:

1. Load the Titanic dataset into Rattle.
2. Go to the **Explore** tab.
3. Choose the **Summary** option.
4. Select the **Show Missing** values report.
5. Press the **Execute** button.

The preceding steps have been illustrated in the following screenshot:



We've obtained a **Missing Value Summary** report as shown in the following screenshot. We're going to look at the report in more detail:

| Missing Value Summary | | | | | | | | | | | | | |
|-----------------------|--------|-----|-------|-------|---------------------|------|----------|----------|-----|-------|-----|---|--|
| | Pclass | Sex | SibSp | Parch | Ticket | Fare | Survived | Embarked | Age | Cabin | | | |
| 183 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 19 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | |
| 529 | 1 | 1 | 1 | 1 | CENTRAL AREA | | | | | | | | |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | |
| 158 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 2 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 177 | 687 | 866 | | |

Rattle timestamp: 2014-09-23
=====

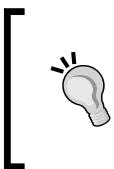
As you can see in the highlighted part of the preceding screenshot, this report has a central area with a column for each variable. In this central area, each row corresponds to a pattern of observations, - **1** means that the value is present and **0** means the values is missing.

Now take a closer look at the first row. Below each variable, there is a **1**; this means that in this kind of observation, all variables have a value. What about the second row? The second row represents the observation that all variables have values except the variable **Age**.

On the left-hand side of the central area, there is a column. Each row in this column has a number representing the number of repetitions of the pattern. Looking at the first and second rows, there are **183** observations with no missing values and **19** observations with a missing value for the variable **Age**.

On the right-hand side of the central area, there is another column. Each row in this column has a number that tells how many missing variables there are in that pattern. Looking again at the first and second rows, they have **0** and **1** missing variables respectively.

Finally, under the central area, there is a row. In this row, each value is the number of total missing values that the variable has. In this example, **Embarked** has **2** missing values and **Age** has **177** missing values.



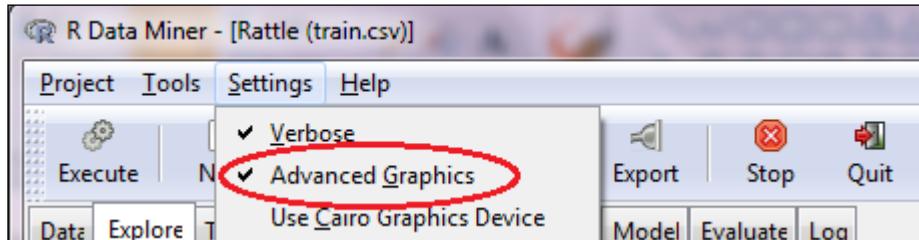
Remember that some datasets have dummy values for missing values. If you have a variable called price and you have observations with the value 0 for this variable, they are probably missing values. The Missing Value Summary report will not show these hidden missing values.

Visualizing distributions

In the last section, we discussed distributions and we saw some measures that describe them. In this section, we're going to see how to visualize distributions. Visualizations are more intuitive than numeric measures and they will help us to understand our data.

Rattle offers two different set of charts depending on the nature of the variables. For numeric variables, we can use **Box Plot**, **Histogram**, **Cumulative**, and **Benford**. And for categorical variables, Rattle provides us with **Bar Plot**, **Dot Plot**, and **Mosaic** charts. We're going to explore the most common visual representations.

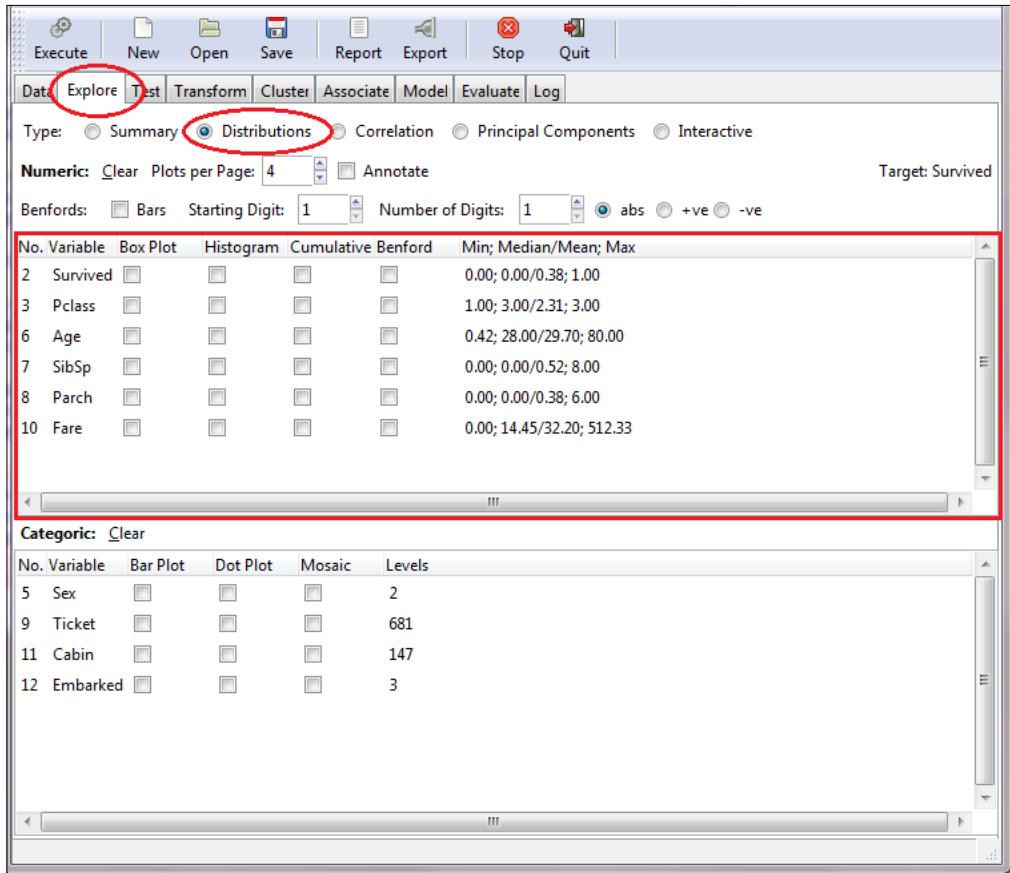
Before using Rattle to plot charts, make sure that the **Advanced Graphics** option is unchecked. With this option checked, some charts like histograms will not be plotted. This is shown in the following screenshot:



Numeric variables

We're going to use the variable **Age** of the Titanic passenger list to show the different types of charts with numeric variables. Load the data set, set the variable **Survived** as target, and go to the **Explore** tab and select the **Distributions** type. The central area of the screen is divided into two panels - the upper panel is reserved for the numeric variables and the lower one for categorical variables.

In the numeric variables area, you can see six variables (**Survived**, **Pclass**, **Age**, **SibSp**, **Parch**, and **Fare**) and four different plots (**Box Plot**, **Histogram**, **Cumulative**, and **Benford**). To plot a chart, you have to select the appropriate checkbox and click on the **Execute** button, as shown in the following screenshot:

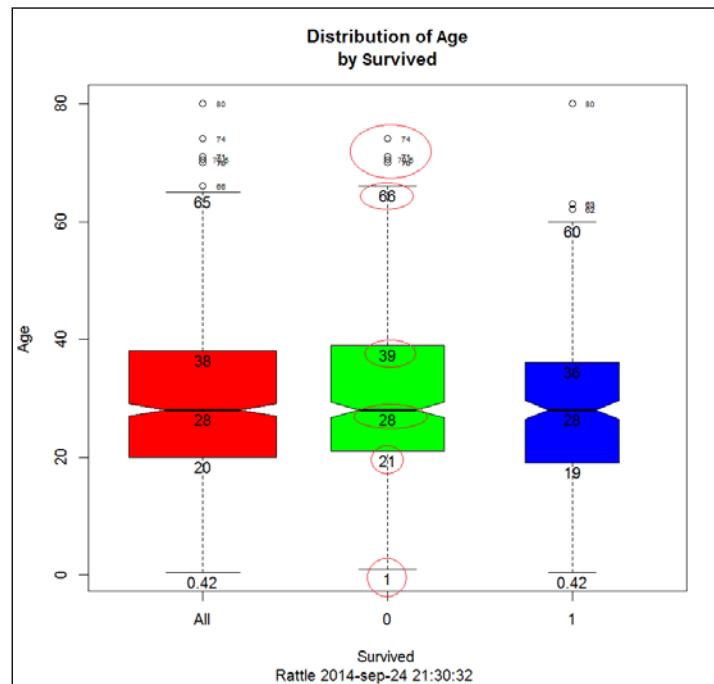


Box plots

The first chart we're going to discuss will be the **Box Plot**. We're going to plot a chart of the variable **Age** of the Titanic's passenger list. Select the **Annotate** checkbox in order to have the values of the data points labeled as shown in the following screenshot:

| Data Explore Test Transform Cluster Associate Model Evaluate Log | | | | | | | | | |
|---|-------------------------------------|--|--|--|-----------------------------------|------------------------------|---|------------------------------|--|
| Type: | <input type="radio"/> Summary | <input checked="" type="radio"/> Distributions | <input type="radio"/> Correlation | <input type="radio"/> Principal Components | <input type="radio"/> Interactive | | | | |
| Numeric: | <input type="checkbox"/> Clear | <input type="checkbox"/> Plots per Page: 4 | <input checked="" type="checkbox"/> Annotate | | | | | | |
| Benfords: | <input type="checkbox"/> | <input type="checkbox"/> Bars | Starting Digit: 1 | <input type="checkbox"/> | Number of Digits: 1 | <input type="checkbox"/> abs | <input checked="" type="checkbox"/> +ve | <input type="checkbox"/> -ve | |
| No. Variable Box Plot Histogram Cumulative Benford Min; Median/Mean; Max | | | | | | | | | |
| 1 Survived | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 0.00; 0.00/0.38; 1.00 | | | | |
| 2 Pclass | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 1.00; 3.00/2.31; 3.00 | | | | |
| 5 Age | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 0.42; 28.00/29.70; 80.00 | | | | |
| 6 SibSp | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 0.00; 0.00/0.52; 8.00 | | | | |
| 7 Parch | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 0.00; 0.00/0.38; 6.00 | | | | |
| 9 Fare | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 0.00; 14.45/32.20; 512.33 | | | | |

These plots summarize the distribution of a variable in a dataset. In the following screenshot, we can see the representation of the variable **Age**:

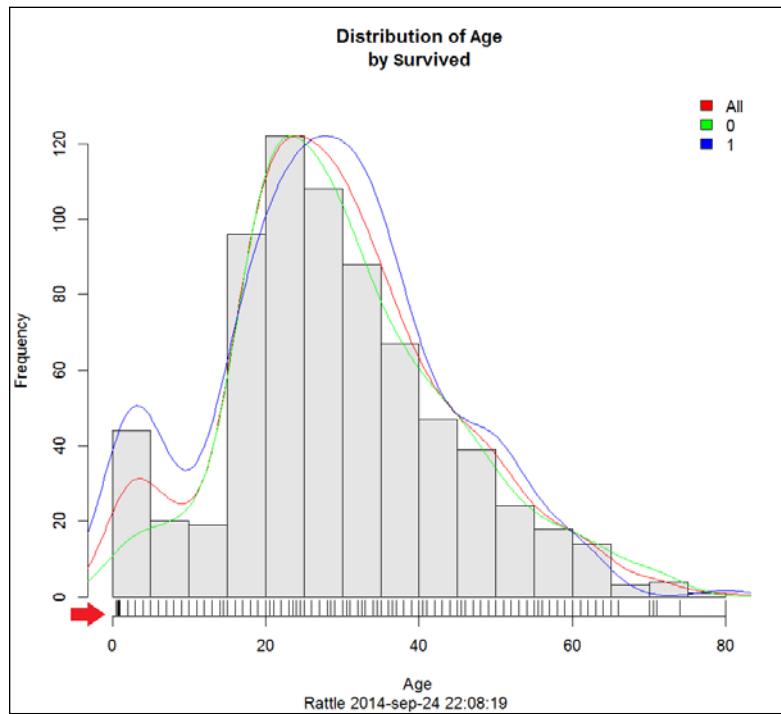


If you have identified the target variable when you loaded the dataset, Rattle will create a plot for all observations and a plot for every possible value of the target variable. In this example, the target variable **Survived** has two possible values, - 0 and 1.

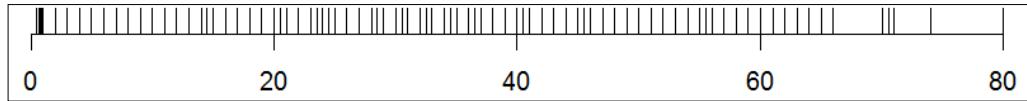
We have highlighted some points of the central plot – the green part. In the center of the plot, the horizontal line labeled with a **28** is the median. The point labeled with a **21** is the first quartile, or Q1, and **39** represents the third quartile, or Q3. In this plot, the interquartile range is $39 - 21 = 18$ ($Q3 - Q1$). The lower and upper points labeled with **1** and **66** are 1.5 times the interquartile range from the median. Points above the point labeled with a **66** are outliers.

Histograms

Histograms give us a quick view of the spread of a distribution. Rattle's histogram combines three charts in one, namely the R histogram (the bars), the density plot (the line), and the rug plot. The rug plot is marked with a red arrow in this screenshot:



This histogram shows us the distribution in terms of age. The vertical bars are the original histogram. Every bar represents an **Age** range and the height of the bar represents the **Frequency** or the number of observations that fall in that age range. The density plot is a more accurate representation of the estimated values. Finally, in the rug plot, every line shows the exact value of an observation, as shown in the following screenshot:

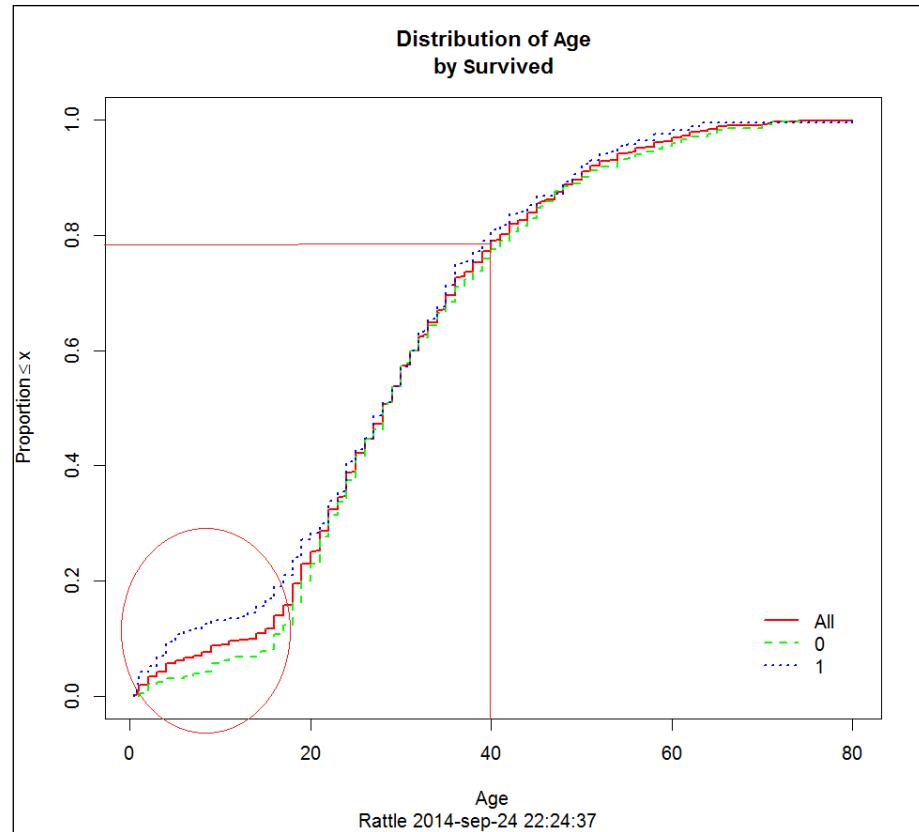


In the preceding histogram, we can see that most people on the Titanic were between **20** and **40** years of age.

Cumulative plots

The cumulative plot represents the percentage of the population that has a value than or equal to the value shown in the *x* axis. I've plotted the cumulative plot for the variable **Age**. If you look at the following screenshot, you can see that nearly 80 percent of the passengers were less than or equal to 40 years old.

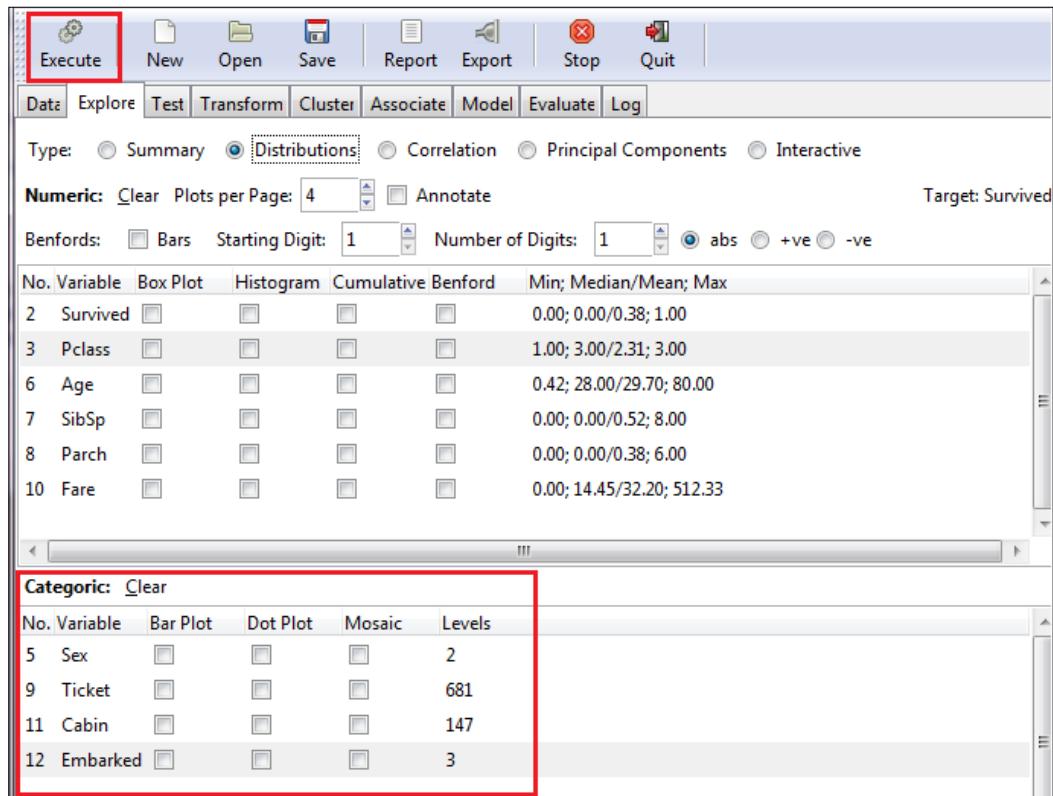
We've circled the younger passengers. In this plot, like in the histogram we plotted before, we see that young people had a greater probability of survival.



Categorical variables

We're now going to explore categorical variables. As with numeric variables, you have to load the Titanic dataset and set **Survived** as the target variable. Then go to the **Explore** tab and select the **Distributions** type.

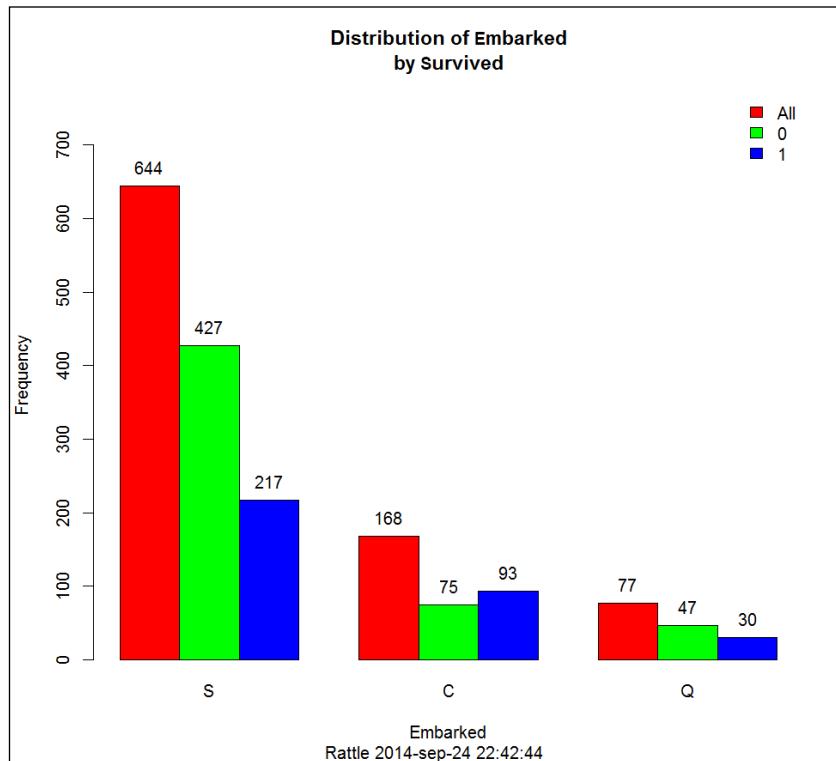
To plot a new graph, you have to check the plot and the variable in the **Categoric** variable panel and click on **Execute**. This is illustrated in the following screenshot:



We'll use the variable **embarked** from the Titanic passenger list to plot a bar plot, a dot plot, and a mosaic plot.

Bar plots

The bar chart is probably the simplest and easiest to understand – it uses vertical or horizontal bars to compare among categories. In the following screenshot, we can see a bar chart of the variable **embarked**:



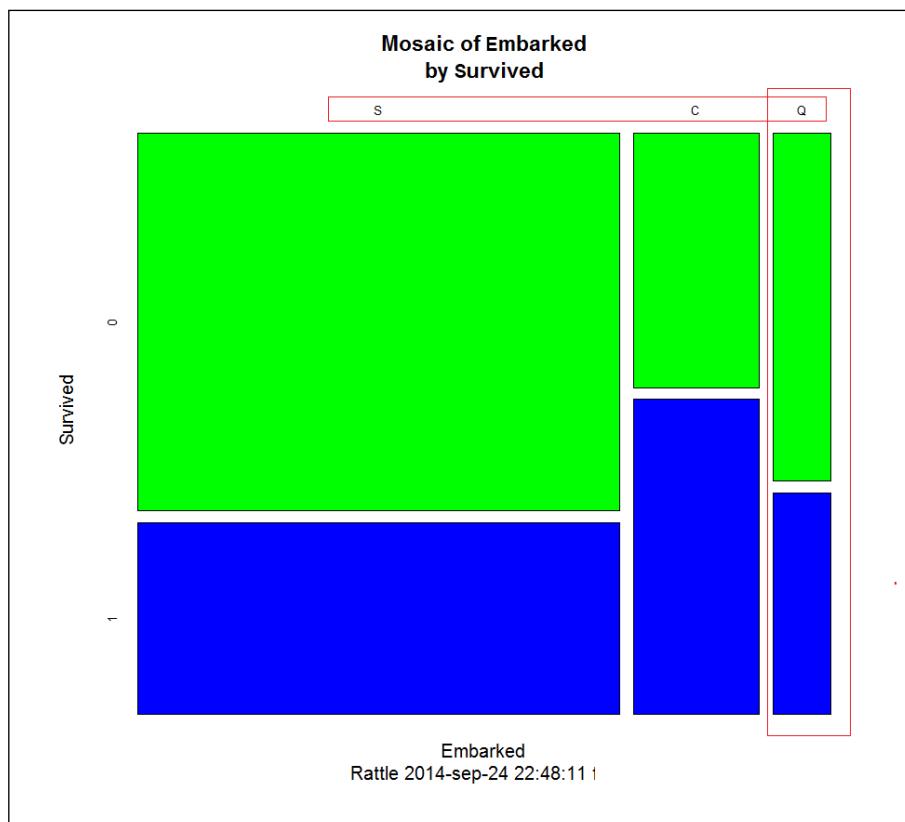
In the previous chapter, we introduced this dataset and we explained that the variable **embarked** has three possible values – **C** for Cherbourg, **Q** for Queenstown, and **S** for Southampton. If you look at this chart, it is quick and easy to see that most of the people (**644**) embarked in Southampton. Looking at the blue and green bars, we can see that around a third of the passengers that embarked at Southampton survived and around half of the passengers who embarked at Cherbourg survived.



Try to create a bar chart of the variable **sex** and you'll discover that 74.2 percent of females survived and only 18.9 percent of the males survived the Titanic disaster.

Mosaic plots

The mosaic plot shows the distribution of the values for a variable. Look at the following screenshot. At the top of the plot, there are three letters – **S**, **C**, and **Q** – representing the three harbors. Below each letter, there is a bar divided into two sub-bars (blue and green). We have highlighted the bar below **Q**, as shown in the following screenshot:



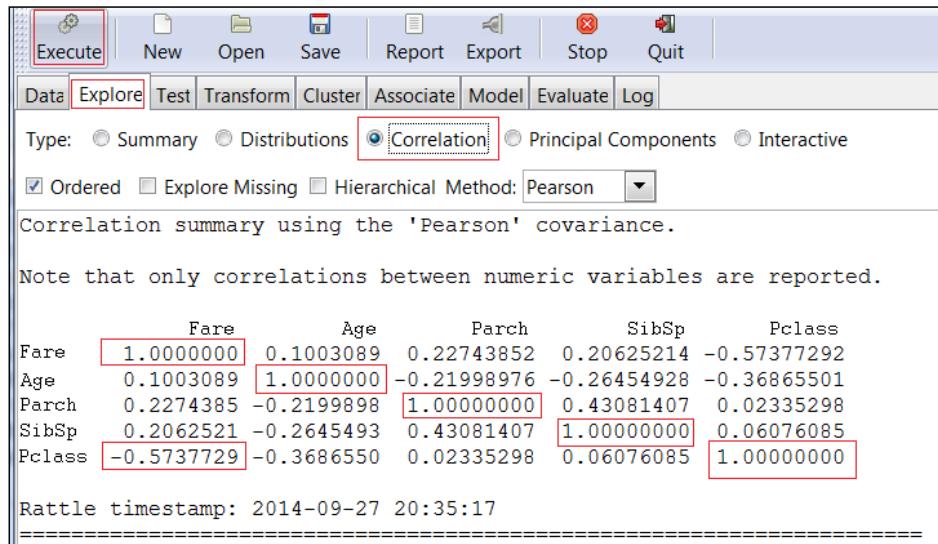
The width of this bar represents the number of occurrences. In our plot, the wider bar is the bar below S. This is the harbor where most of the people embarked. For each harbor, we have a green and a blue bar. The size of the green bar represents the number of people who didn't survive and the blue bar represents the number of people who survived.

As you can see, the mosaic plot gives us a fast understanding about how our data is distributed.

Correlations among input variables

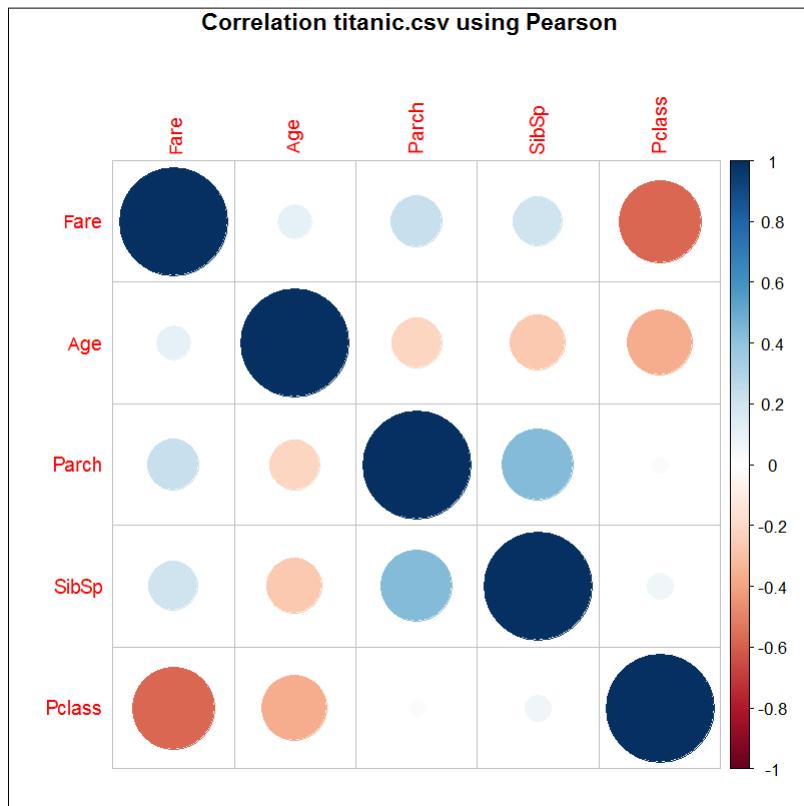
An important step is to identify relationships among input variables. To measure this relationship, we use the correlation coefficient. **Correlation coefficient** is a number between +1 and -1. When two variables have a correlation coefficient close to +1, they have a strong positive correlation. A coefficient of exactly +1 indicates a perfect positive fit. A positive correlation between two variables means that both variables increase and decrease their values simultaneously. A correlation coefficient between two variables close to -1 shows that both variables have strong negative correlation. When two variables have a negative correlation, the value of one of the variables increases when the value of the other variable decreases. A correlation coefficient close to 0 or a weak correlation between two variables means that there is no linear relationship between those variables.

Coming back to the Titanic passenger list, I've selected the **Explore** tab, the **Correlation** sub-option, and I've clicked on the **Execute** button, as shown in this screenshot:



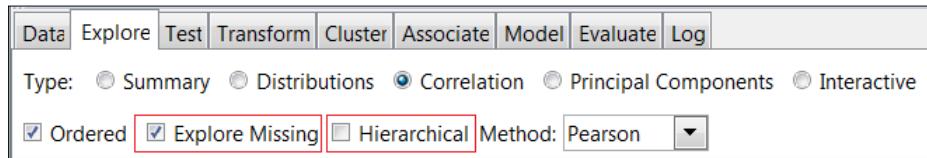
Of course, each variable has a correlation coefficient with itself of 1.0. Now look at the variable **Pclass** (passenger class). This variable has three possible values: 1 (first class), 2 (second class), and 3 (third class). This is a categorical variable because there are three possible groups or categories. These categories are ranked and we're going to use a numeric variable for that. In this way, Rattle can compute the correlation between **Pclass** and other numeric variables. Look at the correlation coefficient between **Fare** and **Pclass**; it is -0.573. Is there any relationship between **Fare** and **Pclass**? A correlation coefficient close to -0.6 indicates that there is some correlation between the two variables. What does this correlation between **Fare** and **Pclass** mean in real life, though? Usually, first class tickets are the most expensive, second class tickets are cheaper, and the third class tickets are the cheapest. Still, why is the relationship between **Pclass** and **Fare** negative? It is because a higher value of **Fare** (higher price) indicates a lower number of the variable **Pclass** (higher class).

The following chart is a visual representation of the correlation coefficients. By looking at the graph, you will see that the correlations coefficients are the same as in the previous report. Note that you need to enable the **Advanced Graphics** option inside the **Settings** menu for this:



The Explore Missing and Hierarchical options

The **Explore Missing** option will help you to detect relationships between missing values in your dataset, as shown in the following screenshot:



When two variables have a strong correlation in missing values, it means that when the value of a variable is not present, the second variable also tends to have a missing value.

The **Hierarchical** option uses a tree diagram graphical to represent the correlation between variables.

Further learning

In this chapter, we've introduced some EDA measures. If you want a more extensive EDA introduction, I recommend the *Exploratory Data Analysis* course on Coursera – www.coursera.org/course/exdata.

If you prefer going to the source, *Exploratory Data Analysis Paperback*, by John W. Tukey, is for you.

Wikipedia offers some useful insights into these EDA statistics concepts.

Summary

This chapter was divided into three main sections depending on how we are looking at data – tables, text summaries, and charts.

When we saw text summaries, we introduced Summary, Describe, Basics, Kurtosis, and Skewness reports. To understand these reports, we needed to remember some basic statistics concepts like mean, median, mode, range, quartile, interquartile range, variance, and standard deviation.

In this chapter, we also introduced some important charts – histograms, correlations, **Box Plot**, and **Bar Chart**.

In the next chapter, we'll learn how to load data into Qlik Sense and how to create data visualizations. We'll use some of the charts we introduced in this chapter. You'll see that Qlik Sense is more powerful for a business user who wants to understand his data and create a graphical representation of his data. Rattle and R are tools closer to statistics and some functionalities, like the correlations analysis, are very powerful; for this reason, we've introduced EDA using Rattle. After the next chapter, you'll be able to choose the tool that you feel more comfortable with for each task.

We'll continue exploring data in the next chapter. After the next chapter, we'll start creating predictions with our data.

4

Creating Your First Qlik Sense Application

In the previous chapters, we've seen how to use Rattle to modify and explore our data. The exploration we've done is mainly a mathematical exploration. Qlik Sense is the perfect tool to explore and understand the data from a business point of view. Qlik Sense is easy and intuitive. In this chapter, we'll create a simple application in order to explore the basics of Qlik Sense.

To create a simple application, we'll follow these steps:

- Download an example dataset
- Learn how to load it into Qlik Sense
- Learn about the Qlik Sense data model and its application structure
- Learn how to create basic charts such as bar and pie charts
- To finish our application, we'll create some filters that will help us to select the desired information
- Finally, we'll learn to explore our dataset using Qlik Sense; at this point, we'll start answering basic business questions

Customer segmentation and customer buying behavior

Segmenting the customers means dividing our customers into groups relevant to our business. Customers are divided based on demography, behavior, and other indicators. Analyzing your customers and dividing them into different groups allows you to be more accurate in your marketing activities.

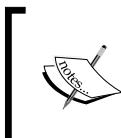
There are different types of customer segmentation; some of them are:

- Geographic segmentation
- Demographic segmentation
- Buying behavior segmentation
- Psychographic segmentation
- Segmentation by benefits
- Cultural segmentation

In this chapter, we'll develop an application that allows us to visually create customer segments based on different variables. In the next chapter, we'll create a system that will automatically segment our customers based on their shopping habits in the main product categories. The main objective of this application is improving our knowledge of our customers to address more effective marketing activities.

Loading data and creating a data model

In order to create an example application, I've downloaded a dataset from the Center for Machine Learning and Intelligent Systems at the University of California, Irvine. They have a dataset repository you can use for training purposes. The datasets are organized by task (clustering, classification, regression, and others), by attribute type, by domain area, and so on. This is a very useful resource to practice your new skills and we'll be using it again in this book.



You can find more information from Bache, K. and Lichman, M. (2013); UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]; Irvine, CA: University of California, School of Information and Computer Science.

In this chapter, we're going to use a dataset called **Wholesale customers Data Set**. The dataset is originated from a larger database – Abreu, N. (2011); Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional; Mestrado em Marketing, ISCTE-IUL, Lisbon. You can find the dataset on this page:

<https://archive.ics.uci.edu/ml/datasets/Wholesale+customers#>

The dataset contains 440 customers (observations) of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. The columns are explained as follows:

- **Fresh:** annual spending (per 1,000) on fresh products
- **Milk:** annual spending (per 1,000) on milk products
- **Grocery:** annual spending (per 1,000) on grocery products
- **Frozen:** annual spending (per 1,000) on frozen products
- **Detergents_Paper:** annual spending per 1,000) on detergents and paper
- **Delicatessen:** annual spending (per 1,000) on delicatessen products
- **Channel:** Horeca (value = 1) or Retail (value = 2)

 In the food industry, Horeca stands for Hotel, Restaurant, or Café, so a business that prepares and serves food.

- **Region:** Lisbon (value = 1), Porto (value = 2), or Other (value = 3)

In the following screenshot, you can see what the data looks like:

| Channel | Region | Fresh | Milk | Grocery | Frozen | Detergents_Paper | Delicassen |
|---------|--------|-------|------|---------|--------|------------------|------------|
| 2 | 3 | 12669 | 9656 | 7561 | 214 | 2674 | 1338 |
| 2 | 3 | 7057 | 9810 | 9568 | 1762 | 3293 | 1776 |
| 2 | 3 | 6353 | 8808 | 7684 | 2405 | 3516 | 7844 |
| 1 | 3 | 13265 | 1196 | 4221 | 6404 | 507 | 1788 |
| 2 | 3 | 22615 | 5410 | 7198 | 3915 | 1777 | 5185 |
| 2 | 3 | 9413 | 8259 | 5126 | 666 | 1795 | 1451 |

An important difference between Rattle, or R, and Qlik Sense is that in Rattle, generally, our dataset is a simple table. Using Qlik Sense, we can easily work with more complex data models. Working with more complex data models allows us to discover hidden relationships. In this example, we have a table with customer data; if we're able to link the customer information with a salesperson or shipping information, our analysis would be richer.

In Qlik Sense Desktop, we have two ways to load data:

- We can use the **Data load editor** option
- We can use the **Quick data load** option

With the **Quick data load** option, you can load data by just dragging and dropping data files, but if you want to transform data into Qlik, you need to do it using the **Data load editor** option. We'll create our data model using only the **Quick data load** option, but we'll also see how we can do the same work using the **Data load editor** option.

Preparing the data

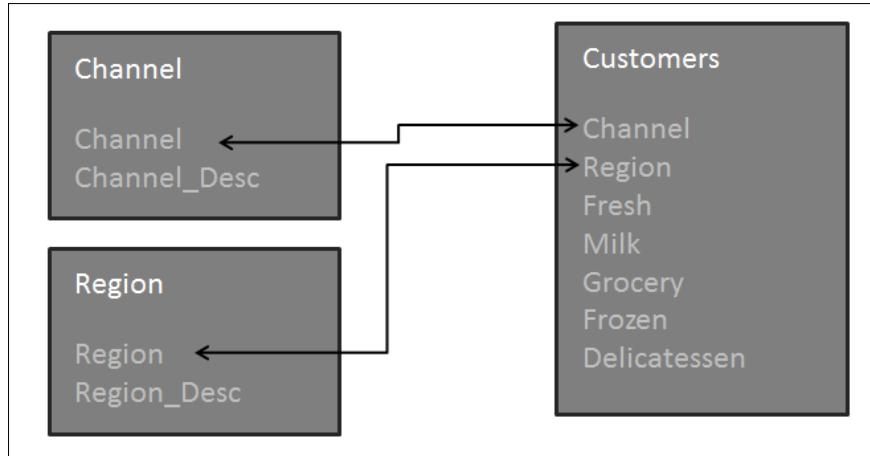
We're going to create a very simple data model.

Our data model has three tables. The main table, *Customers*, is the dataset we've downloaded. We also have two additional tables, *Channel* and *Region*. We're going to use the tables to convert from codes to descriptions; the value **1** in the field *Region* means Lisbon.

The original dataset contains six product categories – **Fresh**, **Frozen**, **Milk**, **Grocery**, **Delicatessen**, and **Detergents_Paper**. The six product categories can be grouped into two main categories – **Food** (**Fresh**, **Milk**, **Grocery**, **Frozen**, and **Delicatessen**) and **Detergents_Paper**. We'll create two new columns in the dataset called **Food** and **Total_Spent**. The new variable **Food** will contain the sum of **Fresh**, **Milk**, **Grocery**, **Frozen**, and **Delicatessen**. The new variable **Total_Spent** will contain the total annual expenditure for each customer. We can create the new fields in two ways. We can use a spreadsheet tool to create these new columns and set it as a CSV file to save the resulting data. We can also use the **Data load editor** to do it. In this example, we'll use the **Data load editor** to create these two variables and a third variable called **Customer_ID**. The variable **Customer_ID** will be a unique identifier for each customer.

In the original dataset, we need six values to represent the annual expenditure of a customer. It's hard to represent six variables in a two-dimensional chart. In the modified dataset, we can use two values – **Food** and **Detergents_Paper** – to summarize the six original values. In this way, we can graphically represent the annual expenditure of a customer by a point in a plane. This is a trick to see your customers in an easy way.

You can see our data model in the following diagram:



In the preceding diagram, we can see that the three tables are associated. To associate two tables, Qlik Sense only needs to find two fields with the same name. If Qlik Sense finds a field called `Channel` in the `Customers` table and a field with the same name in the `Channel` table, its associative engine assumes that the two fields mean the same and associates the tables.

Create two CSV or Excel files containing the following two smaller tables. These tables will have the common columns `Channel` and `Region` when compared with the main `Customers` table:

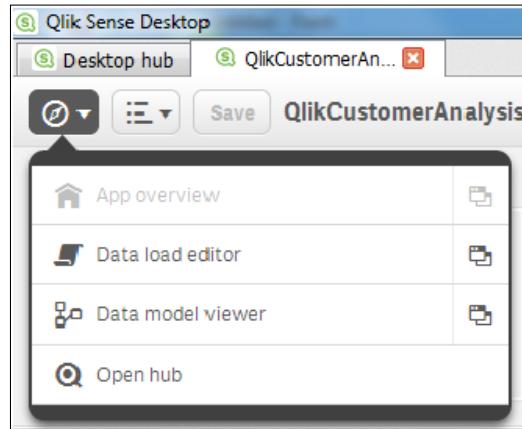
| Channel | Channel_Desc |
|---------|--------------|
| 1 | Horeca |
| 2 | Retail |

| Region | Region_Desc |
|--------|-------------|
| 1 | Lisbon |
| 2 | Porto |
| 3 | Other |

Creating Your First Qlik Sense Application

Open Qlik Sense Desktop and in the pop-up window, select **Create new app**, name it, and open the new application.

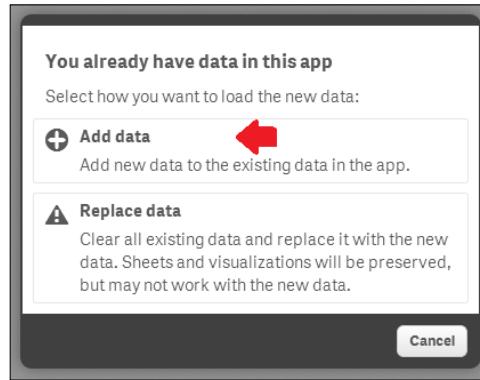
A Qlik Sense application has a main menu. We'll use this menu to move between the **App overview**, **Data load editor**, **Data model viewer**, and **Open hub** options, as shown here:



Drag the customers file you have modified and drop it over your new application. A window showing the data you are going to load will appear; simply click on the **Load data** button. After these instructions are followed, you will see a screen similar to this:

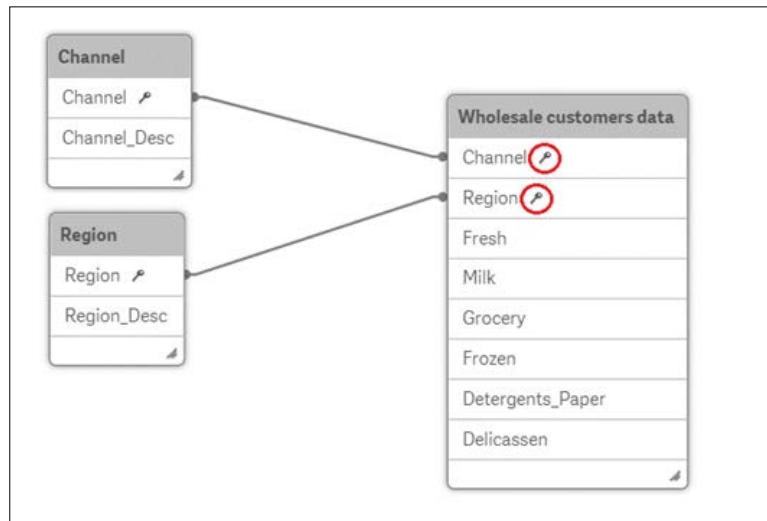
A screenshot of the "Select data from WholesalecustomersdataV2.csv" dialog box. At the top, there are settings for "File format" (Delimited), "Field names" (Embedded field names), "Delimiter" (Semicolon), "Quoting" (MQQ), and "Character set" (Western European). Below these are buttons for "Header size" (with a dropdown showing "0"), "Ignore End-Of-File character?", and "Comment". Under the "Fields" section, there is a checkbox "Select all fields" and a search bar "Search fields". The main area shows a preview of the data with columns: Channel, Region, Fresh, Milk, Grocery, Frozen, Delicatessen, Purer, and Food. The data consists of several rows of numerical values. A red arrow points to the bottom right corner of the dialog box, where there are "Cancel" and "Load data" buttons.

A pop-up window will inform you that the data has been loaded. Close the window and load the **Channel** and **Region** files (the two smaller CSV files that we created earlier). Every time you try to add a new data file, Qlik Sense will ask you if you want to replace or add data; choose **Add data** as shown in the following screenshot:



Now we can review our work. Open the **Data model viewer** option. Qlik Sense opens the data model we've just built.

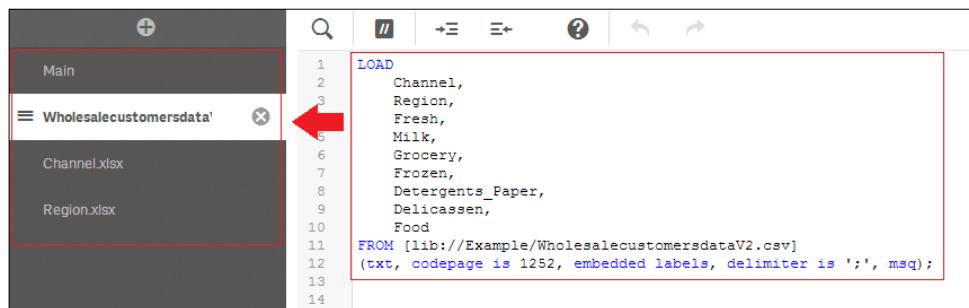
Check the data model we've just created. Close to the fields **Channel** and **Region**, there is an icon representing a key. I've circled the icon to make it easier to identify. This icon means that Qlik uses this field to associate a table with another one, as shown in the following screenshot:



To load this data we've used the Quick data load functionality. This functionality is only present in Qlik Sense Desktop, and is not present in the server version of the product. Qlik Sense, as a platform not as a personal tool, focus on data governance. For an analytic tool data governance are mechanisms to ensure the data loaded in the system meets the organization's standards. For this reason this functionality is not present in in Qlik Sense.

