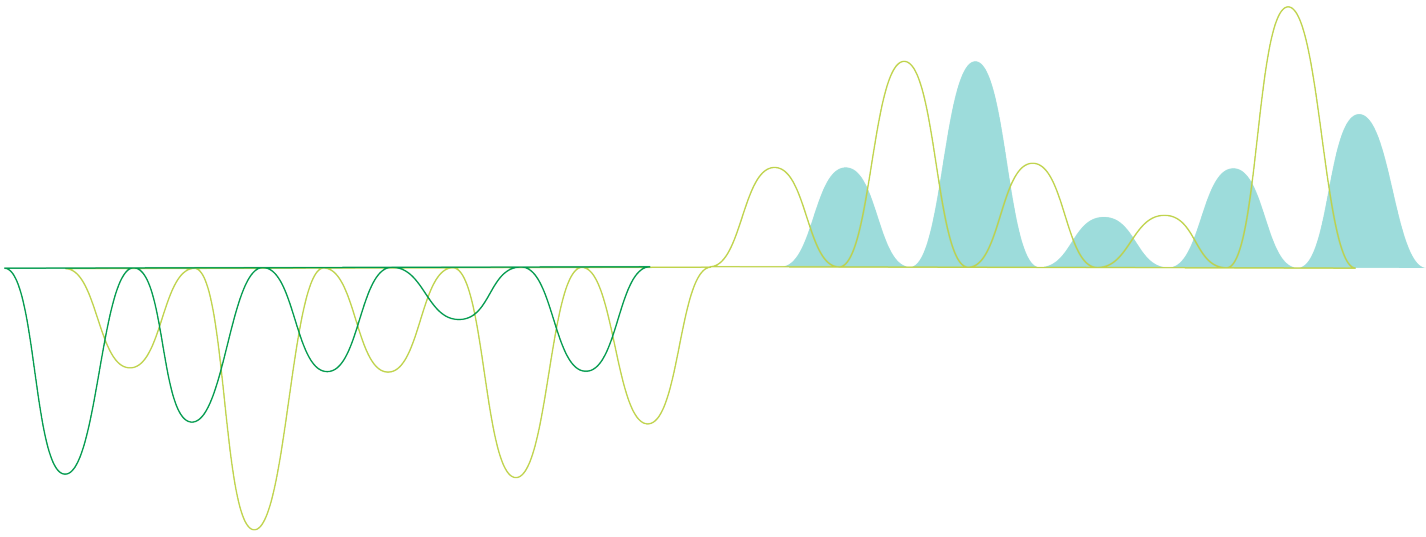
Tutorial - Chart Expressions

Qlik Sense®
May 2023



Contents

1 Welcome to this tutorial!

This tutorial introduces chart expressions in Qlik Sense. Expressions are a combination of functions, fields, and mathematical operators, used to process data and produce a result that can be seen in a visualization.

Chart expressions are mostly used in measures. You can also build visualizations that are more dynamic and powerful by using expressions for titles, subtitles, footnotes, and even dimensions.

1.1 What you will learn

When you have completed the tutorial, you will be comfortable using expressions in visualizations.

1.2 Who should complete this tutorial

You should be familiar with Qlik Sense basics. For example, you have loaded data, created apps, and created visualizations on different sheets.

You will need access to the data load editor and should be allowed to load data in Qlik Sense Enterprise on Windows.

1.3 Lessons in this tutorial

The topics in this tutorial could be completed in any order. However, later topics assume you are familiar with previous topics. The screenshots were taken in Qlik Sense Enterprise SaaS. You may see some visual differences if you are using Qlik Sense Enterprise on a different deployment.

1.4 Further reading and resources

- Qlik offers a wide variety of resources when you want to learn more.
- Qlik online help is available.
- Training, including free online courses, is available in the Qlik Continuous Classroom.
- Discussion forums, blogs, and more can be found in Qlik Community.

2 Using expressions in visualizations

Visualizations in Qlik Sense are built from charts, which are built from dimensions and measures. You can make your visualizations more dynamic and complex with expressions.

Visualizations can have titles, subtitles, footnotes, and other elements to help convey information. The elements that make up a visualization can be simple. For example: a dimension consisting of a field representing data, and a title consisting of text.

Measures are calculations based on fields. For example: **Sum(Cost)** means that all the values of the field **Cost** are aggregated using the function **Sum**. In other words, **Sum(Cost)** is an expression.

2.1 What is an expression?

An expression is a combination of functions, fields, and mathematical operators (+ * / =). Expressions are used to process data in an app in order to produce a result that can be seen in a visualization. They can be simple, involving only basic calculations, or complex, involving functions fields and operators. Expressions are used both in scripts and in chart visualizations.

All measures are expressions. The difference between measures and expressions is that expressions have no name or descriptive data.

You can build visualizations that are more dynamic and powerful by using expressions for dimensions, titles, subtitles, and footnotes. This means, for example, that instead of a static text, the title of a visualization can be generated from an expression whose result changes depending on your selections.

2.2 Where can I use expressions?

When you are editing a visualization, if an fx symbol can be seen in the properties panel, you can use an expression. Click fx to open the expression editor, which is designed to help you build and edit expressions. Expressions can also be entered directly into the expression field.

An expression cannot be saved directly as a master item. However, master measures and master dimensions can contain expressions. If an expression is used in a measure or dimension which is then saved as a master item, the expression in the measure or dimension is preserved.

2.3 When are expressions evaluated?

In a load script, an expression is evaluated as the script executes. In visualizations, expressions are evaluated automatically whenever any of the fields, variables, or functions that the expression contains change value or logical status. A few differences exist between script expressions and chart expressions in terms of syntax and available functions.