Qlik Sense Desktop Quick data load functionality has done a lot of work for us. In order to understand what happened, we'll review the Data load editor and we'll use it to create the three new variables.

Look at the left-hand side vertical bar; Qlik Sense has created a sheet for every file that we've loaded. Look at this sheet and you will find a `LOAD` sentence like the one shown in the following screenshot:

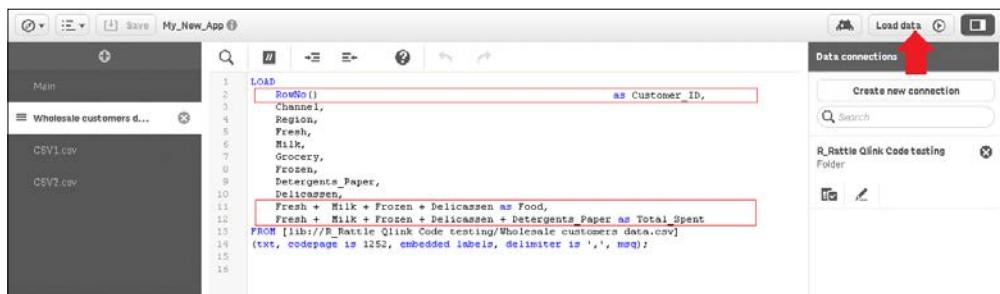


The screenshot shows the Qlik Sense Data Load Editor interface. On the left, there is a vertical list of sheets: 'Main', 'Wholesalecustomersdata' (which is selected and highlighted with a red arrow), 'Channel.xlsx', and 'Region.xlsx'. The main area displays a script editor with the following code:

```
LOAD
    Channel,
    Region,
    Fresh,
    Milk,
    Grocery,
    Frozen,
    Detergents_Paper,
    Delicassen,
    Food
FROM [lib://Example/WholesalecustomersdataV2.csv]
(txt, codepage is 1252, embedded labels, delimiter is ',', msg);
```

This `LOAD` sentence is very useful; if you want, you can modify data here before loading it. As we've said before, we want to add three new columns to this table – **Customer_ID**, **Food**, and **Total_Spent**. We can do this simply by adding the columns to the spreadsheet or by using the **Qlik Data load editor** option. In this example, we're going to use the **Data load editor** option.

Change the code as shown in the following screenshot and click on the **Load data** button. Qlik Sense will reload all data files. With this code, you've calculated the new field in Qlik Sense instead of doing it in the CSV file or the Excel spreadsheet:



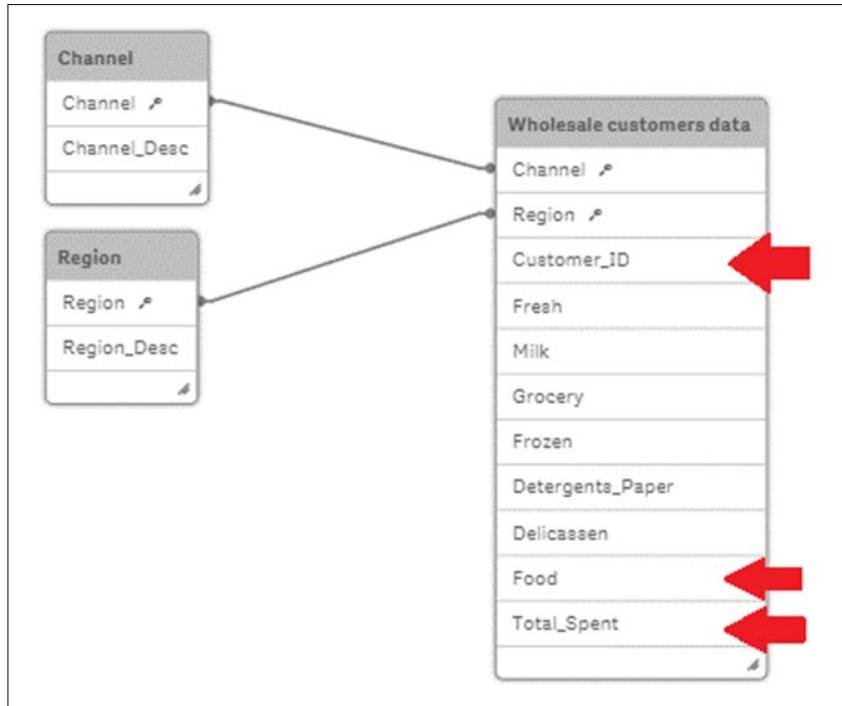
The screenshot shows the Qlik Sense Data Load Editor interface after modifying the script. The code now includes new fields and a calculation:

```
LOAD
    RowNo() as Customer_ID,
    Channel,
    Region,
    Fresh,
    Milk,
    Grocery,
    Frozen,
    Detergents_Paper,
    Delicassen,
    Fresh + Milk + Frozen + Delicassen as Food,
    Fresh + Milk + Frozen + Delicassen + Detergents_Paper as Total_Spent
FROM [lib://R_Rattle Qlik Code testing/Wholesale customers data.csv]
(txt, codepage is 1252, embedded labels, delimiter is ',', msg);
```

A red arrow points to the **Load data** button in the top right corner of the editor.

The first line adds a number and labels it as `Customer_ID`. The function `RowNo()` returns the number of the row. In this dataset, each row is a different customer, so with this line of code, we'll add an identifier to each customer. The last two lines add the new variables `Food` and `Total_Spent`.

Open the **Data model viewer** option again; now you can see the three fields that we've just created. This is shown in the following screenshot:

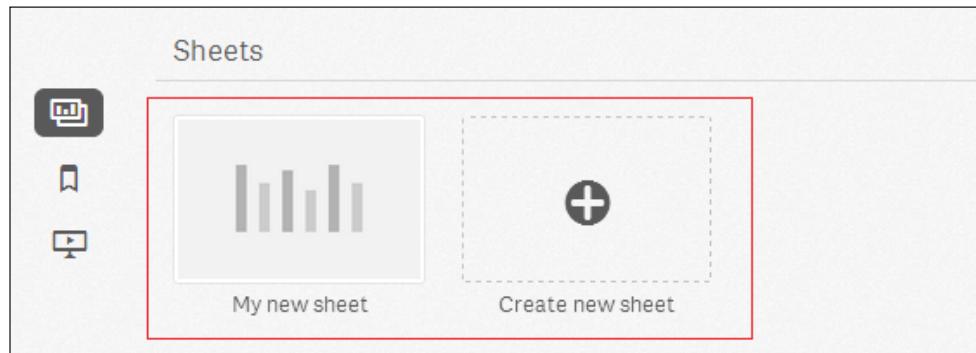


As you've seen, you can create the same data model using your favorite spreadsheet tool and load it to Qlik Sense or use the **Data load editor** option. The **Data load editor** option is a powerful Qlik Sense feature. Like in other self-service visualization tools, Qlik Sense has the option to load data without writing a line of code, but you also have a powerful data loading and transformation tool. Personally, I prefer the **Data load editor** option because it provides me with precise control over my data.

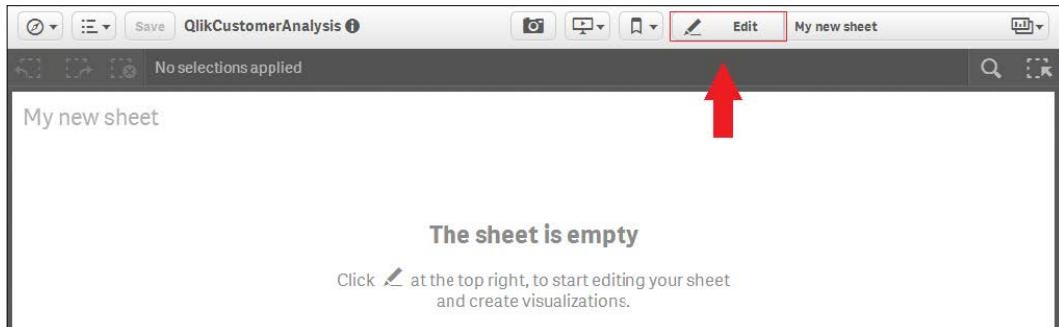
Creating a simple data app

As we've seen in *Chapter 1, Getting Ready with Predictive Analytics*, a Qlik Sense application is based on different sheets. In this section, we'll learn how to add a new sheet in your application and how to add basic charts and filters.

In the Qlik Sense main menu, choose **App overview** to open your application. A new Qlik Sense application always has an empty sheet called **My new sheet**, and you always have the option of adding a new one. This option is shown in the following screenshot:



Now you are on an empty sheet. In order to modify a Qlik Sense sheet, you need to turn the **Edit** mode on. You can do this by clicking on the **Edit** mode to add new visual components, as shown in this screenshot:

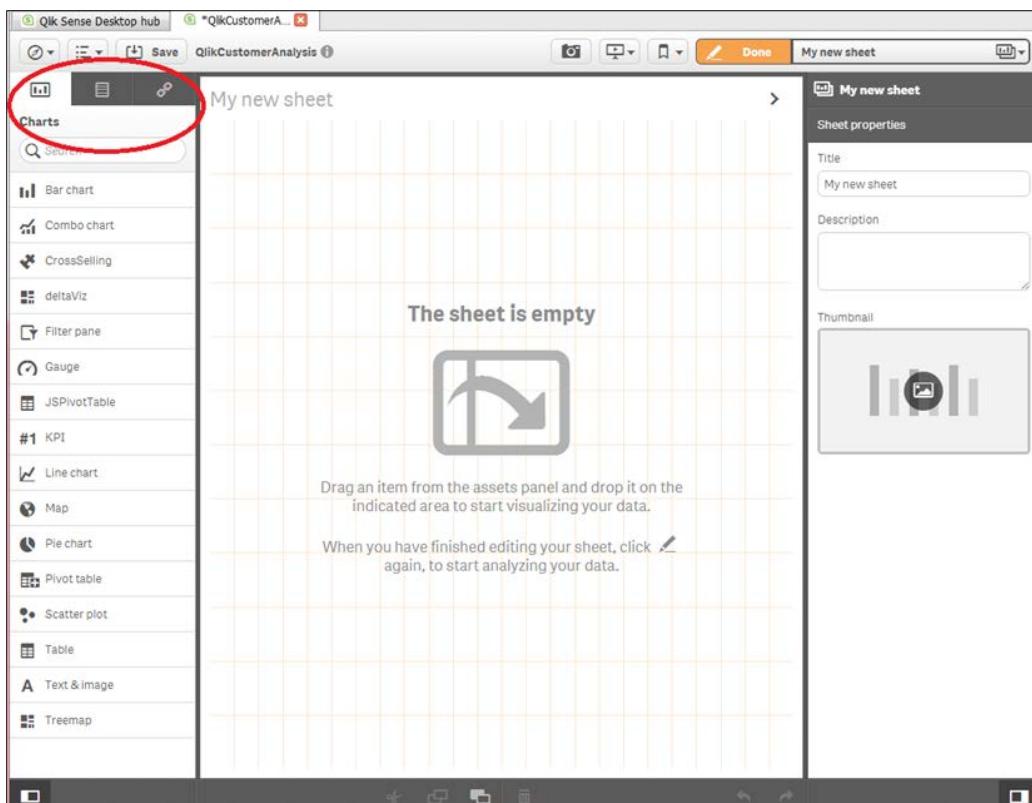


Associative logic

Before learning how to create charts, we'll learn how associative logic works.

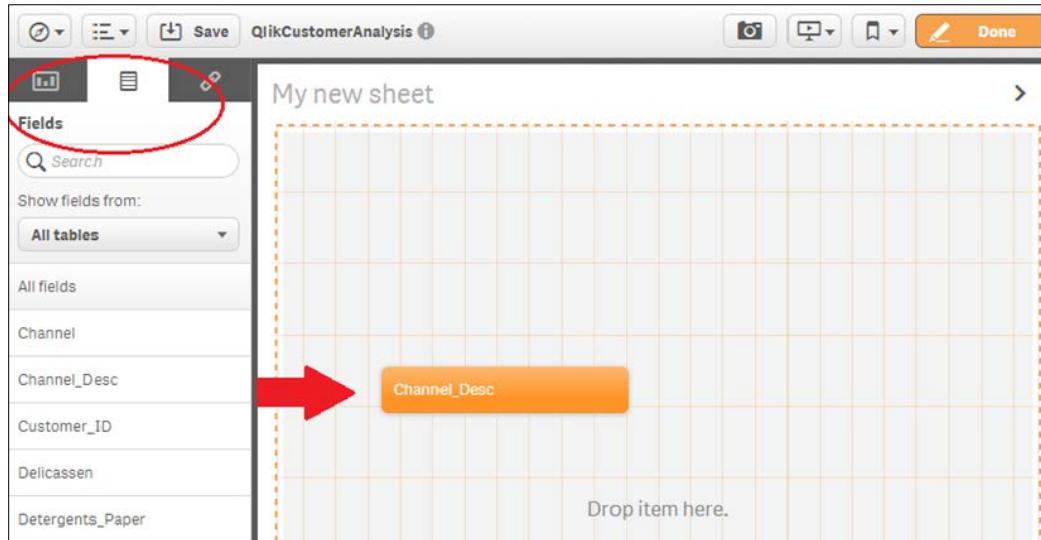
Associative logic is a key functionality in Qlik Sense – it allows a business user without technical knowledge to explore the data.

In the following screenshot, we'll see the Qlik Sense main screen in **the Edit mode**. The screen is divided into three areas. In the center pane, you can see the sheet you're developing; the current screenshot shows an empty sheet. The right-hand pane shows the properties of the active object. In this case, the right-hand pane shows the sheet properties – **Title**, **Description**, and **Thumbnail**. The left-hand pane has a tab row with three options – **Charts**, **Fields**, and **Master items**; in this chapter, we'll use **Charts** and **Fields**, as shown here:



Creating Your First Qlik Sense Application

From the left-hand side tab row, select **Fields**; you'll see all fields in alphabetical order. Drag the **Channel_Desc** field and drop it into the central area, as shown in the following screenshot:



This action will add a filter pane to your new sheet. The filter pane has four arrows – up, down, right, and left. Use these arrows to move and resize the filter pane. You will be able to move and resize all visual objects in Qlik Sense, as shown here:



After placing **Channel_Desc**, add two fields as pane filters in your sheet – **Region_Desc** and **Customer_ID**. Finally, click on the **Done** button to exit the edit mode, as shown in the following screenshot:

The screenshot shows the Qlik Sense edit mode interface. On the left, there's a sidebar with 'Charts' and four chart types: Bar chart, Combo chart, and CrossSelling. The main area is titled 'My new sheet' and contains three columns of data:

| Channel_Desc | Region_Desc | Customer_ID |
|--------------|-------------|-------------|
| Horeca | Lisbon | 1 |
| Retail | Other | 2 |
| | Porto | 3 |
| | | 4 |
| | | 5 |

In the top right corner of the main area, there's a red arrow pointing upwards towards the 'Done' button.

You've learned in *Chapter 1, Getting Ready with Predictive Analysis*, how to use a filter in Qlik Sense. The following screenshot illustrates that the two filters that have been selected, turned green in color. Use these two filters to answer the question: which retail customers do I have in Porto?

The screenshot shows the Qlik Sense view mode interface. At the top, there are two filters: 'Channel_Desc' set to 'Retail' and 'Region_Desc' set to 'Porto'. Below the filters, the 'My new sheet' table is displayed with the following data:

| Channel_Desc | Region_Desc | Customer_ID |
|--------------|-------------|-------------|
| Retail | Porto | 313 |
| Horeca | Lisbon | 316 |
| | Other | 328 |
| | | 332 |
| | | 334 |
| | | 335 |
| | | 336 |
| | | 1 |
| | | 2 |
| | | 3 |

You've selected **Retail** and **Porto** in the **Channel_Desc** and **Region_Desc** filters. In the **Customer_ID** pane, you can see some customers with a white background and some customers with a dark grey background. The customers with the white background are customers associated to **Retail** and **Porto**, so these customers are **Retail** customers from **Porto**. The dark grey customers are not related to **Porto** and **Retail**, so they are not **Retail** customers from **Porto**.

Using filters, a business user with no technical knowledge can ask everything about his dataset.

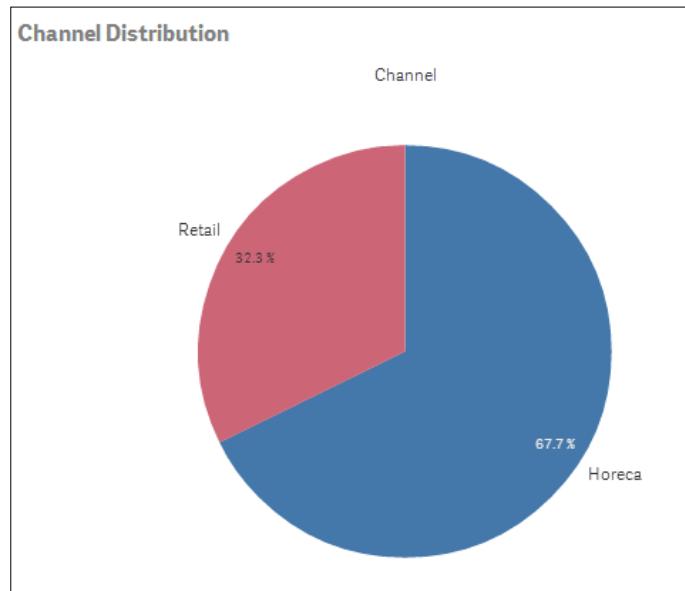
Creating charts

Before starting to create **chart** diagrams, delete the filters we've created in the previous section or create a new sheet.

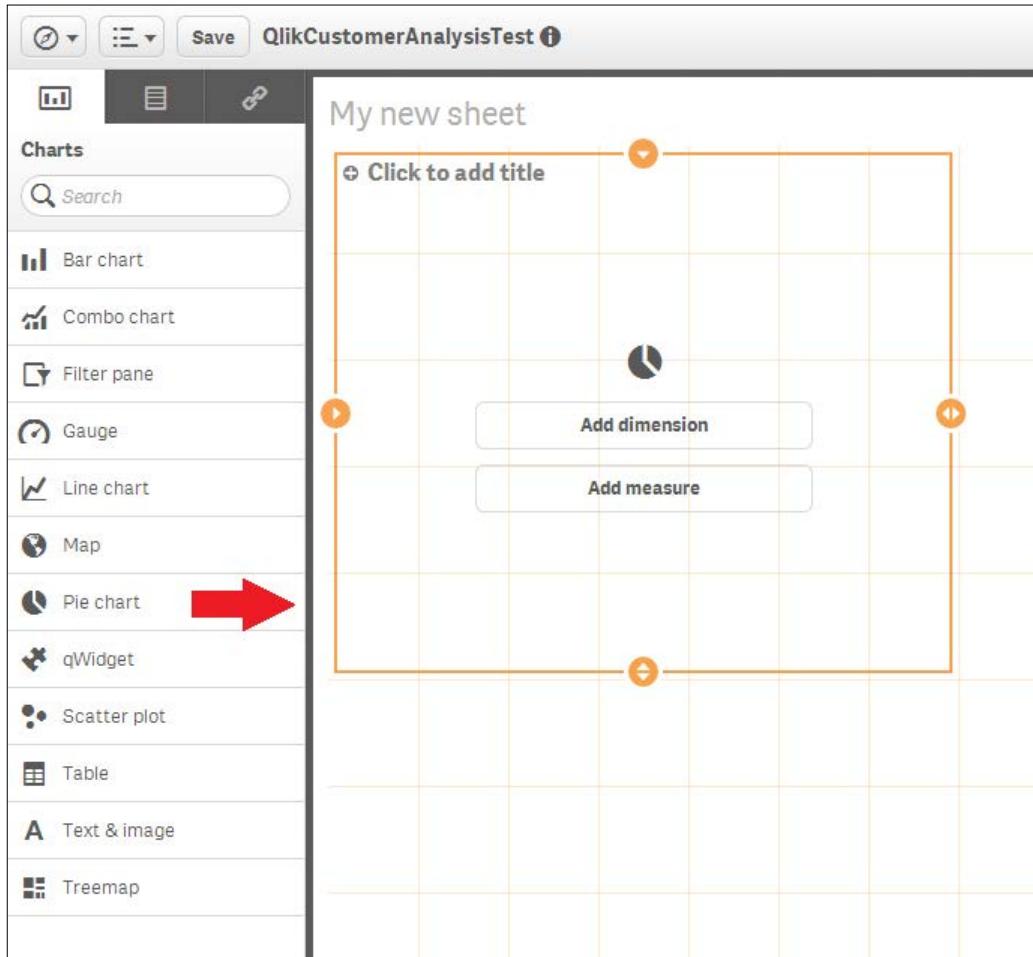
To create a Qlik Sense visualization, you need to know three important things:

- The type of **chart** you are going to use
- The **dimension** objects you are going to use in your analysis
- The metric or metrics

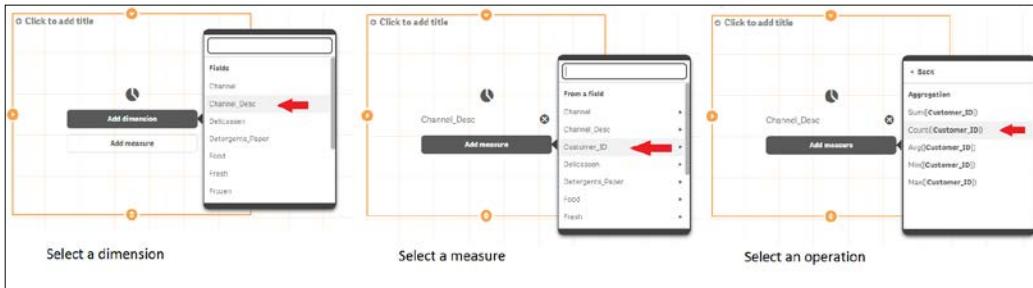
We're going to start with a very basic **chart**. Our objective is to create a **pie chart** like the chart in the following screenshot. This chart explains the distribution of our customers between two channels – **Horeca** and **Retail**:



In this pie chart, the dimension is **Channel** and the measure is the number of customers. On the left-hand side of the following screenshot, there is a bar with all of the different charts that Qlik Sense provides. Drag a pie chart and drop it into the central area as shown in the following screenshot. Change the size of the chart with the orange lines and place it wherever you prefer:



In order to finish the pie chart, you need to choose a **dimension** and a **measure**, and change the **title** to an appropriate one. In order to add **Channel** as the dimension, click on **Add dimension** and select **Channel_Desc**. Finally, add **Count([Customer_ID])** as the measure. This is shown in the following screenshot:

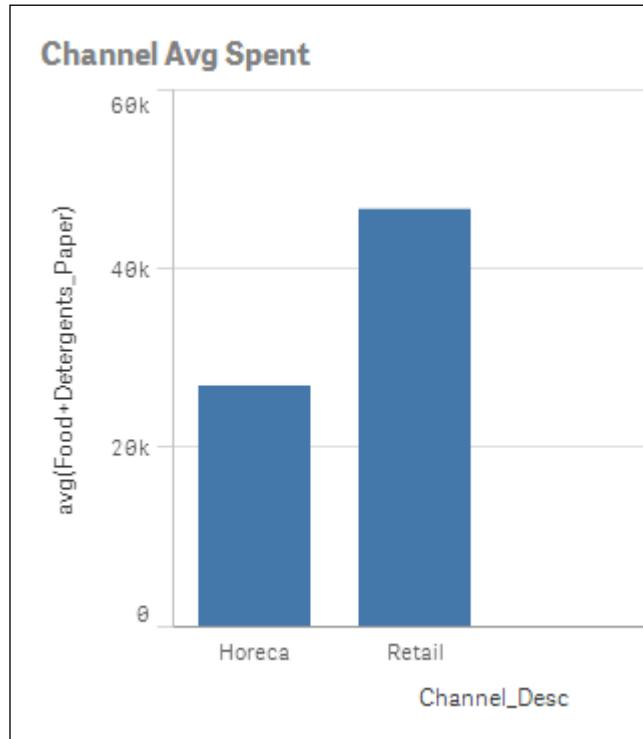


Close the edit mode by clicking on **Done**:



You've finished your first pie chart and it tells you that 67.7 percent of your customers belong to **Horeca** and only 32.3 percent belong to **Retail**.

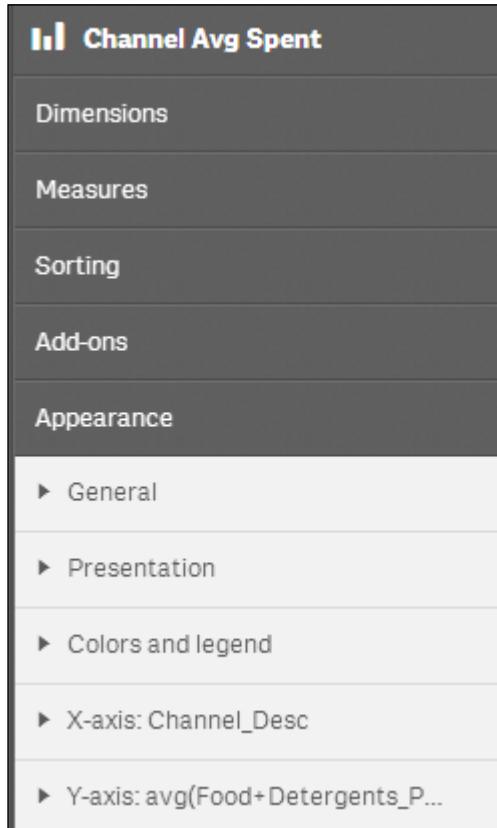
Now I would like to understand which customers spend more money. I will create a bar chart using **Channel_Desc** as the dimension and the average money spent as the measure. Drag-and-drop a bar chart into the central area, change its **title**, and select **Channel_Desc** as the dimension and **avg([Total_Spent])** as the measure. Your bar chart might look similar to the following screenshot:



You've created a chart that tells you that, on average, **Retail** customers spend more money than **Horeca** customers. I would like to improve this bar chart. Turn the **Edit** mode on and click on **Bar chart** to select it. On the right-hand side, you have a bar with the chart properties. For a bar chart, the properties are organized around five areas:

- **Dimensions**
- **Measures**
- **Sorting**
- **Add-ons**
- **Appearance**

These areas are shown in the following screenshot:



Bar charts are a very good measure to see the difference in the average money spent by **Horeca** and **Retail** customers, but it's hard to see the exact value. For this reason, we'll put the exact value in the chart. We're going to change the labels for the dimension and the measure, from **Channel_Desc** and **Avg(Total_Spent)** to **Channel** and **Avg Spent**, by following these steps:

1. Expand the **Dimensions** menu and write a new label in the **Label** text field.
2. Expand the **Measures** menu and write an appropriate name in the **Label** text field.
3. Expand the **Appearance** menu and turn the **Value labels** option on to see the exact value on the chart.

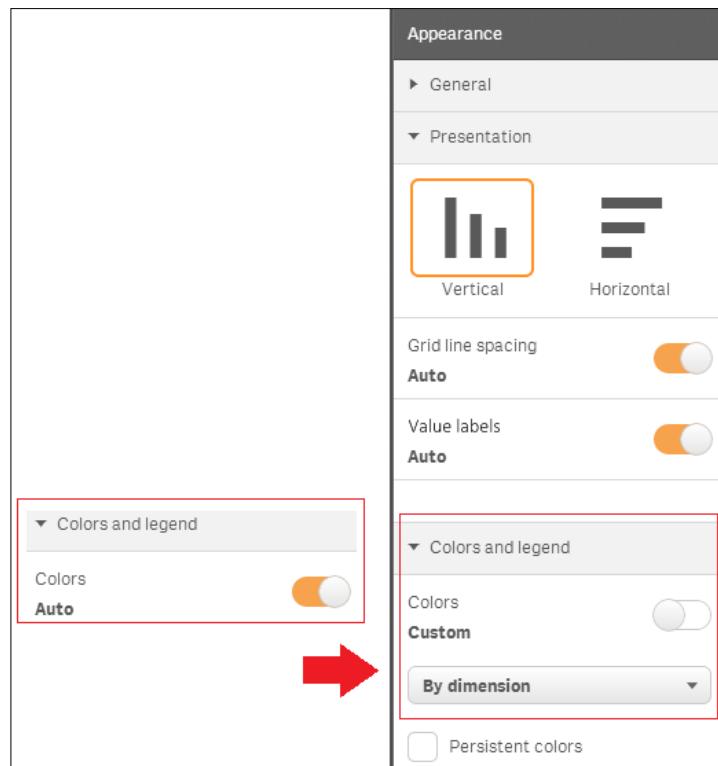
These steps are demonstrated in the following screenshot:

1 – Use Channel_Desc as dimension and label it 'Channel'.

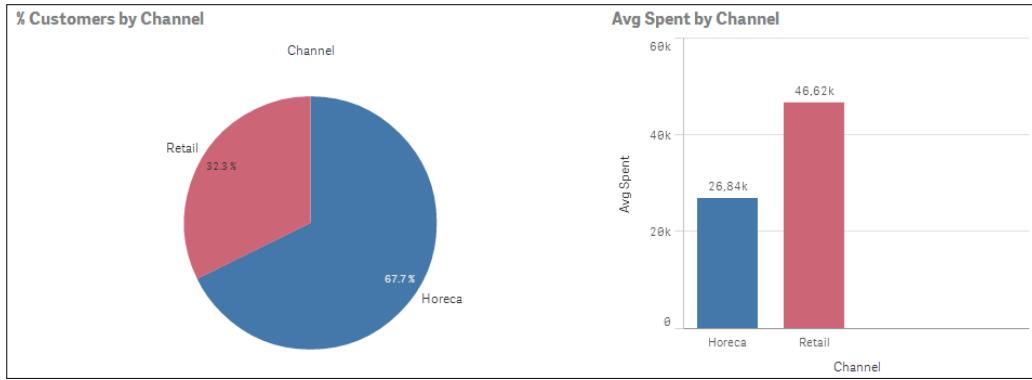
2 – Create a measure and label it 'Avg Spent'.

3 – Turn on 'Value labels'.

Finally, in the pie chart we created before this bar chart, **Retail** and **Horeca** customers appeared colored in blue and red, but in the bar chart, all bars are blue. In order to have a consistent application in terms of color, we'll change the color properties of the bar chart to have the same colors as in the pie chart. As you can see in the following screenshot, we need to turn off the **Auto** mode from the **Color** option and set it to **By dimension**:



Now we've created two plots that explain that we have more **Horeca** customers than **Retail** customers, but on average our **Retail** customers spend more money on our business:

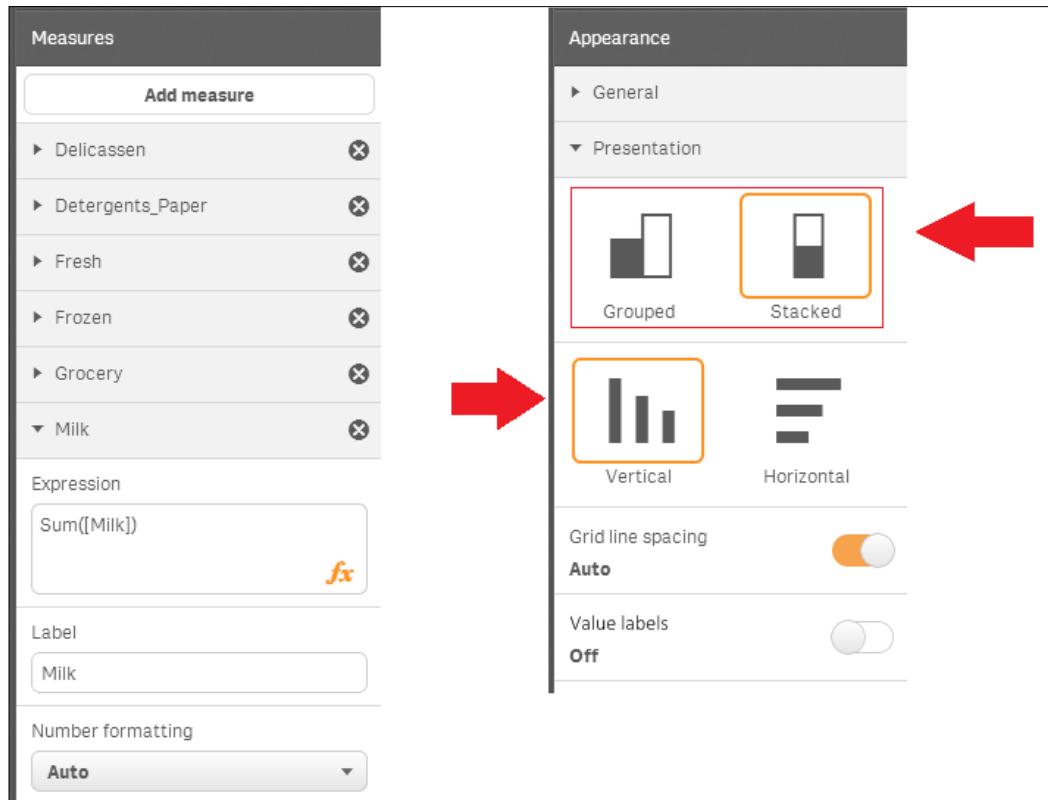


I know that, on average, I'm earning more money with the **Retail** channel customers, but my next question is: in which channel am I earning more money in absolute terms?

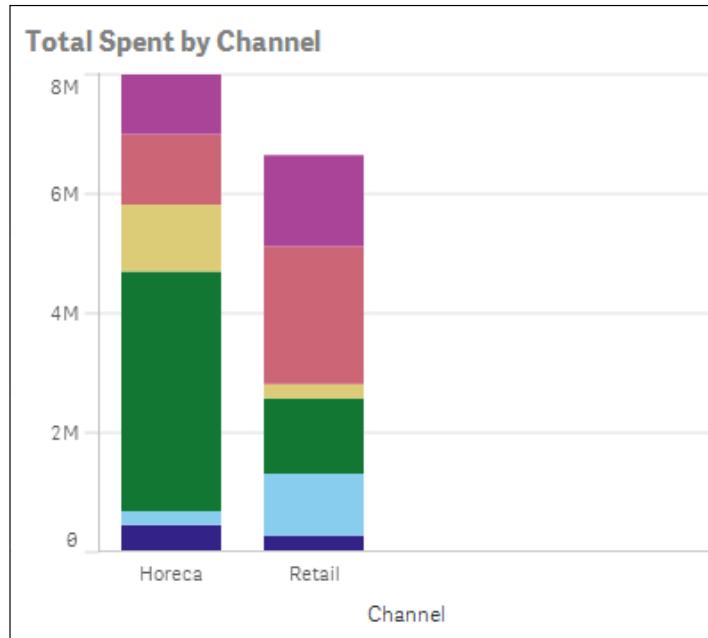
Create a new bar chart with **Channel_Desc** as the dimension and the following six measures:

- **Sum([Delicassen])** – label it Delicatessen
- **Sum([Detergents_Paper])** – label it Detergents
- **Sum([Fresh])** – label it Fresh
- **Sum([Frozen])** – label it Frozen
- **Sum([Grocery])** – label it Grocery
- **Sum([Milk])** – label it Milk

In the **Appearance** menu, expand the **Presentation** submenu. In this bar chart, we'll see six different measures; we can see these measures grouped in six different bars for **Retail** and six for **Horeca**. We also can see the six metrics stacked in one single bar. In this case, stacking the metrics in a single bar has the advantage of us being easily able to see the total money spent. This is illustrated in the following screenshot:

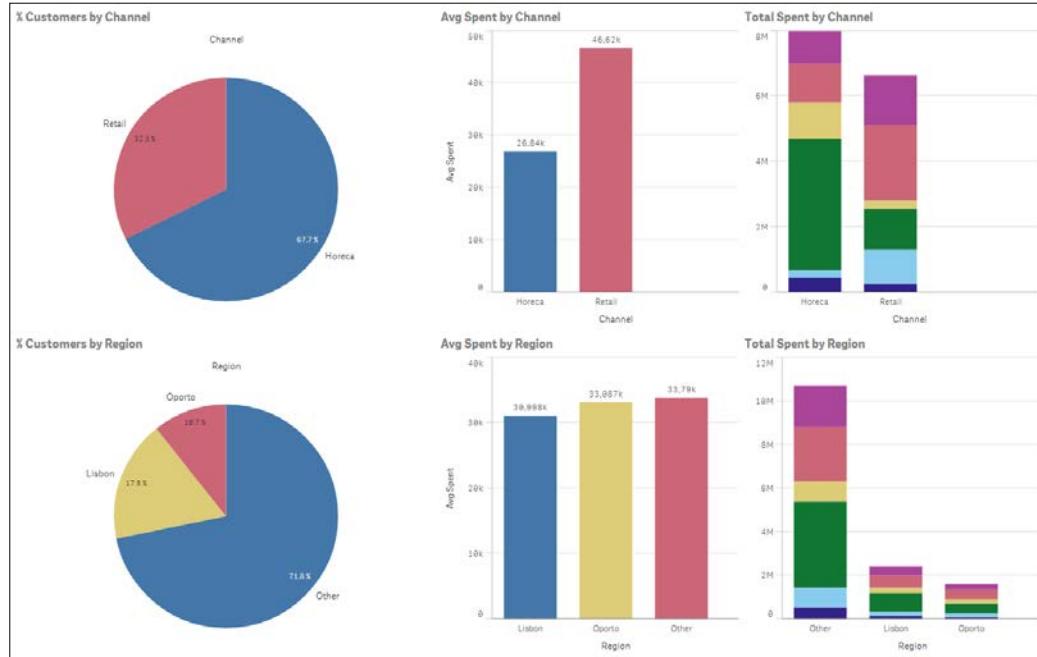


With this new bar chart, you can see that we're earning 8 million with our **Horeca** customers and 6.62 million with our **Retail** customers. We can also see that **Horeca** customers spend 4 million in **Fresh** products. The **Retail** customers spend 2.32 million in **Grocery**.



Creating Your First Qlik Sense Application

We've created a pie chart and two bar charts to understand how **Horeca** and **Retail** customers are buying our products. Now, repeat the three charts by changing the dimension from `Channel_Desc` to `Region_Desc`, as shown in the following screenshot:

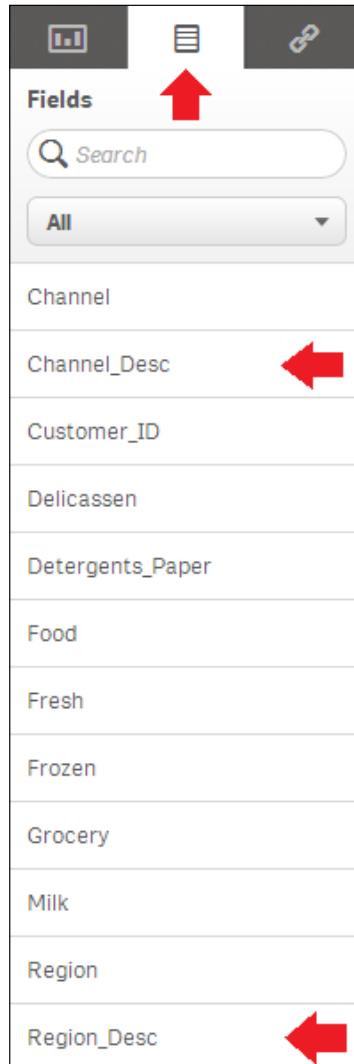


To finish our application, we're going to create a table with our top customers and two filters. Tables are different from regular charts because tables have columns; when you add a new column to a table, Qlik Sense asks you if the new column is a dimension or a measure. To create a table of top customers, drag a table and drop it into the central area. Choose **Customer_ID** as the dimension and **sum(Total_Spent)** as the measure. Limit the number of customers to **10** and sort by **Total Spent**, as shown here:

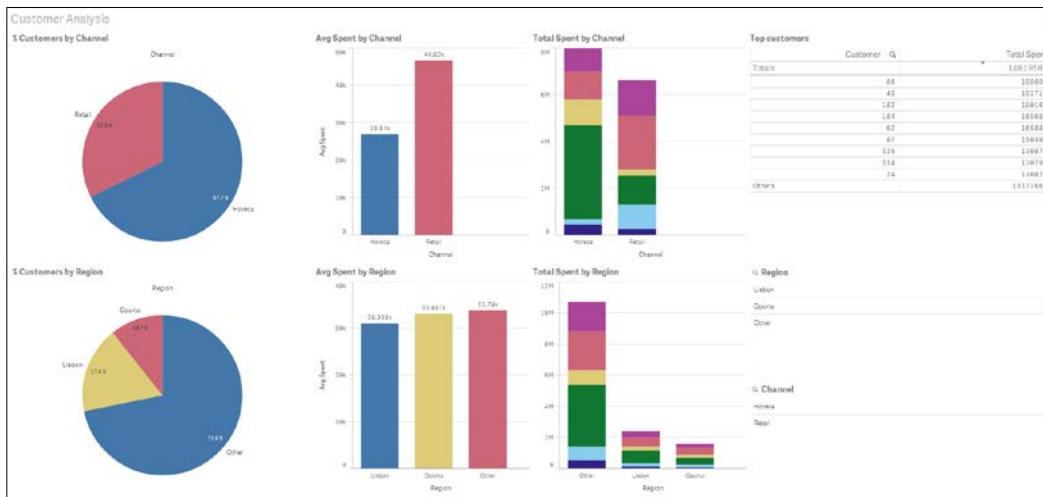
The image consists of three side-by-side screenshots of the Qlik Sense Table editor. Each screenshot shows a different step in the configuration process:

- Left Screenshot:** Shows the 'Columns' section where 'Customer_ID' is selected as the field and 'Customer' is chosen as the label. A red box highlights the 'Customer_ID' field.
- Middle Screenshot:** Shows the 'Expression' section where 'Sum(Total_Spent)' is entered and 'Total Spent' is selected as the label. A red box highlights the 'Sum(Total_Spent)' expression.
- Right Screenshot:** Shows the 'Sorting' section where 'Total Spent' is listed as the primary sort key and 'Customer' is listed as the secondary sort key. A red box highlights the sorting configuration.

To finish the application, we'll add two filters. Go to the left-hand side bar, open the **Fields** list, and drag-and-drop **Channel_Desc** and **Region_Desc** into the central area:



Click on the **Save** button on the top bar and turn the **Edit** mode off. We've finished our first application. In the next section, we'll use this application to answer business questions:



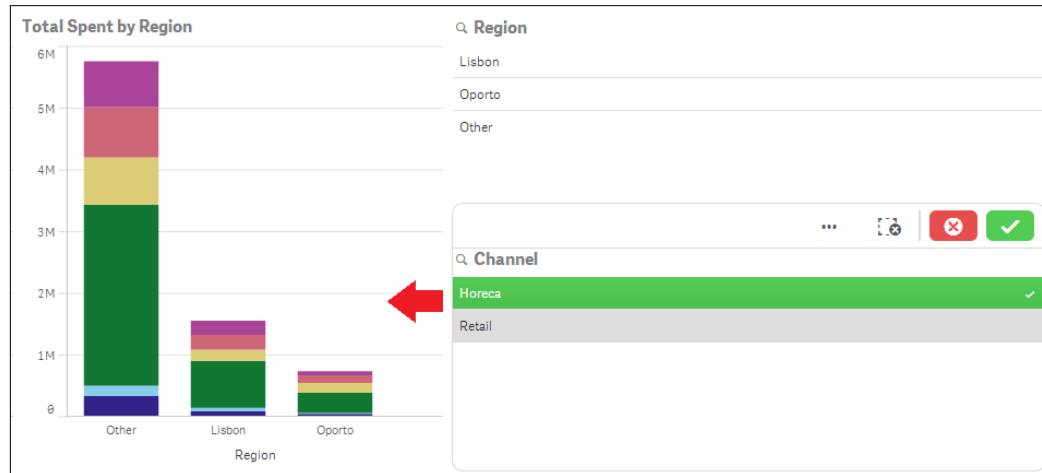
These charts tell us that 67.7 percent of our customers were **Horeca** customers and only 32.3 percent were **Retail** customers. On average, **Retail** customers spent more money on our products, but in absolute values, **Horeca** customers spent more money.

We have three regions – **Lisbon**, **Porto**, and **Others**. 17.5 percent of our customers were located in **Lisbon**, 10.7 percent in **Porto**, and 71.8 percent in **Other** regions. The average money spent is similar in all the regions, but the total spent is higher in **Other** regions than in **Lisbon** and **Porto**.

Now, we're going to use the filters to respond to more questions.

Analyzing your data

Our new application has two filters we can use to get a response to our questions. Select **Horeca** in the **Channel** filters. The application responds by actualizing the data. Now everything you see is related to the **Horeca** customers. Use the green or red buttons to confirm or cancel your selection. This is depicted in the following screenshot:



In the following screenshot, I've selected **Horeca** and **Porto**, and in the **Top Customers** table, I can see the top 10 **Horeca** customers in **Porto**. Now you can use the filters and the visualizations we've created to answer your own questions:

| Top customers | | Total Spent |
|---------------|---|-------------|
| Customer | Q | |
| Totals | | 719150 |
| 326 | | 130877 |
| 312 | | 43784 |
| 325 | | 35629 |
| 295 | | 33410 |
| 329 | | 32124 |
| 333 | | 31972 |
| 324 | | 29064 |
| 297 | | 28687 |
| 323 | | 25311 |
| Others | | 328292 |

Region

Oporto 

Lisbon

Other

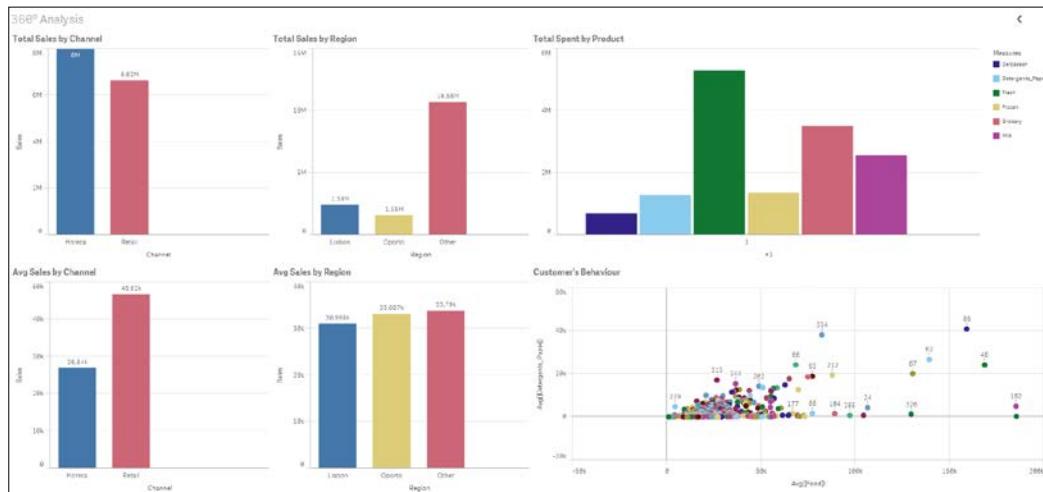
Channel

Horeca 

Retail

[ You can filter the data using the filters we've put in the application or you can filter the data by clicking on the charts.]