3 Which aggregation functions?

Aggregation functions are many-to-one functions. They use the values from many records as input and collapse these into one single value that summarizes all records. Sum(), Count(), Avg(), Min(), and Only() are all aggregation functions.

In Qlik Sense, you need exactly one level of aggregation function in most formulas. This includes chart expressions, text boxes, and labels. If you do not include an aggregation function in your expression, Qlik Sense will automatically assign the Only() function.

- An aggregation function is a function that returns a single value describing some property of several records in your data.
- All expressions, except calculated dimensions, are evaluated as aggregations.
- All field references in expressions must be wrapped in an aggregation function.

You can use the expression editor to create and change expressions in Qlik Sense.

3.1 Consolidating amounts using Sum()

Sum() calculates the total of the values given by the expression or field across the aggregated data.

Let us calculate the total sales that each manager has made, as well at the total sales of all managers.

Inside the app, on the *Which Aggregations?* sheet you will find two tables, a table titled *Sum(), Max(), Min()*, and a table titled *Count()*. We will use each table to create aggregation functions.

Do the following:

1. Select the available Sum(), Max(), Min() table.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.
4. Enter the following: *Sum(Sales)*
5. Click **Apply**.

Table showing total sales per Manager

Sum(), Max (), Min()

Manager	Sum(Sales)
Totals	**$ 104,852,674.81**
Dennis Johnson	$ 15,945,030.85
Stewart Wind	$ 15,422,448.79
Carolyn Halmon	$ 11,363,424.41
John Greg	$ 9,770,909.24
Samantha Allen	$ 7,540,947.33
Amanda Honda	$ 6,436,630.86
Brenda Gibson	$ 6,215,872.87
Kathy Clinton	$ 5,154,950.48
Molly McKenzie	$ 5,079,387.55
John Davis	$ 4,960,997.49

You can see the sales that each manager has made, as well as the total sales of all managers.

*As a best practice, make sure that your data is formatted appropriately. In this case, set the **Number formatting** to **Money**, and the **Format pattern** to $ #,##0;-$ #,##0.*

3.2 Calculating highest sale value using Max()

Max() finds the highest value per row in the aggregated data.

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the *fx* symbol.
   The expression editor opens.
3. Enter the following : *Max (Sales)*
4. Click **Apply**.

Table showing total sales and highest sale per Manager

Sum(), Max (), Min()

Manager	Sum(Sales)	Max(Sales)
Totals	**$ 104,852,674.81**	**$ 555,376.00**
Dennis Johnson	$ 15,945,030.85	$ 285,350.40
Stewart Wind	$ 15,422,448.79	$ 258,946.70
Carolyn Halmon	$ 11,363,424.41	$ 555,376.00
John Greg	$ 9,770,909.24	$ 310,156.07
Samantha Allen	$ 7,540,947.33	$ 52,469.65
Amanda Honda	$ 6,436,630.86	$ 133,568.68
Brenda Gibson	$ 6,215,872.87	$ 119,030.00
Kathy Clinton	$ 5,154,950.48	$ 47,326.42
Molly McKenzie	$ 5,079,387.55	$ 79,134.97
John Davis	$ 4,960,997.49	$ 118,249.47

You can see that the highest sales earnings for each manager, as well as the highest total number.

3.3 Calculating lowest sale value using Min()

Min() finds the lowest value per row, in the aggregated data.

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the *fx* symbol.
   The expression editor opens.
3. Enter the following : *Min (Sales)*
4. Click **Apply**.

Table showing total sales, highest sale, and lowest sale per Manager

Sum(), Max (), Min()

Manager	Sum(Sales)	Max(Sales)	Min(Sales)
Totals	**$ 104,852,674.81**	**$ 555,376.00**	**-$ 27,929.88**
Dennis Johnson	$ 15,945,030.85	$ 285,350.40	-$ 27,929.88
Stewart Wind	$ 15,422,448.79	$ 258,946.70	-$ 1,687.63
Carolyn Halmon	$ 11,363,424.41	$ 555,376.00	-$ 13,749.60
John Greg	$ 9,770,909.24	$ 310,156.07	-$ 17,883.07
Samantha Allen	$ 7,540,947.33	$ 52,469.65	-$ 1,687.91
Amanda Honda	$ 6,436,630.86	$ 133,568.68	-$ 15,122.77
Brenda Gibson	$ 6,215,872.87	$ 119,030.00	-$ 11,903.00
Kathy Clinton	$ 5,154,950.48	$ 47,326.42	-$ 3,418.90
Molly McKenzie	$ 5,079,387.55	$ 79,134.97	-$ 1,631.49
John Davis	$ 4,960,997.49	$ 118,240.47	$ 13,778.70

You can see the lowest sales earnings for each manager, as well as the lowest total number.

3.4 Counting the number of entities using Count()

Count() is used to count the number of values, text and numeric, in each chart dimension.

In our data, each manager is responsible for a number of sales representatives (*Sales Rep Name*). Let us calculate the number of sales representatives.

Do the following:

1. Select the available Count() table.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the *fx* symbol.
   The expression editor opens.
4. Enter the following : *Count([Sales Rep Name])*
5. Click **Apply**.

Table showing Sale Representatives, and total number of Sales Representatives.

Count()

Sales Rep Name	Count([Sales Rep Name])
Totals	**64**
Amalia Craig	1
Amanda Honda	1
Cart Lynch	1
Molly McKenzie	1
Sheila Hein	1
Brenda Gibson	1
Dennis Johnson	1
Ken Roberts	1
Robert Kim	1
William Fisher	1
Cary Frank	1
Edward Smith	1
Lee Chin	1
Ronald Milam	1

You can see that the total number of sales representatives is 64.

Difference between Count()and Count(distinct)

Let us calculate the number of managers.

Do the following:

1. Add a new dimension to your table: *Manager*.
   A single manager is handling more than one sales representative, so the same manager name appears more than once in the table.
2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.
4. Enter the following: *Count(Manager)*
5. Add another measure with the expression: *Count(distinct Manager)*
6. Click **Apply**.

Table showing Sales Representatives, total number of Sales Representatives, Manager responsible for each Sales Representative, incorrect total number of Managers, and correct total number of Managers.

Count()

Sales Rep Name	Count([Sales Rep Name])	Manager	Count(Manager)	Count(distinct Manager)
Totals	**64**		**64**	**18**
Amalia Craig	1	Amanda Honda	1	1
Amanda Honda	1	Amanda Honda	1	1
Cart Lynch	1	Amanda Honda	1	1
Molly McKenzie	1	Amanda Honda	1	1
Sheila Hein	1	Amanda Honda	1	1
Brenda Gibson	1	Brenda Gibson	1	1
Dennis Johnson	1	Brenda Gibson	1	1
Ken Roberts	1	Brenda Gibson	1	1
Robert Kim	1	Brenda Gibson	1	1
William Fisher	1	Brenda Gibson	1	1
Cary Frank	1	Carolyn Halmon	1	1
Edward Smith	1	Carolyn Halmon	1	1
Lee Chin	1	Carolyn Halmon	1	1
Ronald Milam	1	Carolyn Halmon	1	1

You can see that the total number of managers on the column using *Count(Manager)* as an expression was calculated as 64. That is not correct. The total number of managers is correctly calculated as 18 using the *Count(distinct Manager)* expression. Each manager is only counted once, regardless of how many times their name appears on the list.

4 Nested aggregations

Any field name in a chart expression must be enclosed by exactly one aggregation function. If you need to nest aggregations, you can use Aggr() to add a second aggregation level. Aggr() contains an aggregation function as an argument.

4.1 Always one level of aggregation in a function

A typical app may contain:

- one million records in the data
- one hundred rows in a pivot table
- a single KPI, in a gauge or text box

All three numbers may still represent all data, despite the difference in magnitude. The numbers are just different aggregation levels.

Aggregation functions use the values from many records as input and collapse these into one single value that can be seen as a summary of all records. There is one restriction: you cannot use an aggregation function inside another aggregation function. You usually need every field reference to be wrapped in exactly one aggregation function.

The following expressions will work:

- `Sum(Sales)`
- `Sum(Sales)/Count(Order Number)`

The following expression will not work because it is a nested aggregation:

- `Count(Sum(Sales))`

The solution to this comes in the form of the **Aggr()** function. Contrary to its name it is not an aggregation function. It is a "many-to-many" function, like a matrix in mathematics. It converts a table with N records to a table with M records. It returns an array of values. It could also be regarded as a virtual straight table with one measure, and one or several dimensions.

*Use the **Aggr()** function in calculated dimensions if you want to create nested chart aggregations on multiple levels.*

4.2 Using **Aggr()** for nested aggregations

Aggr() returns an array of values for the expression, calculated over the stated dimension or dimensions. For example, the maximum value of sales, per customer, per region. In advanced aggregations, the **Aggr()** function is enclosed in another aggregation function, using the array of results from the **Aggr()** function as input to the aggregation in which it is nested.

When it is used, the **Aggr()** statement produces a virtual table, with one expression grouped by one or more dimensions. The result of this virtual table can then be aggregated further by an outer aggregation function.

4.3 Calculating largest average order value

Let us use a simple **Aggr()** statement in a chart expression.

We want to see our overall metrics at the regional level, but also show two more complex expressions:

- Largest average order value by manager within each region.
- Manager responsible for that largest average order value.

We can easily calculate the average order value for each region using a standard expression `Sum(Sales)/Count([Order Number])`.

Inside the app, on the *Nested Aggregations* sheet you will find a table titled *Aggr() function*.

Do the following:

1. Select the available Aggr() function table.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the *fx* symbol.
   The expression editor opens.
4. Enter the following: *Sum(Sales)/Count([Order Number])*
5. Click **Apply**.

Table showing average order value per region.

Aggr() function

Region	Average order value
Totals	**$ 1,087**
Germany	$ 405
Japan	$ 604
Nordic	$ 641
Spain	$ 577
UK	$ 1,390
USA	$ 1,821

*As a best practice, make sure that your data is formatted appropriately. In this case, in each column we will change the **Label** to represent the calculation. In columns with monetary values we will change the **Number formatting** to **Money**, and the **Format pattern** to $ #,##0;-$ #,##0.*

Our goal is to retrieve the largest average order value for each region. We have to use **Aggr()** to tell Qlik Sense that we want to grab the average order value for each region, per manager, and then display the largest of those. To get the average order value for each region, per manager, we will have to include these dimensions in our **Aggr()** statement:

`Aggr(Sum(Sales)/Count([Order Number]),Region,Manager)`

This expression causes Qlik Sense to produce a virtual table that looks like this:

*Virtual table of **Aggr()** function showing average order value for each region, per manager.*

Virtual table of Aggr() function

Region	Manager	Average order value
Totals		-
Germany	Micheal Williams	$ 3,506
Germany	Dennis Johnson	$ 1,380
Germany	Molly McKenzie	$ 820
Germany	David Laychak	$ 624
Germany	John Davis	$ 456
Germany	Sheila Hein	$ 445
Germany	Amanda Honda	$ 443
Germany	John Greg	$ 436
Germany	Samantha Allen	$ 404
Germany	Stewart Wind	$ 393
Germany	William Fisher	$ 380
Germany	Ken Roberts	$ 379
Germany	Kathy Clinton	$ 335
Germany	Odessa Morris	$ 331

When Qlik Sense calculates the individual average order values for each region, per manager, we will need to find the largest of these values. We do this by wrapping the **Aggr()** function with **Max()**:

```
Max(Aggr(Sum(Sales)/Count([Order Number]),Manager,Region))
```

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the fx symbol.
   The expression editor opens.
3. Enter the following : *Max(Aggr(Sum(Sales)/ Count([Order Number]), Manager, Region))*
4. Click **Apply**.