Finally, I've created a new sheet called **360° Analysis**. In this last sheet, we analyze the customer average money spent and the total money spent in the two different sales channels, the three regions, and the six different product categories. The following screenshot represents all this:



In the bottom-right, you can see a scatter plot. In this chart, each point represents a customer. The *y* axis represents the money spent on **Detergents_Paper** and the *x* axis represents the money spent on **Food**. At the beginning of this chapter, we created the field **Food** to be able to represent the customer's behavior on a plane.

Now try to select the **Retail** customers directly on a chart. All of the charts will update automatically to represent only **Retail** customers.

Finally, cancel your selection and select a single customer. On the **Total Spent by Product** bar chart, you can see the money spent by this customer in each category.

Now it's your turn to try to create new charts with Qlik Sense and explore your customers' characteristics even more.

Further learning

In this chapter, we've learned three main things about Qlik Sense – how to load and transform data, how to make selections to filter data, and how to create basic visualizations.

To find out more information about these features, I suggest you go through this document created by Michael Tarallo on the Qlik Community – <https://community.qlik.com/docs/DOC-6932>.

For data loading, you will find a section in the previous document called *Data Loading & Modeling*, but we especially like *Power of Qlik Script*, a series of three videos. There is a special video to learn how associative logic works, called *Working with Selections*, and in the *Apps & Visualizations* sections, you'll find videos that explain how to create data visualizations.

Qlik Community is a very active users' community around the Qlik platform. You will find a lot of resources related to Qlik Sense on this site. I strongly recommend you to register with *Qlik Community*.

Summary

In this chapter, we saw how to use Qlik Sense to create an application that helps us to analyze our customer data. We used a simple dataset with just 440 customers that we downloaded from the University of California website mentioned earlier. The dataset contained only two dimensions—channel and region—and six measures—**Fresh, Frozen, Milk, Grocery, Delicatessen, and Detergents_Paper**.

We learned how to load a dataset from a CSV file. We created a data model and uploaded additional tables created in a spreadsheet editor. Finally, we saw that we can create additional fields in a table using the spreadsheet editor or the data load editor. We also saw that the data load editor provides a lot of control over the data.

After creating the data model, we learned what associative logic is and how a business user can slice and dice his data using it.

Finally, we learned how to create basic visualizations using Qlik Sense, and we created our first Qlik Sense app.

In the next chapter, we'll use Rattle to segment the customers automatically based on their annual money spent in the different product categories. This segmentation will show us the data in a different light.

In *Chapter 8, Visualizations, Data Applications, Dashboards, and Data Storytelling*, we'll improve our knowledge of Qlik Sense and learn to create useful and attractive visualizations.

5

Clustering and Other Unsupervised Learning Methods

In this chapter, we will:

- Define machine learning
- Introduce unsupervised and supervised methods
- Focus on **K-means**, a classic machine learning algorithm, in detail

We'll use K-means to improve the application we created in *Chapter 4, Creating Your First Qlik Sense Application*. In *Chapter 4, Creating Your First Qlik Sense Application*, we created a Qlik Sense application to understand our customers' behavior. In this chapter, we'll create clusters of customers based on their annual money spent. This will give us a new insight. Being able to group our customers based on their annual money spent will allow us to see the profitability of each customer group and deliver more profitable marketing campaigns or create tailored discounts.

Finally, we'll see hierarchical clustering, different clustering methods, and association rules. Association rules are generally used for market basket analysis.

Machine learning – unsupervised and supervised learning

Machine Learning (ML) is a set of techniques and algorithms that gives computers the ability to learn. These techniques are generic and can be used in various fields. Data mining uses ML techniques to create insights and predictions from data.

In data mining, we usually divide ML methods into two main groups – supervised learning and unsupervised learning. A computer can learn with the help of a teacher (supervised learning) or can discover new knowledge without the assistance of a teacher (unsupervised learning).

In **supervised learning**, the learner is trained with a set of examples (dataset) that contains the right answer; we call it the **training dataset**. We call the dataset that contains the answers a **labeled dataset**, because each observation is labeled with its answer. In supervised learning, you are supervising the computer, giving it the right answers. For example, a bank can try to predict the borrower's chance of defaulting on credit loans based on the experience of past credit loans. The training dataset would contain data from past credit loans, including if the borrower was a defaulter or not.

In **unsupervised learning**, our dataset doesn't have the right answers and the learner tries to discover hidden patterns in the data. In this way, we call it unsupervised learning because we're not supervising the computer by giving it the right answers. A classic example is trying to create a classification of customers. The model tries to discover similarities between customers.

In some machine learning problems, we don't have a dataset that contains past observations. These datasets are not labeled with the correct answers and we call them **unlabeled datasets**.

In traditional data mining, the terms **descriptive analytics** and **predictive analytics** are used for unsupervised learning and supervised learning.

In unsupervised learning, there is no target variable. The objective of unsupervised learning or descriptive analytics is to discover the hidden structure of data. There are two main unsupervised learning techniques offered by Rattle:

- Cluster analysis
- Association analysis

Cluster analysis

Sometimes, we have a group of observations and we need to split it into a number of subsets of similar observations. **Cluster analysis** is a group of techniques that will help you to discover these similarities between observations.

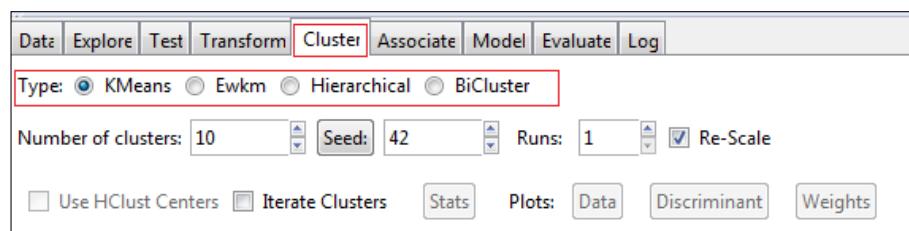
Market segmentation is an example of cluster analysis. You can use cluster analysis when you have a lot of customers and you want to divide them into different market segments, but you don't know how to create these segments.

Remember the application we developed in *Chapter 4, Creating Your First Qlik Sense Application?* We started with a dataset containing 440 customers; each observation contained the money the amount of customer spent in six different product categories. We used Qlik Sense to create an application that helps us to understand our customers. Sometimes, especially with a large amount of customers, we need some help to understand our data. Clustering can help us to create different customer groups based on their buying behavior.

In Rattle's **Cluster** tab, there are four cluster algorithms:

- **KMeans**
- **EwKm**
- **Hierarchical**
- **BiCluster**

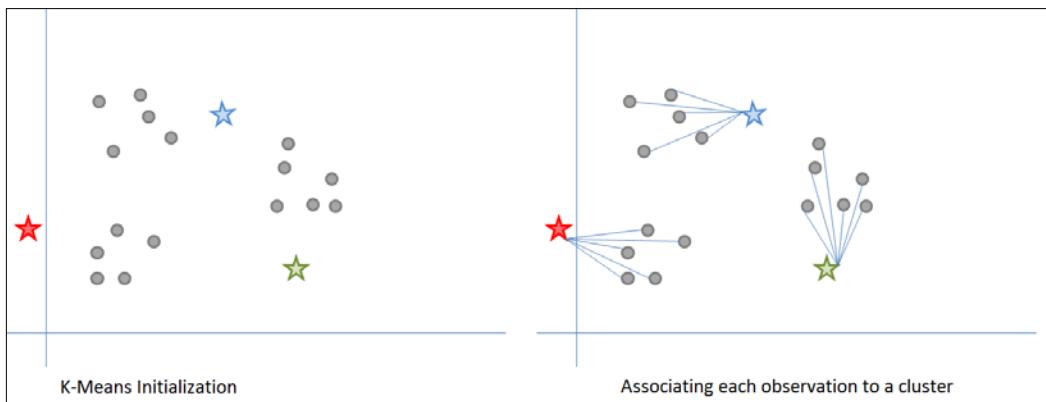
The two most popular families of cluster algorithms are hierarchical clustering and centroid-based clustering:



Centroid-based clustering the using K-means algorithm

I'm going to use K-means as an example of this family because it is the most popular.

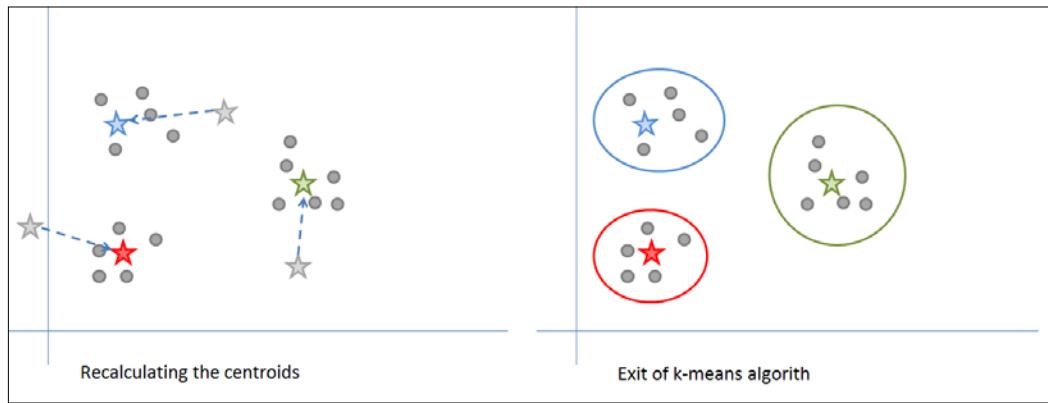
With this algorithm, a cluster is represented by a point or center called the **centroid**. In the initialization step of K-means, we need to create k number of centroids; usually, the centroids are initialized randomly. In the following diagram, the observations or objects are represented with a point and three centroids are represented with three colored stars:



After this initialization step, the algorithm enters into an iteration with two operations. The computer associates each object with the nearest centroid, creating k clusters. Now, the computer has to recalculate the centroids' position. The new position is the mean of each attribute of every cluster member.

This example is very simple, but in real life, when the algorithm associates the observations with the new centroids, some observations move from one cluster to the other.

The algorithm iterates by recalculating centroids and assigning observations to each cluster until some finalization condition is reached, as shown in this diagram:



The inputs of a K-means algorithm are the observations and the number of clusters, k . The final result of a K-means algorithm are k centroids that represent each cluster and the observations associated with each cluster.

The drawbacks of this technique are:

- You need to know or decide the number of clusters, k .
- The result of the algorithm has a big dependence on k .
- The result of the algorithm depends on where the centroids are initialized.
- There is no guarantee that the result is the optimum result. The algorithm can iterate around a local optimum.

In order to avoid a local optimum, you can run the algorithm many times, starting with different centroids' positions. To compare the different runs, you can use the cluster's distortion – the sum of the squared distances between each observation and its centroids.

Customer segmentation with K-means clustering

We're going to use the wholesale customer dataset we downloaded from the Center for Machine Learning and Intelligent Systems at the University of California, Irvine, in *Chapter 4, Creating Your First Qlik Sense Application*. You can download the dataset from here - <https://archive.ics.uci.edu/ml/datasets/Wholesale+customers#>.

As we saw in *Chapter 4, Creating Your First Qlik Sense Application*, the dataset contains 440 customers (observations) of a wholesale distributor. It includes the annual spend in monetary units on six product categories – **Fresh**, **Milk**, **Grocery**, **Frozen**, **Detergents_Paper**, and **Delicatessen**. We've created a new field called **Food** that includes all categories except **Detergents_Paper**, as shown in the following screenshot:

| | A | B | C | D | E | F | G | H | I |
|---|---------|--------|-------|------|---------|--------|-----------|-----------|-------|
| 1 | Channel | Region | Fresh | Milk | Grocery | Frozen | Detergent | Delicasse | Food |
| 2 | 2 | 3 | 12669 | 9656 | 7561 | 214 | 2674 | 1338 | 31438 |
| 3 | 2 | 3 | 7057 | 9810 | 9568 | 1762 | 3293 | 1776 | 29973 |
| 4 | 2 | 3 | 6353 | 8808 | 7684 | 2405 | 3516 | 7844 | 33094 |
| 5 | 1 | 3 | 13265 | 1196 | 4221 | 6404 | 507 | 1788 | 26874 |
| 6 | 2 | 3 | 22615 | 5410 | 7198 | 3915 | 1777 | 5185 | 44323 |
| 7 | 2 | 3 | 9413 | 8259 | 5126 | 666 | 1795 | 1451 | 24915 |

Load the new dataset into Rattle and go to the **Cluster** tab. Remember that, in unsupervised learning, there is no target variable.

I want to create a segmentation based only on buying behavior; for this reason, I set **Region** and **Channel** to **Ignore**, as shown here:

| No. Variable | Data Type Input | Target | Risk | Ident | Ignore | Weight | Comment |
|--------------------|-----------------|----------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-------------|
| 1 Channel | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | Unique: 2 |
| 2 Region | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | Unique: 3 |
| 3 Fresh | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 433 |
| 4 Milk | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 421 |
| 5 Grocery | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 430 |
| 6 Frozen | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 426 |
| 7 Detergents_Paper | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 417 |
| 8 Delicassen | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 403 |
| 9 Food | Numeric | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | Unique: 437 |

In the following screenshot, you can see the options Rattle offers for K-means. The most important one is **Number of clusters**; as we've seen, the analyst has to decide the number of clusters before running K-means:

Type: KMeans Ewkm Hierarchical BiCluster

Number of clusters: 10 Seed: 42 Runs: 1 Re-Scale

Use HClust Centers Iterate Clusters Stats Plots: Data Discriminant Weights

KMeans Clustering

A cluster analysis will identify groups within a dataset. The KMeans clustering algorithm will search for K clusters (which you specify). The resulting K clusters are represented by the mean or average values of each of the variables.

By default KMeans only works with numeric variables.

We have also seen that the initial position of the centroids can have some influence on the result of the algorithm. The position of the centroids is random, but we need to be able to reproduce the same experiment multiple times. When we're creating a model with K-means, we'll iteratively re-run the algorithm, tuning some options in order to improve the performance of the model. In this case, we need to be able to reproduce exactly the same experiment. Under the hood, R has a pseudo-random number generator based on a starting point called **Seed**. If you want to reproduce the exact same experiment, you need to re-run the algorithm using the same **Seed**.

Sometimes, the performance of K-means depends on the initial position of the centroids. For this reason, sometimes you need to able to re-run the model using a different initial position for the centroids. To run the model with different initial positions, you need to run with a different **Seed**.

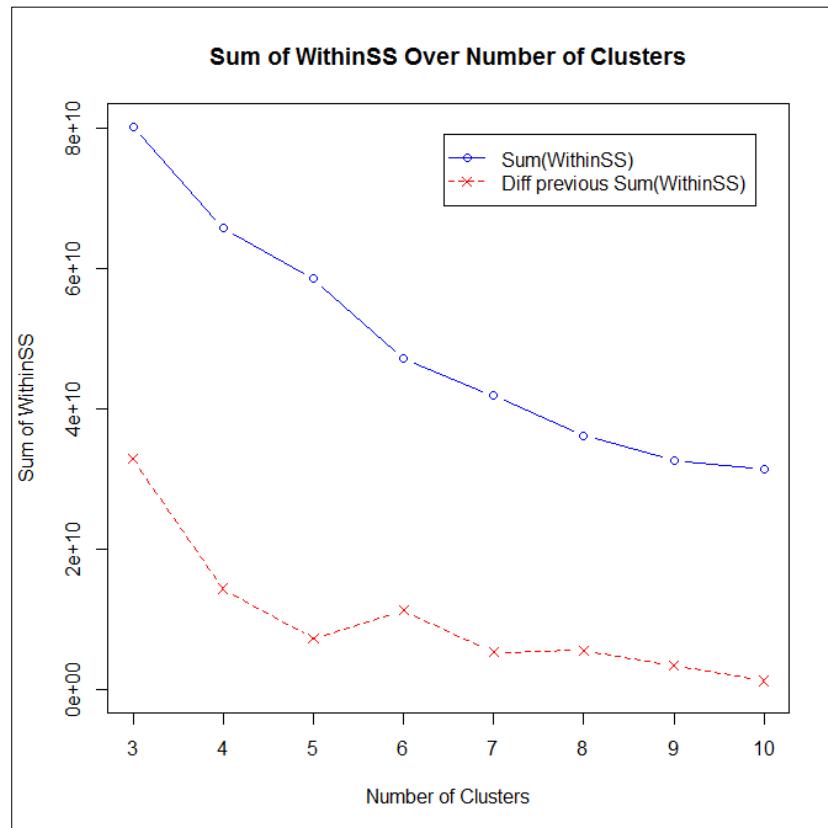
After executing the model, Rattle will show some interesting information. The size of each cluster, the means of the variables in the dataset, the centroid's position, and the **Within cluster sum of squares** value. This measure, also called distortion, is the sum of the squared differences between each point and its centroid. It's a measure of the quality of the model.

Another interesting option is **Runs**; by using this option, Rattle will run the model the specified number of times and will choose the model with the best performance based on the **Within cluster sum of squares** value.

Deciding on the number of clusters can be difficult. To choose the number of clusters, we need a way to evaluate the performance of the algorithm. The sum of the squared distance between the observations and the associated centroid could be a performance measure. Each time we add a centroid to **KMeans**, the sum of the squared difference between the observations and the centroids decreases. The difference in this measure using a different number of centroids is the gain associated to the added centroids. Rattle provides an option to automate this test, called **Iterative Clusters**.

If you set the **Number of clusters** value to **10** and check the **Iterate Clusters** option, Rattle will run **KMeans** iteratively, starting with **3** clusters and finishing with **10** clusters. To compare each iteration, Rattle provides an iteration plot. In the iteration plot, the blue line shows the sum of the squared differences between each observation and its centroid. The red line shows the difference between the current sum of squared distances and the sum of the squared distance of the previous iteration. For example, for four clusters, the red line has a very low value; this is because the difference between the sum of the squared differences with three clusters and with four clusters is very small. In the following screenshot, the peak in the red line suggests that six clusters could be a good choice.

This is because there is an important drop in the **Sum of WithinSS** value at this point:



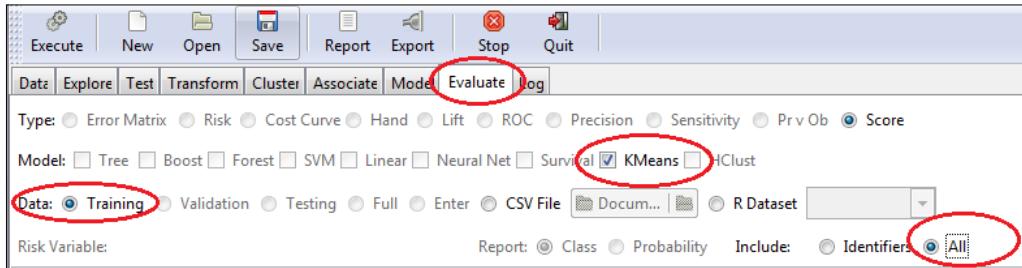
In this way, to finish my model, I only need to set the **Number of clusters** to 6, uncheck the **Re-Scale** checkbox, and click on the **Execute** button:

| |
|--|
| Type: <input checked="" type="radio"/> KMeans <input type="radio"/> Ewkm <input type="radio"/> Hierarchical <input type="radio"/> BiCluster |
| Number of clusters: <input type="text" value="6"/> <input type="button" value="▲"/> <input type="button" value="▼"/> Seed: <input type="text" value="42"/> <input type="button" value="▲"/> <input type="button" value="▼"/> Runs: <input type="text" value="10"/> <input type="button" value="▲"/> <input type="button" value="▼"/> <input type="checkbox"/> Re-Scale |
| <input type="checkbox"/> Use HClust Centers <input type="checkbox"/> Iterate Clusters <input type="button" value="Stats"/> <input type="button" value="Plots: Data"/> <input type="button" value="Discriminant"/> <input type="button" value="Weights"/> |

Finally, Rattle returns the six centroids of my clusters:

| Cluster centers: | | | | | | | | | | |
|------------------|-----------|-----------|-----------|-----------|------------|------------|------------|--|--|--|
| | Fresh | Milk | Grocery | Frozen | Detergents | Paper | Delicassen | | | |
| 1 | 46518.000 | 3438.682 | 4785.091 | 5249.955 | | 801.0455 | 2147.227 | | | |
| 2 | 20996.317 | 3826.433 | 5098.087 | 4157.010 | | 1119.9423 | 1679.712 | | | |
| 3 | 20031.286 | 38084.000 | 56126.143 | 2564.571 | | 27644.5714 | 2548.143 | | | |
| 4 | 60571.667 | 30120.333 | 17314.667 | 38049.333 | | 2153.0000 | 20700.667 | | | |
| 5 | 5996.482 | 3368.597 | 4206.765 | 2418.283 | | 1282.9469 | 1001.305 | | | |
| 6 | 5076.654 | 12288.526 | 18814.526 | 1605.000 | | 8254.3974 | 1830.513 | | | |

Now we have the six centroids and we want Rattle to associate each observation with a centroid. Go to the **Evaluate** tab, select the **KMeans** option, select the **Training** dataset, mark **All** in the report type, and click on the **Execute** button as shown in the following screenshot. This process will generate a CSV file with the original dataset and a new column called **kmeans**. The content of this attribute is a label (a number) representing the cluster associated with the observation (customer), as shown in the following screenshot:



After clicking on the **Execute** button, you will need to choose a folder to save the resulting file to and will have to type in a filename. The generated data inside the CSV file will look similar to the following screenshot:

| Customer_ID | Channel | Region | Fresh | Milk | Grocery | Frozen | Detergents | Paper | Delicassen | kmeans |
|-------------|---------|--------|-------|-------|---------|--------|------------|-------|------------|--------|
| 1 | 2 | 3 | 12669 | 9656 | 7561 | 214 | | 2674 | 1338 | 5 |
| 2 | 2 | 3 | 7057 | 9810 | 9568 | 1762 | | 3293 | 1776 | 5 |
| 3 | 2 | 3 | 6353 | 8808 | 7684 | 2405 | | 3516 | 7844 | 5 |
| 4 | 1 | 3 | 13265 | 1196 | 4221 | 6404 | | 507 | 1788 | 2 |
| 5 | 2 | 3 | 22615 | 5410 | 7198 | 3915 | | 1777 | 5185 | 2 |
| 6 | 2 | 3 | 9413 | 8259 | 5126 | 666 | | 1795 | 1451 | 5 |
| 7 | 2 | 3 | 12126 | 3199 | 6975 | 480 | | 3140 | 545 | 5 |
| 8 | 2 | 3 | 7579 | 4956 | 9426 | 1669 | | 3321 | 2566 | 5 |
| 9 | 1 | 3 | 5963 | 3648 | 6192 | 425 | | 1716 | 750 | 5 |
| 10 | 2 | 3 | 6006 | 11093 | 18881 | 1159 | | 7425 | 2098 | 6 |

In the previous screenshot, you can see ten lines of the resulting file; note that the last column is **kmeans**.

Preparing the data in Qlik Sense

Our objective is to create the same data model that we created in *Chapter 4, Creating Your First Qlik Sense Application*, but using the new CSV file with the kmeans column. We have two options – we can reproduce all the steps we performed in the previous chapter to create a new application with the new file or we can update the Qlik Sense application we developed in the previous chapter.

We're going to update our application by replacing the customer data file with this new data file. Save the new file in the same folder as the original file, open the Qlik Sense application, and go to **Data load editor**.

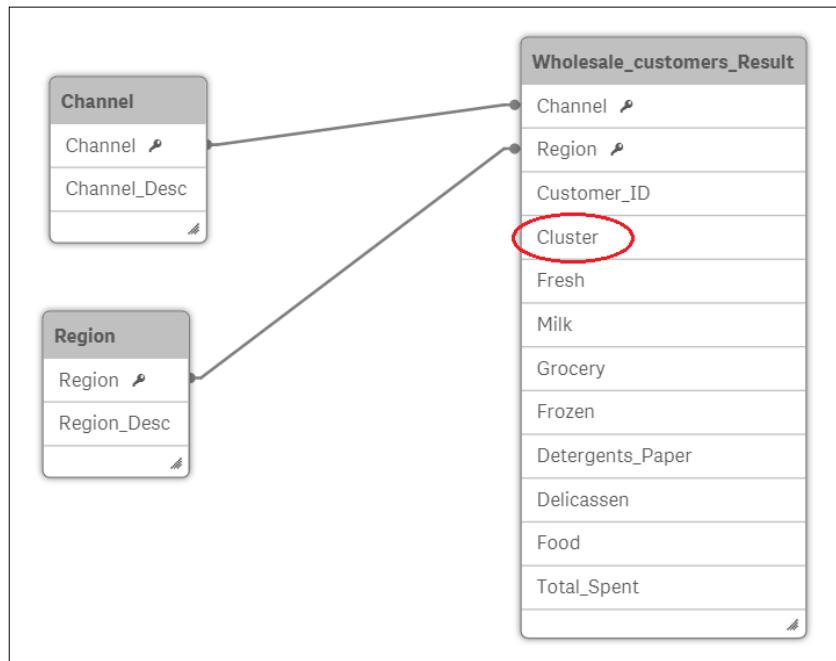
There are two differences between the original file and this one. In the original file, we added a line to create a customer identifier called `Customer_ID`, and in this second file we have this field in the dataset. The second difference is that in this new file we have the `kmeans` column.

From **Data load editor**, go to the **Wholesale customer data** sheet, modify line 2, and add line 3. In line 2, we just load the content of `Customer_ID`, and in line 3, we load the content of the `kmeans` field and rename it to **Cluster**, as shown in the following screenshot. Finally, update the name of the file to be the new one and click on the **Load data** button:

```

LOAD
1 Customer_ID,
2 kmeans as Cluster
3 Channel,
4 Region,
5 Fresh,
6 Milk,
7 Grocery,
8 Frozen,
9 Detergents_Paper,
10 Delicassen,
11 as Food,
12 Fresh + Milk + Grocery + Frozen + Delicassen
13 as Total_Spent
14 FROM [lib://Wholesale customers/Wholesale_customers_Results.csv]
15 (txt, codepage is 1252, embedded labels, delimiter is ',', header)
  
```

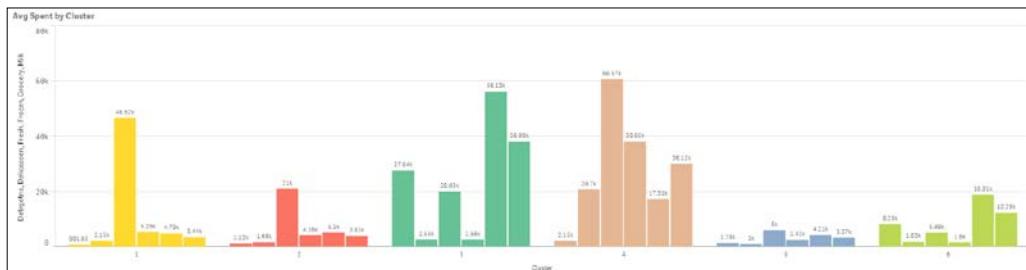
When the data load process finishes, open the data model viewer to check your data model, as shown here:



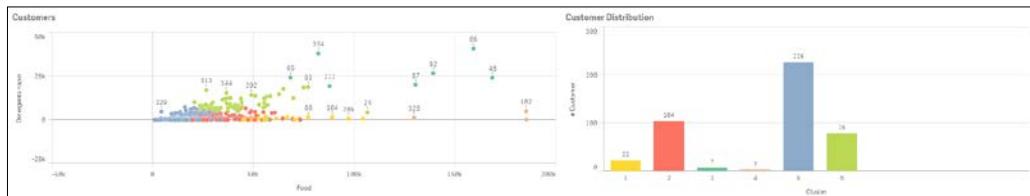
Note that you have the same data model with a new field called **Cluster**.

Creating a customer segmentation sheet in Qlik Sense

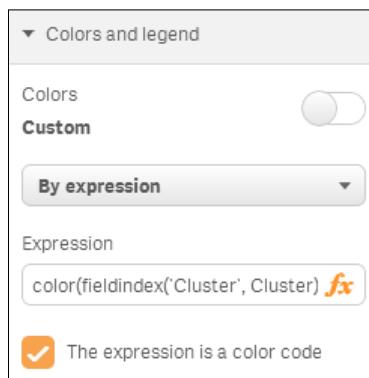
Now we can add a sheet to the application we developed in *Chapter 4, Creating Your First Qlik Sense Application*. We'll add three charts to see our clusters and how our customers are distributed in our clusters. The first chart will describe the buying behavior of each cluster, as shown here:



The second chart will show all customers distributed in a scatter plot, and in the last chart we'll see the number of customers that belong to each cluster, as shown here:



I'll start with the chart to the bottom-right; it's a bar chart with `Cluster` as the dimension and `Count ([Customer_ID])` as the measure. This simple bar chart has something special – colors. Each customer's cluster has a special color code that we use in all charts. In this way, cluster 5 is blue in the three charts. To obtain this effect, we use this expression to define the color as `color(fieldindex('Cluster', Cluster))`, which is shown in the following screenshot:



You can find this color trick and more in this interesting blog by Rob Wunderlich - <http://qlikviewcookbook.com/>.

My second chart is the one at the top. I copied the previous chart and pasted it onto a free place. I kept the dimension but I changed the measure by using six new measures:

- Avg ([Detergents_Paper])
- Avg ([Delicassen])
- Avg ([Fresh])
- Avg ([Frozen])
- Avg ([Grocery])
- Avg ([Milk])

I placed my last chart at the bottom-left. I used a scatter plot to represent all of my 440 customers. I wanted to show the money spent by each customer on food and detergents, and its cluster. I used the y axis to show the money spent on detergents and the x axis for the money spent on food. Finally, I used colors to highlight the cluster. The dimension is Customer_Id and the measures are Delicassen+Fresh+Frozen+Grocery+Milk (or Food) and [Detergents_Paper]. As the final step, I reused the color expression from the earlier charts.

Now our first Qlik Sense application has two sheets – the original one is 100 percent Qlik Sense and helps us to understand our customers, channels, and regions. This new sheet uses clustering to give us a different point of view; this second sheet groups the customers by their similar buying behavior. All this information is useful to deliver better campaigns to our customers. Cluster 5 is our least profitable cluster, but is the biggest one with 227 customers. The main difference between cluster 5 and cluster 2 is the amount of money spent on fresh products. Can we deliver any offer to customers in cluster 5 to try to sell more fresh products?

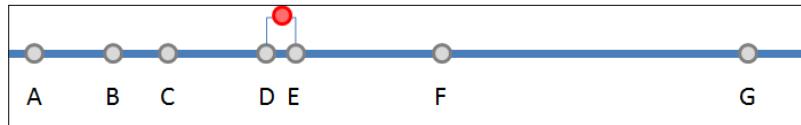
Select retail customers and ask yourself, who are our best retail customers? To which cluster do they belong? Are they buying all our product categories?

Hierarchical clustering

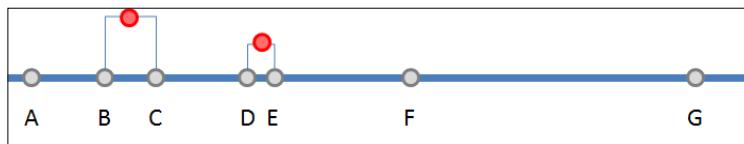
Hierarchical clustering tries to group objects based on their similarity. To explain how this algorithm works, we're going to start with seven points (or observations) lying in a straight line:



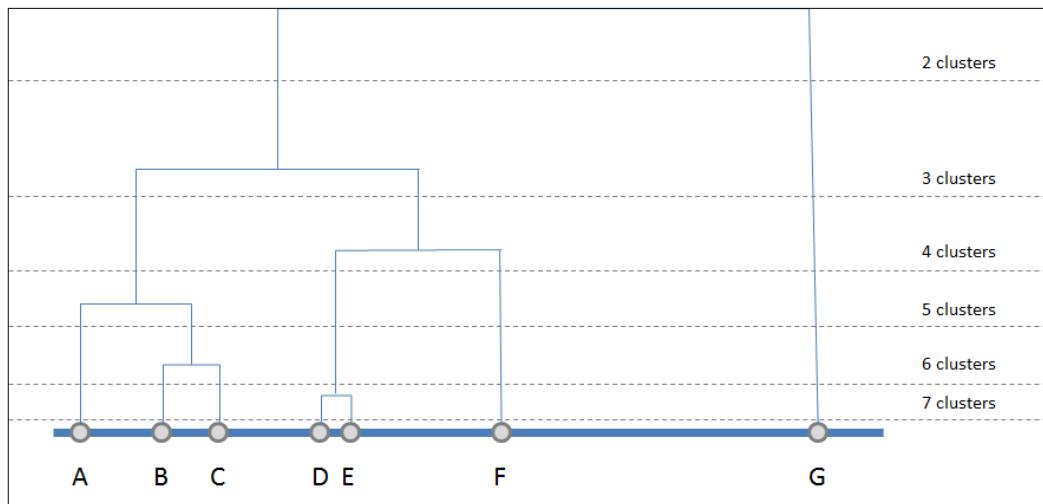
We start by calculating the **distance** between each point. I'll come back later to the term distance; in this example, distance is the difference between two positions in the line. The points **D** and **E** are the ones with the smallest distance in between, so we group them in a cluster, as shown in this diagram:



Now, we substitute point **D** and point **E** for their mean (red point) and we look for the two points with the next smallest distance in between. In this second iteration, the closest points are **B** and **C**, as shown in this diagram:



We continue iterating until we've grouped all observations in the dataset, as shown here:

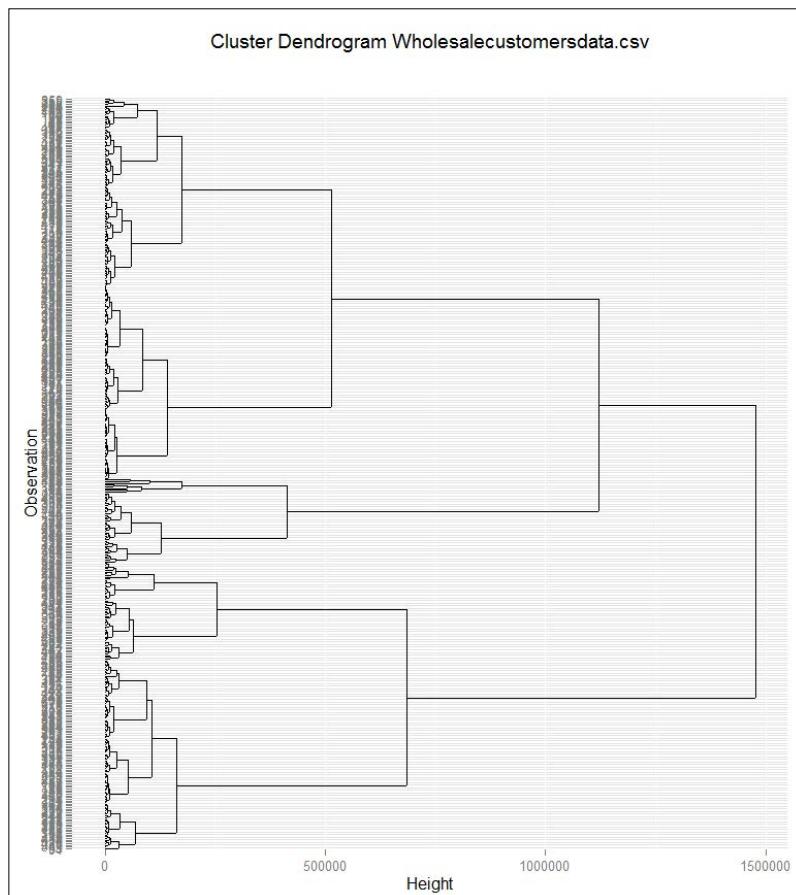


Note that, in this algorithm, we can decide on the number of clusters after running the algorithm. If we divide the dataset into two clusters, the first cluster is point **G** and the second cluster is **A, B, C, D, E, and F**. This gives the analyst the opportunity to see the big picture before deciding on the number of clusters.

The lowest level of clustering is a trivial one; in this example, seven clusters with one point in each one.

The chart I've created while explaining the algorithm is a basic form of a dendrogram. The **dendrogram** is a tree diagram used in Rattle and in other tools to illustrate the layout of the clusters produced by hierarchical clustering.

In the following screenshot, we can see the dendrogram created by Rattle for the wholesale customer dataset. In Rattle's dendrogram, the *y* axis represent all observations or customers in the dataset, and the *x* axis represents the distance between the clusters:



Association analysis

Association rules or association analysis is also an important topic in data mining. This is an unsupervised method, so we start with an unlabeled dataset. An **unlabeled dataset** is a dataset without a variable that gives us the right answer. Association analysis attempts to find relationships between different entities. The classic example of association rules is market basket analysis. This means using a database of transactions in a supermarket to find items that are bought together. For example, a person who buys potatoes and burgers usually buys beer. This insight could be used to optimize the supermarket layout.

Online stores are also a good example of association analysis. They usually suggest to you a new item based on the items you have bought. They analyze online transactions to find patterns in the buyer's behavior.

These algorithms assume all variables are categorical; they perform poorly with numeric variables. Association methods need a lot of time to be completed; they use a lot of CPU and memory. Remember that Rattle runs on R and the R engine loads all data into RAM memory.

Suppose we have a dataset such as the following:

| | | | | | |
|---------------|---------|----------|----------|---------|----------|
| Transaction 1 | Burger | Chicken | Potatoes | | |
| Transaction 2 | Burger | Onions | | | |
| Transaction 3 | Onions | Boots | | | |
| Transaction 4 | Burger | Onions | Chicken | | |
| Transaction 5 | Burger | Onions | Chicken | Clothes | Potatoes |
| Transaction 6 | Chicken | Clothes | Potatoes | | |
| Transaction 7 | Chicken | Potatoes | Clothes | | |

Our objective is to discover items that are purchased together. We'll create rules and we'll represent these rules like this:

Chicken, Potatoes → Clothes

This rule means that when a customer buys **Chicken** and **Potatoes**, he tends to buy **Clothes**.

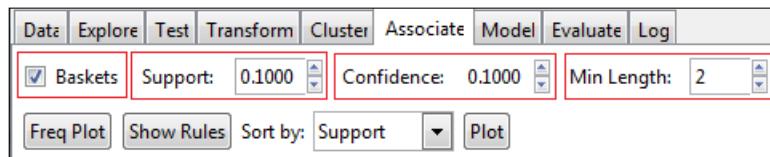
As we'll see, the output of the model will be a set of rules. We need a way to evaluate the quality or interest of a rule. There are different measures, but we'll use only a few of them. Rattle provides three measures:

- **Support**
- **Confidence**
- **Lift**

Support indicates how often the rule appears in the whole dataset. In our dataset, the rule Chicken, Potatoes → Clothes has a support of 48.57 percent (3 occurrences / 7 transactions).

Confidence measures how strong rules or associations are between items. In this dataset, the rule Chicken, Potatoes → Clothes has a confidence of 1. The items Chicken and Potatoes appear three times in the dataset and the items Chicken, Potatoes, and Clothes appear three times in the dataset; and $3/3 = 1$. A confidence close to 1 indicates a strong association.

In the following screenshot, I've highlighted the options on the **Associate** tab we have to choose from before executing an association method in Rattle:



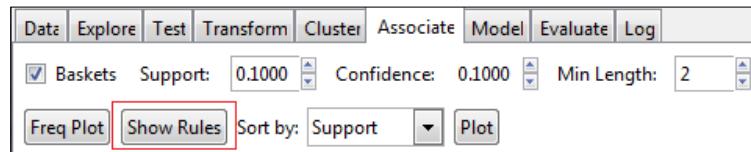
The first option is the **Baskets** checkbox. Depending on the kind of input data, we'll decide whether or not to check this option. If the option is checked, such as in the preceding screenshot, Rattle needs an identification variable and a target variable. After this example, we'll try another example without this option.

The second option is the minimum **Support** value; by default, it is set to 0.1. Rattle will not return rules with a lower **Support** value than the one you have set in this text box. If you choose a higher value, Rattle will only return rules that appear many times in your dataset. If you choose a lower value, Rattle will return rules that appear in your dataset only a few times. Usually, if you set a high value for **Support**, the system will return only the obvious relationships. I suggest you start with a high **Support** value and execute the methods many times with a lower value in each execution. In this way, in each execution, new rules will appear that you can analyze.

The third parameter you have to set is **Confidence**. This parameter tells you how strong the rule is.

Finally, the length is the number of items that contains a rule. A rule like Beer è Chips has length of two. The default option for **Min Length** is 2. If you set this variable to 2, Rattle will return all rules with two or more items in it.

After executing the model, you can see the rules created by Rattle by clicking on the **Show Rules** button, as illustrated here:



Rattle provides a very simple dataset to test the association rules in a file called `dvdtrans.csv`. Test the dataset to learn about association rules.

Further learning

In this chapter, we introduced supervised and unsupervised learning, the two main subgroups of machine learning algorithms; if you want to learn more about machine learning, I suggest you complete a MOOC course called *Machine Learning* at Coursera:

<https://www.coursera.org/learn/machine-learning>

The acronym **MOOC** stands for **Massive Open Online Course**; these are courses open to participation via the Internet. These courses are generally free. Coursera is one of the leading platforms for MOOC courses.

Machine Learning is a great course designed and taught by Andrew Ng, Associate Professor at Stanford University; Chief Scientist at Baidu; and Chairman and Co-founder at Coursera. This course is really interesting.

A very interesting book is *Machine Learning with R* by Brett Lantz, *Packt Publishing*. This book contains a very interesting chapter about clustering and K-means with R.

Summary

In this chapter, we were introduced to machine learning, and supervised and unsupervised methods. We focused on unsupervised methods and covered centroid-based clustering, hierarchical clustering, and association rules.

We completed the application we started in *Chapter 4, Creating Your First Qlik Sense Application*. We used a simple dataset, but we saw how a clustering algorithm can complement a 100 percent Qlik Sense approach by adding more information.

In the next chapter, we'll cover supervised methods and we'll use decision trees as an example.

6

Decision Trees and Other Supervised Learning Methods

In the previous chapter, we introduced Machine Learning, unsupervised methods, and supervised methods. We focused on unsupervised learning and described some algorithms, we also concentrated on classifiers. We took time to study cluster analysis, focusing on centroids-based algorithms, and we also looked at hierarchical clustering.

We used Rattle to process customer data in order to create different clusters of customers, and then, we used Qlik Sense to visualize these different clusters.

The objective of this chapter is to introduce you to supervised learning. As I explained in the previous chapter, in supervised learning, the computer analyzes a set of examples to *learn* how to predict the output of a new situation.

We'll focus on Decision Tree Learning, or Decision Trees, because they're widely used and the knowledge *learned* by the tree is easy to translate to rules in any software, such as Qlik Sense. These rules are easy to understand for human experts.

In supervised learning, we split the dataset into three datasets – training, validation, and test. The training dataset usually contains 70 percent of the original observations, our algorithm will use this dataset in the training phase to *learn by example*. Each of the validation and test datasets usually contains 15 percent of the original observations. We'll use the validation dataset to fine-tune our algorithm, and finally, after the fine-tuning, we'll use the test dataset to evaluate the final performance of our algorithm. These three datasets match with the three phases of a supervised algorithm – training, validation (or tuning), and test (or performance evaluation).

In this chapter:

- We'll describe the main concepts of Decision Tree Learning.
- We'll review the algorithm and the possible applications, and we'll see examples based on these algorithms.
- Then, we'll use Rattle and Qlik Sense to create an application to classify new loan applications into low risk applications and high risk applications. We'll load that data into Qlik Sense and create a few example visualizations.
- After Decision Trees, we'll look at ensemble methods and Supported Vector Machines.
- Finally, we'll look at Neural Networks, which can be used as supervised or unsupervised learning and statistics methods such as Regression or Survival Analysis.

Partitioning datasets and model optimization

As we've explained, in supervised learning, we split the dataset in three subsets – training, validation, and testing:



To create the model or learner, Rattle uses the training dataset. After creating a model, we use the validation data to evaluate its performance. To improve the performance, depending on the algorithm we're using, we can use different tuning options. After tuning, we rebuild the model and evaluate its performance again. This is an iterative process; we create the model and evaluate it until we're fine with its performance.

For simplicity, in this chapter, we'll see only model creation, and in the following chapter, we'll see model optimization, but in real life, this is an iterative process.

The examples in this chapter will not have any optimization.

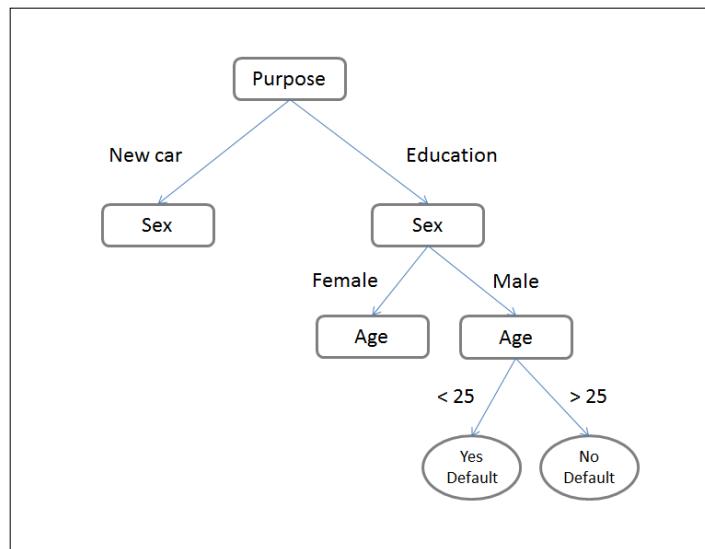
Finally, when you're happy with the model, you can use the testing dataset to confirm its performance. You need to use the testing dataset because you've used the validation dataset to optimize the model. You need to be sure that the optimizations you've done, work for all data, not just for the validation data.