Table showing region, average order value, and largest average order value for each region, per manager.

Aggr() function

Region	Average order value	Largest average order value
Totals	**$ 1,087**	**$ 12,338**
Germany	$ 405	$ 3,506
Japan	$ 604	$ 2,182
Nordic	$ 641	$ 2,554
Spain	$ 577	$ 1,639
UK	$ 1,390	$ 12,338
USA	$ 1,821	$ 8,615

You can see the largest average order value for all managers at the region level. This is the first of our two complex expressions! The next requirement is to have the name of the manager responsible for these large average order values displayed next to the values themselves.

To do this, we will use the same **Aggr()** function as before, but this time together with the **FirstSortedValue()** function. The **FirstSortedValue()** function tells Qlik Sense to provide us with the manager, for the specific dimension specified in the second portion of the function:

```
FirstSortedValue(Manager,-Aggr(Sum(Sales)/Count(Order Number),Manager,Region))
```

*There is one small, but very important, part of the expression: there is a minus symbol before the **Aggr()** expression. Within a **FirstSortedValue()** function, you can specify the sort order of the array of data. In this case, the minus symbol tells Qlik Sense to sort from largest to smallest.*

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *FirstSortedValue(Manager,-Aggr(Sum(Sales)/ Count([Order Number]), Manager, Region))*
4. Click **Apply**.

Table showing region, average order value, largest average order value for each region, and manager responsible for that order value.

Aggr() function

Region	Average order value	Largest average order value	Manager
Totals	**$ 1,087**	**$ 12,338**	**Dennis Johnson**
Germany	$ 405	$ 3,506	Micheal Williams
Japan	$ 604	$ 2,182	Brenda Gibson
Nordic	$ 641	$ 2,554	Kathy Clinton
Spain	$ 577	$ 1,639	Micheal Williams
UK	$ 1,390	$ 12,338	Dennis Johnson
USA	$ 1,821	$ 8,615	Carolyn Halmon

5 Naked field references

A field is considered naked when it is not enclosed in an aggregation function.

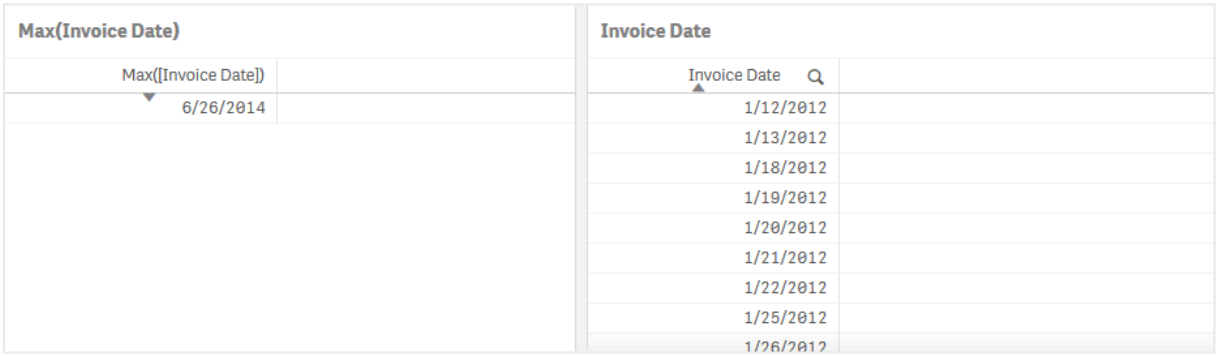
A naked field reference is an array, possibly containing several values. If so Qlik Sense will evaluate it as NULL, not knowing which of these values you want.

5.1 Always use an aggregation function in your expression

If you find that your expression does not evaluate correctly, there is a high chance that it does not have an aggregation function.

A field reference in an expression is an array of values. For example:

*Two tables, one showing that **Max(**Invoice Date**)** is a single value, and one showing that Invoice Date is an array of values.*



You must enclose the field *Invoice Date* in an aggregation function to make it collapse into a single value.

If you do not use an aggregation function on your expression, Qlik Sense will use the **Only()** function by default. If the field reference returns several values, Qlik Sense will interpret it as NULL.

Splitting invoice dates using the **If()** function

The **If()** function is often used for conditional aggregations. It returns a value depending on whether the condition provided within the function evaluates as True or False.

Inside the app, on the *Naked field references* sheet you will find a table titled *Using If() on Invoice dates*.

Do the following:

1. Select the available table titled Using If() on Invoice dates.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.

4. Enter the following: *If([Invoice Date]>= Date(41323), 'After', 'Before')*
5. Click **Apply**.

Table showing invoice dates being split by a reference date.