Rattle splits the data randomly to assure that each dataset is representative, but when we optimize the model and test it again, we need to be able to repeat the same experiment exactly, with the same data. In this way, we'll be able to know if we're improving the model performance. To solve this problem, Rattle splits the dataset using a pseudo random number generator. Every time we split the dataset using the same **Seed**, we'll have the same subsets.

Decision Tree Learning

Decision Tree Learning uses past observations to learn how to classify them and also try to predict the class of a new observation. For example, in a bank, we may have historical information on the granting of loans. Usually, past loan information includes a customer profile and whether the customer defaulted or not. Based on this information, the algorithm can learn to predict whether a new customer will default.

We usually represent a Decision Tree as we did in the following diagram. The root node is at the top, and the leaves of the tree are at the bottom, the leaves represent a decision. In order to create rules from a tree, we need to start from the root node, and then we work downwards, towards the leaves. The following diagram represents a sample Decision Tree:



After studying the preceding diagram of a Decision Tree, we can obtain these rules:

```
If Purpose = 'Education' AND Sex = 'male' AND Age > 25 Then No  
Default  
If Purpose = 'Education' AND Sex = 'male' AND Age < 25 Then Yes  
Default
```

As you can see, a tree is easy to translate to a set of *rules* or *If then* sentences. This is very useful for calculation by Rattle, or any other language, or system, such as Qlik Sense.

Finally, a human expert can understand the rules and the knowledge learned by the algorithm; in this way, a credit manager can understand and review why a computer has classified a loan application as dangerous or not dangerous.

In short, the main advantages of Decision Tree Learning are:

- The technique is simple
- It requires little data preparation
- The result is simple to understand for a human expert
- It is easy to visually represent

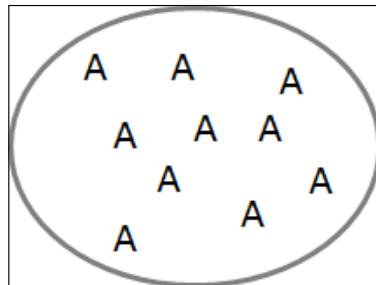
On the other hand, the main disadvantages are:

- **Unstable:** A little change in the input data can produce a big change in the output.
- **Overfitting:** Sometimes, Decision Tree Learners create very complex trees that do not generalize the data well. In other words, the algorithm learns how to classify the learning dataset, but fails to classify new observations.

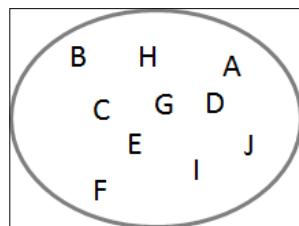
Entropy and information gain

Before we explain how to create a Decision Tree, we need to introduce two important concepts – entropy and information gain.

Entropy measures the homogeneity of a dataset. Imagine a dataset with 10 observations with one attribute, as shown in the following diagram, the value of this attribute is **A** for the 10 observations. This dataset is completely homogenous and is easy to predict the value of the next observation, it'll probably be **A**:

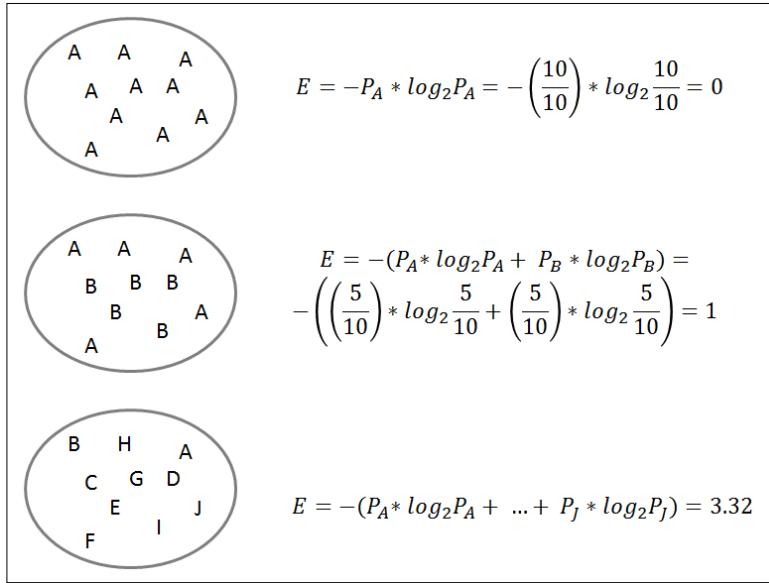


The entropy in a dataset that is completely homogenous is zero. Now, imagine a similar dataset, but in this dataset each observation has a different value, as shown in the following diagram:



Now, the dataset is very heterogeneous and it's hard to predict the following observation. In this dataset, the entropy is higher. The formula to calculate the entropy is $Entropy = -\sum_i P_x \log_2 P_x$, where P_x is the probability of x .

Try to calculate the entropy for the following datasets:



Now, we understand how entropy helps us to know the level of predictability of a dataset. A dataset with a low entropy level is very predictable; a dataset with a high level of entropy is very hard to predict. We're ready to understand information gain and how entropy and information gain can help us to create a Decision Tree.

The **information gain** is a measure of the decrease of entropy you achieve when you split a dataset. We use it in the process of building a Decision Tree. We're going to use an example to understand this concept. In this example, our objective will be to create a tree to classify loan applications depending on its probability of defaulting, into low risk applications and high risk applications. Our dataset has three input variables: Purpose, Sex, and Age, and one output variable, Default?.

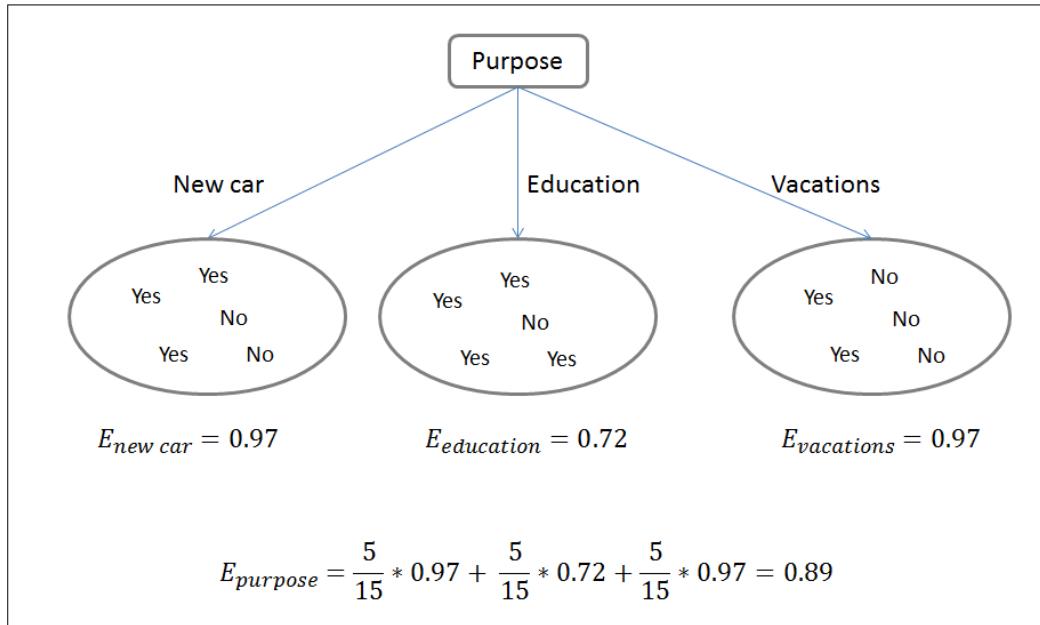
The following image shows the dataset:

| Purpose | Sex | Age | Default? |
|-----------|--------|---------|----------|
| New Car | Male | < 25 | Yes |
| New Car | Male | 25 - 65 | No |
| New Car | Female | < 25 | Yes |
| New Car | Female | 25 - 65 | Yes |
| New Car | Male | 25 - 65 | No |
| Education | Male | < 25 | Yes |
| Education | Male | < 25 | No |
| Education | Female | < 25 | Yes |
| Education | Female | < 25 | Yes |
| Education | Female | < 25 | Yes |
| Vacations | Female | < 25 | Yes |
| Vacations | Female | > 65 | No |
| Vacations | Female | < 25 | Yes |
| Vacations | Male | 25 - 65 | No |
| Vacations | Male | > 65 | No |

To create the Decision Tree, we will start by choosing an attribute for the root node. This attribute will split our dataset into two datasets. We will choose the attribute that adds more predictability or reduces the entropy. We will start calculating the entropy for the current dataset:

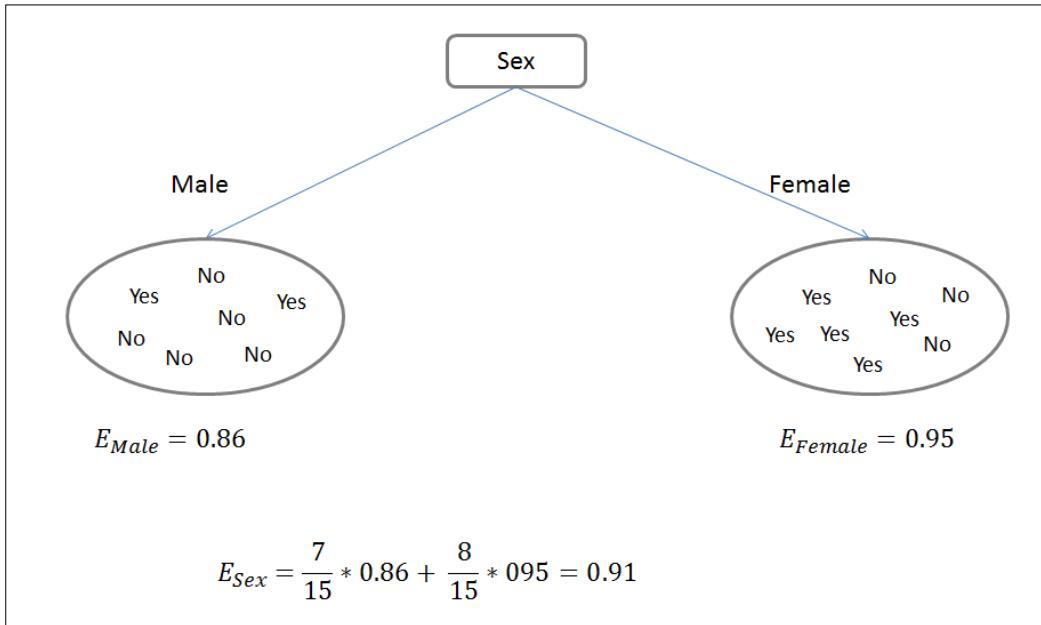
$$E = -(P_{yes} * \log_2 P_{yes} + P_{No} * \log_2 P_{No}) = -\left(\frac{9}{15} * \log_2 \frac{9}{15} + \frac{6}{15} * \log_2 \frac{6}{15}\right) = 0.97$$

We will start with an entropy of 0.97; our objective is to try to reduce the entropy to increase the predictability. What happens if we choose the attribute `Purpose` for our root node? By choosing `Purpose` for our root node, we will divide the dataset in three datasets. Each dataset contains five observations. We can calculate the entropy of each dataset and aggregate it to have a global entropy value.



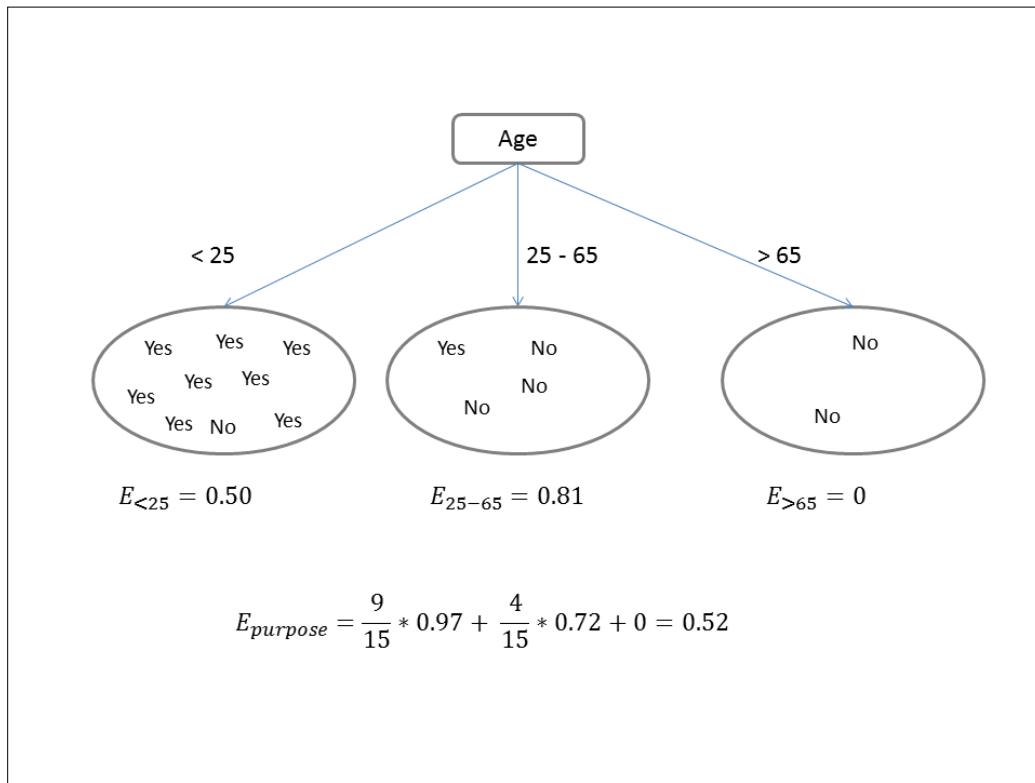
The original entropy was 0.97. If we use `Purpose` for our root node and divide the dataset into three sets, the entropy will be 0.89, so our new dataset will be more predictable. The difference between the original entropy and the new entropy is the information gain. In this example, the information gain is 0.08. However, what happens if we choose `Sex` or `Age` for our root node?

If we use `Sex` to split the dataset, we create two datasets. The `male` dataset contains seven observations and the `female` dataset contains eight observations; the new entropy is 0.91. In this case, the information gain is 0.06, so `Purpose` is a better option than `Sex` to split the dataset. Splitting the dataset by `Purpose`, the result becomes more predictable. This is illustrated in the following diagram:



Finally, if we use `Age` to split the dataset, we will obtain three subsets. The subset that contains young people (< 25) contains nine observations, the subset with middle-aged people contains four observations, and finally, the subset with people older than 65 years contains two observations. In this case, the entropy is 0.52 and the information gain is 0.45.

The attribute `Age` has the higher information gain; we will choose it for our root node, as illustrated in the following diagram:



We've divided our dataset into three subsets, divided by `Age`.

After the root node, we need to choose a second attribute to split our three datasets and create a deeper tree.

Underfitting and overfitting

Underfitting and overfitting are problems not just with a classifier but for all supervised methods.

Imagine you have a classifier with just one rule that tries to distinguish between healthy and not healthy patients. The rule is as follows:

If Temperature < 37 then Healthy

This classifier will classify all patients with a lower temperature than 37 degrees, as healthy. This classifier will have a huge error rate. The tree that represents this rule will have only the root node and two branches, with a leaf in each branch.

Underfitting occurs when the tree is too short to classify a new observation correctly; the rules are too general.

On the other hand, if we have a dataset with many attributes, and if we generate a very deep Decision Tree, we risk the fact that our Tree fits well with the training dataset, but not able to predict new examples. In our previous example, we can have a rule such as this:

```
If Temperature<27 and Sintom_A = V ..... and Sintom_B = Y ....Age=12  
and ... and Eyes = Blue and Height = 182 and Weight=74.6 then  
Healthy
```

In this case, the rule is too specific. What happens if the weight of the next patient is 76? The classifier will not be able to classify the new patient correctly. The Tree is too deep and the rules are too specific; this problem is called **overfitting**. We'll see a very low error rate on training data, but a high error rate on test data.

We'll come back to overfitting in the next chapter.

Using a Decision Tree to classify credit risks

In this section, we will create a model to classify credit risks. In this section, we will create the model; we won't look at the performance of the model. We'll evaluate the performance of the model and improve it in the next chapter.

As we did before, to create this example, we'll download a dataset from the UCI Machine Learning Repository. We'll use a dataset called *Statlog (German Credit Data) Dataset*. The source of the dataset is Professor Dr. Hans Hofmann from Institut für Statistik und Ökonometrie, Universität Hamburg. The dataset classifies people described by a set of attributes as good or bad credit risks.

The dataset is downloaded from the following link:

<https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29>

In the following screenshot, you can see the original form of this dataset. The screenshot shows us the top ten lines of the dataset. The dataset doesn't have a header line. It contains 20 attributes, and the last column is the target variable—1 for Good Credit and 2 for Bad Credit. The attributes are separated by a blank space, as shown in this screenshot:

```

1 A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 A121 67 A143 A152 2 A173 1 A192 A201 1
2 A12 48 A32 A43 5951 A61 A73 2 A92 A101 2 A121 22 A143 A152 1 A173 1 A191 A201 2
3 A14 12 A34 A46 2096 A61 A74 2 A93 A101 3 A121 49 A143 A152 1 A172 2 A191 A201 1
4 A11 42 A32 A42 7882 A61 A74 2 A93 A103 4 A122 45 A143 A153 1 A173 2 A191 A201 1
5 A11 24 A33 A40 4870 A61 A73 3 A93 A101 4 A124 53 A143 A153 2 A173 2 A191 A201 2
6 A14 36 A32 A46 9055 A65 A73 2 A93 A101 4 A124 35 A143 A153 1 A172 2 A192 A201 1
7 A14 24 A32 A42 2835 A63 A75 3 A93 A101 4 A122 53 A143 A152 1 A173 1 A191 A201 1
8 A12 36 A32 A41 6948 A61 A73 2 A93 A101 2 A123 35 A143 A151 1 A174 1 A192 A201 1
9 A14 12 A32 A43 3059 A64 A74 2 A91 A101 4 A121 61 A143 A152 1 A172 1 A191 A201 1
10 A12 30 A34 A40 5234 A61 A71 4 A94 A101 2 A123 28 A143 A152 2 A174 1 A191 A201 2

```

We prefer to work with a CSV file with a header line and the attributes separated by commas. For this reason, before loading the dataset into Rattle, we work it a little, with a spreadsheet editor, to transform the original file.

To label each column, we used the following information document provided with the dataset:

| Column | Label | Values |
|--------|-------------------------------------|--|
| 1 | Status of existing checking account | A11:... < 0 DM A12: 0 <= ... < 200 DM A13: ... >= 200 DM /salary assignments for at least 1 year A14: no checking account |
| 2 | Duration in months | Numeric |
| 3 | Credit history | A30: no credits taken/all credits paid back duly A31: all credits at this bank paid back duly A32: existing credits paid back duly till now A33: delay in paying off in the past A34: critical account/other credits existing (not at this bank) |

| Column | Label | Values |
|--------|---|---|
| 4 | Purpose | A40: car (new) A41: car (used) A42: furniture/equipment A43: radio/television A44: domestic appliances A45: repairs A46: education A47: (vacation - does not exist?) A48: retraining A49: business A410: others |
| 5 | Credit amount | Numeric |
| 6 | Savings account/bonds | A61: ... < 100 DM A62: 100 <= ... < 500 DM A63: 500 <= ... < 1000 DM A64: >= 1000 DM A65: unknown/ no savings account |
| 7 | Present employment since | A71: unemployed A72: ... < 1 year A73: 1 <= ... < 4 years A74: 4 <= ... < 7 years A75: .. >= 7 years |
| 8 | Installment rate in percentage of disposable income | Numeric |
| 9 | Personal status and sex | A91: male: divorced/separated A92: female: divorced/separated/married A93: male: single A94: male: married/widowed A95: female: single |
| 10 | Other debtors/guarantors | A101: none A102: co-applicant A103: guarantor |

| Column | Label | Values |
|--------|--|--|
| 11 | Present residence since | Numeric |
| 12 | Property | A121: real estate A122: if not A121: building society savings agreement/life insurance A123: if not A121/A122: car or other, not in attribute 6 A124: unknown/no property |
| 13 | Age in years | Numeric |
| 14 | Other installment plans | A141: bank A142: stores A143: none |
| 15 | Housing | A151: rent A152: own A153: for free |
| 16 | Number of existing credits at this bank | Numeric |
| 17 | Job | A171: unemployed/unskilled - non-resident A172: unskilled - resident A173: skilled employee/official A174: management/self-employed/highly qualified employee/officer |
| 18 | Number of people being liable to provide maintenance for | Numeric |
| 19 | Telephone | A191: none A192: yes, registered under the customers name |
| 20 | foreign worker | A201: yes A202: no |
| 21 | Target | 1: Good 2: Bad |

To create our classifier, we will start by loading the data into Rattle and identifying the target variable. During the data load, we'll split the dataset into three datasets—the training dataset, the validation dataset, and the testing dataset. As we've explained in this chapter, we'll use the training dataset to create our model, the validation dataset to tune it, and the testing dataset to evaluate the final performance. We'll come back to this in the next chapter when we look at cross-validation. The following screenshot shows how to split the original dataset into three datasets:

The screenshot shows the Rattle software interface with the 'Partition' tab selected. The 'Source' dropdown is set to 'Spreadsheet'. The 'Filename' field contains 'GermanCreditCS...'. The 'Separator' is set to ';' and 'Decimal:' to '.'. The 'Header' checkbox is checked. The 'Partition' field is set to '70/15/15', and the 'Seed' is set to '42'. Below these settings is a table titled 'Target Data Type' with columns for 'No.', 'Variable', 'Data Type', 'Input', 'Target', 'Risk', 'Ident', 'Ignore', 'Weight', and 'Comment'. The 'Input' column has a radio button for 'Auto' (selected) and checkboxes for 'Categoric', 'Numeric', and 'Survival'. The 'Target' column has a radio button for 'Auto' (selected) and checkboxes for 'Categoric', 'Numeric', and 'Survival'. The 'Comment' column lists unique counts for each variable: Age.in.years (53), Credit.amount (921), Credit.history (5), Duration.in.month (33), foreign.worker (2), Housing (3), Installment.rate.in.percentage.of.disposable.income (4), Job (4), Number.of.existing.credits.at.this.bank (4), Number.of.people.being.liable.to.provide.maintenance.for (2), Other.debtors...guarantors (3), Other.installment.plans (3), Personal.status.and.sex (4), Present.employment.since (5), Present.residence.since (4), Property (4), Purpose (10), Savings.account.bonds (5), Status.of.existing.checking.account (4), Telephone (2), and Target (2).

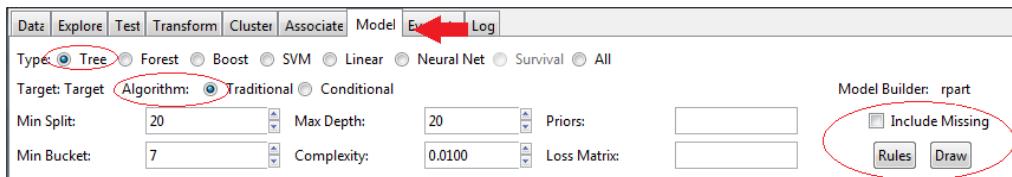
| No. | Variable | Data Type | Input | Target | Risk | Ident | Ignore | Weight | Comment |
|-----|--|-----------|----------------------------------|----------------------------------|-----------------------|-----------------------|-----------------------|-----------------------|-------------|
| 1 | Age.in.years | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 53 |
| 2 | Credit.amount | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 921 |
| 3 | Credit.history | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 5 |
| 4 | Duration.in.month | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 33 |
| 5 | foreign.worker | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 6 | Housing | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 |
| 7 | Installment.rate.in.percentage.of.disposable.income | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 8 | Job | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 9 | Number.of.existing.credits.at.this.bank | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 10 | Number.of.people.being.liable.to.provide.maintenance.for | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 11 | Other.debtors...guarantors | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 |
| 12 | Other.installment.plans | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 3 |
| 13 | Personal.status.and.sex | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 14 | Present.employment.since | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 5 |
| 15 | Present.residence.since | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 16 | Property | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 17 | Purpose | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 10 |
| 18 | Savings.account.bonds | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 5 |
| 19 | Status.of.existing.checking.account | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 4 |
| 20 | Telephone | Categoric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 21 | Target | Numeric | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |

To create a Decision Tree, after loading the credit data, go to the **Model** tab. In this section, we will use **Tree**. We'll see the other models later in this chapter.

In the following screenshot, we can see that, to create a Decision Tree, Rattle offers us two algorithms, traditional and conditional. The traditional algorithm works as we've seen in this chapter. The conditional algorithm helps to address overfitting; this algorithm can work better than the traditional algorithm in many cases. To optimize our Tree, Rattle has six parameters. As we'll see in the next chapter, one of the most common problems of supervised learning is overfitting; these parameters will help us to avoid it by reducing the complexity of the resulting Tree:

- **Min Split:** This is the minimum number of observations needed to create a new branch.
- **Min Bucket:** This is the minimum number of observations in each leaf.
- **Max Depth:** This is the maximum depth of the tree.
- **Complexity:** With this parameter, we will control the minimum *gain* needed to create a new branch. If the value is high, the resulting tree will be simple; if the value is low, the resulting tree will be more complex.
- **Priors:** Sometimes, the distribution of the target variable doesn't match with the real distribution. Imagine a dataset with a lot of sick patients. We can use this parameter to inform Rattle of the correct distribution of the target variable.
- **Loss Matrix:** In the next chapter, we'll see that in some cases, we need to distinguish between different kinds of misclassifications or errors. This parameter will help us to address this problem.

Finally, we've two important buttons: **Rules** and **Draw**. We will use these buttons just after creating our first tree, as shown here:



Set the following parameters as in the previous screenshot and press **Execute**:

- **Min Split:** 20
- **Max Depth:** 20
- **Min Bucket:** 7
- **Complexity:** 0.0100

Rattle will create a tree and will show the new tree in the screen. In the following screenshot, we've shown the root node and the first two branches of the tree:

```
Summary of the Decision Tree model for Classification (built using 'rpart'):

n= 700

node), split, n, loss, yval, (yprob)
  * denotes terminal node

1) root 700 209 1 (0.70142857 0.29857143)
  2) Status.of.existing.checking.account=A13,A14 326 44 1 (0.86503067 0.13496933) *
  3) Status.of.existing.checking.account=A11,A12 374 165 1 (0.55882353 0.44117647)
```

In the second line, `n= 700` is the size of the training set. Remember that our original dataset has 1,000 observations, but we've divided the complete dataset into training (70 percent), validation (15 percent), and testing (15 percent). For this reason, the size of the training dataset is 700.

In the fifth line, we see the root node. The number 1) is the node; `root` denotes that this is the root node; 700 is the number of observations; 209 is the number of observations misclassified, 1 is the default value for the target variable, and `(0.70142857 0.29857143)` is the distribution of the target variable. In our example, 0.7014 of the observations are classified as 1 (good credit risk) and 0.2986 are classified as 2 (bad credit risk).

```
1) root 700 209 1 (0.70142857 0.29857143)
```

The following lines show us the second and third nodes:

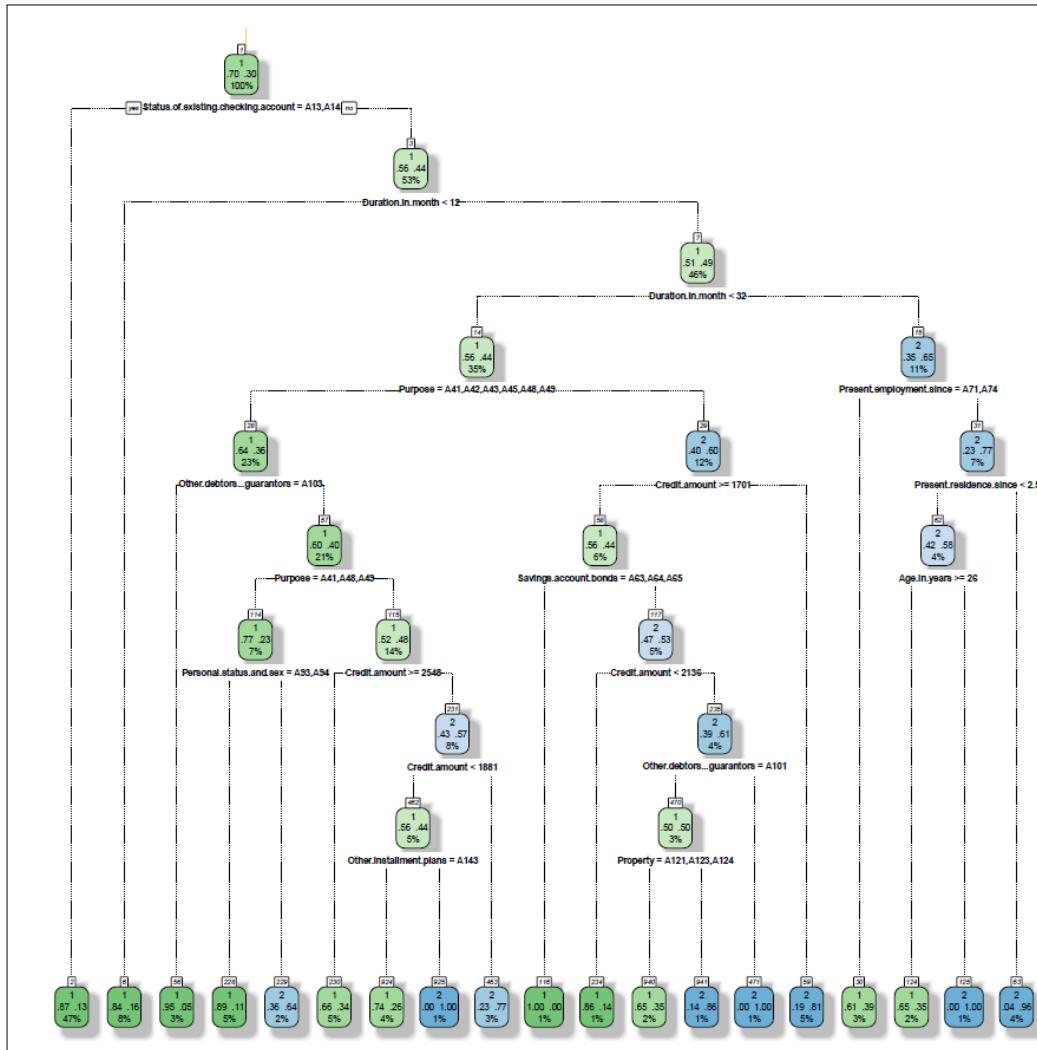
```
2) status.of.existing.checking.account=A13,A14 326 44 1 (0.86503067
0.13496933) *
3) status.of.existing.checking.account=A11,A12 374 165 1 (0.55882353
0.44117647)
```

In this node, the symbol * in the second node indicates that it's a leaf. The attribute `Status.of.existing.checking.account` is used by Rattle to create a branch. If the value of this attribute is A13 (≥ 200 DM/salary assignments for at least 1 year) or A14 (no checking account), the observation belongs to the second node. This second node is a leaf with 326 observations classified as 1 (good credit risk) and 44 observations are misclassified.

Decision Trees and Other Supervised Learning Methods

If the value of the attribute is $A_{11} (\dots < 0 \text{ DM})$ or $A_{12} (0 \leq \dots < 200 \text{ DM})$, the observation belongs to the third node. This node has 374 observations, but it's not a leaf node, so under this node, we'll have more branches.

Now, press the **Draw** button, and you'll have a graphical representation of the same tree, as shown here:



As we've seen, one advantage of trees is that it is easy to convert trees into rules that are easy to translate to other languages such as SQL, or Qlik Sense. Now push the **Rules** button to create the set of rules.

In our example, Rattle generates 19 rules. In the next chapter we'll see how to evaluate the performance of this model. Now, we'll focus on understanding the rules and how to use them. In the following screenshot, we see the first rule:

```
=====
Tree as rules:

Rule number: 125 [Target=2 cover=9 (1%) prob=1.00]
  Status.of.existing.checking.account=A11,A12
  Duration.in.month>=11.5
  Duration.in.month>=31.5
  Present.employment.since=A72,A73,A75
  Present.residence.since< 2.5
  Age.in.years< 26.5
```

The rule we see in the previous screenshot is the rule number 125. There are 9 observations that fall into this rule (`cover=9`); these 9 observations are 1 percent of the dataset. When an observation falls under this rule, the probability that the value of the target variable is 1 (Target=2), is 1.0 (`prob=1.0`).

This rule looks very specific because it fits perfectly into a small number of observations; we'll improve it in the following chapter.

Using Rattle to score new loan applications

As we've explained before, we will call scores to the process of predicting the output for new examples. We've two options to score new observations with our Decision Tree; we can code the Decision Tree rules in Qlik Sense or we can use Rattle to automatically score new observations.

As you have seen before, the rules are easy to translate to an *If then* structure that is easy to implement in any language. Imagine Rattle provides you with a set of 10 rules and the first rule is as follows:

The screenshot shows the Rattle software interface with the 'Model' tab selected. The configuration settings include:

- Type: Tree
- Target: Target
- Algorithm: Traditional
- Model Builder: rpart
- Min Split: 20
- Max Depth: 30
- Priors: (empty)
- Min Bucket: 7
- Complexity: 0.0100
- Loss Matrix: (empty)
- Include Missing: checked
- Rules: (highlighted)
- Draw: (disabled)

Tree as rules:

```

Rule number: 125 [Target=2 cover=9 (1%) prob=1.00]
  Status.of.existing.checking.account=A11,A12
  Duration.in.month>=11.5
  Duration.in.month>=31.5
  Present.employment.since=A72,A73,A75
  Present.residence.since< 2.5
  Age.in.years< 26.5

```

In the following screenshot, we see how we can create a new attribute called **Prediction**. In this example, we will just see the implementation of the rule 109 using the Qlik Sense **Data load editor**, but we can use the *If then* structures to implement all the rules, as shown here:

```

// Rule number: 125 [Target=2 cover=9 (1%) prob=1.00]
if ("Status of existing checking account" = 'A11' OR "Status of existing checking account" = 'A12') AND
("Duration in month")>=11.5 AND "Duration in month">=31.5) AND
("Present residence since" = 'A72' OR "Present residence since" = 'A73' OR "Present residence since" = 'A75') AND
( "Present residence since" < 2.5) AND
("Age in years" < 26.5), 2, 0) AS Prediction,

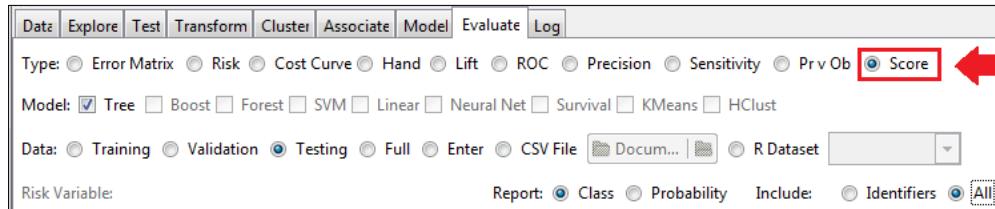
```

Now, we have a new attribute in our table called **Prediction** that gives us a prediction for the credit risk.

Rattle provides us an option to automatically score new observations. Using this option, we don't need to manually code the rules; for this reason, we will use Rattle to score new credit applications in this example.

In the Rattle's **Evaluate** tab, there are different types of evaluation. In this section, we will use **Score**, as shown in the following screenshot. Under the type of evaluation, there is the model we will use. In our example, we've only built a Tree model, for this reason, we will choose **Tree**.

Under the model, we must choose the data we want to score. The two most usual options are **Testing** and **CSV File**. We can score new observations contained in a CSV file by selecting the **CSV File** option. In our example, we will use the **Testing** option to score the testing:



Finally, we have to choose the type of report we want to create. Choose **Class** and a category will be created for each observation. In the **Include** option, choose **All** to include all variables in the report. Press the **Execute** button and Rattle will create a CSV file with all original variables and a new one called `rpart`, as shown in this screenshot:

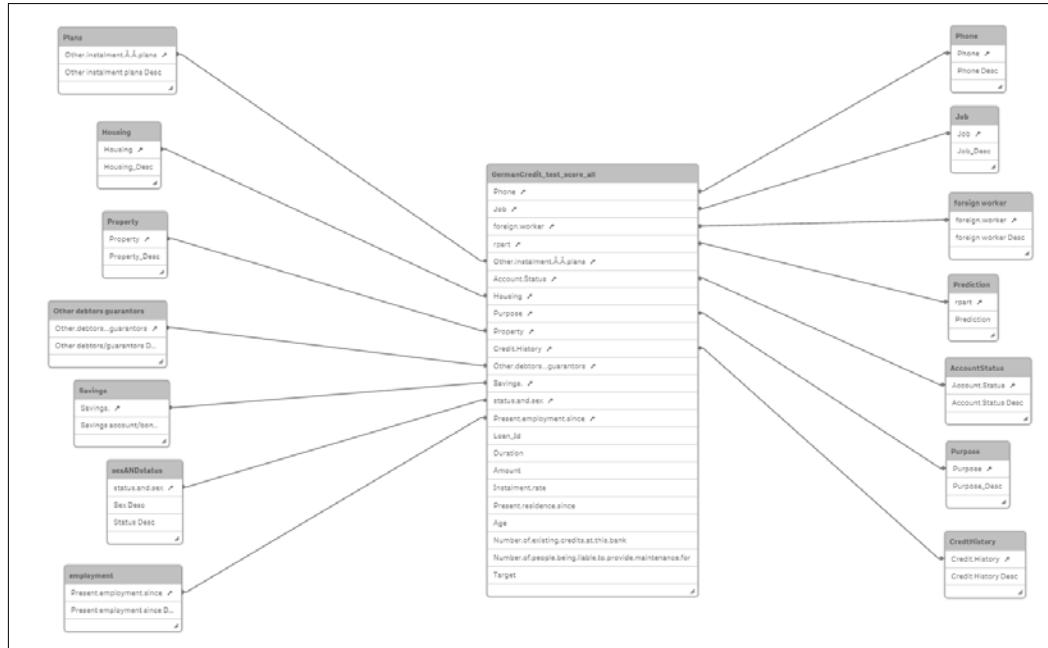
| | Account.S.Duratio | Credit.His.Purpose | Amount | Savings. | Present.e | Instalment | status.an | Other.det | Present.r | Property | Age | Other.intlHousing | Number.c.Job | Number.c.Phone | foreign.w.Target | rpart |
|---|-------------------|--------------------|--------|-----------|-----------|------------|-----------|-----------|-----------|----------|--------|-------------------|--------------|----------------|------------------|-------|
| 1 | A14 | 12 A32 | A43 | 3059 A64 | A74 | 2 A92 | A101 | 4 A121 | 61 A143 | A152 | 1 A172 | 1 A191 | A201 | 1 | 1 | |
| 2 | A14 | 30 A30 | A49 | 8072 A65 | A72 | 2 A93 | A101 | 3 A123 | 25 A141 | A152 | 3 A173 | 1 A191 | A201 | 1 | 1 | |
| 3 | A11 | 24 A32 | A41 | 12579 A61 | A75 | 4 A92 | A101 | 2 A124 | 44 A148 | A153 | 1 A174 | 1 A192 | A201 | 2 | 2 | |
| 4 | A12 | 6 A32 | A42 | 1374 A61 | A73 | 1 A93 | A101 | 2 A121 | 36 A141 | A152 | 1 A172 | 1 A192 | A201 | 1 | 1 | |
| 5 | A11 | 18 A32 | A49 | 1913 A64 | A72 | 3 A94 | A101 | 3 A121 | 36 A141 | A152 | 1 A173 | 1 A192 | A201 | 1 | 1 | |
| 6 | A12 | 12 A34 | A49 | 1264 A65 | A75 | 4 A93 | A101 | 4 A124 | 57 A143 | A151 | 1 A172 | 1 A191 | A201 | 1 | 1 | |
| 7 | A14 | 36 A32 | A43 | 2299 A63 | A75 | 4 A93 | A101 | 4 A123 | 39 A143 | A152 | 1 A173 | 1 A191 | A201 | 1 | 1 | |
| 8 | A14 | | | | | | | | | | | | | | | |

Now, we have a file containing all the variables of the testing dataset and a prediction for each observation. In the next section, we will use Qlik Sense to create a visual application for the business user. With this application, the business users will be able to access new applications information.

Creating a Qlik Sense application to predict credit risks

In the previous section, we've created a Decision Tree using Rattle and we've scored the testing dataset using the model we created. In this section, we'll use Qlik Sense to build a visual application to explore new loan applications.

The German Credit dataset contains two different types of input variables, numeric, and categorical. In a categorical variable such as Purpose, each observation contains a value, and possible values for Purpose are A40, A41, A42, A43, A44, and A45. Each value has meaning, for example A40 means a new car. In order to help the user to understand and explore the data, we want to translate all categorical values to its meaning. Like in *Chapter 4, Creating Your First Qlik Sense Application*, we'll add a description in separate tables and we'll build a data model, such as the following screenshot:



Remember that to link two tables, Qlik Sense needs two fields with exactly the same names.

Now, we need to create a table for each categorical variable containing the original value and its translation. For the variable Purpose, we'll create a table like the following:

| Purpose | Purpose_Description |
|---------|---------------------|
| A40 | car (new) |
| A41 | car (used) |
| A42 | furniture/equipment |
| A43 | radio/television |
| A44 | domestic appliances |
| A45 | repairs |
| A46 | education |
| A47 | vacation |
| A48 | retraining |
| A49 | business |
| A410 | others |

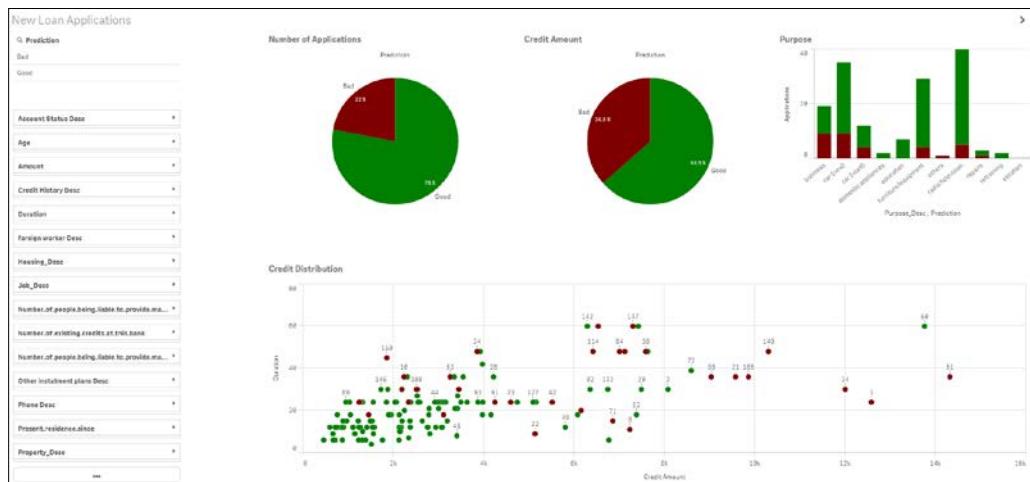
Use a spreadsheet tool such as Microsoft Excel, to create a file that contains a sheet for each categorical variable.

Now, we've two files with one file containing the scored testing dataset, and a file with all the descriptions for the categorical variables.

You've learned in *Chapter 4, Creating Your First Qlik Sense Application*, about how to load data into Qlik Sense. In this example, we have a file with 14 sheets or tables. If you want to load all sheets, you can select all sheets in the data load wizard, like in the following screenshot:



After loading the data, we create a visual application for the business user. You've learned in *Chapter 4, Creating Your First Qlik Sense Application*, and, *Chapter 5, Clustering and Other Unsupervised Learning Methods*, on how to create this application. One benefit of Qlik Sense is that it gives self-service data visualization; it means that each user can create his own charts depending on his interests. You can create the application you want; as an example, we've created an application with two sheets. The first sheet is an overview and the second sheet contains a table to see all the details of new applications, as shown in the following screenshot:



Ensemble classifiers

Thomas G Dietterich defines Ensemble methods as follows:

"Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a (weighted) vote of their prediction."

You can get more information from <http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf>.

Ensemble methods create a set of weak classifiers and combine them into a strong classifier. A weak classifier is a classifier that performs slightly better than a classifier that randomly guesses the prediction. Rattle offers two types of ensemble models: Random Forest and Boosting.

Boosting

Boosting is an ensemble method, so it creates a set of different classifiers. Imagine that you have m classifiers, we can define a classifier x as:

$$\text{Tree}_x(\text{new observation}) = \text{Classification}_x$$

When we need to evaluate a new observation, we can calculate the average of all m tree's predictions using the following formula:

$$\text{Tree}(\text{new observation}) = \frac{\sum_{i=1}^m \text{Tree}_i(\text{new observation})}{m}$$

We can improve this evaluation by adding a weight to each tree, as shown here in this formula:

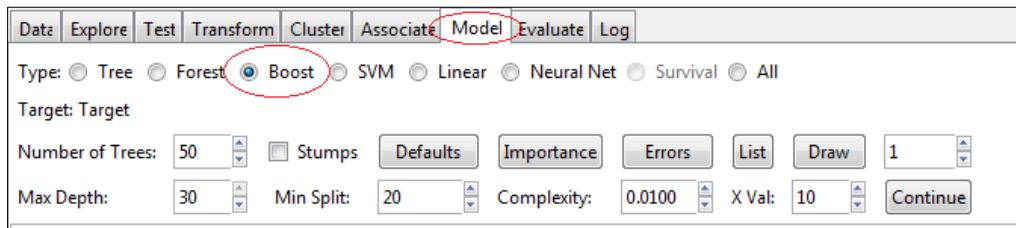
$$\text{Tree}(\text{new observation}) = \sum_{i=1}^m \text{Weight}_i * \text{Tree}_i(\text{new observation})$$

We can use this mechanism for Regression and also for classification (for example, if the result is higher or equal to one, the observation belongs to class X). For classification, we can also use other mechanisms such as the majority of vote.

Now, we know to ensemble a set of classifiers, but we need to create this set of classifiers. To create different classifiers, we will use different subsets of data.

The most usual boosting algorithm is AdaBoost or adaptive boosting. This algorithm was created by Yoav Freund and Robert Schapire in 1997. In AdaBoost, we start by assigning equal weights to all observations. To create the first tree, we will select a random set of observations. After creating and evaluating the model, we will increase the weight of misclassified observations in order to *boost* misclassified observations. Now, we can create the second leaner or model by selecting a new random set of observations. After creating each leaner, we will increase the weight of misclassified observations.

To create a boosting model, you have to go to Rattle's **Model** tab and select the **Boost** type, as shown in the following screenshot:



Rattle offers different options, the most important ones are:

- **Number of Trees:** This is the number of different trees the algorithm will ensemble
- **Max Depth:** This is the maximum depth of the final tree
- **Min Split:** In the **Tree** option, this is the minimum number of observations needed to create a new branch
- **Complexity:** In the **Tree** option, this parameter controls the minimum gain in terms of complexity needed to create a new branch

In an ensemble algorithm, **Out-Of-Bag Error** or OOB Error is a good approach to performance. OOBs are observations that are not in the subset used to create each tree. Rattle uses all observations that aren't in the subset used to create the trees to validate the model. When we see an OOB Error of 0.123, it means that our model correctly classifies 87.7 percent of the observation that wasn't in the random training set.

When we execute the Boost model, Rattle returns information in text form. See highlighted in the following screenshot, the **Out-of-Bag Error** and **iteration** values. In our example, the iteration is 46. We ran the example with the variable **Number of Trees** set to 46; a iteration of 46 and OOB error 0.123 means that with 46 trees, we can achieve a OOB error of 0.123. If we increase the **Number of Trees** value to 50, we won't be able to achieve a better OOB Error, as shown here:

```
Summary of the Ada Boost model:

Call:
ada(Target ~ ., data = crs$dataset[crs$train, c(crs$input, crs$target)],
    control = rpart.control(maxdepth = 30, cp = 0.01, minsplit = 20,
                           xval = 10), iter = 50)

Loss: exponential Method: discrete   Iteration: 50

Final Confusion Matrix for Data:
          Final Prediction
True value   1     2
           1 481 10
           2  54 155

Train Error: 0.091

Out-Of-Bag Error:  0.123  iteration= 46
```

If you press the button; **Errors**, Rattle will plot the OOB Error, and you will see how it evolves when you add more trees to the model.

Finally, we've created a set of trees; we can see each tree in the text format or as a plot by pushing the **List** or **Draw** button and select the number of trees in the text field, as shown here:



Random Forest

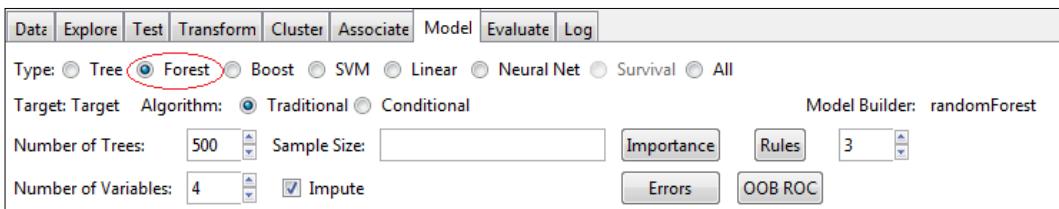
Random Forest is an ensemble method developed by *Leo Breiman* and *Adele Cutler* in 2001. This is an ensemble algorithm, so we don't have a single tree, we've a set of trees or a forest. Random Forest combines this set of trees in a single model to improve the performance.

The main idea is to produce a set of trees, introducing some randomness in each tree, and combine all of them to produce a better prediction. The randomness is produced in two different ways; in the set of variables used to split the data in branches, and in the observations used to create (train) the tree.

The basic algorithm, is the same as we've already seen in this chapter, but now we will create a fixed number of trees. We need to decide the **Number of Trees** value and Rattle will create this many number of trees for us. Before it creates a tree, it randomly selects a set of observations to train the classifier and uses a randomly chosen subset of input variables to split the data. The final result is a set of trees.

In order to predict a new observation, Random Forest evaluates the new observations for each tree. If the target attribute is categorical (classification), Random Forest will choose the most frequent as its prediction. If the target variable is numerical (Regression), the average of all predictions will be chosen.

To create a Random Forest model in Rattle, you have to load a dataset, go to the **Model** tab, and choose **Forest** as the model type, as shown here:



After selecting **Forest** as the model type, press the **Execute** button to create the model. After executing Random Forest, Rattle creates a summary. The summary contains the following:

- The number of observations used to train the model
- Whether the model includes observations with missing values
- The type of trees—Classification or Regression

- The number of trees created
- The number of variables used at each split
- The OOB estimate of the error rate
- The confusion matrix

We'll see the confusion matrix in the next chapter. Now, we have to see the OOB estimate of the error rate. In the following screenshot, we will see 23 percent as the **OOB estimate of error rate**. This rate is high, and means our model has low performance, as indicated in this screenshot:

```
Summary of the Random Forest Model
=====
Number of observations used to build the model: 700
Missing value imputation is active. ←

Call:
randomForest(formula = as.factor(Target) ~ .,
              data = crs$dataset[crs$sample, c(crs$input, crs$target)],
              ntree = 500, mtry = 4, importance = TRUE, replace = FALSE, na.action = na.roughfix)

      Type of random forest: classification
      Number of trees: 500
No. of variables tried at each split: 4 ←

      OOB estimate of  error rate: 23% ←

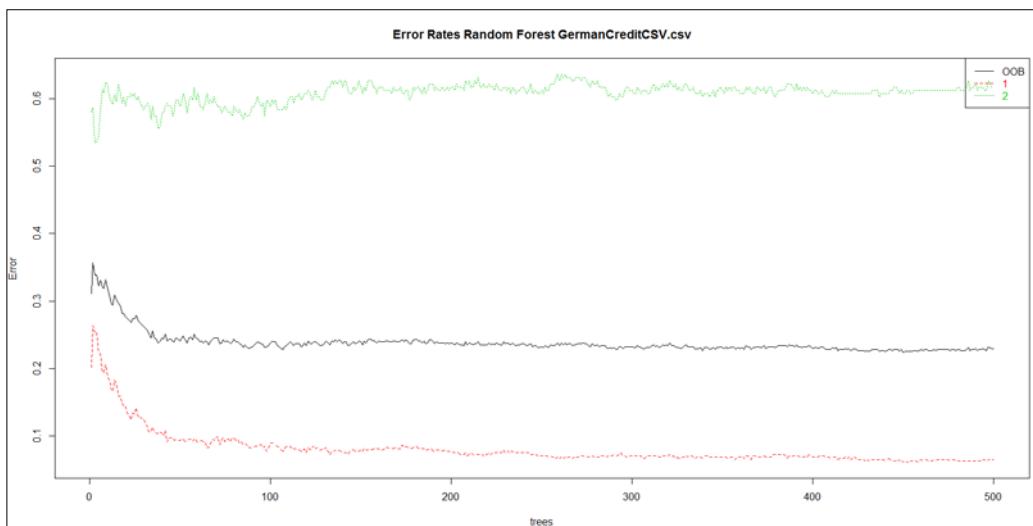
Confusion matrix:
  1  2 class.error
1 459 32  0.06517312
2 129 80  0.61722488
```

Rattle offers you some options to improve the model. You can choose **Number of Trees** that the model will create and **Number of Variables**; in between, Rattle will choose one to split the dataset. We can use the **Impute** checkbox to control whether observations with missing values are ignored or not.

We've four buttons: **Importance**, **Rules**, **Errors** and **OOB ROC**. You've seen before that by pressing the **Rules** button, Rattle converts a tree to a set of rules. In this case, we have a set of trees. For this reason, we've a text field where we can indicate the number of trees. If we write three in the text field and press **Rules**, Rattle will convert the tree number three, to a set of rules.

The **Importance** button creates a plot that shows two measures of the variable importance prediction accuracy and Gini index. The Gini index is a measure of the inequality of a distribution. These two measures help us to understand which variables are more useful for predicting a new observation.

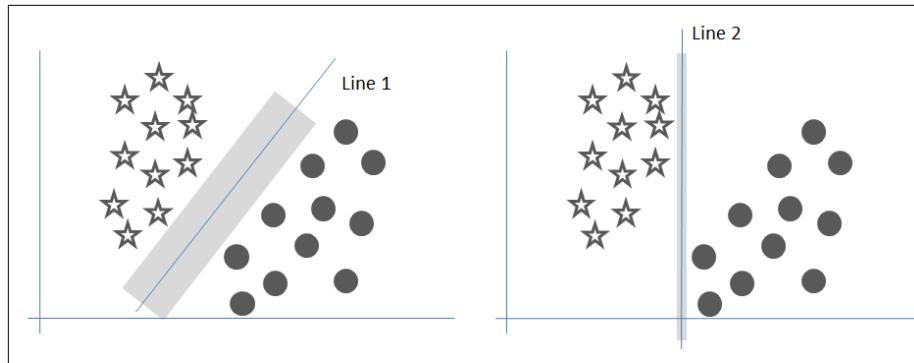
There is a trade-off between the error rate and the time needed to create the model. If we choose to create a small number of trees, Rattle will need a small amount of time to create it, but the error rate will be high. If we choose creating a large number of trees, the time needed will be higher, but the error rate will be lower. A good tool to choose the number of trees is the error plot. Press the **Error** button and Rattle will create an error plot. This plot shows us the evolution of the OOB estimate of error rate when you increase the number of trees. Look at the following screenshot; when the number of trees is small, the error rate is high; by incrementing the number of trees, we will decrease the error rate. In the same plot, we can see the error rate associated with each prediction:



Finally, **Sample Size** can be used to limit the number of observations necessary to create each tree, or in a classifier, can be used to define the number of observations of each class that will be used. In our credit example, we've 1,000 observations; 700 are classified as 1, and 300 are classified as 2. On average, the number of observations classified as 1 in each sample dataset will be 70 percent. We can use **Sample Size** to create a different distribution. If you use 200, 200 as the **Sample Size** value, each sample dataset will contain 200 observations classified as 1 and 200 as 2.

Supported Vector Machines

Supported Vector Machine (SVM) is a supervised method used for Classification and Regression. SVM looks for the *best separation* between observations that belong to different classes. The best separation is the one with the higher margin. In the following diagram, you can see a dataset with two classes of observations: stars and circles. On the left-hand side, the observations are divided by a line called **Line 1**. On the right-hand side, there is the same dataset divided by a line called **Line 2**:



In this example, our dataset only has two attributes and can be represented in a plane and divided by a simple line, but usually our datasets are more complex and have a lot of attributes. We need to use a hyperplane to divide them.

Usually, different classes in a dataset are not easily separable as on our previous example. For this reason, *Kernel Functions* are used to preprocess the attributes of the observations to convert into easier separable observations. The performance of the model depends on the Kernel Function used.

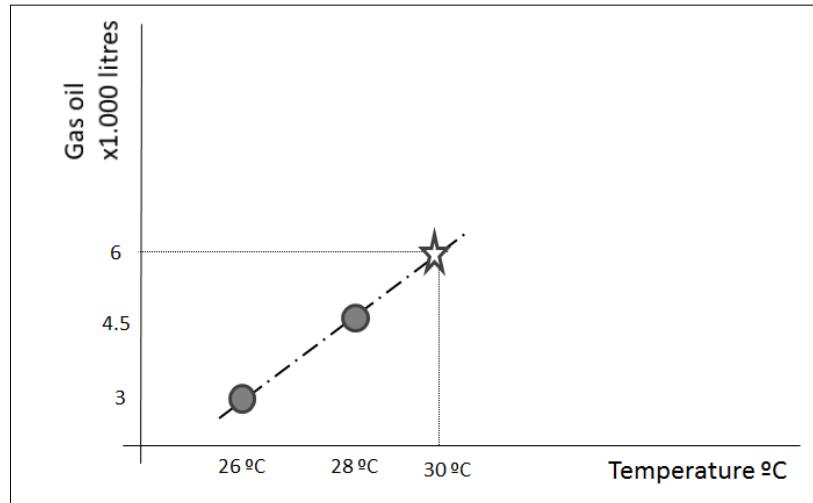
Other models

In this section, we will see other models provided by Rattle in the **Model** tab, which aren't supervised learning. These methods are Linear and Logistic Regression, Neural Networks, and Survival Analysis.

Linear and Logistic Regression

Linear Regression is a statistical method to describe the relationship between one or more input variables and one or more output variables. The objective is to create a formula that models the relationships between input and output variables; in this way, we can use this formula to predict new observations.

Imagine you are the manager of a marina, your marina has a gas station and you need to predict the amount of gas oil you will sell during a summer day. On the Mediterranean coast, during the summer, the amount of gas oil sold is correlated to the temperature. The reason is obvious, on sunny days, the temperature rises and more tourists want to use their boats. The example is illustrated in this diagram:



Based on past experience, we know that on a day with a temperature of 26°C, the gas station sold 3,000 liters, and on a 28°C day, it sold 4,500 liters. In this example, we can assume that the relationship between **Gas oil Sold** and **Temperature** is described by this equation:

$$\text{Gas oil Sold} = a + b * \text{Temperature}$$

This equation describes the relationship between Gas oil Sold and Temperature. Using this equation and a weather forecast, we can predict the gas oil our gas station will sell tomorrow.

When our dataset has one output variable, we will call it **Simple Linear Regression**. If we have more than one output variable, we call it **Multiple Linear Regression**.

We use Linear Regression when the target variable is numerical. In classification tasks, when the target variable is categorical, we will use Logistic Regression. When the target variable has two possible values or classes, we can use binary or binomial Logistic Regression.

Neural Networks

The Neural Network model is a method that can be used as supervised or unsupervised learning. Neural Networks are especially useful for pattern recognition and time series prediction; real-world applications include facial recognition, character recognition, or stock prices.

Neural Networks are inspired by the human brain and have three main layers:

- **Input Layer:** This layer receives the input data and passes them to the hidden layer.
- **Hidden Layer:** This layer contains a number of interconnected nodes. These nodes, called neurons, are mathematical functions that create the predictions.
- **Output Layer:** This layer creates the final prediction from the predictions done in the hidden layer.

Neural Networks work especially well, when the number of input or attributes in the observations is high. An important disadvantage of Neural Networks is that the Hidden Layer is a black box and the algorithm doesn't explain the value of the prediction.

Further learning

You can find a great introduction to Data Science in *Introduction to Data Science*, a very interesting Coursera course. The instructor is Bill Howe, Director of Research Scalable Data Analytics at the University of Washington. You can find more details from the following location:

<https://www.coursera.org/course/datasci>

The course has a very interesting section about Machine Learning. In this section, you will find a very intuitive introduction to entropy, information gain, and Decision Tree Learning.

In the last chapter, we've referenced *Machine Learning with R*, Brett Lantz, Packt Publishing. In this book, you can find a section about Decision Trees and the author develops an example using the German Credit dataset.

Summary

In this chapter, we've seen the concept of entropy and information gain. We've learned to create a Decision Tree with these concepts. After this, we've used Rattle to create a model to predict credit risk. We've translated our tree to rules, and seen how to code them in Qlik Sense.

After Decision Trees, we saw how ensemble models combine a set of learners to create a better model. We've focused on two ensemble models: Random Forest and Boosting.

Then, we've introduced Supported Vector Machines, and finally, we've covered other methods such as Regression and Neural Networks.

During this entire chapter, we didn't worry about the model performance, we just created the models. However, we avoided looking at the prediction accuracy of all these different models. In the next chapter, we'll learn how to compare the performance of different models and see how to optimize a model.

In real life, model creation and optimization are iterative processes. You can create a model and evaluate its performance. Then, you have to test different model parameters, evaluate the performance again, and compare the new performance with your previous performance. In this book, model creation and optimization are split into different chapters for simplicity.

7

Model Evaluation

In the previous chapter, we've seen how to create supervised learning methods. We divided our datasets into three subsets – **training**, **validation**, and **testing**. We also used the training dataset to train our models, and in this chapter, we'll use the validation dataset to measure the model performance and to compare different models.

In this chapter, we'll explore different methods for measuring the predictive power of a model.

As we've seen before, there are two kinds of predictive models: regression and classification. In a regression model, the output variable is a numeric variable; in a classification model, the output variable is a categorical variable. We'll start this chapter with cross-validation. After this, we'll measure the performance in regression methods, and then, we'll move on to classification performance.

Cross-validation

Cross-validation is a very useful technique to evaluate the performance of a supervised method. We will randomly split our dataset into k sub-datasets called folds (usually, 5 to 10). We will choose a fold for testing and keep the rest for training. We will train the model using the other $k-1$ folds and test it with a fold. We will repeat this process of training and testing k times, each time keeping a different folder for testing.

In each iteration, we will create a model and obtain a performance measure such as accuracy. When we've finished, we have k measures of performance, and we can obtain the performance of the modeling technique by calculating the average.

Using Rattle, we can split the original dataset into training, validation, and testing. Some R packages implement cross-validation when creating the model. If the model we are creating, uses cross-validation, we can skip the creation of the validation dataset and only create the training and testing datasets.

When we build a tree, such as in the previous chapter, Rattle uses the `RPART` package. This package implements cross-validation. For this reason, after building the model, the last information that Rattle gives us is the complexity table, which is shown here:

| Root node error: 209/700 = 0.29857 | | | | | |
|------------------------------------|----------|--------|-----------|---------|----------|
| n= 700 | | | | | |
| | CP | nsplit | rel error | xerror | xstd |
| 1 | 0.036683 | 0 | 1.00000 | 1.00000 | 0.057932 |
| 2 | 0.023923 | 4 | 0.80861 | 0.99522 | 0.057852 |
| 3 | 0.016746 | 6 | 0.76077 | 0.93780 | 0.056839 |
| 4 | 0.012759 | 8 | 0.72727 | 0.96172 | 0.057273 |
| 5 | 0.011962 | 14 | 0.61244 | 0.96651 | 0.057358 |
| 6 | 0.010000 | 18 | 0.56459 | 0.96651 | 0.057358 |

In the complexity table, we can see the **Complexity Parameter (CP)** that we've discussed in the previous chapter, the number of splits (**nsplit**), the error in the training set (**error**), and the cross-validated error (**xerror**). Notice that by incrementing the number of splits, we will reduce the error on the training dataset very fast, but we're not interested in that because a low error in the training dataset doesn't assure a low error with the new observation. We want to reduce the cross-validated error. We can see that on row **3** with **6** folds, we've the lowest cross-validated error; after row **3**, the error increases. For this reason, Rattle stops splitting the dataset.

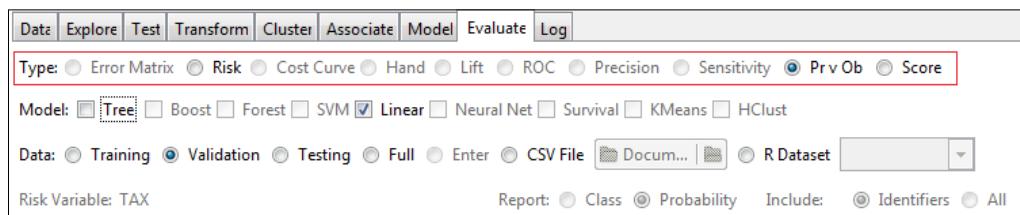
Regression performance

To measure the performance of a regression, the distance between the predicted outputs and the actual outputs, is a good model performance measure.

Rattle offers us a good way to see predicted values versus the actual value—the Predicted versus Observed plot. To test this plot, you need to create a regression model. You can download a sample dataset from the **UCI Machine Learning Repository** (<http://archive.ics.uci.edu/ml>; Irvine, CA: University of California, School of Information and Computer Science), or from Kaggle (<http://www.kaggle.com/>). On some websites, such as the UCI Machine Learning Repository, the datasets are classified by the task you want to perform with the dataset.

Predicted versus Observed Plot

Imagine we have to create a model to predict the price of a house. Click on the **Evaluate** tab:

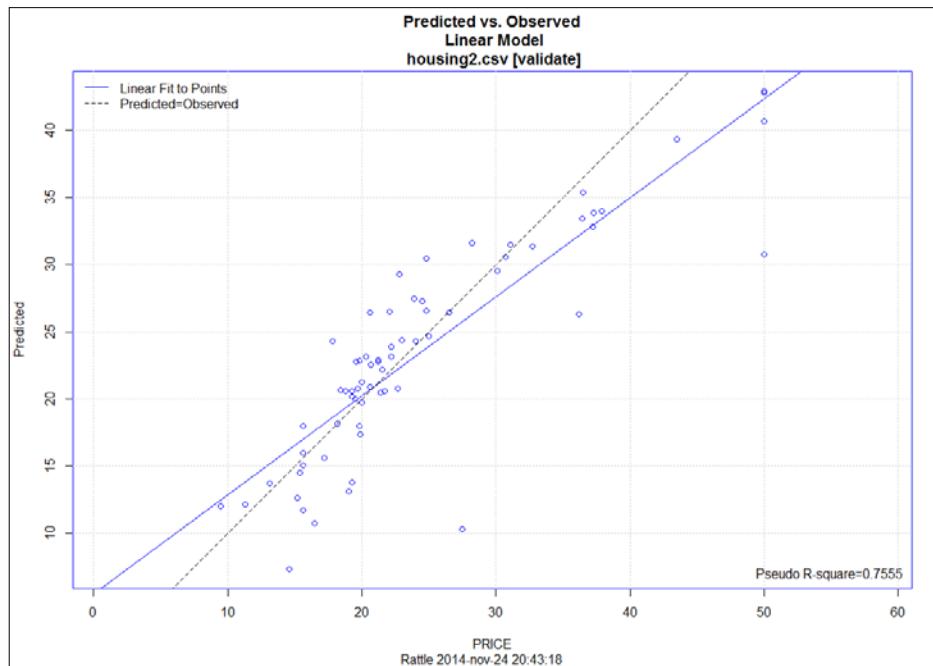


Rattle's **Evaluate** tab offers us two good options for a regression model as shown in the preceding screenshot:

- **Predicted versus Observed Plot:** We will use this option to compare predicted values versus actual values.
- **Score:** As we've seen before, this option creates predictions for the selected dataset:

Model Evaluation

After creating the model, go to the **Evaluate** tab, select your **Model**, the **Validation** dataset, the **Pr v Ob** option, press **Execute**, and Rattle will build a **Predicted vs. Observed** plot for you, as shown here:



This plot shows a set of points; each point is an observation in the y axes, where we can see the predicted value, and in the x axes, we can see the actual value. We can also see a dotted line; this line represents a perfect prediction, when predicted values are the same as the actual values. The last line is a linear fit to points.

Finally, **Pseudo R-square** is an approach to **R-square**. This measures the variance explained by the model. R-square is a number from 0 to 1; an R-square close to 1 means that the model has strong predictive power. When the model doesn't provide a good prediction, R-square is close to 0. In the same way, a **Pseudo R-square** close to 1 is good; a measure close to 0 means low performance.

Measuring the performance of classifiers

In this section, we'll see how to measure the performance of a classifier. In the example we saw in the previous chapter, a Decision Tree can predict that a new customer will not default, but actually he/she does default. We need a mechanism to evaluate the error rate of a decision tree; this mechanism is the confusion matrix or the error matrix.

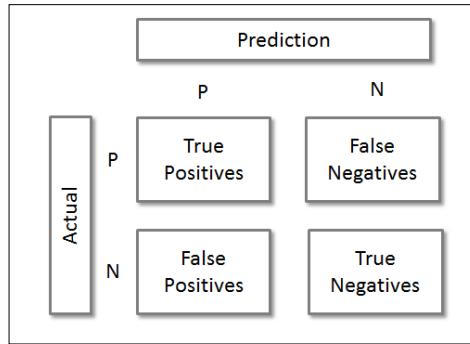
Confusion matrix, accuracy, sensitivity, and specificity

Coming back to our loan example, imagine you have classified 1000 loans using a Decision Tree. For each loan, our classifier has added a label with the value *yes* or *no*, depending upon whether the algorithm predicts that the customer will default or not. In order to generalize, we will use the terms positive or negative classification. In our loans example, we have a positive classified observation when our classifier predicted that a customer will default, so the value of the **Default?** Attribute is **Yes**.

In this scenario, there are four types of predictions, listed as follows:

- **True Positive:** The observation has been correctly classified as positive. In our example, the classifier predicts that the customer will default and the customer defaults.
- **False Positive:** The observation has been incorrectly classified as positive. In our example, the classifier predicts that the customer will default but the customer doesn't default.
- **True Negative:** The observation has been correctly classified as negative. In our example, the classifier predicts that the customer will not default and the customer doesn't default.
- **False Negative:** The observation has been incorrectly classified as negative. In our example, the classifier predicts that the customer will not default but the customer defaults.

We will call this classification; the confusion matrix; or the error matrix because we represent it using a matrix, and this, gives us an idea of the prediction error. The following diagram illustrates the confusion matrix:

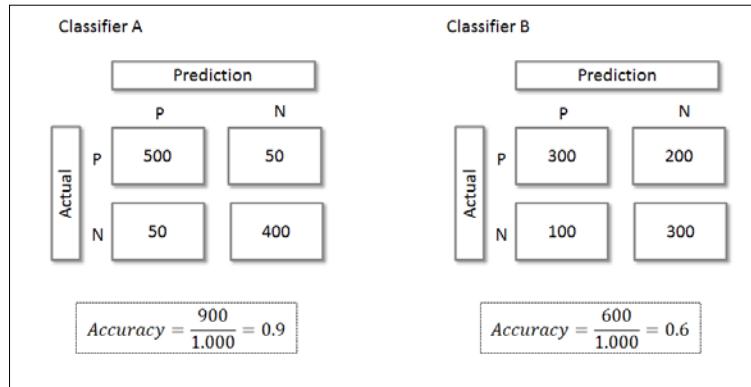


As you can see in the matrix, **True Positives** and **True Negatives** are correctly classified by the algorithm, whereas **False Positives** and **False Negatives** are incorrectly classified. In order to evaluate the performance of the classifier, the first measure is the **accuracy**. To calculate the accuracy, we will divide the total correctly classified observations by all the observations:

$$\text{Accuracy} = \frac{\text{Correctly classified observations}}{\text{Total Predicted observations}}$$

The accuracy is a number between 0 and 1. If the accuracy is 0, it means that the classifier has failed in all the predictions; if the accuracy is 1, it means that the classifier has classified correctly, all the observations. When the accuracy is close to one, the performance of the classifier is good, and when the accuracy is close to zero, the performance is bad. If you use a model that randomly guesses whether a credit is classified as bad or good risk (binary classification), the accuracy should be 0.5.

In the following diagram, are the results of two classifiers, **Classifier A** and **Classifier B**, both having 1000 observations, but the **Accuracy** value is very different. In the following example, the performance of **Classifier A** is better than the performance of **Classifier B**:



There is no rule for whether an accuracy rate is good or not, because it depends on the problem that you are solving. In order to know if your model has good performance, you have to compare with another model. The first step can be to compare the performance of your model against a random classifier or a simple classifier as baseline. Imagine that by exploring your data, you have discovered that young people have a greater probability of defaulting than older people. You can create a naive classifier that predicts Default?=Yes for young people and Default?=No for older people. If your assumption is true, the accuracy of your naive classifier will be greater than 0.5 (random classifier); maybe 0.6 or 0.7. You can use the performance of this naive model as the baseline to measure the performance of your model.

The opposite measure is the **error rate** – the percentage of observations misclassified. Accuracy and error rate are good measures towards understanding the performance of a classifier, but keep in mind that they do not distinguish between the types of errors. Imagine you have developed a classifier that classifies tumor images between malignant and benignant. A false positive is a tumor classified as malignant, but that actually benign. If you have a false positive, then probably the doctor will ask for additional tests to confirm the diagnosis, and he will discover that the image was misclassified; but what happens if you have a false negative? If you have a false negative, a malignant tumor will be classified as benign. Probably, the doctor won't ask for additional tests and a malignant tumor would be treated as a benign tumor.

As you can see, for this domain, a false negative is more dangerous than a false positive. For this reason, we need a way to differentiate between the kinds of misclassifications.

We can use the **Sensitivity** or **True Positive Rate** to measure the ability of a classifier to correctly classify positive observations:

$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

If a classifier has a high sensitivity, this means that when an observation is classified as positive, the probability of error is low.

In order to measure the ability of a classifier to correctly classify a negative case, we will use **Specificity** or **True Negative Rate**:

$$\text{Sensitivity} = \frac{\text{True Negative}}{\text{Negative}} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}}$$

Now, we'll use the model that we built in the previous chapter to explore Rattle's confusion matrix options, to classify the risk of loan applications into two classes: 1 (good credit risk) and 2 (bad credit risk). Load the dataset we used in *Chapter 6, Decision Trees and Other Supervised Learning Methods*, and create two different models. I've created a decision tree (Minimum Split = 30, Minimum Bucket = 20, and Maximum Depth = 10) and a Random Forest model (default parameters).

Now, go to Rattle's **Evaluate** tab. You have three rows of options. In the top row (highlighted in the following screenshot), you can choose the type of evaluation you want to perform. Choose **Error Matrix** (confusion matrix). The middle row is for the model you want to evaluate. As explained earlier, I've created two models, a decision tree and a random forest, so I have the option of calculating the confusion matrix for these models. In the following screenshot, you can see that I've selected **Tree** and **Forest** because I want to compare the confusion matrix of both the models.

Finally, the bottom row allows you to choose the dataset that we'll use to perform the evaluation. By default, Rattle chooses the **Validation** dataset. Usually, we use this dataset to optimize the model, as shown in this screenshot:

```

Data Explore Test Transform Cluster Associate Model Evaluate Log
Type:  Error Matrix  Risk  Cost Curve  Hand  Lift  ROC  Precision  Sensitivity  Prv Ob  Score
Model:  Tree  Boost  Forest  SVM  Linear  Neural Net  Survival  KMeans  HClust
Data:  Training  Validation  Testing  Full  Enter  CSV File  Document...  R Dataset 
Risk Variable: Report:  Class  Probability Include:  Identifiers  All

Error matrix for the Decision Tree model on GermanCreditCSV.csv [validate] (counts):
Predicted
Actual   1   2
1  96   5
2  36  13

Error matrix for the Decision Tree model on GermanCreditCSV.csv [validate] (proportions):
Predicted
Actual   1   2   Error
1  0.64  0.03  0.05
2  0.24  0.09  0.73

Overall error: 0.2733333, Averaged class error: 0.2752525

Rattle timestamp: 2014-11-21 10:12:06 fgs
=====
Error matrix for the Random Forest model on GermanCreditCSV.csv [validate] (counts):
Predicted
Actual   1   2
1  90  11
2  34  15

Error matrix for the Random Forest model on GermanCreditCSV.csv [validate] (proportions):
Predicted
Actual   1   2   Error
1  0.60  0.07  0.11
2  0.23  0.10  0.69

Overall error: 0.3, Averaged class error: 0.3486352

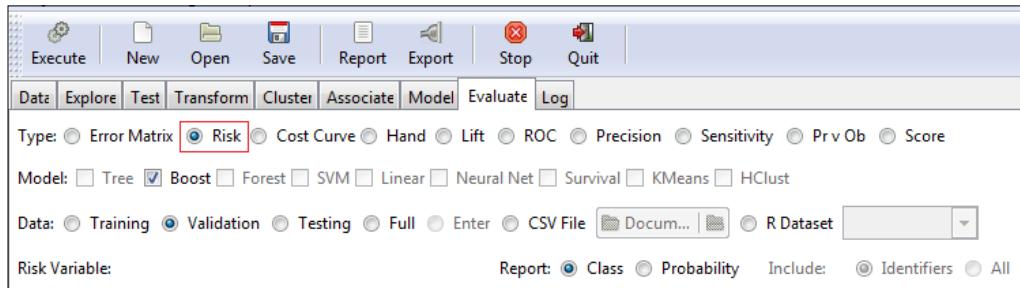
Rattle timestamp: 2014-11-21 10:12:06
=====
```

In the previous screenshot, we can see the output of the **Error Matrix** option. For each model, we can see the confusion matrix with the total number of observations and the percentage of observations. After the matrix, Rattle provides us with the error rate or **Overall error**.

Risk Chart

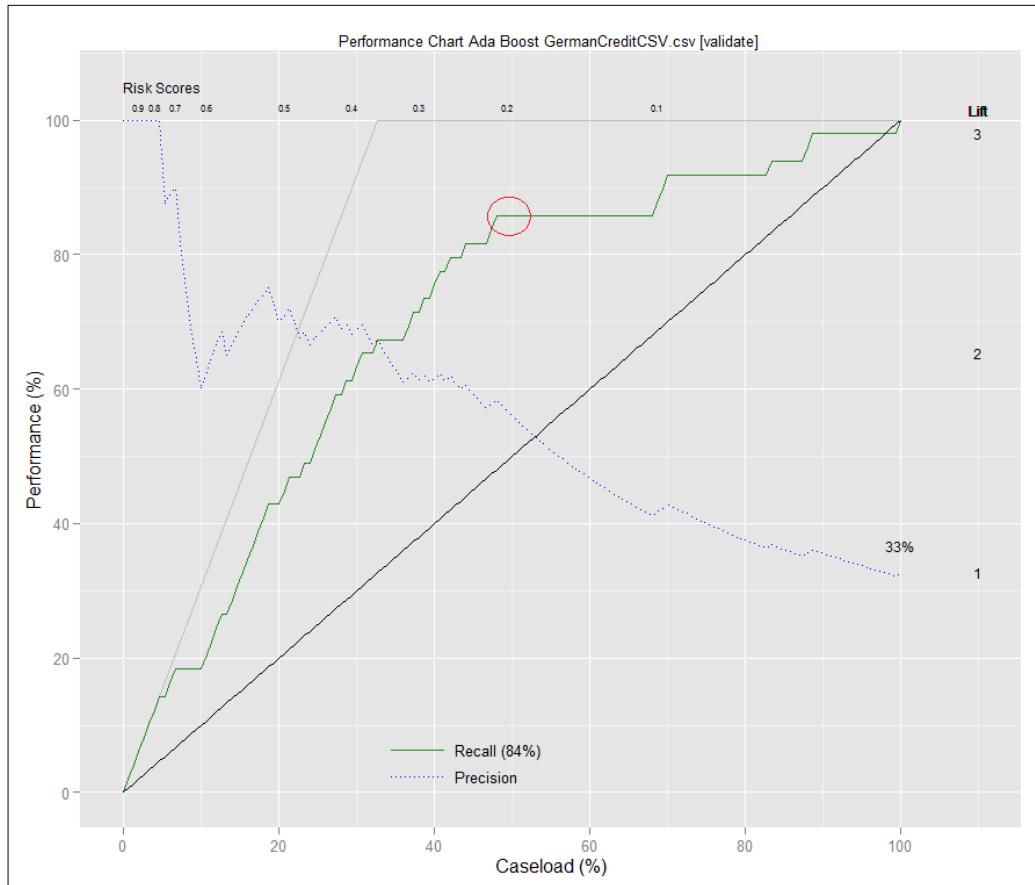
For binary classification models, Risk Chart, or Cumulative Gain Chart, is a good way to measure model performance.

To obtain a Risk Chart, after creating a binary classification model, go to the **Evaluate** tab, choose the **Risk** type and the **Validation** dataset, and press **Execute**, as shown here:



In the following screenshot, we can see a Risk Chart for our credit example. In this example, we've 1000 credit applications, 700 classified as a good risk and 300 classified as bad risk. We have a risk score of 33 percent. The risk of giving credit to an application classified as bad risk is 33 percent.

Imagine we want to use our model to choose some applications to inspect before granting a credit. We would like that our model helps us to choose the most risky applications. The Risk Chart will explain whether our model is appropriate for that task. The following screenshot demonstrates this:



In order to understand a Risk Chart, we need to know two important concepts – **Precision** and **Recall**. In a classification problem, precision is the percentage of positive observations that are correctly classified by the model:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} - \text{False Positive}}$$

A model with a high recall will be able to find the positive observations in our dataset; Recall and Sensitivity are the same. The formula is shown here:

$$Recall = Sensitivity = \frac{True\ Positive}{Positive} = \frac{True\ Positive}{True\ Positive - False\ Negative}$$

In the *y* axes, we can see the percentage of positive observations or **Performance (%)** applications classified as bad risk.

In the *x* axes, we can see the percentage of observations or **Caseload (%)**: 1000 credit applications.

In this plot, the diagonal line is a baseline, if we review credit applications randomly. To review 50 percent of bad risk applications, we need to review 50 percent of applications.

The **Recall** line shows us how the model ranks the applications. If we review 500 applications selected by the model (50 percent of the whole dataset), we'll review approximately 84 percent of the bad risk applications.

ROC Curve

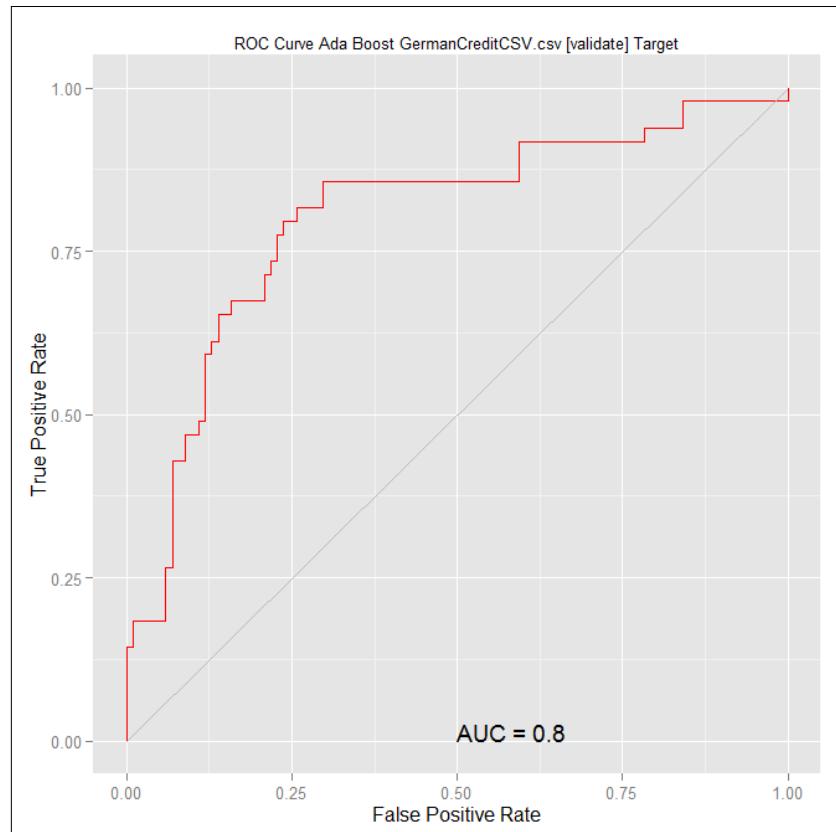
The ROC Curve is a chart that shows the performance of a binary classification model. This chart plots **True Positive Rate** (sensitivity) versus the **False Positive Rate** of our model.

Imagine that we want to develop a binary classification model to classify new loan applications into high risk applications and low risk applications. We have a model that returns the probability of fraud. We need to choose a threshold to split applications between low and high risk. For example, if the probability of fraud is higher or equal than 0.7 we predict *high risk*, and *low risk* if the probability is lower than 0.7. We can try with different thresholds and we'll discover that with higher thresholds, we obtain lower true positive rates and higher false positives rates, and with lower thresholds, we will obtain higher true positive rates and lower false positive rates. Obviously, there is a trade-off between true positive rate and false positive rate; the ROC Curve represents this trade-off for our model and helps us understand the model performance.

A good way to measure the accuracy of a classifier is the Area Under the ROC Curve or **AUC**. An area of 1 represents a perfect classifier; an area of 0.5 represents a random classifier. The ROC Curve of our model will be a number between 0 and 1. A rule of thumb to understand the **AUC** is:

- 1 to 0.90: Excellent
- 0.90 to 0.80: Good
- 0.70 to 0.80: Correct
- 0.70 to 0.60: Poor
- 0.60 to 0.50: Bad

In the following screenshot, you can see the ROC Curve of our credit risk example. The diagonal line is the ROC Curve for a random classifier, and we can use it as a baseline to compare the performance of our model. As you can see in the screenshot, the area under the baseline curve is 0.5. In the screenshot, you can see that the Area Under the Curve or **AUC** is 0.8:



Further learning

In the previous chapter, I recommended that you pursue *Introduction to Data Science*, a Coursera course by Bill Howe, Director of Research Scalable Data Analytics at the University of Washington. The seventh lecture of this course has two interesting videos on Overfitting, Evaluation, and Cross-validation. This is an introductory course and the videos are very intuitive.

Also, a very nice introductory book is *Data Science for Business* written by Foster Provost and Tom Fawcett, O'Reilly Media. This is a great book for a manager who needs to understand data science. The book has a nice chapter about overfitting and model evaluation.

Summary

In this chapter, we saw different ways of analyzing the performance of supervised models. We started with regression model evaluation and then we moved on to classification models performance.

For regression models, we saw that the difference between predicted values and actual values is the most important measure. In this way, Rattle provides the Predicted versus Observed Plot.

We discovered that in classification, a false positive is different from a false negative. Based on this difference, we can create a confusion matrix and evaluate the performance of a classifier using different mechanisms such as a Risk Chart, or ROC Curve.

In this chapter, we've worked with Rattle because it provides the necessary tools to evaluate the performance of a model developed with it. In the next chapter, we'll come back to Qlik Sense to learn the key concepts of data visualization and learn how to communicate with data.

8

Visualizations, Data Applications, Dashboards, and Data Storytelling

In this chapter, we'll focus on data visualization. Representing your data and insights visually will help business users to understand it. A good data application allows the business user to explore data, understand it, and discover new things.

Usually, a dashboard is a visual representation of the most important **Key Performance Indicator (KPI)** of a company, department, or business process. A dashboard can be created on a sheet of paper, in a spreadsheet, or with the help of a data visualization tool such as Qlik Sense.

When you create a dashboard with Qlik Sense, you create a live application. A Qlik Sense application can take the form of a dashboard or an analytic application. As you will see in this chapter, Qlik Sense allows you to combine this dashboard, analysis, and reporting in the same data application.

In data visualizations; charts, tables and other visualizations are the building blocks. We'll start by describing the visualizations provided by Qlik Sense and look at some basic rules to create them. We're not going to explain how to create a chart in Qlik Sense, because we did that in *Chapter 4, Creating Your First Qlik Sense Application*.

After data visualization, we'll discuss how to create a data application. We'll focus on the important features of Qlik Sense and on one particular approach to data application design.

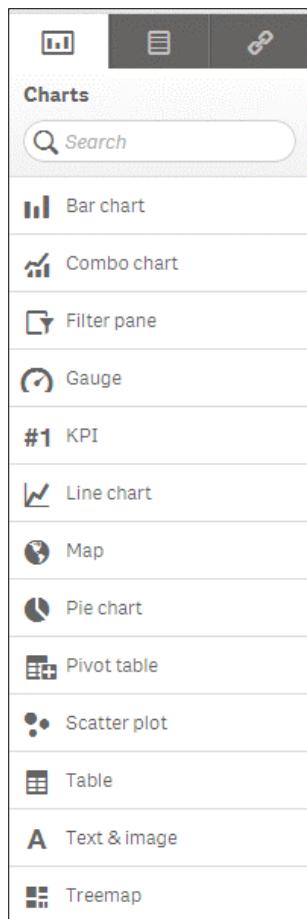
Finally, we'll focus on data storytelling, a way to present your data and conclusions.

Data visualization in Qlik Sense

In *Chapter 4, Creating Your First Qlik Sense Application*, we saw how to create a Qlik Sense application. We also saw how to load data and how to create charts. The objective of this chapter is not to create a lot of charts. We are going to explore, in detail, a bar chart to explore the configuration options and we'll describe the charts provided by Qlik Sense and when to use the different data visualizations. We'll also look at some ideas on how to create data visualizations.

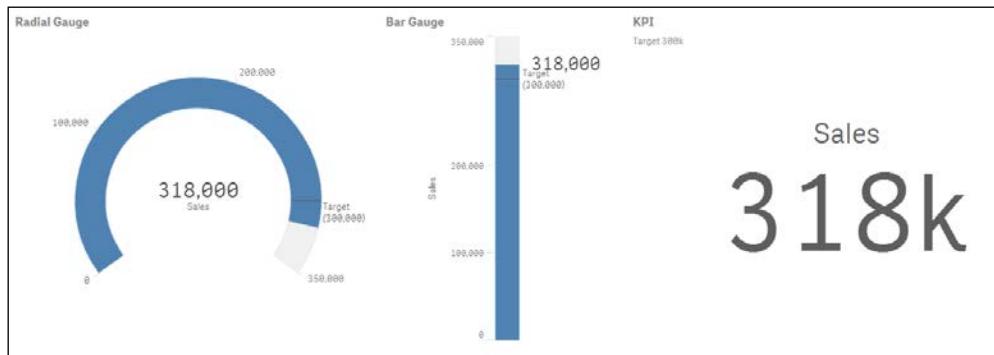
Visualization toolbox

To see all the available options, open a Qlik Sense application in **Edit** mode and you will find all the options in the left hand pane. In the following screenshot, we can see the **Charts pane**:



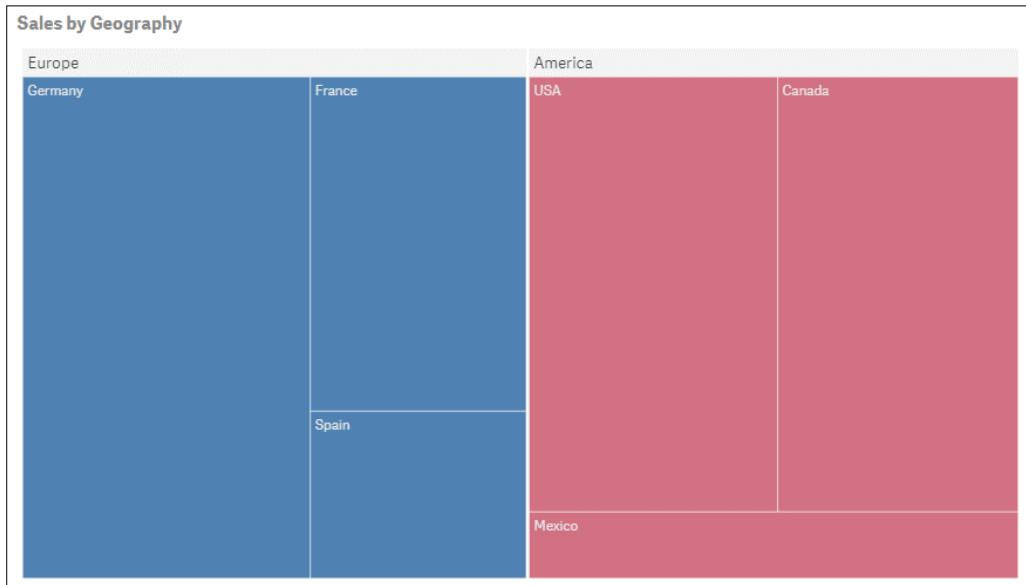
Qlik Sense provides the following default charts:

- **Bar chart:** This is the simplest chart, it helps us to answer questions such as "Who are my best customers?" and "Who are the top performing salespeople?".
- **Combo chart:** In this chart, you can combine bars, lines, and points or symbols. I like to use it to represent metrics of different types such as sales and margins; you can use bars to represent the amount of sales and points to represent the margin.
- **Filter pane:** You can use a filter pane to contain the most common filters such as the year, month, or country; in this way, the user has easy access to the most common filters.
- **Gauge and KPI:** These charts only represents a metric with no dimensions. The Gauge chart shows the metric like a speedometer (**Radial Gauge**) or like a thermometer (**Bar Gauge**) and the KPI chart shows the metric as a text. Be careful when representing a single metric; if a dashboard tells us that the amount of sales is € 318,000, we don't know if the performance is high or low. If you compare the amount of sales with another metric like the target or the last year sales, this will add context to our metric. In the following screenshot you can see the same metric as a Radial Gauge, Bar Gauge, and KPI.