Using If() on Invoice dates

Date	if([Invoice Date]>= Date(41323), 'After', 'Before')
Totals	**Before**
2/10/2013	Before
2/11/2013	Before
2/12/2013	Before
2/13/2013	Before
2/14/2013	Before
2/17/2013	Before
2/18/2013	After
2/19/2013	After
2/20/2013	After
2/21/2013	After
2/24/2013	After
2/25/2013	After

This expression tests if the *Invoice Date* is before the reference date 2/18/2013 and returns 'Before' if it is. If the date is after or equal to the reference date 2/18/2013, 'After' is returned. The reference date is expressed as the integer number 41323.

5.2 Avoiding naked field references

At first glance, this expression looks correct:

```
If([Invoice Date]>=Date(41323),'After','Before')
```

It should evaluate invoice dates after the reference date, return 'After' or else return 'Before'. However, *Invoice Date* is a naked field reference, it does not have an aggregation function, and as such is an array with several values and will evaluate to NULL. In the previous example, there was only one *Invoice Date* per *Date* value in our table, so the expression calculated correctly.

Let's see how a similar expression calculates under a different dimensional value, and how to solve the naked field reference issue:

Avoiding naked field references in an **If()** function

We will be using a similar expression as before:

```
If([Invoice Date]>=Date(41323),Sum(Sales))
```

This time the function sums the sales after the reference date.

Inside the app, on the *Naked field references* sheet you will find a table titled *Sum(Amount)*.

Do the following:

1. Select the available Sum(Amount) table.
   The properties panel opens.

2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.
4. Enter the following: *If([Invoice Date]>= 41323, Sum(Sales))*
5. Click **Apply**.

*Table showing year, sum of sales for each year, and the results of the expression using the **If()** function.*

Sum(Amount)

Year	Sum(Sales)	If([Invoice Date]>= Date(41323), Sum(Sales))
Totals	**$ 104,852,675**	**-**
2012	$ 40,173,302	-
2013	$ 42,753,991	-
2014	$ 21,925,382	-

*Keep the **Label** intact on the measures to show the differences between each expression. In columns with monetary values, change the **Number formatting** to **Money**, and the **Format pattern** to $ #,##0;-$ #,##0.*

For each year there is an array of invoice dates that come after the reference date. Since our expression lacks an aggregation function it evaluates to NULL. A correct expression should use an aggregation function such as **Min()** or **Max()** in the first parameter of the **If()** function:

```
If(Max([Invoice Date])>= Date(41323),Sum(Sales))
```

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *If([Invoice Date]>= Date(41323), Sum(Sales))*
4. Click **Apply**.

*Table showing year, sum of sales for each year, and the results of the different expressions using the **If()** function.*

Sum(Amount)

Year	Sum(Sales)	If([Invoice Date]>= Date(41323), Sum(Sales))	If(Max([Invoice Date])>= Date(41323), Sum(Sales))
Totals	**$ 104,852,675**	**-**	**$ 104,852,675**
2012	$ 40,173,302	-	-
2013	$ 42,753,991	-	$ 42,753,991
2014	$ 21,925,382	-	$ 21,925,382

Alternatively, the **If()** function can be put inside the **Sum()** function:

```
Sum(If([Invoice Date]>= Date(41323),Sales) )
```

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *Sum(If([Invoice Date]>= Date(41323), Sales))*
4. Click **Apply**.

*Table showing year, sum of sales for each year, and the results of the different expressions using the **If()** function.*

Sum(Amount)				
Year	Sum(Sales)	If([Invoice Date]>= Date(41323), Sum(Sales))	If(Max([Invoice Date])>= Date(41323), Sum(Sales))	Sum(If([Invoice Date]>= Date(41323), Sales))
Totals	**$ 104,852,675**	**-**	**$ 104,852,675**	**$ 58,563,348**
2012	$ 40,173,302	-	-	$ 0
2013	$ 42,753,991	-	$ 42,753,991	$ 36,637,967
2014	$ 21,925,382	-	$ 21,925,382	$ 21,925,382

In the second to last expression, the **If()** function was evaluated once per dimensional value. In the last expression, it is evaluated once per row in the raw data. The difference in how the function is evaluated causes the results to be different, but both return an answer. The first expression simply evaluates to NULL. The picture above shows the difference between the expressions, using 2/18/2013 as the reference date.

6 The importance of Only()

Only() returns a value if there is only one possible value in the group. This value will be the result of the aggregation. Qlik Sense defaults to **Only()** if no aggregation function is specified.

If there is a one-to-one relationship between the chart dimension and the parameter, the **Only()** function returns the only possible value. If there are several values, it returns NULL. For example, searching for the only product where the unit price =12 will return NULL if more than one product has a unit price of 12.

The following images show the difference between one-to-one and one-to-many relationships:

A table showing one-to-one relationship between Manager Number and Manager

One-to-one relationship

Manager Number	Manager
104	Amanda Honda
109	Brenda Gibson
111	Carolyn Halmon
118	David Laychak
121	Dennis Johnson
132	John Davis
134	John Greg
144	Kathy Clinton
145	Ken Roberts
157	Micheal Williams
159	Molly McKenzie
160	Odessa Morris
169	Samantha Allen
176	Sheila Hein
179	Stephanie Reagan
181	Stewart Wind
184	Viginia Mountain
185	William Fisher

A table showing one-to-many relationship of Sales Rep Name and Manager.

One-to-many relationship

Sales Rep Name	Manager
Amalia Craig	Amanda Honda
Amanda Honda	Amanda Honda
Cart Lynch	Amanda Honda
Molly McKenzie	Amanda Honda
Sheila Hein	Amanda Honda
Brenda Gibson	Brenda Gibson
Dennis Johnson	Brenda Gibson
Ken Roberts	Brenda Gibson
Robert Kim	Brenda Gibson
William Fisher	Brenda Gibson
Cary Frank	Carolyn Halmon
Edward Smith	Carolyn Halmon
Lee Chin	Carolyn Halmon
Ronald Milam	Carolyn Halmon
Amelia Fields	David Laychak
Deborah Halmon	David Laychak
Judy Rowlett	David Laychak
Angelen Carter	Dennis Johnson
Dennis Fisher	Dennis Johnson

The **Only()** function is an aggregation function. It uses many records as input and returns one value only, similarly to **Sum()** or **Count()**. Qlik Sense uses aggregations in virtually all its calculations. The expression in a chart, in a sort expression, in a text box, in an advanced search, and in a calculated label are all aggregations and cannot be calculated without involving an aggregation function.

But what if a user enters an expression that lacks an explicit aggregation function? For example, if the sort expression is set to *Date*? Or if there is an advanced search for customers who have bought beer and wine products using the expression `=[Product Type]='Beer and Wine'`?

This is where the **Only()** function affects the calculation. If there is no explicit aggregation function in the expression, Qlik Sense uses the **Only()** function implicitly. In the above cases, **`Only(`**`Date`**`)`** is used as sort expression and **`Only(`**`[Product Type]`**`)`**`='Beer and Wine'` is used as the search criterion.

Sometimes the new expression returns a result that the user does not expect. Both of the examples above will work when there is only one possible value of *Date* or *Product Type*, but neither of them will work for cases when there is more than one value.

6.1 Different expressions using **Only()**

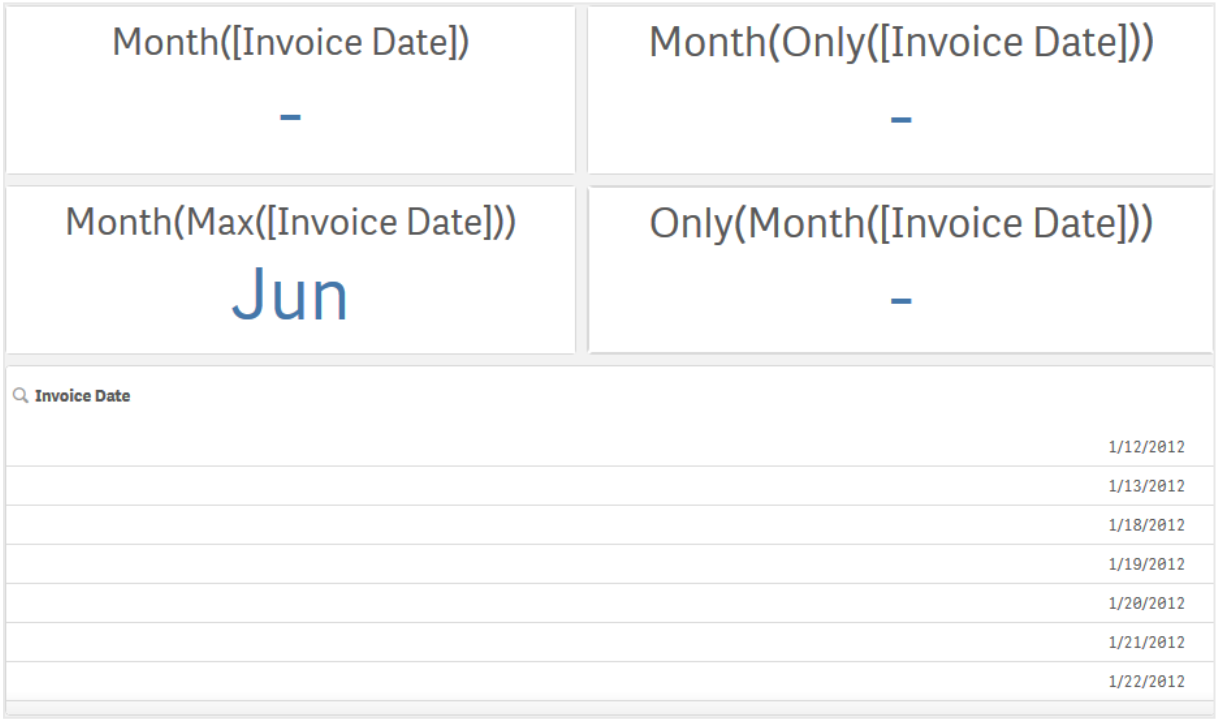
We will create four KPIs with similar expressions. This way, we can compare how having naked field references, or having **Only()** in a different position in our expression can have a big impact on your selection results.

Inside the app, on the *Importance of Only()* sheet you will find a filter pane with *Invoice Date* as the dimension.

Do the following:

1. Create a KPI.
2. Click **Add measure**. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *Month([Invoice Date])*
4. Create three more KPIs with measures: *Month(Only([Invoice Date]))*, *Month(Max([Invoice Date]))*, and *Only(Month([Invoice Date]))*.
5. Click **Apply**.

Four KPIs and a filter pane showing three different but similar expressions.



*In each KPI the **Number formatting** has been set to **Measure expression**.*

When you have a naked field reference, the **Only()** function is inserted at the lowest level. That means that the first two KPIs, *Month([Invoice Date])* and *Month(Only([Invoice Date]))*, will be interpreted the same and will always give the same result.