- **Line chart:** This chart is very useful when showing the evolution of a metric over time.
- **Map:** A different way to discover patterns in data is to visualize data in a map. By adding a location to your business data, you are providing a new context that can help you to obtain new insights. In Qlik Sense, there are two types of map polygons or area maps, as well as slippy maps or point maps.
- **Pie chart:** Sometimes, we need to see how a metric such as sales is distributed; the pie chart is a very effective method to show distributions. Be careful, if our dimension has a lot of values, this chart can get confusing.

- **Scatter plot:** To discover if two quantitative values are correlated and to which degree, we can use a scatter plot. With a scatter plot, we can see if there is a correlation between two variables.
- **Table and Pivot Table:** Tables are very useful when studying exact numbers or details.
- **Text & image:** By using text and image objects we can combine text, images, and measures. This is very useful in the main sheet of a dashboard. Usually, you place the headlines in this first sheet, and you can use the font size and type to provide more relevance to a text or value.
- **Treemap:** A treemap is a very useful tool to represent hierarchical (tree-structured) data. In a treemap, each branch is plotted as a rectangle, split by other branches or leaves. Finally, each leaf is plotted as a rectangle, the size of this rectangle depends on the measure's value. In the following screenshot, **Europe** and **America** are two branches and the countries are leaves:



- **Extensions:** Qlik Sense allows developers to create *Extensions*, which are additional developments that extend Qlik Sense functionality. There are several websites where you can find extensions, such as the following:
 - **Qlik Community** (<http://community.qlik.com/>): This is a very active community about Qlik Sense and Qlik View.

- **Qlik Market** (<http://market.qlik.com/>): This is an exchange platform for useful solutions. There are three types of solutions: **Connectors**, **Applications**, and **Extensions**. This site is a very useful resource. Some extensions are free and some of them are chargeable.
- **Qlik Branch** (<http://branch.qlik.com/>): This is a collaborative site for developers to share their projects and extensions. You can find extensions and other developments that add functionalities to Qlik Sense.

You can find detailed instructions on how to use each chart on YouTube. Qlik has a channel called **Qlik** and there is a very good series called **Qlik Sense Desktop Tutorials**.



Qlik Sense charts are Responsible and Smart. The charts adapt themselves to the space to show information to the user in the smartest possible way. These kinds of charts are very useful for mobile devices. We're using Qlik Desktop, which is a PC based tool but, when using Qlik Sense Enterprise, we can access and develop from a PC, tablet, or smartphone and the Smart Charts adapt to the available space. The ability of the charts to adapt to space is very important for mobility purposes. If we access Qlik Sense with a browser from any device, from a cell phone to a desktop, every component in Qlik Sense is able to adapt to the available display screen space.

Creating a bar chart

A good visualization or chart has to be *clear*, *complete*, and *necessary*. The user needs to be able to understand the visualization, for this reason you have to remember the following:

- Avoid complex charts
- Include legends whenever necessary
- Label axes
- Include titles, subtitles, and footnotes wherever necessary

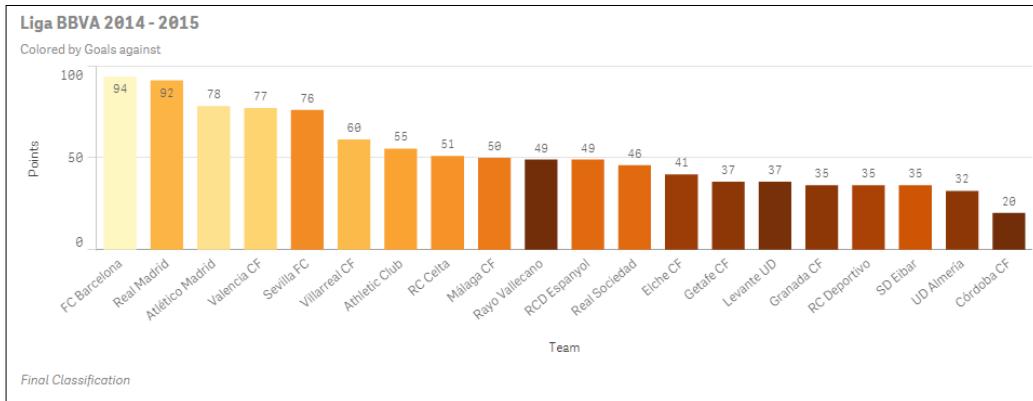
Each visualization is a message and it's important that each chart has a complete meaning of its own. Finally, use only the necessary chart to avoid hiding the important ones in an ocean of plots.

In this section, we will use a simple bar chart to explore most of the configuration options of a Qlik Sense chart. In a bar chart, you have different values side-by-side. This chart is very useful when looking at the difference between dimensions. You can look over your sales in the different regions and it is very easy to compare actual versus planned, or see the differences between regions. You can use this chart for rank analysis or to show the top values, like the best sales representatives or the bestselling products.

To create a bar chart, I've created a small dataset with the final classification of Liga BBVA 2014-2015, the Spanish soccer league, as shown in the following screenshot:

| Rank | Team | Points | Played | Won | Drawn | Lost | Golas For | Goals Against |
|------|-----------------|--------|--------|-----|-------|------|-----------|---------------|
| 1 | FC Barcelona | 94 | 38 | 30 | 4 | 4 | 110 | 21 |
| 2 | Real Madrid | 92 | 38 | 30 | 2 | 6 | 118 | 38 |
| 3 | Atlético Madrid | 78 | 38 | 23 | 9 | 6 | 67 | 29 |
| 4 | Valencia CF | 77 | 38 | 22 | 11 | 5 | 70 | 32 |
| 5 | Sevilla FC | 76 | 38 | 23 | 7 | 8 | 71 | 45 |
| 6 | Villarreal CF | 60 | 38 | 16 | 12 | 10 | 48 | 37 |
| 7 | Athletic Club | 55 | 38 | 15 | 10 | 13 | 42 | 41 |
| 8 | RC Celta | 51 | 38 | 13 | 12 | 13 | 47 | 44 |
| 9 | Málaga CF | 50 | 38 | 14 | 8 | 16 | 42 | 48 |
| 10 | RCD Espanyol | 49 | 38 | 13 | 10 | 15 | 47 | 51 |
| 11 | Rayo Vallecano | 49 | 38 | 15 | 4 | 19 | 46 | 68 |
| 12 | Real Sociedad | 46 | 38 | 11 | 13 | 14 | 44 | 51 |
| 13 | Elche CF | 41 | 38 | 11 | 8 | 19 | 35 | 62 |
| 14 | Levante UD | 37 | 38 | 9 | 10 | 19 | 34 | 67 |
| 15 | Getafe CF | 37 | 38 | 10 | 7 | 21 | 33 | 64 |
| 16 | RC Deportivo | 35 | 38 | 7 | 14 | 17 | 35 | 60 |
| 17 | Granada CF | 35 | 38 | 7 | 14 | 17 | 29 | 64 |
| 18 | SD Eibar | 35 | 38 | 9 | 8 | 21 | 34 | 55 |
| 19 | UD Almería | 32 | 38 | 8 | 8 | 22 | 35 | 64 |
| 20 | Córdoba CF | 20 | 38 | 3 | 11 | 24 | 22 | 68 |

Our objective is to create a chart like the following:



This chart shows us the final classification, the number of points for each team and we used color to represent *goals against*. The chart tells us that FC Barcelona won the league and Real Madrid was second. Real Madrid and FC Barcelona were very close because the final number of points are very similar. In general, the teams with more goals against, perform worse; for this reason the bars on the right are darker than the bars on the left. There are two exceptions: Real Madrid and Rayo Vallecano. Real Madrid have more goals against than Atlético de Madrid or Valencia but have a better placing; probably they scored a lot of goals. Rayo Vallecano have the same issue as Real Madrid; they have a lot of goals against. As you can see, bar charts are very simple but they can provide a lot of information.

From the previous chapters, you know how to load data and how to create a bar chart. In this chart, we used **Team** for the dimension and **Points** for the measure.

To personalize bar charts, we've four main menus:

- **Data**
- **Sorting**
- **Add-ons**
- **Appearance**

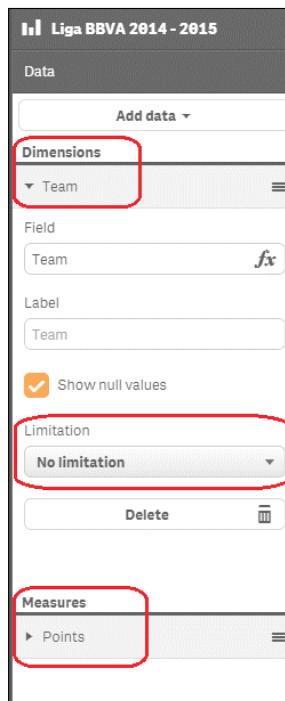
These menus are illustrated in the following screenshot:



In this section, we're going to describe these four menus; they are similar in all charts.

The Data menu

To create a chart to represent the final placing at the end of the season, the first important decision is what to represent in the chart. The dimension is very obvious, we choose the variable **Team**. For measure, there are different options, such as **Rank**, or **Points**. If we use **Rank** as the measure, we'll see the final rank, but if we use **Points**, we'll see the final rank and the difference in points between teams. In this example, we've used **Points** because it provides us with more information:



From the previous chapters, you already know how to set **Points** and **Team** as the measure and dimension. In this example, we're going to focus on a few additional settings that will help us customize the chart.

In the **Dimensions** box, we see two interesting options, as follows:

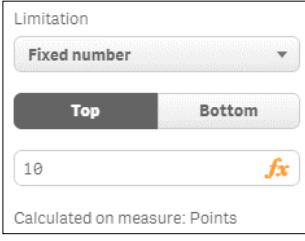
- The **Show null values** checkbox
- The **Limitation** checklist

The **Show null values** checkbox tells us what to do when the value of the dimension is null. In the following piece from the original dataset, we've deleted the name of a football team; if the checkbox is checked, Qlik Sense will place a - in the place of the team name. If the checkbox is unchecked, Qlik Sense will draw RC Celta and RCD Espanyol, and will omit the ninth team, as illustrated here:

| | | | | | | | | |
|----|--------------|----|----|----|----|----|----|----|
| 8 | RC Celta | 51 | 38 | 13 | 12 | 13 | 47 | 44 |
| 9 | | 50 | 38 | 14 | 8 | 16 | 42 | 48 |
| 10 | RCD Espanyol | 49 | 38 | 13 | 10 | 15 | 47 | 51 |

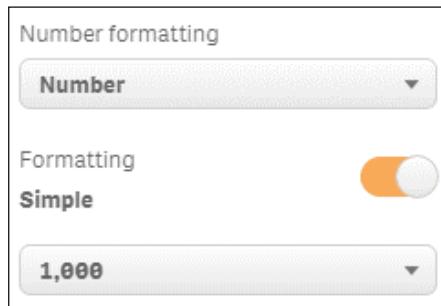
This check is very useful when looking for problems in your dataset. Imagine that you have a dataset with sales transactions; each row contains product, amount sold, customer, and salesman. If you plot a chart with amount sold as the measure and Product, Customer or Salesman as the dimension, a value of amount sold with - as label, this means that, in the original dataset, one or more rows have missing values for the dimension; as happened with the ninth team in our previous example.

The second important setting in dimensions is the **Limitation** listbox. This setting limits the number of values that Qlik Sense plots. There are four possible options, as depicted in the following table:

| Limitation | Illustration | Description |
|---------------|---|---|
| No limitation |  | Qlik Sense will plot all possible dimensions; in our example Qlik Sense will plot all teams. |
| Fixed number |  | Qlik Sense will plot a fixed number of values (teams). This option is useful to create a Top 10 analysis. |

| Limitation | Illustration | Description |
|-----------------------|--|---|
| Exact value | <p>Limitation</p> <p>Exact value</p> <p>>= > < <=</p> <p>0 fx</p> <p>Calculated on measure: Points</p> | Qlik Sense will plot only the Teams that have a number of points that agree to the condition. |
| Relative value | <p>Limitation</p> <p>Relative value</p> <p>>= > < <=</p> <p>0% fx</p> <p>Calculated on measure: Points</p> | Qlik Sense will plot only the Teams that have a number of points that agree to the condition. |

An important setting for the measure is **Number formatting**. By default, this setting option is **Auto**; generally Qlik Sense selects the best option, but if you want to be sure how the numbers are going to be formatted, you can set it however you wish. An example setting is shown here:



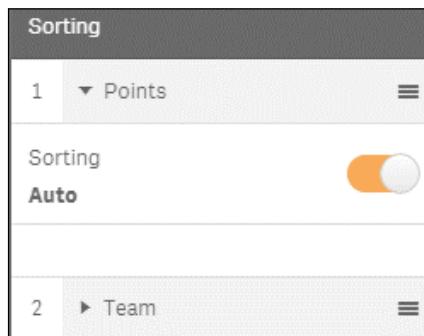
You can choose between these different **Number formatting** settings:

- **Auto**
- **Number**
- **Money**
- **Date**
- **Duration**
- **Custom**

For **Money**, **Date** and **Duration**, Qlik Sense Desktop takes the format from the first lines in **Load Script**. Qlik Sense Desktop automatically creates these lines from your computer's local settings. You can use the **Custom** option to create your own formatting mask.

The Sorting menu

The **Sorting** menu allows us to personalize how values are sorted in our bar chart. This menu looks like this:



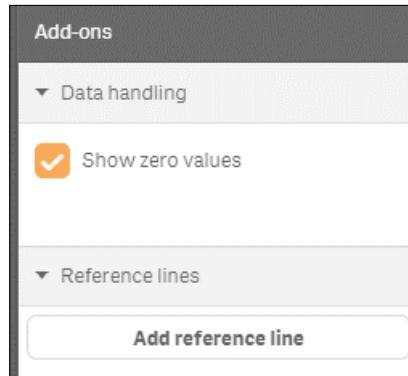
In this example, we can sort the values by the measure, **Points**, or by the dimension, **Team**. Qlik Sense Desktop chooses the top variable to sort the values; in this case we will use **Points** to sort the values. If two teams have the same amount of points, they will be sorted by the second variable; in this case, **Team**.

If we place **Team** at the top of the variable list, the values will be sorted alphabetically by the name of the team.

As you can see in the previous screenshot, Qlik Sense Desktop sets the sorting type, by default, to **Auto**. You can set this option to **Custom** and personalize how you want to sort the values.

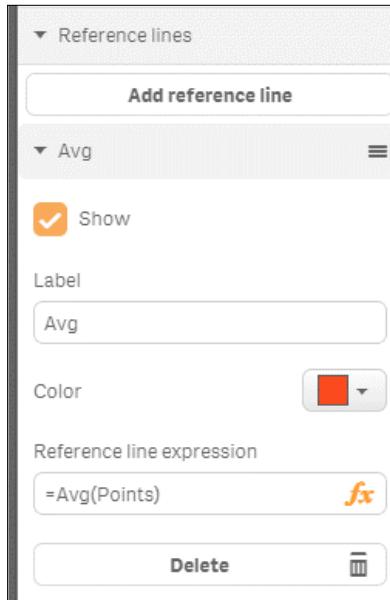
The Add-ons menu

This menu provides us with the personalization options, **Data handling** and **Reference lines**, as shown in this screenshot:



Data Handling has only one option: **Show zero values**. If this option is checked, as shown in the preceding screenshot, Qlik Sense Desktop plots a team with zero points. If we uncheck this option, Qlik Sense Desktop does not plot a team with zero points.

With **Reference lines** we can add horizontal lines to our bar chart. In the following screenshot, you can see how to set a line by taking the points average:



The Appearance menu

The **Appearance** menu has five options, as listed here:

- **General**
- **Presentation**
- **Colors and legend**
- **X-axis**
- **Y-axis**

This menu is illustrated in the following screenshot:

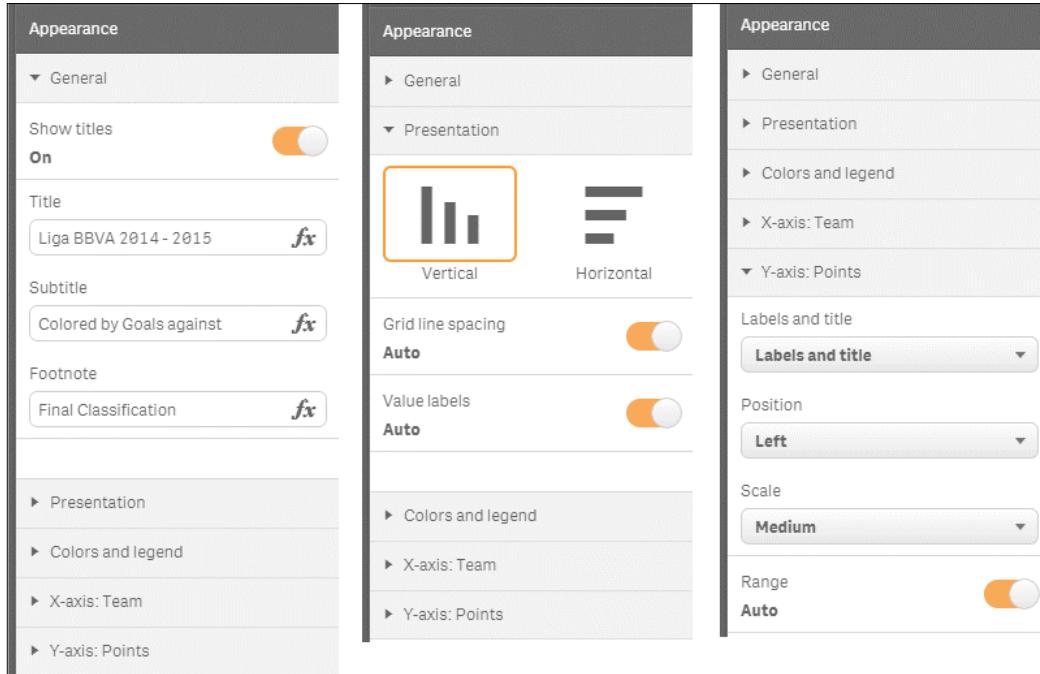


Under the **General** submenu, there are the **Title**, **Subtitle**, and **Footnote** options. When you see an **fx** symbol in the textbox, it means that you can use an expression. We're going to use this functionality in the following example.

In the **Presentation** submenu, we can choose between **Vertical** and **Horizontal** orientation, the type of grid, and whether we use the **Value labels** switch, to plot the value of each bar in the bar chart. In this example, if you turn on this switch, the amount of points for each team will be plotted in each bar.

The **X-axis** and **Y-axis** submenus allow us to personalize labels and titles for each axis. If the variable is numeric, such as **Points**, we can also personalize the axis range.

The **Appearance** menu is further illustrated as follows:



Finally, using the **Colors and legend** submenu, we can personalize the color of each bar. Qlik Sense Desktop offers us four options:

- **Single color:** All bars have the same color.
- **By dimension:** Each different value in the dimension variable has a different color. In our example, each team has a different color. We used this option in *Chapter 4, Creating Your First Qlik Sense Application*.

- **By measure:** In our example, Qlik Sense Desktop uses the variable **Points** to color each bar.
- **By expression:** In this option, we can use a new expression to color the bars. This option is very useful because it allows us to code more information in the same plot. In this example we used the variable **Goals Against**; in this way we discovered that Real Madrid and Rayo Vallecano conceded an unusual number of goals. The expression can return a number, as in this example, or a color code, as in the example in *Chapter 5, Clustering and Other Unsupervised Learning Methods*.

In the next section, we're going to use title and subtitle expressions to add information to our chart.

Data analysis, data applications, and dashboards

Before creating a data application, you have to define who is going to use your application and what the objective of the application is. Patricia L. Saporito, in *Applied Insurance Analytics*, Pearson Education LTD defines three main user groups:

- **Executive management:** They need key metrics, very visual applications and should be able to access the analysis from different devices
- **Middle managers:** They need key metrics and the ability to navigate from aggregate data to detailed data
- **Analysts:** This group needs to be able to manipulate data on the lowest grain and create new metrics

You also need to be familiar with the metrics you are going to use in your application. A special type of metric is a **Key Performance Indicator (KPI)**. KPIs explain the performance of a business process or activity; Sales versus Objective would be a good KPI for a **Sales Performance** dashboard. As you can see, KPIs are metrics that illustrate performance but, to understand the reasons of this performance, we need intermediate metrics; Opportunity Win Rate would be a good metric to help us to understand the Sales versus Objective metric.

In this section, we'll discuss how to structure data applications with Qlik Sense Desktop. We'll start by describing the important data analysis characteristics of Qlik Sense Desktop. Then we'll focus on how to structure a Qlik Sense Desktop application.

Qlik Sense data analysis

Qlik Sense and QlikView are different tools. QlikView is a great tool for developing guided analytic applications, and Qlik Sense is a tool for self-service visualization and data analysis, but QlikView and Qlik Sense share the same principles in their analytic engine. Qlik Sense is a new tool but its analytic engine is the new version of the successful QlikView engine. In this section, we'll review the most important characteristics of this engine and how these characteristics influence our analysis.

In-memory analysis

Traditional **Business Intelligence (BI)** or data analysis tools access the data through a database. Databases are based on filesystems, and data access in a filesystem is not fast. For this reason, traditional BI doesn't provide a great interactive experience. When a user makes a selection, the BI tool executes a query to the database, the database then needs to access a lot of rows, sometimes millions of rows in the filesystem and aggregate them to provide the result; this entire process is time consuming and the user has to wait for the information. The result is that the user perceives a lack of interactivity and doesn't use the tool. To avoid this, BI tools aggregate data to reduce the number of rows. For example, in a sales analysis application, the raw data is sales tickets; each row in the dataset is a line ticket. If you aggregate the sales information by store and day, you have the amount of sales in a week for a given store. You have reduced the number of rows and you will improve your response time, but you have lost information and your analysis will be limited.

In order to provide a good level of interactivity and avoid aggregating data, Qlik Sense works in a different way to the traditional BI. When you open a Qlik Sense application, the engine loads all the data for you to the RAM. This memory is very fast and Qlik offers great performance without aggregating data. Thanks to its *in-memory* approach, Qlik Sense is a very interactive tool and we can take advantage of the most detailed atomic data. Keeping the most atomic data in our dataset allows us to develop our data application using a **Dashboard Analysis and Reporting (DAR)** approach.

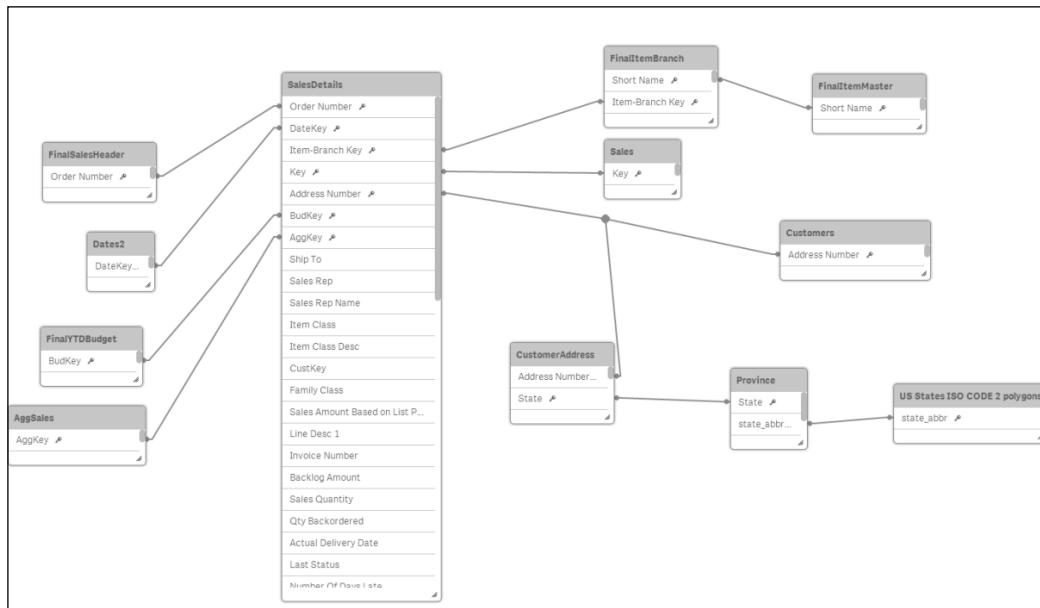
Qlik introduced in-memory analysis in 1993. Today, other vendors are developing in-memory tools to analyze data. In my opinion, the QIX Associative Data Indexing Engine makes best use of these decades of experience.

Associative experience

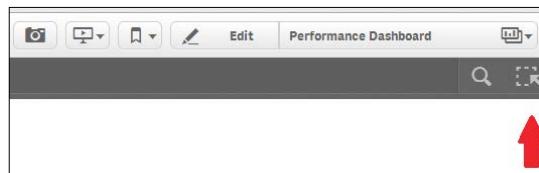
In a traditional database data model, different tables are linked together with pre-declared keys and the user only can explore his/her data through predefined paths.

The Qlik Sense engine uses a different approach. In Qlik's engine, all data is associated, and the user can consult the analytic engine without any predefined paths. After installing Qlik Sense, you have three example applications. We're going to use an example application to understand the magic of associative logic. Open Qlik Sense, open the **Sales Discovery** app, and open the Data model.

The data model is organized around a table called **SalesDetails** that contains the most atomic data of a sale – price, quantity, discount, product, customer, and so on. This table contains the field we want to measure and we call it the **fact table**:



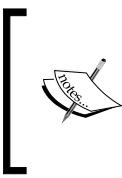
Our objective is to obtain Californian customers who bought Alcoholic Beverages in January 2014. Go to **App overview** and open any sheet in the application. In the top right square, click the squared icon to open a filter screen, marked by the red arrow in the following screenshot:



As you can see in the following screenshot, we use the search box to look for a filter. Enter **Year** in the search box; Qlik Sense will give you the **Year** filter, select **2014**. Repeat this for **Alcoholic Beverages**, **CA**, and **Jan**. Finally, enter **customer** in the search box to obtain the **Customer** filter; in this filter, you see a white list of customers. These customers are associated with **Jan, 2014, CA, Alcoholic Beverage**; this means that these customers have bought **Alcoholic Beverages** in California. After this white list, there is a grey list of customers. These customers are not associated with my selections; this means that these customers have not bought **Alcoholic Beverages** in January **2014** or they are not from California:

A screenshot of the Qlik Sense filter interface. At the top, it says "CURRENT SELECTIONS". There are four dropdown filters: "Year" (2014 selected), "Product Group Desc" (Alcoholic Beverages selected), "state_abbr" (CA selected), and "Month" (Jan selected). Below these, there is a search bar with the placeholder "customer" which has a red oval around it. On the left side, under "APP DIMENSIONS", there is a list of various companies and organizations, with "Customer" being the first item and also having a red oval around it.

Thanks to the associative logic, a business user can freely explore his/her information and obtain answers to his questions. Now try posing and answering your own questions.



A lot of people has written articles about the Associative Experience, I especially like the one written by *Michael Tarallo*, *The Associative Experience - Revisited*: <https://communityqlik.com/blogs/theqlikviewblog/2014/04/10/the-associative-experience--revisited>

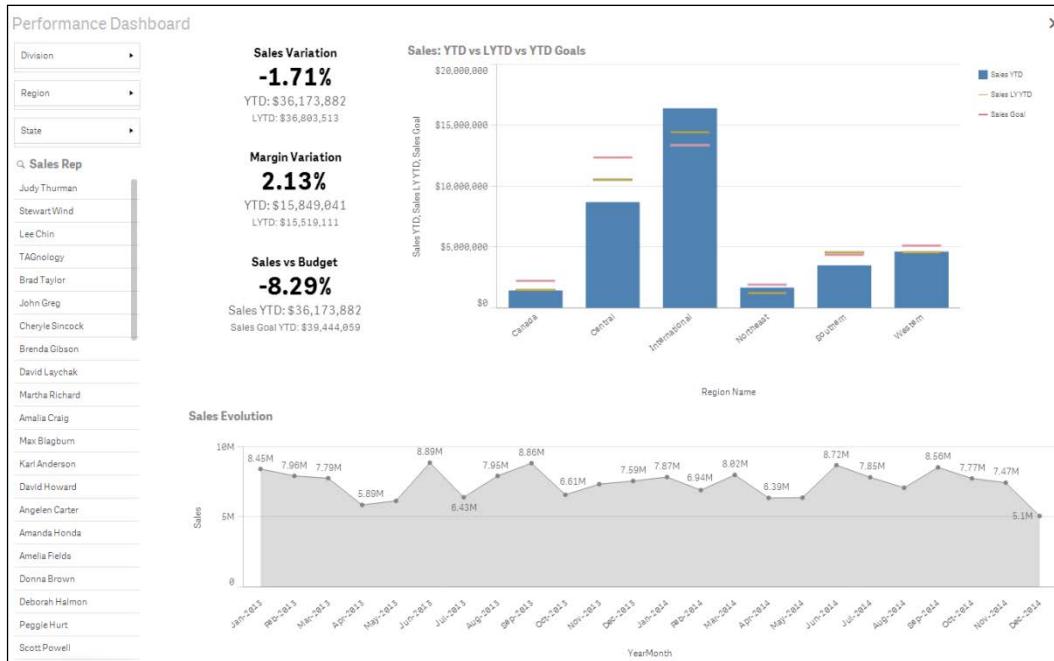
Data applications and dashboards

There are different approaches to designing data applications. As we've seen, the design of the application is defined by the audience, the business objective, and the data, but a very popular approach in QlikView and Qlik Sense is **DAR**.

The DAR approach

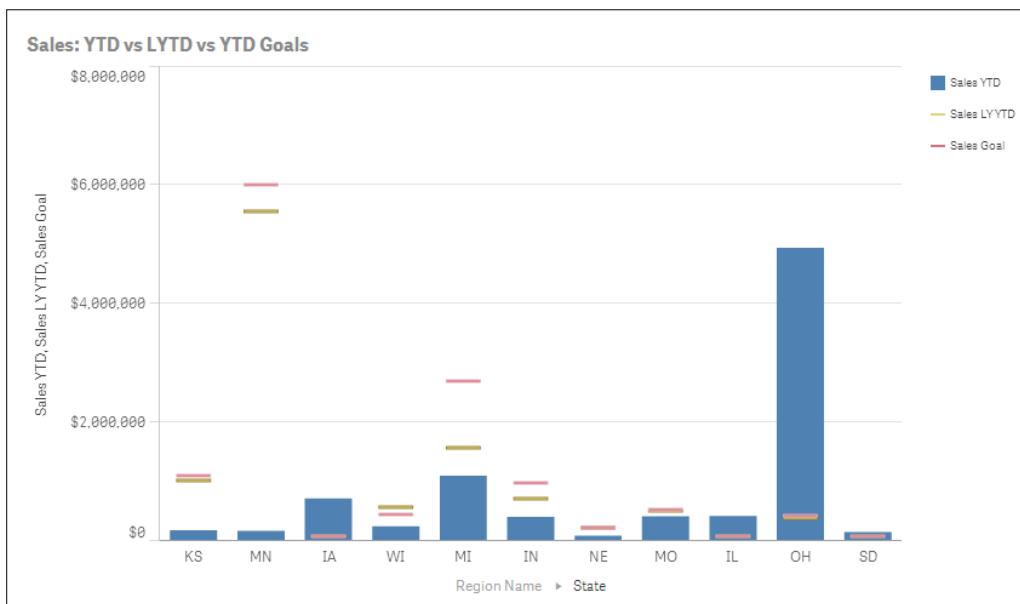
DAR stands for **Dashboard, Analysis and Reporting** and is a very useful approach proposed by Qlik to develop data analysis applications in Qlik Sense. We'll use the **Sales Discovery** app to explain this concept.

In a DAR application, the first sheet offers us the headlines of our application. This part of the application is the dashboard. In the following screenshot, you can see the **Performance Dashboard** sheet from the **Sales Discovery** application:



In the central area, we can see that, during 2014, the company sold **1.71%** less than in the same period of 2013, the margin was **2.13%**, and sales were **8.29%** below target. We can also see a bar chart that explains how the different regions are performing. This sheet is designed to explain performance and where we have to focus our attention. Just by looking at this sheet, we can see that sales performance is very low and that the **Central** region is the one with the worst performance.

In the dashboard we focus only on a few KPI, as in the headlines in a newspaper. Users will use this sheet to discover a performance gap, and then they'll move to the analysis part. In the same initial sheet, select the **Central** region and the sheet will become the **Performance Dashboard** for the **Central** region. Now you can see that sales are **30.07%** below budget and the following bar chart shows the performance of the states in the **Central** region. This chart tells us that we should focus on Minnesota (**MN**) so select **MN** in the bar chart:



Now, to continue analyzing this low performance in Minnesota, go to the **Top Customers** sheet. You've moved to a different sheet but Qlik Sense keeps your current selection. You are in the **Top Customers** sheet, analyzing data from **Central Region** and Minnesota. In the top bar of the screen, you can see the filters, as shown in this screenshot:



The **Top Customers** bar chart shows the best customers from 2013 and 2014. Now select the year **2013** and then the year **2014** and compare the Top Customers. You can see that the difference in sales between **2013** and **2014** is that **Paracel** and **Renegade info Crew** were the best customers in 2013, but in 2014 they bought less, as depicted in the following screenshot:



We started looking at sales deviation in our dashboard. Then, in the analysis phase, we discovered the reasons for this deviation.

Finally, we're going to enter into the reporting phase where we'll obtain the most granular information that is responsible for the deviation.

In this case, I would like to find out the details of purchases of these customers. Select these customers and go to the **Transactions** sheet. At the top of the screenshot, you can see that we've selected the **Central** region, the **MN** state, **Paracel**, and **Renegade info Crew**. In the following screenshot, you can see the table of all transactions in Minnesota by these customers:

The screenshot shows a Qlik Sense dashboard titled "Transactions". At the top, there are three filters: "Region Name" set to "Central", "State" set to "MN", and "Customer" set to "Paracel, Reneg...". On the left, there is a sidebar with a "Division" dropdown, a "Region" dropdown (set to "Sales Rep"), and a "State" dropdown. Below these are several names listed under "Sales Rep": Judy Thurman, John Greg, Ronald Milam, Amalia Craig, Amanda Honda, Amelia Fields, Angela Carter, Bima Malek, Brad Taylor, Brenda Gibson, and Brenda Kaylor. The main area displays a table of transaction data with the following columns: Invoice Number, Order Number, Customer, Promised Delivery Date, Sales, and Sales Quantity. The table includes a header row and a "Totals" row at the bottom. The data shows various transactions for Paracel customers across different dates and times.

| Invoice Number | Order Number | Customer | Promised Delivery Date | Sales | Sales Quantity |
|----------------|--------------|--------------------|------------------------|-------|----------------|
| 100102 | 200114 | Paracel | 6/10/2004 | 38148 | (\$775.00) |
| 100264 | 200290 | Paracel | 7/22/2004 | 38198 | (\$340.48) |
| 100534 | 200573 | Paracel | 10/9/2004 | 38269 | (\$544.97) |
| 100542 | 200582 | Paracel | 10/10/2004 | 38278 | (\$1,269.78) |
| 100779 | 200988 | Paracel | 4/30/2004 | 38107 | \$9,843.00 |
| 100780 | 200998 | Paracel | 4/30/2004 | 38107 | \$53,626.00 |
| 100781 | 201056 | Paracel | 4/30/2004 | 38107 | \$18,683.00 |
| 100781 | 201056 | Paracel | 4/30/2004 | 38114 | \$0.00 |
| 100782 | 201057 | Paracel | 4/30/2004 | 38107 | \$18,603.00 |
| 100782 | 201057 | Paracel | 4/30/2004 | 38114 | \$0.00 |
| 100797 | 201284 | Paracel | 4/30/2004 | 38107 | \$71,915.00 |
| 101425 | 200888 | Paracel | 5/7/2004 | 38114 | \$12,426.00 |
| 101428 | 200982 | Paracel | 5/7/2004 | 38114 | \$0.00 |
| 101443 | 201284 | Paracel | 5/7/2004 | 38114 | \$22,710.00 |
| 102023 | 201045 | Renegade info Crew | 5/13/2004 | 38120 | \$0.00 |
| 102183 | 201865 | paracel | 5/14/2004 | 38121 | \$1,904.00 |
| 102183 | 201865 | Paracel | 5/14/2004 | 38128 | \$0.00 |
| 102187 | 201991 | Paracel | 5/14/2004 | 38114 | \$0.00 |
| 102187 | 201991 | Paracel | 5/14/2004 | 38123 | \$1,426.57 |

As we've seen, we started at the dashboard level where we saw low performance. Then we analyzed this deviation and, finally, we reported the details or the reasons for this deviation.

Data storytelling with Qlik Sense

Generally, when you present your conclusion to a group of executives, your presentation competes for their time and attention with a lot of different things. Using your data to explain a story is powerful because stories have the following characteristics:

- Memorable
- Impactful
- Personal

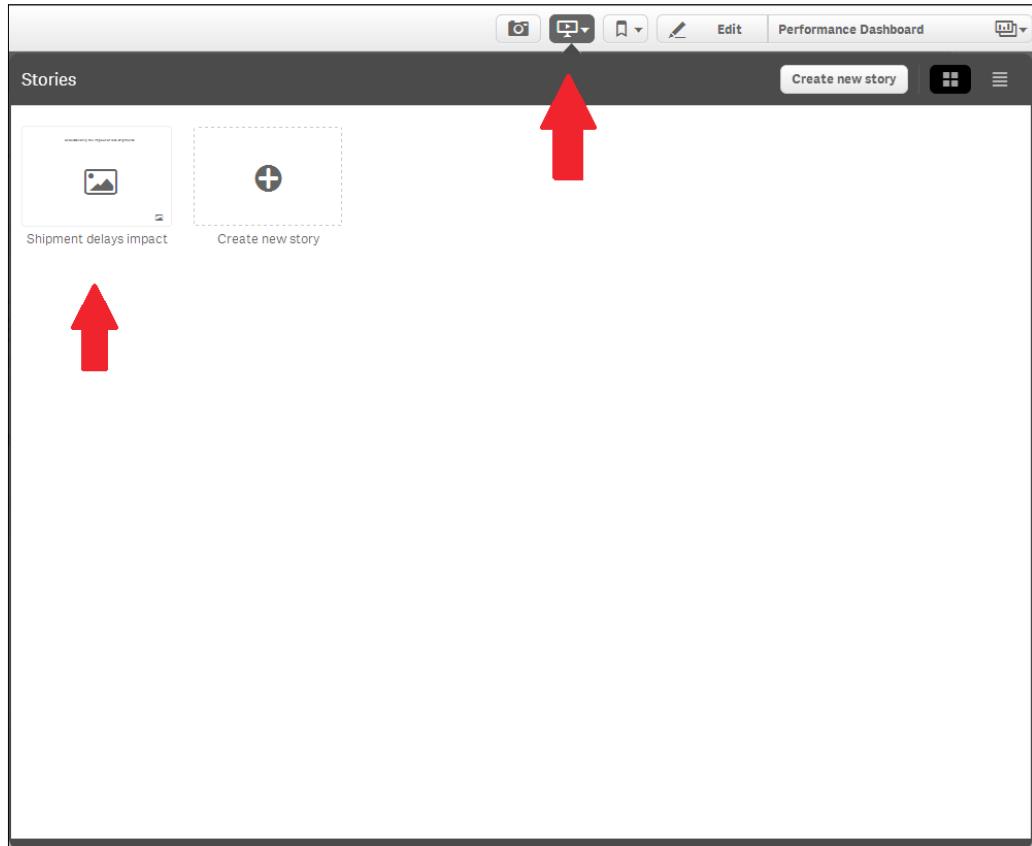
Using storytelling to present your data or insights can help you to do the following:

- Keep the interest of your audience
- Explain complex concepts
- Convince people
- Make your presentation memorable

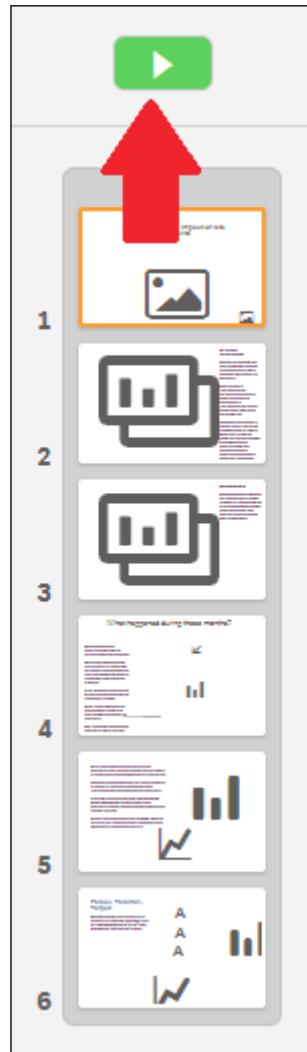
There is a lot of good literature around storytelling and data storytelling. The objective of this section is just present this Qlik Sense functionality but I would like to share my trick to prepare presentations. Usually, in our daily work, we need to prepare a presentation within a limited preparation time. Personally, I use a notepad (not a digital one) and I write down my outline in seven fundamental building blocks:

- **Audience:** Everything starts with the audience. A presentation for IT is very different from a presentation for a business unit. Ask yourself, what are the concerns of your audience? Why are they interested in your story?
- **Objective:** What is your objective? Do you want your audience to make a decision? Is it just an informative presentation? Having your objective in mind is really important when creating your story.
- **Key messages:** Time and attention are very limited; writing down your key messages will help you to focus your story.
- **The story:** Draw your story and choose the data visualization you are going to use. Choose the visualizations that help you to explain your story. Are they relevant? Take time restrictions into consideration.
- **The link between the slides or between visualizations:** Links between slides are sentences that avoid breaking the flow of the story between slides. Different slides in a story can break the flow of the story and make your presentation just a collection of slides. For this reason, I prepare every link between two slides.
- **Review everything:** After we have noted down the preceding building blocks, we have to ask ourselves the following questions:
 - Are the key messages clearly explained in the story?
 - Are you focusing only on key messages?
 - Do the visualizations support the story?
 - Are the visualizations clear?
 - Does the story support the objective and the key messages?
 - Is your story and its content suitable for your audience?