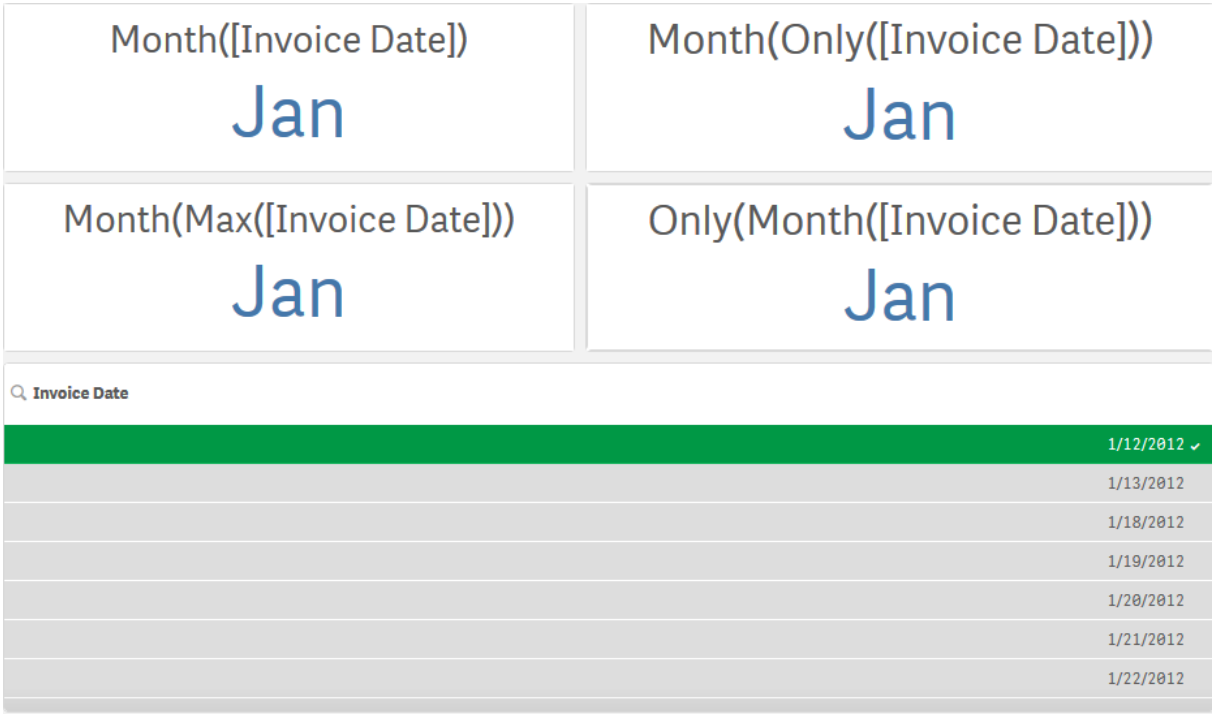
As you can see three of the four KPIs return NULL. The third KPI, *Month(Max([Invoice Date]))*, already returns a value, even though no selection has been made.

When you write expressions you should always ask yourself which aggregation you want to use, or which value you want to use if there are several values. If you want to use NULL to represent several values, you can leave the expression as is. For numbers, you probably want to use **Sum()**, **Avg()**, **Min()**, or **Max()** instead. For strings you may want to use **Only()** or **MinString()**.

Do the following:

1. Stop editing the sheet.
2. In the filter pane, select date in the month of January.
3. Confirm the selection by clicking ✔.

The KPI results change when a single selection is made.

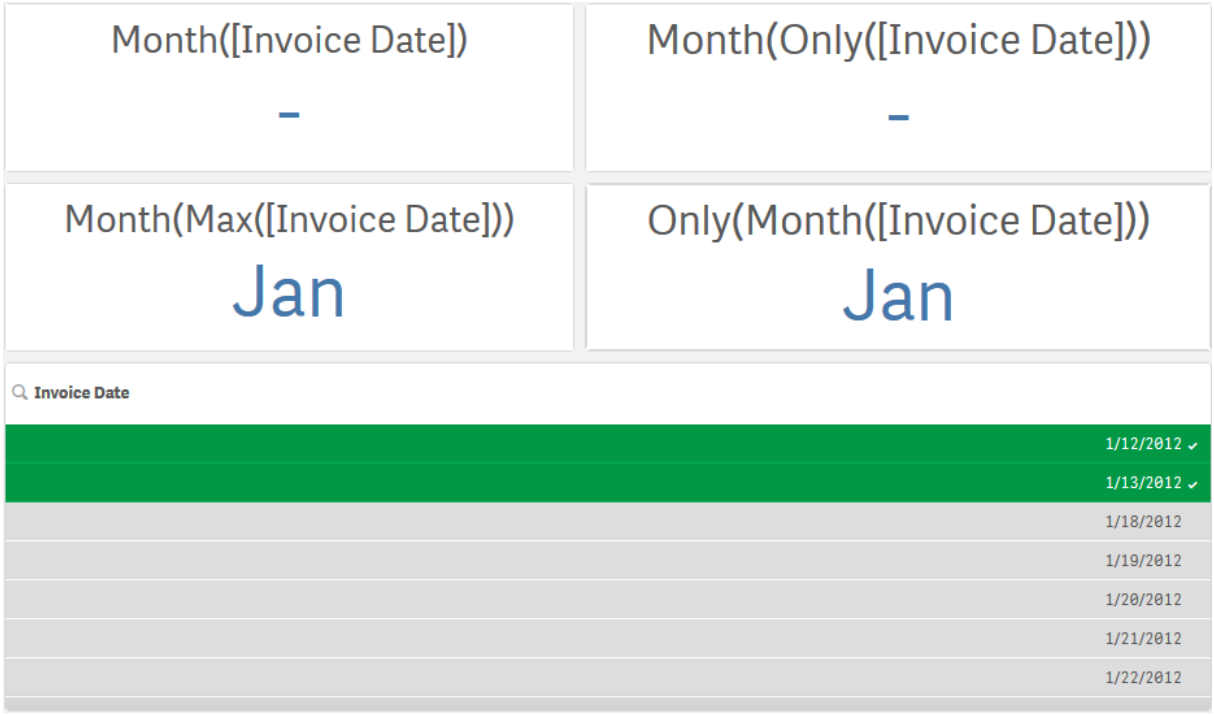


When a single selection is made, all of the KPIs return the correct answer. Even if the expression contains a naked field reference, such as the expression in *Month([Invoice Date])*, the fact that we have made a unique selection allows it to return the proper value.