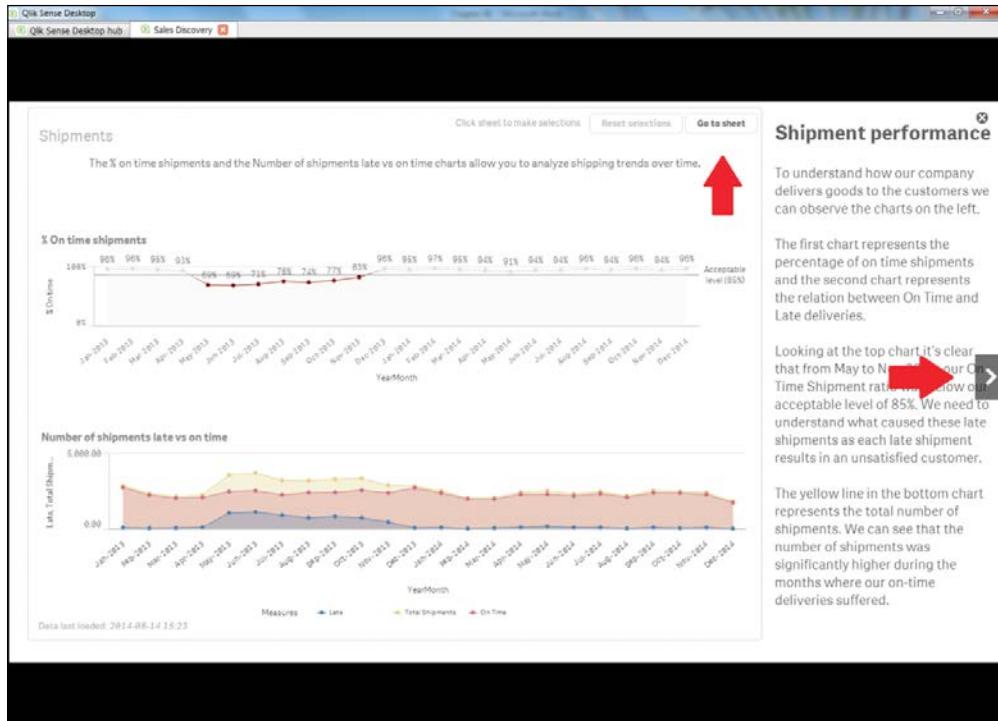
Qlik Sense Desktop has a great tool for data storytelling. To open a data story, click the **Stories** button and choose the **Shipment delays impact** story, as shown in this screenshot:



As you can see in the following screenshot, press the **Play** button to start the story:



The presentation mode has arrow buttons at the right and at the left, to go forward or backward in your presentation. When you present your insights, people will have questions about the data; by pressing the **Go to sheet** button, you can open the data application and show them the data, as illustrated in this screenshot:

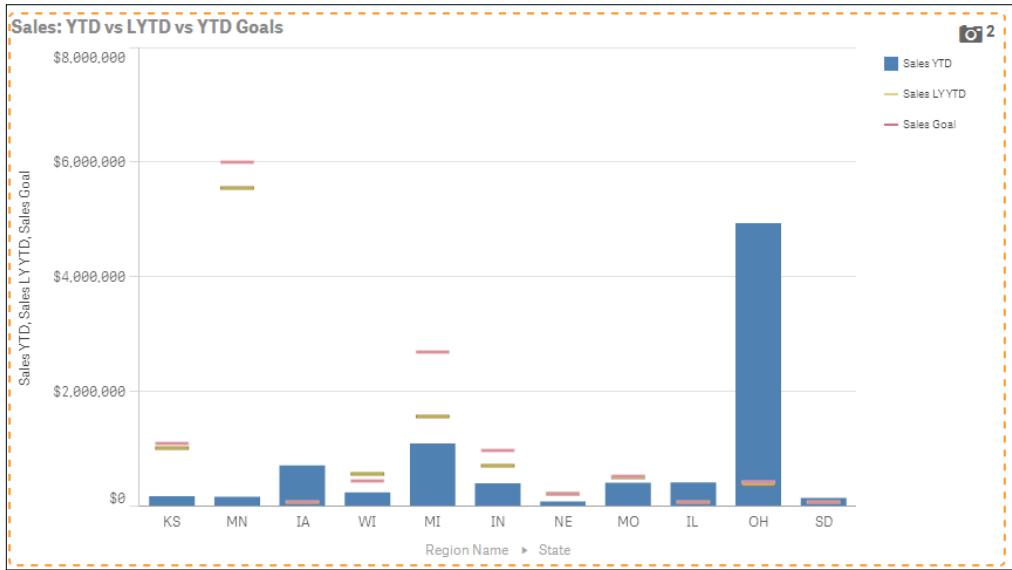


Creating a new story

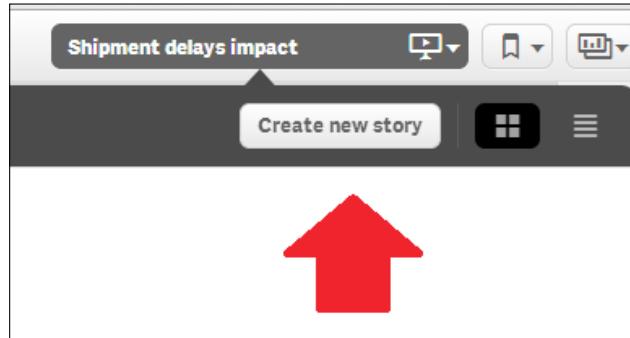
To create a new story, you need to take some snapshots of your data visualizations. Go to the application, open a sheet, and press the camera button, marked by the arrow:



Now the charts are in snapshot mode and the visualizations are highlighted with a dotted line; you just need to click on the chart and Qlik Sense will take a snapshot that you will be able to use to create a story, as shown here:



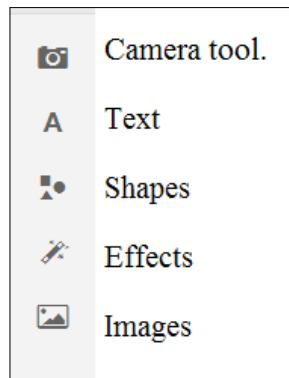
Finally, we have some data snapshots we can use to create a new story. We have to go to the top area of the screen, press the **stories** button, and press the **Create new story** button:



Choose a name for the story and start creating it. To create your story, you have five main components:

- **Camera Tool:** Using this tool, you can capture the charts
- **Text:** Using this tool, you can add titles or paragraphs
- **Shapes:** Use this tool, to point to any data
- **Effects:** You can use this tool to add effects to the charts to highlight special data
- **Images:** Using this tool, add images from the media library

The following screenshot shows the five components:

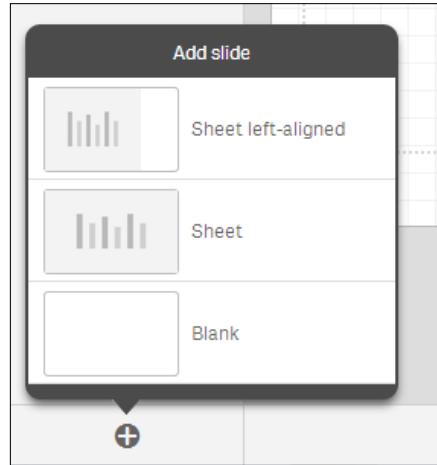


Now you can create a sheet using these components just by dragging and dropping the components into the sheet.

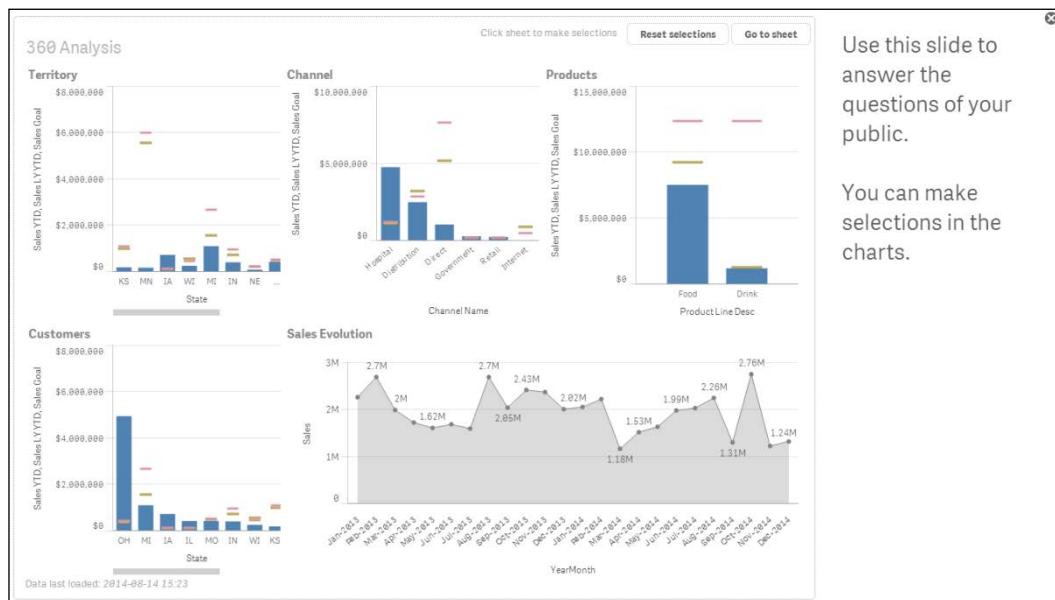
We've added a blank slide and we've added charts, effects, text, and images but we can also add live slides by embedding a sheet into a slide. In the bottom left area, click the plus sign to add a new slide. There are three types of slides:

- **Blank:** This is a slide like the one we've just created
- **Sheet left-aligned:** This is a slide with a left aligned embedded sheet
- **Sheet:** This is a slide with an embedded sheet

These slides are illustrated in the following screenshot:



Select **Sheet left-aligned** or **Sheet** and choose a sheet for your slide. You will get a blank sheet into which you can drag and drop components, as well as customizing them according to your preference. The following screenshot shows a **360 Analysis** sheet that has been customized:



The **Go to Sheet** button and the inclusion of a live sheet in a slide give the opportunity to bring your story to life and handle audience questions.

Further learning

We've introduced data visualization. If you are interested in improving your knowledge of this huge topic, Stephen Few and Edward R. Tufte are probably the best authors to consult. Some great books on this topic are:

- *Learning QlikView Data Visualization* by Karl Pover, PACKT Publishing.
This is a QlikView book but I had a great time with it and the principles are still valid.
- *Information Dashboard Design, Displaying data for at-a-glance monitoring*,
Stephen Few, Analytics Press.
- *Visualize this The FlowingData Guide to Design, Visualization and Statistics*,
by Nathan Yau, Wiley Publishing, Inc.

We've seen the **DAR** approach or **Dashboard, Analysis and Reporting** relating to the design of data applications.

To gain a deeper understanding of the DAR methodology, have a look at a technical paper on the Qlik Design Blog:

<https://community.qlik.com/blogs/qlikviewdesignblog/2013/11/08/dar-methodology>

Finally, related to data storytelling, I found this book helpful:

Strategic Storytelling, How to Create Persuasive Business Presentations, by Dave McKinsey, CreateSpace Independent Publishing Platform.

Summary

In this chapter, we've focused on Qlik Sense and how to communicate insights and the analyses you've made.

We started with data visualization; we created a bar chart and discussed other charts.

After data visualization, we discussed Qlik Sense functionalities which make it a great tool, not just for data visualization, but also for data analysis. These characteristics are the in-memory engine, the associative logic, and the fact that Qlik Sense doesn't need to pre-aggregate data; in Qlik Sense, you can load atomic data in memory and you can aggregate at runtime.

We've also seen the **DAR** approach or **Dashboard, Analysis and Reporting** in the design of data applications.

Finally, we defined data storytelling and reviewed Qlik Sense data storytelling capabilities.

In the next chapter, we'll create a data application. We'll start by defining the planning of the application, exploring the data, creating our predictions with Rattle, and finally, we'll create a data application using Qlik Sense.

9

Developing a Complete Application

A business needs to forecast demand in order to plan supplies and resources. Some industries have been using predictive analytics to forecast demand for many years and, with today's analytics revolution, a lot of new organizations are taking advantage of these techniques to do it.

In this chapter, we'll use a bike sharing dataset to create an application to analyze rental activity, and after this, we'll add information on demand forecasting by using Linear Regression.

We'll divide this chapter into four sections:

- Initially, we'll download and study our dataset. To explore the dataset we'll use Qlik Sense. We've previously used Rattle to explore our data, but as I've mentioned, you can explore the dataset with both tools depending on your preference. I prefer Qlik Sense for exploration. After this chapter, you'll be able to decide which tool you prefer for exploration.
- After we understand the data, we'll build a Qlik Sense application that will help users to explore the data.
- After creating the Qlik Sense application, we'll load the data into Rattle to add demand forecasting information.
- Finally, we'll add the forecast information to our Qlik Sense application.

Understanding the bike rental problem

As in other examples, we're going to use a dataset from the School of Information and Computer Science at the University of California. The dataset is taken from the location specified here:



Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science.



In this chapter, we will use the Bike Sharing Dataset. You can access the dataset here <https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>. This dataset contains the hourly and daily count of rental bikes between the years of 2011 and 2012 in the Washington bikeshare system, with the corresponding weather and seasonal information. This dataset was originally taken from the location specified here:



Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.



Bike sharing systems allow people to borrow a bike from a station and return it to another station on a very short term basis.

In this bike sharing system, there are two types of users – registered and casual. **Registered** users are long-term or mid-term members of a bike sharing program. Usually, **casual** users are members of the program for just one day.

The objective is to create an application that enables control of the activity of the bike sharing system and the following week's forecast based on weather and seasonal information.

Applications that control activity are useful to different kinds of businesses. If you have information about the cost and income related to this activity, you can create an application that controls activity, costs, incomes, and margins. This application could be very useful when optimizing processes.

Demand forecasting is very common in predictive analytics. We can use different methods to forecast the demand of bike rental, but we're going to use Linear Regression. With this last example, we have now covered three main techniques – K-means for Clustering, Decision Trees, and Linear Regression.

We introduced Linear Regression in *Chapter 6, Decision Trees and Other Supervised Learning Methods*, as a method to find a statistical model that describes past data. In this way, we can use the model to predict new data.

The dataset contains two files, `day.csv` and `hour.csv`. The first file contains rental, seasonal and weather data aggregated by day. The second file contains the same data, but aggregated by hour:

- `instant`: This is the recording index
- `dteday`: This is the date
- `season`: This is the season (1: spring, 2: summer, 3: fall, 4: winter)
- `yr`: This is the year (0: 2011, 1: 2012)
- `mnth`: This is the month (1 to 12)
- `hr`: This is the hour (0 to 23)
- `holiday`: This states whether the day is a holiday or not
Source: <http://dchr.dc.gov/page/holiday-schedule>
- `weekday`: The day of the week
- `workingday`: If the day is neither a weekend nor a holiday, then it is 1, otherwise it is 0
- `weathersit`: This provides the following four options:
 - 1: Clear, Few clouds, Partly cloudy
 - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
 - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
 - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- `temp`: The normalized temperature in Celsius, the values are divided by 41 (max)
- `atemp`: The normalized feeling temperature in Celsius, the values are divided by 50 (max)
- `hum`: The normalized humidity, the values are divided by 100 (max)
- `windspeed`: The normalized wind speed, the values are divided by 67 (max)
- `casual`: The count of casual users
- `registered`: The count of registered users
- `cnt`: The count of total rental bikes, including both casual and registered

This dataset has three dependent variables, `casual`, `registered` and `cnt`; 13 independent variables, `dteday`, `season`, `yr`, `mnth`, `hr`, `holiday`, `weekday`, `workingday`, `weathersit`, `temp`, `atemp`, `hum` and `windspeed`, and one identity variable, `instant`.

If we can find the relationship between the weather variables and the number of users, we'll be able to predict the demand using a weather forecast.

For this example, we're going to use the daily file and the dependent variable `cnt`. We're going to ignore the hourly information and `registered` and `casual` variables.

In order to add some context to our analysis, we will assume a target of 4,000 users per day. We have the same capacity (the number of bikes) every day, so on the days with less than 4,000 users, we are wasting our capacity.

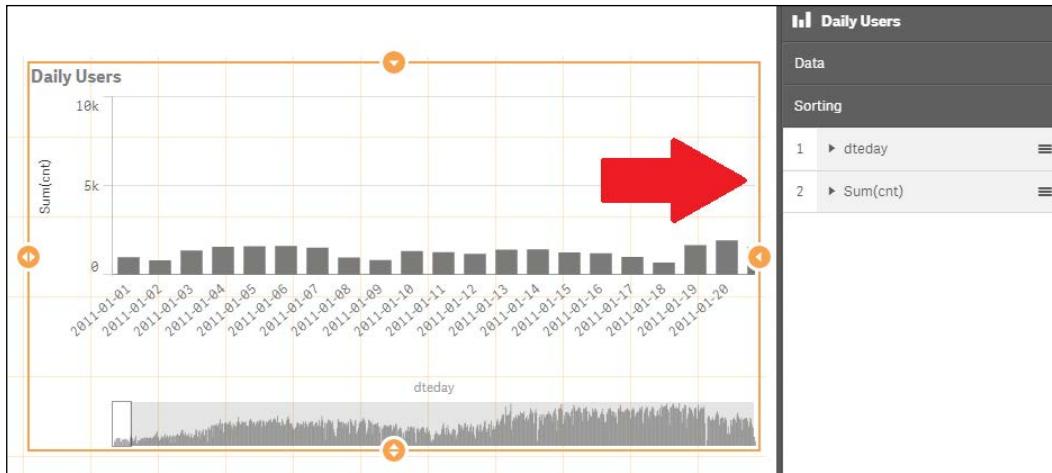
Exploring the data with Qlik Sense

We've seen in previous chapters how to load data into Qlik Sense and how to create charts. In this chapter, we'll discuss different charts, but we're not going to describe how to load the data and how to create charts. In this section, we will build a Qlik Sense application using the data contained in the `day.csv` file. Before starting to build the application, we will explore it.

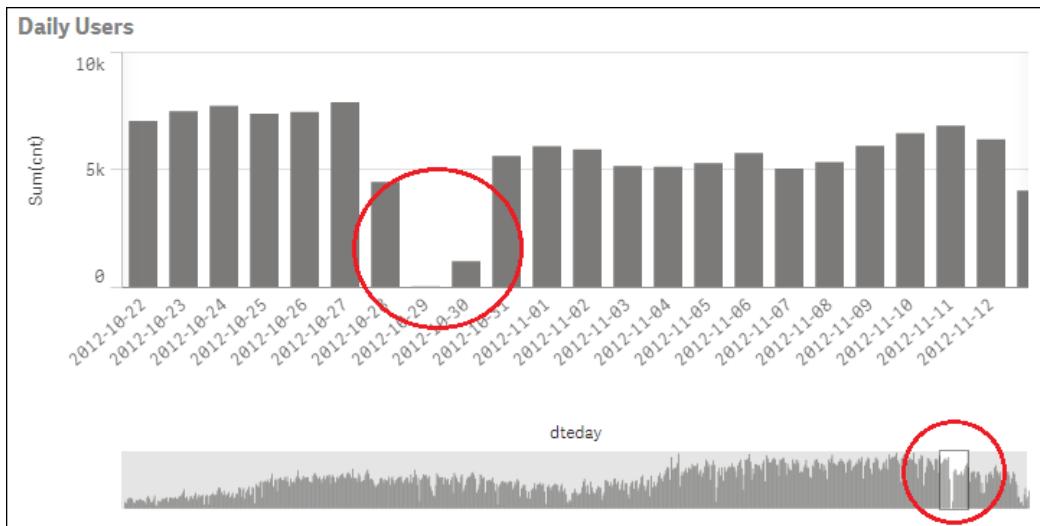
Checking for temporal patterns

We can start creating a chart to see the evolution of the number of rentals, as shown in the following screenshot. We've used a bar chart with the variable `dteday` as **Dimension** and `sum(cnt)`, the total number of users, as **Measure**:

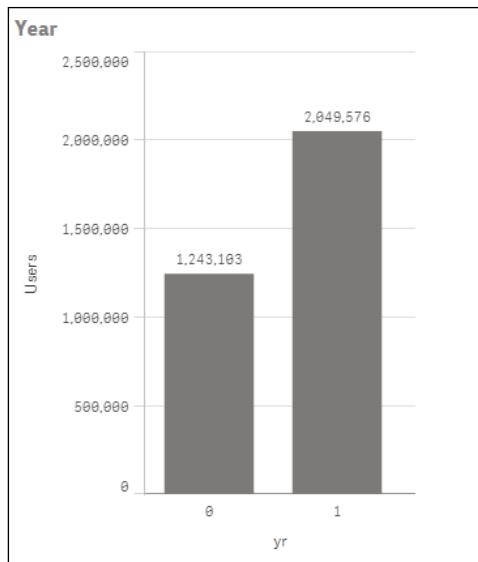
In the bar chart, we usually use the measure to order the bars. If we use a bar chart to see the sales performance of different countries, we will have to order based on the amount of sales. In this case, I will have to order the bars by the `dteday` dimension. To ensure I'm using this variable to order my bar chart, I need to check the sorting order. As you can see in the following screenshot, I set `dteday` as the first variable to sort the data:



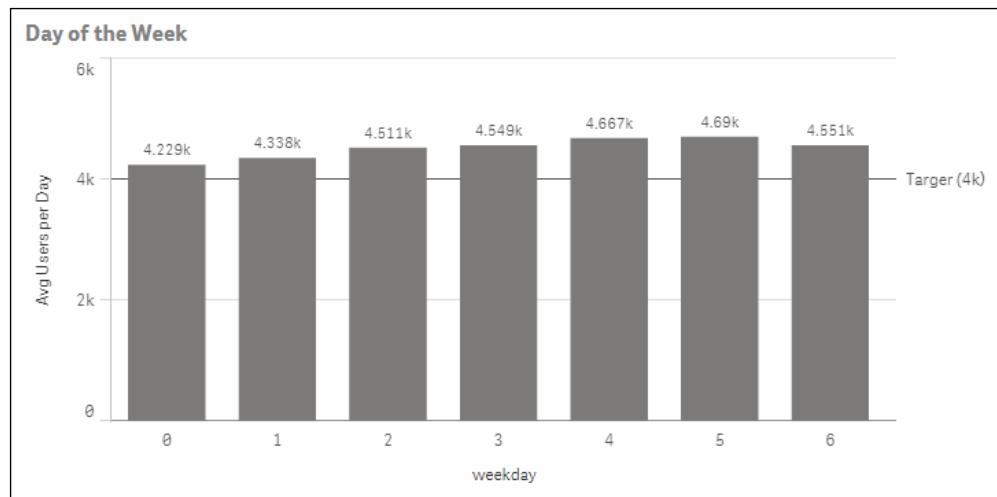
Looking at the following chart, we discover that on 29th October 2012, there was an unusually low number of rentals; on that day the system reports only 22 rentals. The reason for this unusual behavior is that on 29th October 2012, and 30th October, 2012, Hurricane Sandy hit Washington DC. We'll delete these two days from our dataset, before loading it into Rattle to avoid anomalies:



In the previous chart, we can also see that, in 2012, the bike rental increased, compared to 2011. We can replace the dimension `dteday` by `yr` to see the difference between 2011 (`yr = 0`) and 2012 (`yr = 1`). As you can see in the following screenshot, in 2011 the number of users was 1.243.103 and in 2012 the number was 2.049.576:



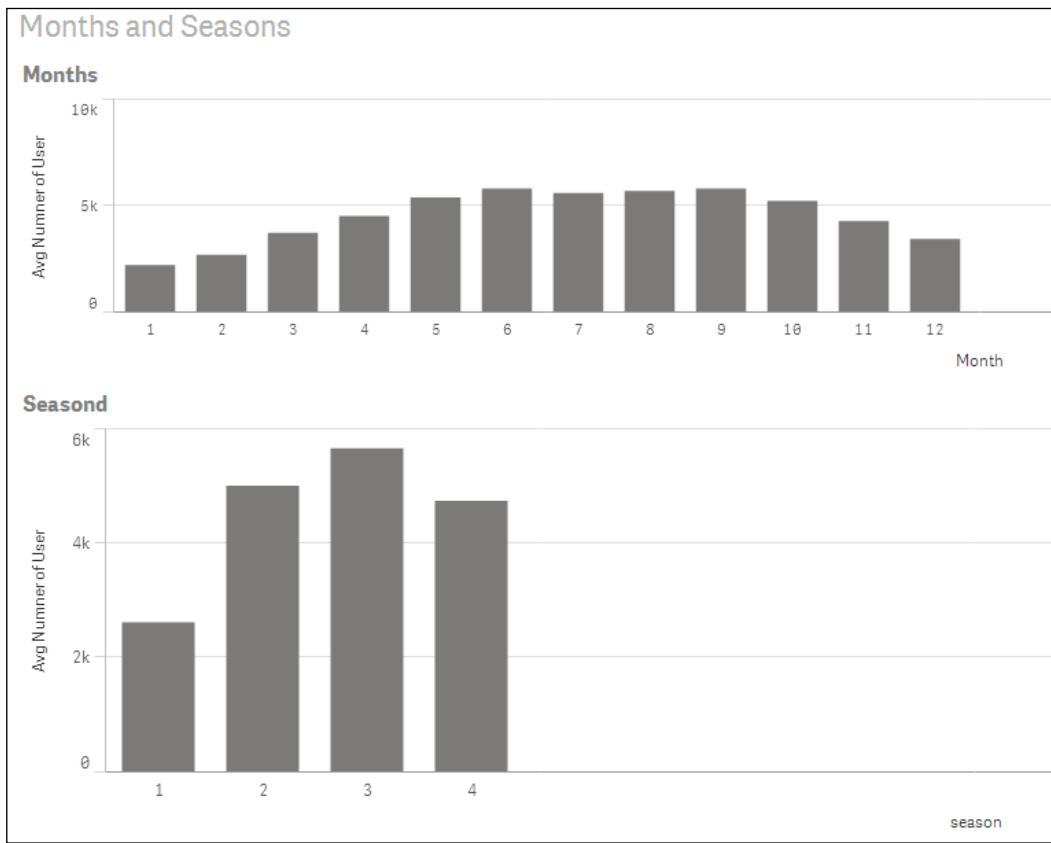
In the next chart, we'll use the average number of rental users and the weekday as the dimension to discover a behavior pattern for weekdays:



Looking at the preceding chart, we can see that the day of the week has a weak influence on the number of users. During the weekend, the average number of users is slightly lower.

As you can see, I've added the target to the chart. You learned in the previous chapter how to add reference lines to our charts.

In the following screenshot, you can see the relation between the variables `mnth` and `season` and the dependent variables `registered` and `casual`:



As you can see, there is a relationship between the month and the number of rentals. During the cooler months of January, December and February, demand is at its lowest and, during the warmer months of July, August and September, demand is higher. But, is it a linear relationship?

A regression model assumes a lineal relationship between the independent variable and the dependent variable. Usually, the relationship between the size of a house and its price is lineal; a big house has a higher price than a small house.

January (month 1) and December (month 12) have a smaller number of registered users than July (month 7). The relationship between the month, season, and the dependent variable is not lineal. We'll transform these variables later.

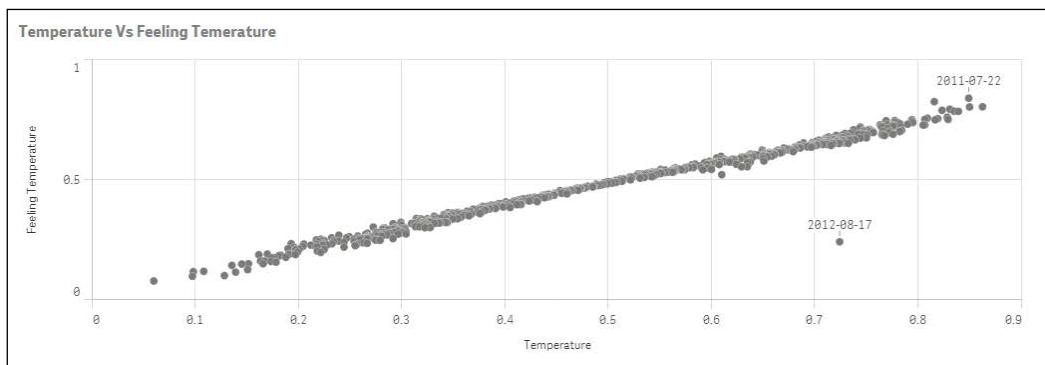
There is also a relationship between `workingday`, `holiday` and the dependent variables. Create the charts to check it.

Visual correlation analysis

We can use Rattle for correlation analysis, but we can also use Qlik Sense to do a very intuitive correlation analysis.

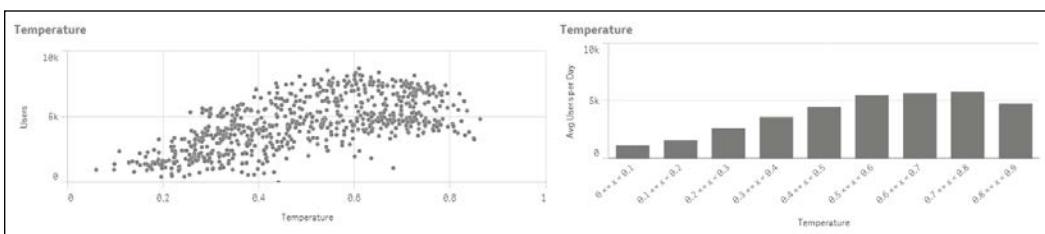
Karl Pover wrote a very interesting chapter on *Learning QlikView Data Visualization, Packt Publishing*, about how to use scatter plots for correlation analysis.

Create a scatter plot using `dteday` as the dimension and `temp` and `atemp` as measures, and you will get a chart similar to the following screenshot:



This chart shows us the relationship between temperature and feeling temperature. Of course, there is a strong relationship between the two variables.

Now, we will use a scatter plot and a bar chart to explore the correlation between temperature, wind speed, humidity, and the total number of users, as shown in the following screenshot. We will start with the variable temperature:



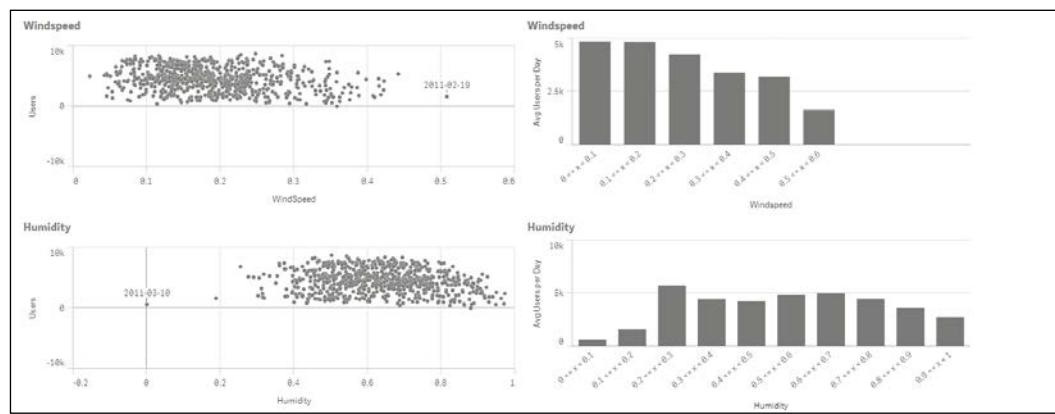
The left chart is a scatter plot much like the previous screenshot; each point is a different day, and we can clearly see that there is a strong relationship between temperature and the number of users. Users prefer warm days.

The chart on the right-hand side helps us to better understand the relationship between temperature and the average number of users. The measure of the chart is the average number of users, and the dimension is the range of the temperature. To create this dimension, I used the function `class`, as shown in this screenshot:

The screenshot shows a 'Data' interface with a 'Dimensions' section. A dimension named 'Temperature' is selected. In the 'Field' input field, the formula `= class(temp, 0.1)` is entered, with an orange 'fx' button to its right. This entire input field is highlighted with a red oval.

The `class` function creates the different ranges and the second parameter (`0.1`) is the width of every class.

Now, repeat these two charts for the variables `hum` and `windspeed`. You will discover that the relationships between humidity, wind speed, and the number of users are weaker than the previously used variables, as shown in this screenshot:

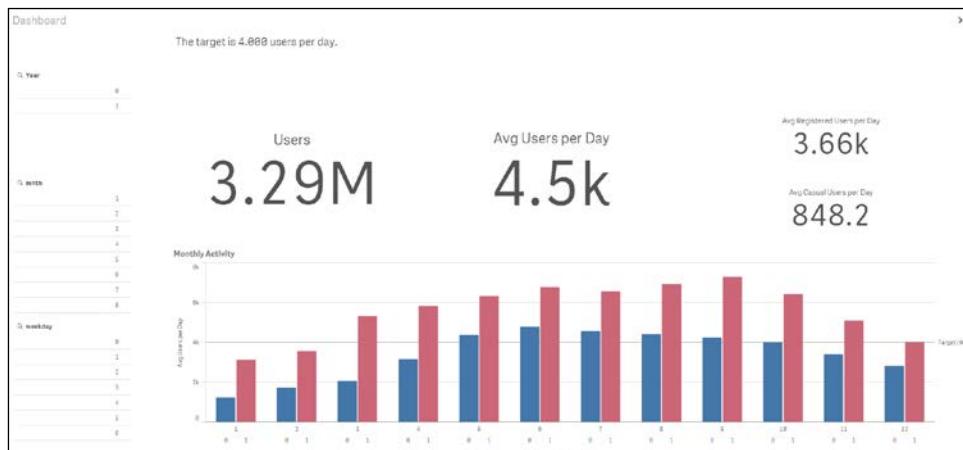


We've used Qlik Sense to explore and understand the data, and we've discovered that there is a strong relationship between the number of total users and the temperature. During the warmest months, the number of users increases, and during the coldest months, the number of users decreases. Now, we'll create an app to help users analyze the demand.

Creating a Qlik Sense App to control the activity

In the previous chapter, we explored the DAR approach to develop Qlik Sense applications; in this section, we will use the same approach.

We'll start with the dashboard. At the center, we'll place the most important information or measures. The most important details are the average number of users and the total number of users; these KPIs will occupy the main area of our application. The number of registered and casual users is also important, and we'll keep space for them in the dashboard, as shown here:

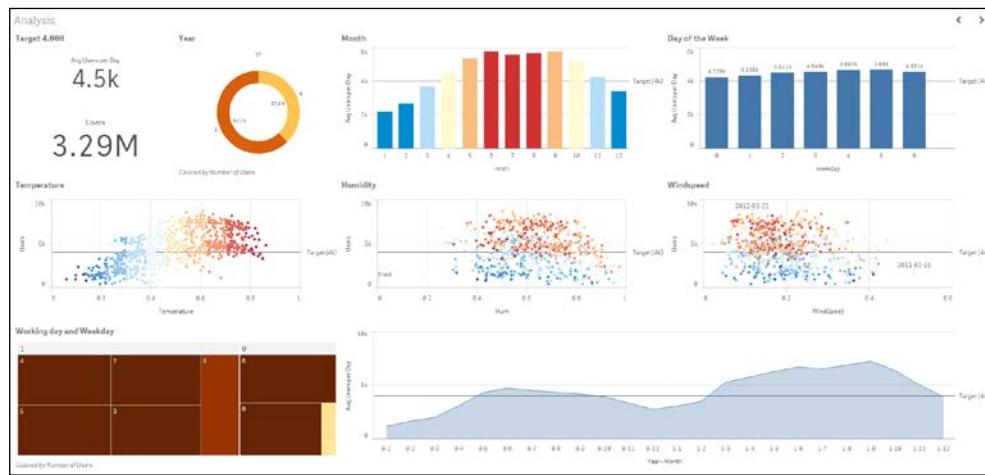


During our analysis, we discovered that, in 2012, the number of users increased, and that, during the colder months, we had a lack of activity. I've used a bar chart to show you the difference between 2011 and 2012 and the lack of activity during the winter.

After the dashboard, you will need an analysis sheet to analyze the months that have a lack of activity. In this analysis sheet, we'll include charts with just two metrics—the average number of users and the total number of users; we'll combine these metrics with the most important dimensions. The user will be able to see the same problem from different points of view and make selections in the charts to see the effect in the other dimensions.

In the previous section, we discovered that the variable that has a greater influence on the activity is temperature. For this reason, I decided to use color to add this measure in all charts.

In the central row, you can see the relationship between the activity and the humidity, and the wind speed and temperature. Using the color to add the temperature to the charts, the users will keep focused on this colored variable, as shown in this screenshot:



In this analysis sheet, the user can use the chart as a filter. Go to the temperature scatter plot and choose the days that have under 4,000 users, as shown in the following screenshot. After analyzing low performing days, you will have discovered that the days of the week with the worst performance is Saturday.



Use the month chart to filter January, February, and March, and you will see that, during these months, Saturday and Sunday are days with very low performance. Perhaps we can plan a promotional activity during these days to improve our activity level.

Finally, in our reporting sheet, we can place one or more tables to be able to see detailed information about the activity each day.

Using Rattle to forecast the demand

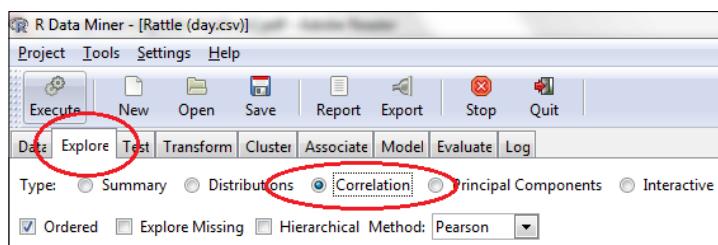
In this section, we'll use Rattle for a quick correlation analysis and to create a model to forecast the bike demand.

Correlation Analysis with Rattle

Our dataset has three possible target variables: `cnt`, `registered`, and `casual`. Rattle doesn't handle multiple targets. We can create a model for `registered` and a second model for `casual` and add both values to have the total number of users, or we can build a model for `cnt` (the total amount). We will only create a model for `cnt` because we're interested in the level of activity and this variable will provide it.

Load the dataset into Rattle and set `instant` to `Ident` and `dteday`, `registered` and `casual` to `Ignore`, and set the rest of variables to `Input`.

To perform the correlation analysis, go to the **Explore** tab, select the **Correlation** radio button and, finally, press the **Execute** button, as shown in this screenshot:



Rattle will return us a correlation matrix and a correlation plot. The correlation matrix is a matrix that displays the correlations between variables. In the following screenshot, you can see a reduced correlation matrix. We've ignored some variables to create a simple version of the correlation matrix. The correlation index is a number between -1 and 1. When the coefficient is 1, the variables are perfectly correlated. If the coefficient is -1, the variables are perfectly and negatively related. If the coefficient is 0, the variables are not related and are not useful when predicting a new value. In order to describe the different correlation coefficients, we can use these rules:

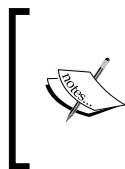
- greater than 0.7: A very strong positive relationship
- 0.4 to 0.7: A strong positive relationship
- 0.3 to 0.4: A moderate positive relationship
- 0.2 to 0.3: A weak positive relationship
- 0.2 to -0.2: A negligible relationship
- -0.2 to -0.3: A weak negative relationship
- -0.3 to -0.4: A moderate negative relationship
- -0.4 to -0.7: A strong negative relationship
- less than -0.7: A very strong negative relationship

The following screenshot shows the correlation summary:

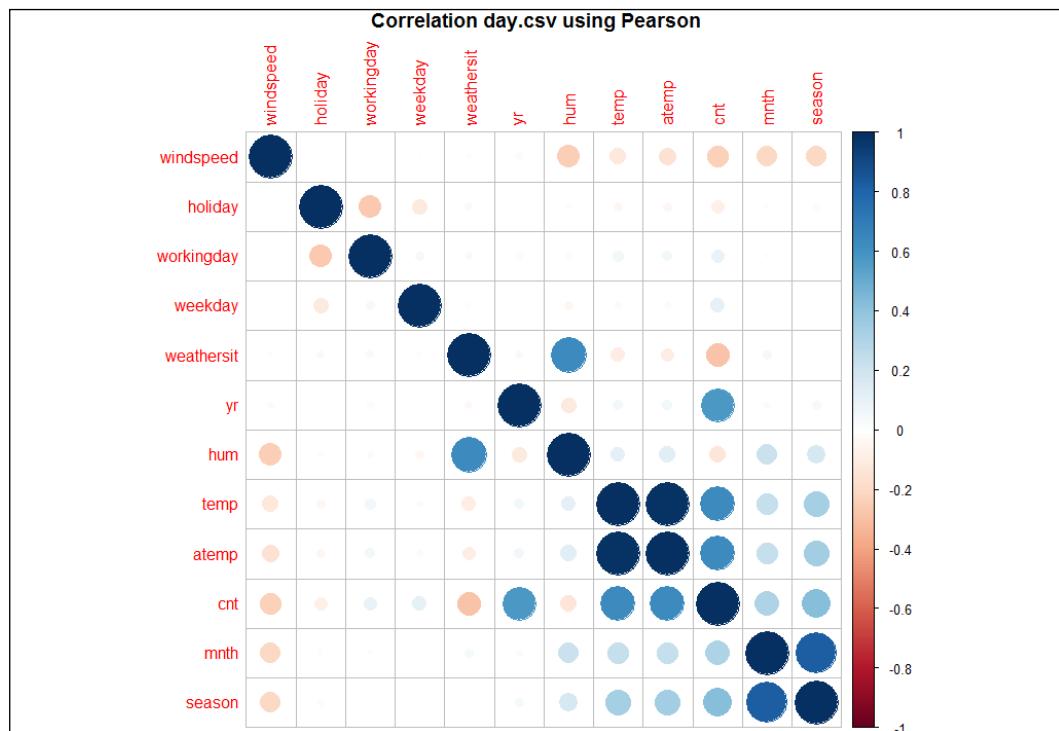
| Correlation summary using the 'Pearson' covariance. | | | | | | |
|---|---------------|--------------|--------------|---------------|--------------|--|
| Note that only correlations between numeric variables are reported. | | | | | | |
| windspeed | holiday | workingday | weekday | weathersit | | |
| windspeed | 1.000000000 | -0.002138282 | -0.001560627 | 0.0072097243 | 0.019696893 | |
| holiday | -0.002138282 | 1.000000000 | -0.265080924 | -0.1167294529 | -0.035618710 | |
| workingday | -0.001560627 | -0.265080924 | 1.000000000 | 0.0409517959 | 0.036544724 | |
| weekday | 0.007209724 | -0.116729453 | 0.040951796 | 1.000000000 | 0.018991914 | |
| weathersit | 0.019696893 | -0.035618710 | 0.036544724 | 0.0189919137 | 1.000000000 | |
| yr | -0.026605114 | 0.001759249 | 0.022853155 | -0.0007176284 | -0.031865947 | |
| hum | -0.249789799 | -0.019085193 | -0.022536858 | -0.0421432549 | 0.635618502 | |
| temp | -0.129407940 | -0.042122706 | 0.059877976 | 0.0277297718 | -0.098747332 | |
| atemp | -0.157154074 | -0.047527991 | 0.057232999 | 0.0236027677 | -0.094983995 | |
| cnt | -0.232535932 | -0.083692129 | 0.095895392 | 0.1040646910 | -0.288818791 | |
| mnth | -0.205734775 | 0.019274288 | -0.010169220 | 0.0073114706 | 0.041825772 | |
| season | -0.206270773 | -0.024476785 | -0.001246263 | 0.0030517333 | 0.007360641 | |
| yr | hum | temp | atemp | cnt | | |
| windspeed | -0.0266051144 | -0.24978980 | -0.12940794 | -0.15715407 | -0.23253593 | |
| holiday | 0.0017592489 | -0.01908519 | -0.04212271 | -0.04752799 | -0.08369213 | |
| workingday | 0.0228531545 | -0.02253686 | 0.05987798 | 0.05723300 | 0.09589539 | |
| weekday | -0.0007176284 | -0.04214325 | 0.02772977 | 0.02360277 | 0.10406469 | |
| weathersit | 0.0318659473 | 0.63561850 | 0.09874733 | -0.09498399 | -0.28881879 | |
| yr | 1.000000000 | -0.11515648 | 0.05666152 | 0.05191096 | 0.57115393 | |
| hum | -0.1151564848 | 1.00000000 | 0.11056228 | 0.12845472 | -0.13944589 | |
| temp | 0.0566615209 | 0.11056228 | 1.00000000 | 0.98977296 | 0.63082829 | |
| atemp | 0.0519109640 | 0.12845472 | 0.98977296 | 1.00000000 | 0.63077395 | |
| cnt | 0.5711539293 | -0.13944589 | 0.63082829 | 0.63077395 | 1.00000000 | |
| mnth | 0.0292674143 | 0.21963639 | 0.23059473 | 0.23885273 | 0.30162668 | |
| season | 0.0410655502 | 0.17984302 | 0.33149921 | 0.34015894 | 0.42383148 | |
| mnth | season | | | | | |
| windspeed | 0.205734775 | -0.206270773 | | | | |
| holiday | 0.019274288 | -0.024476785 | | | | |
| workingday | -0.010169220 | -0.001246263 | | | | |
| weekday | 0.007311471 | 0.003051733 | | | | |
| weathersit | 0.041825772 | 0.007360641 | | | | |
| yr | 0.029267414 | 0.041065550 | | | | |
| hum | 0.219636389 | 0.179843016 | | | | |
| temp | 0.230594733 | 0.331499211 | | | | |
| atemp | 0.238852732 | 0.340158943 | | | | |
| cnt | 0.301626677 | 0.423831483 | | | | |
| mnth | 1.000000000 | 0.829079430 | | | | |
| season | 0.829079430 | 1.000000000 | | | | |
| Rattle timestamp: 2015-06-25 02:53:38 | | | | | | |
| ===== | | | | | | |

Notice that there is a strong correlation between **temp** (temperature) and **atemp** (feeling temperature) and that our target variable **cnt** has a strong correlation with variables **temp**, **atemp**, and **yr**. We already knew this because we discovered that the correlation helps explore the dataset with Qlik Sense.

In the following screenshot, we can see the correlation plot. The color and size of the ball explains the predictive power of each variable. For the dependent variable **cnt**, the independent variables, **yr**, **atemp**, **temp**, and **season**, are strong predictors. The independent variable **mnth** is a moderate predictor. Finally, **windspeed** and **weathersit**, are weak predictors:



Please note that you need to tick **Verbose**, and **Advanced Graphics** in the **Settings** menu to generate the following screenshot. The ticking of **Use Cairo Graphics Device** is optional. Also note that the inner Rattle window **R Data Miner - [Rattle]**, should not be maximized because the generated screenshot tends to go behind the current window.



As we've seen in Qlik Sense, variables **temp** and **atemp** have a strong correlation. We will use just one of them; for this reason, we will set **atemp** to **Ignore**.

Building a model

Creating a predictive model is an iterative process; we start by creating a model with the training dataset. Then we evaluate the performance with the validation dataset, and we modify the model and create a new model. Finally, when we feel comfortable, we can test the performance of our model with the testing dataset.

In this example, we won't tune the performance of the model.

Go to the **Data** tab and set **cnt** to **target** and **atemp** to **Ignore**. To create the model, go to the **Model** tab, select the **Linear** radio button and press **Execute**, as shown in the following screenshot:

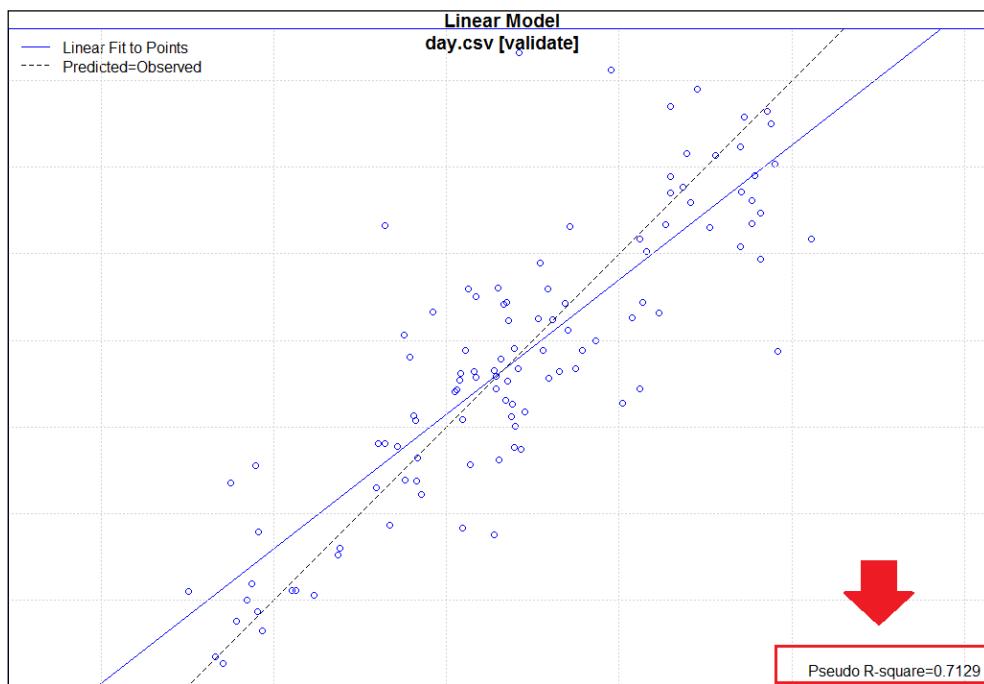
```
R Data Miner - [Rattle (day.csv)]
Project Tools Settings Help
Execute New Open Save Report Export Stop Quit
Data Explore Test Transform Cluster Associate Model Evaluate Log
Type: Tree Forest Boost SVM Linear Neural Net Survival All
Numeric Generalized Poisson Logistic Probit Multinomial
Plot
Summary of the Linear Regression model (built using lm):
Call:
lm(formula = cnt ~ ., data = crs$dataset[crs$train, c(crs$input,
  crs$target)])
Residuals:
    Min      1Q  Median      3Q     Max 
-3911.9 -442.6   48.7  530.5 2384.4 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1795.80    270.92   6.629 8.80e-11 ***
season       479.75    63.87   7.512 2.70e-13 ***
yr          2020.45   77.54   26.057 < 2e-16 ***
mnth        -28.06   19.90  -1.410 0.159294    
holiday     -486.35   229.61  -2.118 0.034657 *  
weekday      81.88    19.24   4.256 2.49e-05 ***
workingday   182.99   85.69   2.135 0.033218 *  
weathersit   -588.79   96.27  -6.116 1.94e-09 ***
temp         5283.09   226.19  23.357 < 2e-16 ***
hum          -1338.03   382.82  -3.495 0.000516 *** 
windspeed    -3256.34   518.01  -6.286 7.09e-10 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 863.2 on 500 degrees of freedom
Multiple R-squared:  0.8122,    Adjusted R-squared:  0.8085 
F-statistic: 216.3 on 10 and 500 DF,  p-value: < 2.2e-16
```

After the execution, examine the summary report provided by Rattle, starting with the **Residuals** section. The maximum error is **2384.4** and the minimum is **-3911.9**. This error looks very high. Notice that 50 percent of the occurrences are between the first and the third quartile, so 50 percent of the time our error of trying to estimate users is between **-442.6** and **530.5**.

In the coefficients section, the last column shows us the significance of each variable or the predictive power—*** for strong predictive power and * for weak predictive power.

The last section, Adjusted R-squared, explains how accurate our model is when predicting the values of the dependent variable. In our example, the value of R-squared is **0.8122**, which means that 81.22 percent of the variation of `cnt` is explained by our model.

To have a reference of the model performance, we need to test its predictive power against the validation dataset. As we saw in *Chapter 7, Model Evaluation*, to evaluate the performance of a regression model, we can use a plot called Predicted versus Observed Plot (**Pr v Ob**), as shown here:



We've created a model that has **Pseudo R-square = 0.7129**. In the next section, we'll try to improve the performance of the model by transforming numeric variables with a non-lineal relationship with the dependent variables.

Improving performance

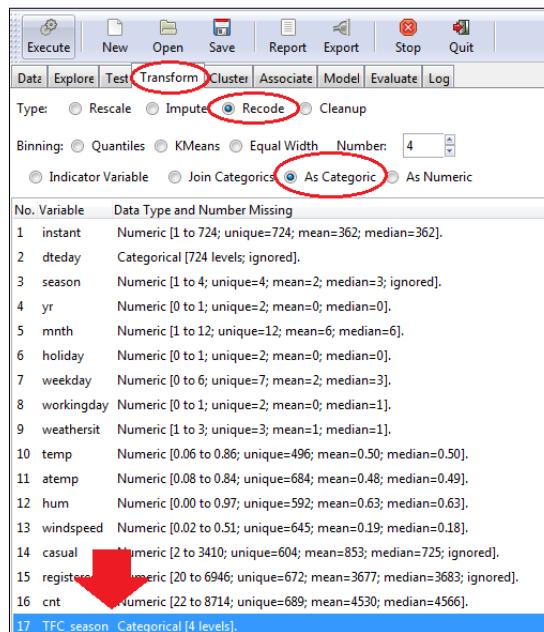
Now, we will enter in an iterative process, tune the model, and check the performance with the validation dataset until we achieve the performance needed. We will see how to make a transformation with an example. In this section, we will only make a small transformation.

As we've seen in the previous section, we have numeric variables that have a non-lineal relationship with the target variable.

To explain how to transform the variables, we're going to use the variable, `season`. As we've seen, `season` is a numeric variable that has a non-lineal relationship with `registered`. We would like to transform the numeric variable with four different values to four flags or indicators, as shown in the following table:

| Season | Spring flag | Summer flag | Fall flag | Winter flag |
|--------|-------------|-------------|-----------|-------------|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

We're going to transform the variable in two different steps. Go to the **Transform** tab, select the variable, `season`, select the **Recode** radio button, select the **As Categorical** radio button and press **Execute**, as shown in the following screenshot:



Rattle has added a new variable called **TFC_season**; this is a categorical variable. For the second step, go to the **Transform** tab, select the variable **TFC_season**, select the **Recode** Radio button, select the **Indicator Variable** Radio button and press **Execute**. After these operations are done, you will see the following result:

| | | |
|----|----------------------|---|
| 16 | cnt | Numeric [22 to 8714; unique=696; mean=4504; median=4548]. |
| 17 | TFC_season | Categorical [4 levels; ignored]. |
| 18 | TIN_TFC_season_.1.1. | Numeric [0.00 to 1.00; unique=2; mean=0.25; median=0.00]. |
| 19 | TIN_TFC_season_.1.2. | Numeric [0.00 to 1.00; unique=2; mean=0.25; median=0.00]. |
| 20 | TIN_TFC_season_.2.3. | Numeric [0.00 to 1.00; unique=2; mean=0.26; median=0.00]. |
| 21 | TIN_TFC_season_.3.4. | Numeric [0.00 to 1.00; unique=2; mean=0.24; median=0.00]. |

Remapped variables added to the dataset with 'TIN_' prefix.

Now Rattle has created four new numeric variables. Each variable has two possible values, 0 or 1.

You need to create the model again but, before that, ensure that the new flag variables are selected as input, as shown in this screenshot:

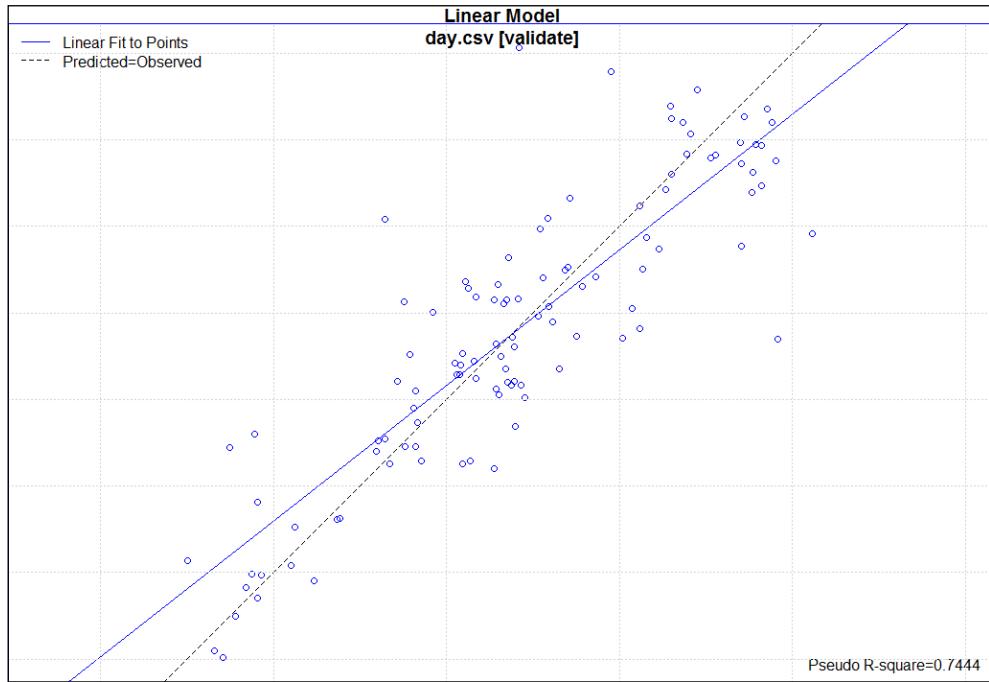
| | | | | | | | | | |
|----|----------------------|-----------|----------------------------------|----------------------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------|
| 17 | TFC_season | Categoric | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | Unique: 4 |
| 18 | TIN_TFC_season_.1.1. | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 19 | TIN_TFC_season_.1.2. | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 20 | TIN_TFC_season_.2.3. | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |
| 21 | TIN_TFC_season_.3.4. | Numeric | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Unique: 2 |

After creating the new model, review the new R-squared and Adjusted R-squared values and you will see some improvement in performance.

As we've explained before, the process of improving or tuning a model is iterative. After the performance evaluation, you can transform your data or modify the model and evaluate the performance again until your performance is acceptable.

Model evaluation

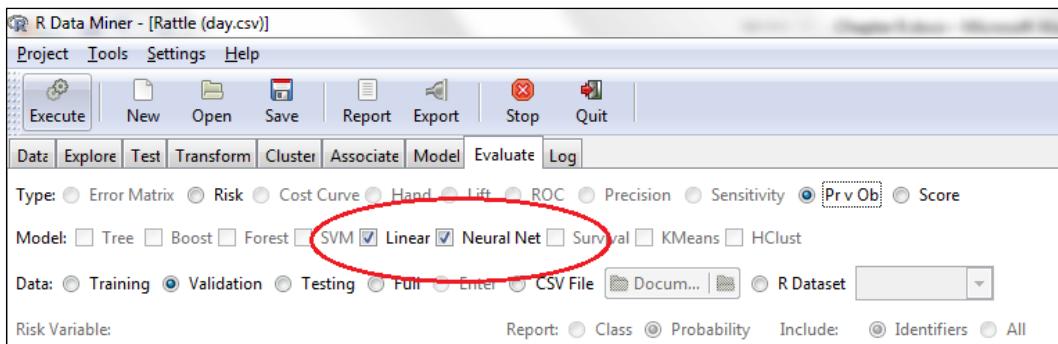
As we saw in *Chapter 7, Model Evaluation*, to evaluate the performance of a regression model, we can use a plot called Predicted versus Observed Plot (**Pr v Ob**), as shown here:



We've quickly developed a model that achieved a Pseudo R-square of **0.744**. We did a small optimization in the model; we can improve the performance by working with the different variables.

After improving the model using the training dataset to build the model and the validation dataset to evaluate this performance, we need to confirm the performance of our model by creating a Predicted versus Observed Plot with the test dataset. We can do that to detect overfitting.

A very interesting feature of Rattle is that we can run multiple models and evaluate the performance of the different models. Go to the **Model** tab and build a Neural Network model. Now, return to the **Evaluate** tab and select the **Linear** and **Neural Net** checkboxes and press the **Execute** button, you can compare the two different models, as shown in the following screenshot. As we saw in *Chapter 6, Decision Trees and Other Supervised Learning Methods*, Neural Networks are a kind of algorithm inspired by biological Neural Networks which are used to approximate functions:



Now that we have a model to predict the demand of rental bikes, we want to add demand forecast information to our Qlik Sense application.

The first step is to predict the new values, or forecast the demand. The second and final step is to load the data into Qlik Sense.

Scoring new data

We have three main options to score new data, these are listed as follows:

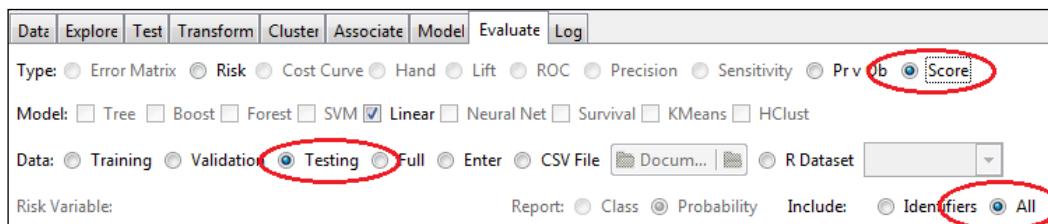
- **Qlik Sense:**
 - We've found the coefficients for the regression; we can use it in Qlik Sense during the data loading or we can create a measure with the formula.
 - This option has a great disadvantage; we would normally prefer to re-evaluate the model with new data. If we change the model, we will have to update the Qlik Sense app.
- **R:**
 - We can save our model in Rattle and then we can load the model in R.
 - We can use the `predict()` function to score new data. This is a good option but the last option is the easiest of all.

- **Rattle:**

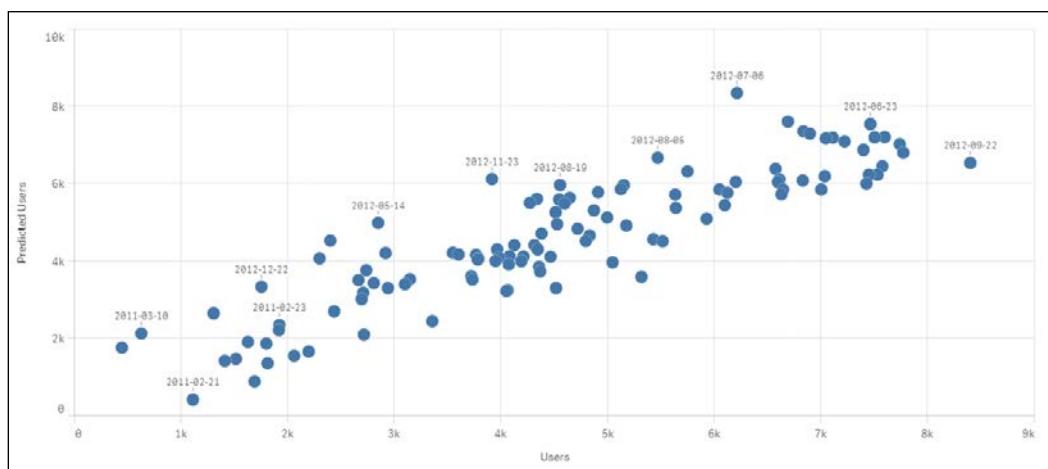
- In Rattle's **Evaluate** tab, we can use the Score option. Using this option, we can score the training, the validation or the testing dataset, or we can load a dataset from a CSV file.

In a real case, we will have the weather forecast for the next week, and we'll use Rattle to load the data from a CSV file. In this example, we don't have next week's forecast and we'll score the testing dataset and the complete dataset.

Go to the **Evaluate** tab, select **Score**, **Testing**, and **All** and press the **Execute** button. Rattle will score the testing dataset for you and will write the prediction in a CSV file. This file will include all original variables and a new variable called `glm` with the predicted value, as shown in the following screenshot. Before finishing, you need to confirm the location and name of the file:

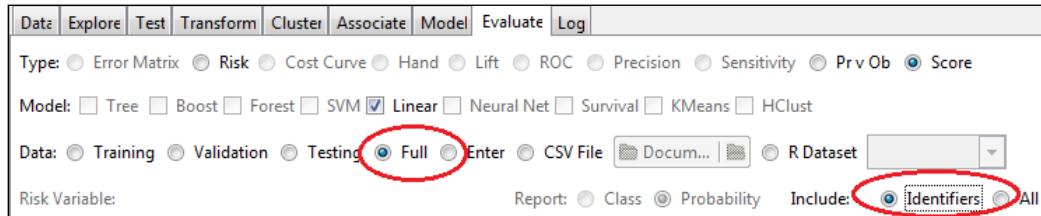


Now, we will check the performance of our model by using Qlik Sense. Create a new Qlik Sense application and load the file generated by Rattle. This file contains 111 rows and the testing dataset, and each row contains all the original data and the predicted value. Create a scatter plot with `dteday` as the dimension and `glm` and `cnt` as measures, as shown here:



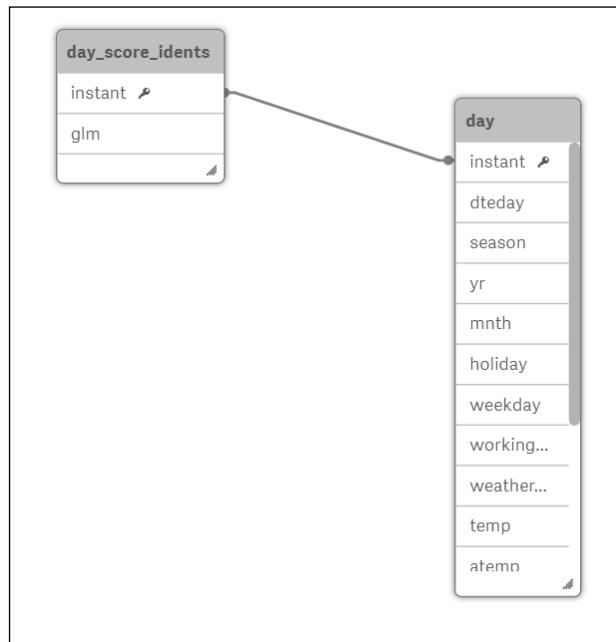
We've created a Predicted versus Observed Plot with the testing dataset. This chart and the one we created in Rattle gives us an idea of the predictive power of our model. We don't need this plot because we've created it in Rattle.

Now, come back to Rattle to score the complete dataset. Select the **Full** dataset and report just the **Identifiers** included, as shown here:

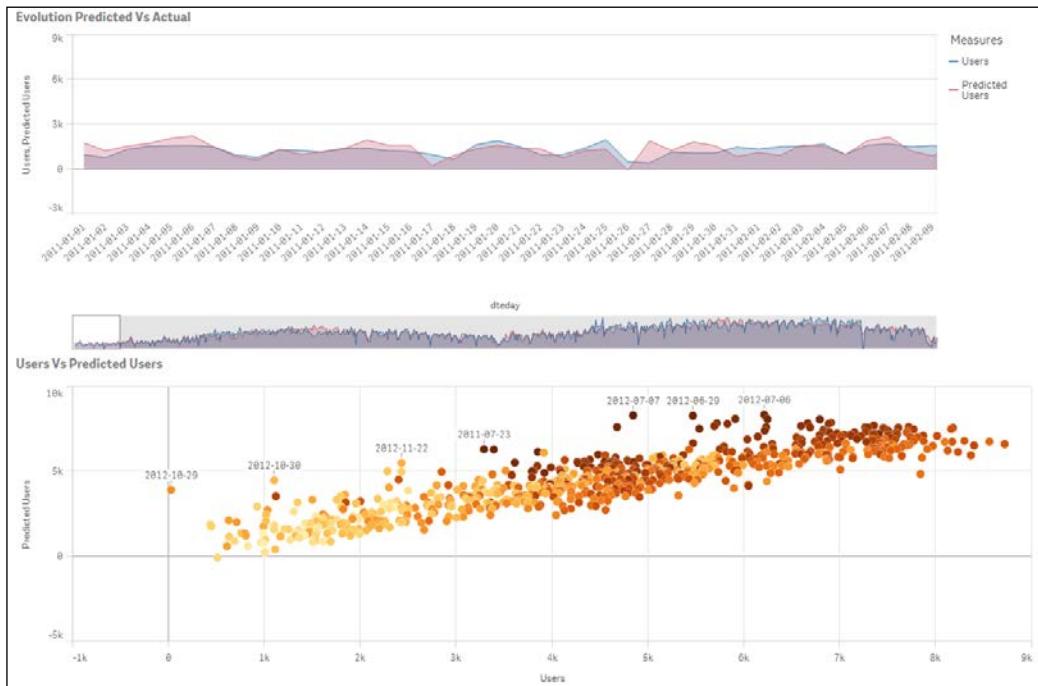


Rattle will create a file with 731 observation with all the original columns and a new column with the predicted value called `glm`.

We will load this information in to our original bikes application. Go to Qlik Sense, open the original application, and drag and drop this second file into the application. Qlik Sense will ask you if you want to add new data or replace the current data, if yes, then select **Add data**. After loading the data, open **Data model viewer**, as shown in the following screenshot:

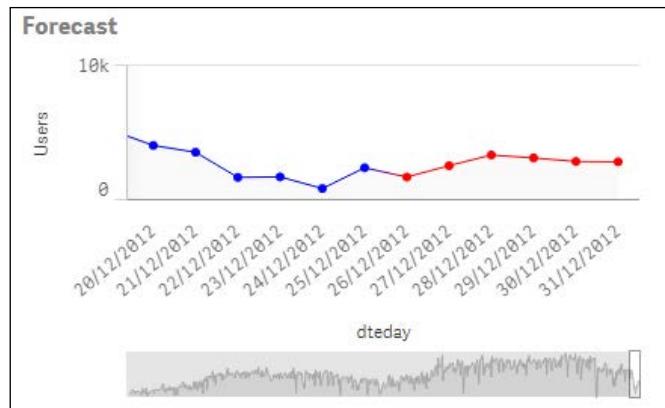


Rattle has created a file that contains variables, the identifier **ident** and the prediction **glm**. When we load this file into Qlik Sense, it creates an association with our original table using the field **ident**. Finally, create two charts to show the predictive power of our model. In the first chart, I used a line chart with **dteday** as the dimension and **cnt** and **glm** as measures, as shown here:



Be careful with these plots, we've added the training dataset and the validation and testing ones. We used training and validation datasets to build the model, so this plot doesn't provide us a real idea of the predictive power of our model. The previous plots we did just with the testing dataset gave us a real idea of the performance.

In the real world, we will load the weather forecast for the following week into Rattle and we will score it. By loading historic and forecast data into Qlik Sense, we will be able to create visualizations that are similar to the following screenshot, which shows historic and forecast data together:



Further learning

Predictive Analytics and Data Visualizations are huge topics and in this book I just introduced the key concepts.

Related to Qlik Sense, we forgot a lot of important things. In my opinion, the data load editor is among the most powerful. In order to learn more about Qlik Sense and the **Data load editor**, you may want to check out:

Learning Qlik® Sense: The Official Guide, Christopher Ilacqua, Henric Cronström, James Richardson. Packt Publishing.

In order to increase your knowledge and learn more about Qlik Sence, I suggested you some books and web sites. A very good source of information and advice can be the Qlik Community:

<https://community.qlik.com>

This is the most active community in the BI landscape.

You also learned how to use Rattle to create predictive models. In this book, we've focused on Clustering, Decision Trees, and Linear Regression; these are the most common predictive techniques. You can use these techniques in your own datasets to provide insights. To improve your Rattle knowledge, the best book is *Data Mining with Rattle and R, The Art of Excavating Data for Knowledge Discovery, Graham Williams, Springer*.

The videos listed on this page are also a good source of knowledge:

<http://rattle.togaware.com/rattle-videos.html>

Rattle also has a very interesting users group:

<https://groups.google.com/forum/#!forum/rattle-users>

Finally, if you prefer to learn predictive techniques with R, try *Machine Learning with R*, Brett Lantz, Packt Publishing.

A funny and very good way to improve your skills in predictive analytics is Kaggle. This is the world's largest community of data scientists. In this community, you can find *data science* competitions. We've not used the term data science in this book; there are a lot of new terms around analytics, and we tried to focus on just a few to avoid confusion. Currently, we use this term to refer to an engineering area dedicated to collecting, cleaning, and manipulating data to unearth new knowledge. At www.kaggle.com, you can find different types of competitions. There are introductory competitions for beginners, and there are competitions with monetary prizes. You can access a competition, download the data and the problem description, and create your own solutions. An example of a Kaggle competition is the bike sharing example we used in this chapter.

Finally, in *Chapter 1, Getting Ready with Predictive Analytics*, we introduced *Competing on Analytics*, by Thomas H. Davenport and Jeanne G Harris, Harvard Business Review Press; if you are worried about how to apply predictive analytics at a business level, start with this book.

Summary

In this chapter, we used Qlik Sense to explore the bike sharing dataset. In Qlik Sense, we saw different ways of doing an intuitive correlation analysis.

After this, we created an application to analyze the rental activity. The application had three sheets following the DAR approach.

Finally, we used Rattle to confirm the correlation analysis we did with Qlik Sense, then we created a predictive model to forecast the demand depending on the weather forecast.

With Rattle, we used the variable `cnt` as the target variable; it would be very interesting to repeat the exercise using the variables `registered` and `casual`.

You've arrived at the end of the book, so by now you should understand the basics of predictive analytics and data visualization and have gained some expertise in using Rattle and Qlik Sense Desktop.

Now you can use Qlik Sense to quickly analyze business data. With Qlik Sense you can discover hidden patterns in your data and create powerful visualizations to present the conclusions of your analysis.

Bibliography

This course is a blend of text and quizzes, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Learning Qlik Sense: The Official Guide Second Edition, Dr. Christopher Ilacqua, Dr. Henric Cronström and James Richardson*
- *Qlik Sense Cookbook, Philip Hand and Neeraj Kharpatne*
- *Predictive Analytics using Rattle and Qlik Sense, Ferran Garcia Pagans*

Index

Symbol

= sign

using, with variables 419-421

A

access pass 170

age of oldest case, in queue

calculating, MinString() function
used 388, 389

analysis interface, authoring engaging applications

about 81
bar charts, defining 85, 86
dimensions, adding 84
measures, adding 84
sheet, creating 82, 83
storytelling 86
visualizations, adding 83, 84

analytics 515

application library, authoring engaging applications

about 86, 87
fields, to be exposed 87
KPIs, defining 88
library entries, creating 88, 89

application life cycle

application authoring, starting 16, 17
continuing 21
overview 15

applications

connectivity management 177
extensions, importing 174
importing 172, 173
managing 171

monitoring 181

security rules 178-180

streams, defining 176

system management 178

tasks 177, 178

user directories 175

users 175

approved sheets

about 337
creating 337, 338

architecture, Qlik Sense

about 163
applications 166
clients 164
nodes 167
services 164
streams 167

association analysis 619-621

associative logic

about 581
working 581-583

Attribute-Relation File Format (ARFF) 534

authentication 182

authoring engaging applications

analysis interface 81
application library 86
communication problem 68, 69
creating 70, 71
data, loading 72, 73
preparations 67
process 70
publishing 104
requirement specifications 68
requisites 67, 68
step-wise implementation 69, 70

authorization 182

B

bar chart

about 92, 93
creating 675, 677
personalizing 677

best practices, data visualization

about 89
analysis 90
dashboard 90
dimensions 91, 92
graphs 91
measures 91, 92
reporting 90
sheet, structuring 91

Bike Sharing Dataset

reference 704

bike sharing system

bike rental problem 704-706
data, exploring with Qlik Sense 706

binning 545

bookmarks 28

boosting method 647-649

Business Discovery 4

business intelligence (BI)

about 3
evolution 13

business problem 207

C

Calendar fields

generating, Declare function used 396-398
generating, Derive function used 396-398

Capventis Redmond Pie-Gauge

reference link 350

casual users 704

categorical variable

about 534, 563, 564
bar chart 565
mosaic plot 566, 567

centroid 606

charts

Add-ons menu 682
Appearance menu 683, 684
bar chart, creating 589-593
creating 584-597
Data menu 678-681

limitations, applying to 364-367
pie chart, creating 584-588

Sorting menu 681

classifiers performance

accuracy 661
confusion matrix 661-665
measuring 661
Risk Chart, obtaining 666-668
ROC Curve 668, 669
sensitivity 661
specificity 661
types of predictions 661

cleanup options 549

clients, Qlik Sense

about 164
hub 165
Qlik Management Console (QMC) 165, 166

cloud

content 122
features 121

cloud sharing 122

cluster 167

cluster analysis

about 605
centroid-based clustering, K-means
algorithm used 606, 607
customer segmentation sheet, creating in
Qlik Sense 614-616
customer segmentation, with K-means
clustering 608-612
data, preparing in Qlik Sense 613, 614

code

packaging, in script files 320-322

ColorMix1 function

using 300, 302

Comma Separated Value (CSV) 534

community sheets

about 337
creating 337, 338

comparison 306, 308

comparison sets

using, in Set Analysis 424-427

Complexity Parameter (CP) 658

components, applications

bookmarks 28
data storytelling 28, 29
sheets 26, 27

composition 302-304
Comprehensive R Archive Network (CRAN) 519
com-qliktech-peoplechart.js file 151
com-qliktech-peoplechart.qext file 151
concat()
 used, for capturing list of field values 434, 435
Concat function
 about 332
 for storing multiple field values in single cell 332, 333
 used, for displaying string of field values as dimensions 386, 387
connectivity 177
contents
 adding, to story 288-293
 creating 288
content security 183
correlation, among input variables 567, 568
correlation analysis, with Rattle 714-716
correlation coefficient 567
credit risks
 classifying, with Decision Tree 633-641
cross-validation
 about 657
 implementing 658
CSV file
 loading 535-538
cumulative figures, in trendlines charts
 plotting, RangeSum() function used 390-392
currency Exchange Rate Calendar
 creating, Peek() function used 404-408
custom components
 creating, within Qlik Sense visualizations 451-453
 reference link, for creating 454
customer buying behavior 571
customer segmentations
 about 571
 types 572

D

DAR methodology
 reference link 700

Dashboard Analysis and Reporting (DAR) 686-692
dashboards 685, 689
data
 about 255
 analyzing 598, 600
 exporting 550
 extracting, from custom databases 267-269
 extracting, from databases and data files 256-263
 extracting, from web files 263-265
 Impute option, used for dealing with missing values 541-545
 loading 534, 535, 572
 preparing 574
 rescaling 540, 541
 using, with extensions 454-460
data analysis, Qlik Sense
 about 686
 associative logic 688
 in memory analysis 686
data applications 685, 689
data discovery
 about 6, 13
 contributing to 39
 data interaction 7, 8
 empowered user 7
 new sheet, creating 48
 private bookmarks, creating 40-42
 private sheets, creating 42-45
 private sheets, publishing 46, 47
 private sheets, sharing 42, 43
 traditional business intelligence architecture 8-10
data discovery consumption requirements
 about 23, 24
 application components, exploring 26
 associative experience, leveraging 30
 associative experience, navigating 30
 hub 25
 Library, extending with 38
 streams 26
data exploring, with Qlik Sense
 about 706
 temporal patterns, checking 706-709
 visual correlation analysis 710-712

data load editor
help, invoking 269, 270

data loading, authoring engaging applications
about 72, 73
additional tables, loading 73-75
database connection, creating 78, 79
data connectors 80, 81
Data load editor, using 76, 77

data model
checking 577-579
creating 573-577

data model viewer
about 117, 118
associations, viewing 272, 273
data model, viewing 272
data, previewing 271
master library, creating from 274-281
preview mode, using 119
table meta data 273

data points
navigating, in scatter chart 369-371

Data Science
reference URL 655

dataset
about 532
atemp 705
casual 705
cnt 705
dteday 705
hr 705
hum 705
instant 705
mnth 705
partitioning 624
reference 608
registered 705
season 705
temp 705
variable description 533
weathersit 705
weekday 705
windspeed 705
workingday 705
yr 705

data storytelling, Qlik Sense
about 692
audience 693
key messages 693
links 693
new story, creating 696-700
objective 693
reviewing 693
story 693

data, structuring
about 111
normalization 111, 112
snowflake schema 113, 114
star schema 113, 114

data transformation
about 538, 539
binning 545, 546
indicator variables 547
Rattle, used 539
variables, recoding 545

data visualization
about 515
best practices 89
books, for references 700

data visualization, Qlik Sense
about 672
bar chart, creating 675-677
visualization toolbox 672

Decision Tree
creating 629
loan applications, scoring with
Rattle 641-643
Qlik Sense application, creating 644, 645
URL 633
using, for credit risk classification 633-641

Decision Tree Learning
about 625, 626
advantages 626
disadvantages 626

Declare function
used, for generating Calendar
fields 396-398

default charts, Qlik Sense
Bar chart 673
Combo chart 673

Extensions 674
Filter pane 673
Line chart 673
Map 673
Pie chart 673
Pivot Table 674
Scatter plot 674
Table 674
Text & image 674
Treemap 674

Demographic Data Discover application
about 241
analysis 245
developing 249, 250
dimensions 250
features 243-245
global selector, for making selections 248, 249
lasso selector, for making selections 246, 247
measures 250, 251
problem analysis 241-243

dendrogram 618

deployment, Qlik Sense
about 167
access passes 169
access rules 171
license 169
multinode 168
single node 168
tokens 169, 170

Derive function
used, for generating Calendar fields 396-398

descriptive analytics 604

dimension 84

dimensionless bar charts
creating, in Qlik Sense 356-358

dimensions, Travel Expense Discovery application
about 234
measures 235
visualizations 238

disadvantages, Decision Tree Learning
overfitting 626
unstable 626

distributions
about 308
categorical variables 563
numeric variables 558
visualizing 558

dynamic charts
creating, in Qlik Sense 498-500

E

E() function
using, in Set Analysis 435-438

embedded functions
using, in Set Analysis 427-429

embedded sheets
adding, to story 293, 294

Engine API Explorer 156-159

Ensemble methods
about 646
boosting 647-649
Random Forest 650-652
Supported Vector Machine (SVM) 653
URL 646

entropy 626-632

environment
installing 519

error rate 663

Exploratory Data Analysis (EDA) 551

Explore Missing option 569

expression editor
help, invoking 269, 270

extended interval match
used, for handling Slowly Changing Dimensions 374-380

Extension editor
using, in Qlik Dev Hub 481-484

extensions
data, using with 454-460

F

fact table 687

fields, used as dimensions
intervals 92
nominals 92
ordinals 92
ratios 92

field values list
capturing, concat() used 434, 435
files
extracting from folders, For Each loop used 402-404
files, Qlik Sense extension
.JS file 442
.QEXT file 442
FirstSortedValue() function
used, for identifying median in quartile range 394, 395
flags
using, in Set Analysis expressions 417-419
For Each loop
for loading data, from multiple files 330-332
used, for extracting files from folders 402-404
Fractile() function
used, for generating quartiles 392-394

G

gauge object 95
General Public License (GNU) 517
geographical map 98, 99
geo maps
creating, in Qlik Sense 342-347
Global Defined Data Sources 18
Graphical User Interface (GUI) 518
graph types, data visualization
bar chart 92, 93
colors 102
gauge object 95
geographical map 98, 99
KPI object 94
line chart 94
pie chart 93, 94
scatter chart 95, 97
sorting 101
tables 99, 100
tree map 98

H

hierarchical clustering 616-618
Hierarchical option 569

HTML visualization extension
creating, for Qlik Sense 442-447
hub 15 25, 165
human resource app
career and compensation 208
costs, training 211
Courses table 216
developing 215
dimensions 216
employee satisfaction and retention 208
Employees table 215
features 208, 209
global selector, using 212-214
health and safety 208
Hierarchy table 216
Map shapes table 216
measures 217, 218
recruitment 208
sheet 210
Survey table 216
training 208
Training table 216
Human Resources analysis
about 207
preparing 208
HyperCube 459

I

Image
creating 361
indicator variables
about 547
As Category option 548
As Numeric option 548
Join Categories option 548
information gain 626-632
input variables 533

K

Kaggle
about 532
URL 532, 659
keyboard shortcuts, Qlik Sense
Desktop 509, 510
key fields 70

- Keyhole Markup Language (KML)** 342
Key Performance Indicator (KPI) 671, 685
key tables, Travel Expense Discovery application
 Airfare table 232
 Budget table 233
 Department table 232
 Expenses table 231
 LinkTable 233
 PerDiemsRates table 231
K-means 603
KPI object
 about 94
 using, in Qlik Sense 350-353
kurtosis
 about 554
 URL 555
- L**
- labeled dataset** 604
latest record, for dimensional value
 identifying, Previous() function
 used 380, 381
Library 18
License Monitor 181
limitations
 applying, to charts 364-367
line chart 94
line-level table 386
Load Script 18
login access pass 170
Logistic Regression 654, 655
Lower Confidence Level 556
- M**
- Machine Learning (ML)**
 about 603
 association analysis 619-621
 cluster analysis 605
 hierarchical clustering 616, 618
 supervised learning 604
 unsupervised learning 604
maps generation, Qlik Sense
 URL 347
- Mashup editor**
 about 142
 pre-built templates 142, 143
mashups
 generating, Qlik Dev Hub used 485-490
master library
 creating, from data model viewer 274-281
 using, in edit mode 281, 282
measure 84
measures of central tendency
 mean 552
 median 552
 mode 552
measures of dispersion
 about 553
 quartiles 553
 range 553
 standard deviation 554
 variance 554
median, in quartile range
 identifying, FirstSortedValue() function
 used 394, 395
menus, charts
 Add-ons menu 682
 Appearance menu 683, 684
 Data menu 678-681
 Sorting menu 681
Minstring() function
 used, for calculating age of oldest case in queue 388, 389
model evaluation
 about 721
 new data, scoring 722-725
 performing 721, 722
model optimization 624
models
 Linear Regression 654
 Logistic Regression 654
 Neural Networks 655
MOOC course
 URL 621
moving annual total (MAT)
 about 398
 figure, setting up 399-401
 reference link 398

MS PowerPoint
stories, exporting to 470-473
multi measure expression
creating, in Set Analysis 429-431
Multiple Linear Regression 655
My Workspace 17

N

NetworkDays() function
used, for calculating working days in calendar month 382-385
Neural Network model
about 655
hidden layer 655
input layer 655
output layer 655
new sheet, data discovery
Combo chart object, creating 50-53
creating 48
predefined visualization, adding to 48, 49
publishing 54, 55
node 167
nominal categorical variable 534
numeric variables
about 558
Box Plot 560, 561
cumulative plot 562
histogram 561, 562

O

Open Database Connectivity (ODBC) 78, 257, 534
Operations Monitor 181
ordinal categorical variable 534
output variables 533
overfitting 632, 633

P

Peek() function
used, for creating currency Exchange Rate Calendar 404-408
used, for creating Trial Balance sheet 408-412
peoplechart.css file 151

peoplechart-properties.js file 151
performance measure
highlighting, in bar chart 295, 297
persistent colors
associating, to field values 298-300
P() function
using, in Set Analysis 435-438
pie chart 93, 94
point in time analysis
with sets 421-423
predictive analytics
about 515 604
process, steps 516
Previous() function
used, for identifying latest record for dimensional value 380, 381
private sheets
about 337
creating 337, 338
properties panel
defining, in Qlik Sense visualizations 448-451

Q

QIX engine 107
Qlik 675
Qlik Analytic Platform (QAP) 135, 136
Qlik Branch
about 160, 161
URL 160, 675
Qlik Community
URL 674
Qlik DataMarket
using 495-498
Qlik Developer Hub
about 137, 474
Engine API Explorer 138
Extension editor, using in 481-484
mashup editor 478, 479
mashup editor 138
single configurator 137
used, for generating mashups 485-490
using, in Qlik Sense 2.1.1 474
web mashups 138-150
working 479

- Qlik home page**
URL 524
- Qlik Management Console (QMC)** 165, 166
- Qlik Market**
URL 675
- QlikMarket data**
adding 129-133
using 129
- Qlik Sense**
about 313, 722
administering 163
data analysis 686
data storytelling 693-696
data visualization 672
default charts 673, 674
defining 5, 6
dimensionless bar charts,
 creating in 356-358
deployment 167
dynamic charts, using in 498-500
geo maps, creating in 342-347
HTML visualization extension, creating
 for 442-447
KPI object, using in 350-353
licensing 167
references 727
variables, defining in 466-469
- Qlik Sense 2.1.1**
Qlik Dev Hub, using in 474
visual exploration capability, using
 in 464, 465
- Qlik Sense application**
about 255
components 17, 18
creating 712-714
creating, for predicting credit risks 643-646
embedding, on website 490-494
global filtering 35-37
global search 34
navigation 30, 31
publishing, created in Qlik Sense
 desktop 336, 337
publishing, to Qlik Sense Cloud 338-342
sharing 18-20
smart visualizations 32, 33
- Qlik Sense application, using in cloud**
about 122
app, creating in Qlik Sense Cloud 125
app, sharing in Qlik Sense Cloud 126, 127
apps, maintaining 128
app, uploading from desktop 123-125
- Qlik Sense client**
extending 151-156
- Qlik Sense Cloud**
account, URL 339
Qlik Sense application, publishing
 to 338-342
- Qlik Sense data model**
about 108, 109
multitable data model, creating 109, 110
pitfalls 115, 116, 117
tables, linking 110
- Qlik Sense Desktop**
about 518
exploring 526-529
installing 523-525
keyboard shortcuts 509, 510
Legacy mode, activating 266, 267
URL 530
ways of using 517
- Qlik Sense Desktop Tutorials** 675
- Qlik Sense for Developers help**
documentation
reference 135
- Qlik Sense Installer file**
URL 336
- Qlik Sense Management Console (QMC)** 15
- Qlik Sense script**
debugging efficiently 315-319
- Qlik Sense Server 2.1.1**
URL 336
- Qlik Sense visualizations**
custom components, creating
 within 451-453
properties panel, defining in 448-451
- Qlikview** 268
- QlikView applications**
migrating, from Qlik Sense 102, 103
script changes 103
user interface changes 103

QlikView.Next project
about 5
themes 5

Qlik way
about 11
calculation on demand 11-13
color coding 11
freedom of data navigation 11

quartiles
about 553
generating, Fractile() function
used 392-394
URL 554

qvd file 265

QVS (QlikView Script File) 320

R

R
about 517, 722
downloading 519
installation, testing with R Console 521
installing 520

Random Forest 650-652

range 553

Rangesum() function
used, for plotting cumulative figures in trendlines charts 390-392

Rattle
about 518, 723
downloading 521
installing 521-523
models 653
used, for scoring loan applications 641-643

Rattle, using, for forecast
about 714
correlation analysis 714,-716
model, creating 717, 718
performance, improving 719, 720

R Console
starting, for testing R installation 521

Reference Lines
adding, to trendline charts 359, 361

Reference Lines in Sales
versus Target gauge chart 347-349

registered users 704

regression performance
measuring 659
predicted, versus observed plot 659, 660

relationships 305

reload time, of application
optimizing 328, 330

requirement specifications, authoring
engaging applications
data 68
dimensions 68
KPIs 68
security 68

rescaling
about 540
data 540, 541

Responsive Web Design (RWD) 24

Risk Chart
about 666
obtaining 666-668

ROC Curve 668, 669

roles, variable
ident 534
identifier 534
Ignore 534
input 534
risk 534
target 534

R-Square 660

S

Sales Discovery application
360-degree customer view 189
about 185
AggSales table 201
application features 187
bottom five customers, reviewing 193, 194
building 199
business problem, analyzing 185
customer sales, analyzing 197, 198
customers, filtering 189, 191
Customers table 200
features 186
productive sales representatives,
analyzing 195
products, analyzing 196
SalesDetails table 200

shipments for top customers,
 reviewing 191
 top customers, analyzing 188
 US States ISO CODE 2 polygons table 201

Sales Discovery Library
 analyzing 201
 dimensions 202
 measures 203, 204
 visualizations 205

SAP connector 268

scatter chart
 about 95-97
 data points, navigating in 369-371

script
 structuring 314, 315

script files
 code, packaging 320-322

search strings
 using, inside set modifier 431-433

security
 about 182
 authentication 182
 authorization 182
 content security 183

services, Qlik Sense
 Qlik Sense Engine Service (The QIX engine) 164
 Qlik Sense Printing Service 164
 Qlik Sense Proxy Service (QPS) 164
 Qlik Sense Repository Database 164
 Qlik Sense Repository Service (QRS) 164
 Qlik Sense Scheduler Service (QSS) 164
 Qlik Sense Service Dispatcher 164

Set Analysis
 = sign, using with variables 419-421
 about 413, 414
 comparison sets, using in 424-427
 E() function, using in 435-438
 embedded functions, using in 427-429
 multi measure expression, creating
 in 429-431
 P() function, using in 435-438
 syntax, cracking for 414, 415

Set Analysis expressions
 about 416
 flags, using in 417-419

set modifier
 about 416
 search strings, using inside 431-433

sheets
 about 26
 base sheets 26
 community sheets 27
 my sheets 27

simple data app
 creating 580

Simple Linear Regression 655

single configurator
 Qlik Sense application, embedding on website 490-494

skewness
 about 554
 URL 555

Slowly Changing Dimensions
 handling, extended interval match
 used 374-380

smart data load profiling
 using 503-507

Smart Search
 using 501-503

snapshots
 creating 286-288

standard deviation
 about 554
 reference link 361

Standard Error 556

Statistical Process Control (SPC) 361

stories
 creating 55
 defining 56, 57
 exporting, to MS PowerPoint 470-473
 media library 62-64
 publishing 64
 shapes, adding 61, 62
 snapshots, creating 58-60
 text, adding 60

storytelling 86

streams 26, 167

string of field values, as dimension
 displaying, Concat() function used 386, 387

sub routines
 using, in Qlik Sense 322-325

summary reports

- about 552
- measures of central tendency 552
- measures of dispersion 553
- measures of shape of distribution 554
- supervised learning** 604
- Supported Vector Machine (SVM)** 653
- syntax**
 - cracking, for Set Analysis 414, 415

T**tables**

- about 99
- pivot table 100
- standard table 99

Target gauge chart

- versus Reference Lines in Sales 347-349

target variables 533**tasks**

- about 177
- reload tasks 178
- user synchronizations 178

testing 657**Text**

- creating 361-364

text summaries

- about 552
- missing values, displaying 556, 557
- summary reports 552

thumbnails

- adding 367-369

tokens 170**training** 657**training dataset** 604**Travel Expense Discovery application**

- business problem, analyzing 221
- data, analyzing 225, 226
- developing 230
- dimensions 234
- expenses overspent, analyzing 224
- expenses, tracking 223
- features 222, 223
- key tables, examining 231
- travel expense story, creating 227

travel expense story

- analysis, sharing 228
- creating 227
- finishing 229, 230
- overview, creating 228

tree maps

- about 353
- creating 354-356

trendline charts

- Reference Lines, adding to 359-361

Trial Balance sheet

- creating, Peek() function used 408-412

types of predictions, classifiers performance

- False Negative 661
- False Positive 661
- True Negative 661
- True Positive 661

U**UCI Machine Learning Repository**

- reference 659

UI calculation speed

- optimizing 326, 327

underfitting 632, 633**unlabeled dataset** 619**unsupervised learning** 604**Upper Confidence Level** 556**user groups**

- analysts 685
- executive management 685
- middle managers 685

user types

- consumer 179
- contributor 179
- developer 179

V**validation** 657**variables**

- = sign, using with 419-421
- about 534
- defining, in Qlik Sense 466-469

variance 554

visual exploration capability
using, in Qlik Sense 2.1.1 464, 465

visualizations

structuring 309-312

visualization toolbox 672

W

web mashups 138-141, 146-150

website

Qlik Sense application, embedding
on 490-494

Weka 534

working days, in calendar month

calculating, NetworkDays() function
used 382-385