Do the following:

1. In the filter pane, select an additional date in the month of January.
2. Confirm the selection by clicking ✔.

The KPI results change when two selections are made with both dates in the month of January.

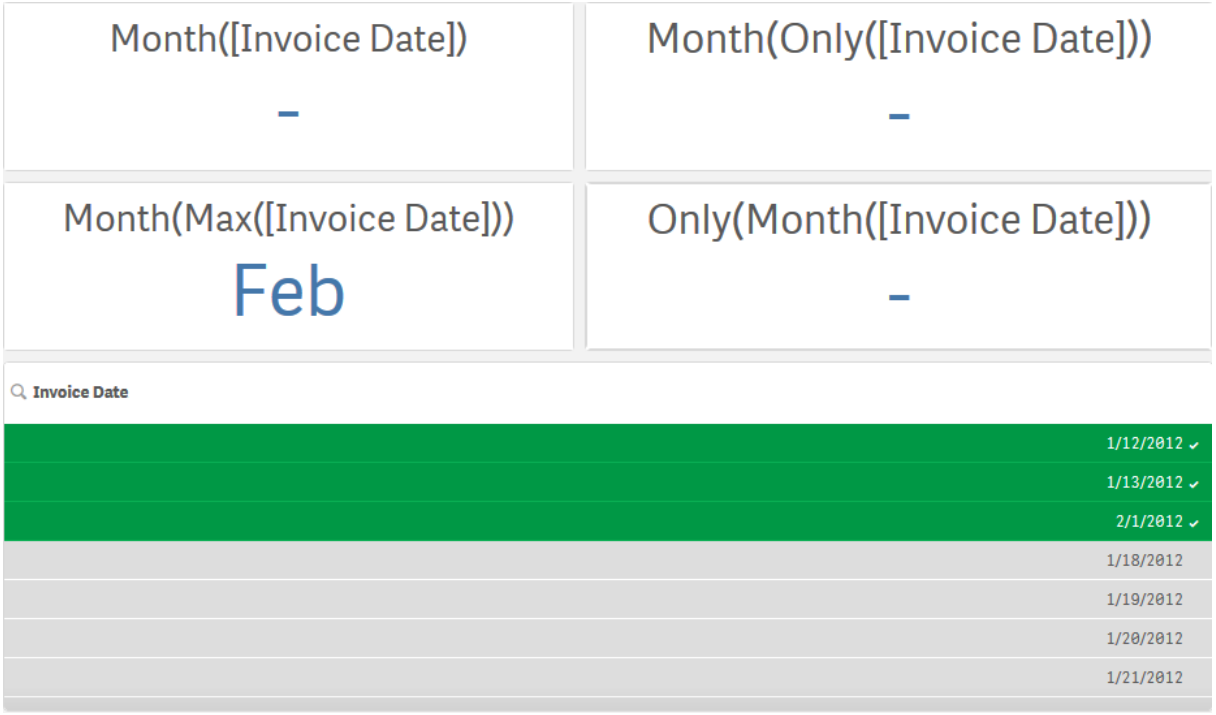


The first two KPIs return NULL, and the other two KPIs return the proper value of January. Specifically, the fourth KPI returns a correct answer because both the date selections we made are for dates in January.

Do the following:

1. In the filter pane, select an additional date, in a month other than January.
2. Confirm the selection by clicking ✔.

The KPI results change when multiple selections are made with dates in different months.



When multiple selections are made, using dates in different months, only the third KPI returns a value. It returns the value of the largest month from the selection made, according to the expression *Month(Max ([Invoice Date]))*. Since Only() is inserted automatically in expressions with naked field references you cannot always assume that the lowest level will be appropriate for your expression. The placement of Only() is important.

7 Examples from real life

Visualizations in Qlik Sense can give you insight in your data. Using expressions in your charts can bring results that specifically apply to your work. The range of functions in Qlik Sense allow you to customize your expressions to fit your needs, even if the option is not readily available.

7.1 Calculating the gross margin percentage

We define the margin as the difference between our sales and the cost of making these sales. We will calculate the margin for each month, as well as what percentage of the monthly sales is our margin.

To calculate the margin percentage we can use the following expression:

```
(Sum(Sales)-Sum(Cost))/Sum(Sales)
```

The expression can be simplified further

```
1-Sum(Cost)/Sum(Sales)
```

Inside the app, on the *Examples from real life* sheet you will find a table titled *Margin*.

Do the following:

1. Select the available table titled Margin.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.
4. Enter the following: *Sum(Sales)*
5. Add three more measures with the expressions: *Sum(Cost)*, *Sum(Sales) - Sum(Cost)*, and *1 - Sum (Cost)/Sum(Sales)*.
6. Click **Apply**.

Table showing sum of sales, and sum of cost per month, as well as calculated margin per month in both amount and percentage forms

Margin

Month	Sum(Sales)	Sum(Cost)	Calculated Margin	Margin %
Totals	**$ 104,852,675**	**$ 61,571,565**	**$ 43,281,110**	**41%**
2012-Jan	$ 1,773,750	$ 1,122,474	$ 651,276	37%
2012-Feb	$ 3,867,568	$ 2,352,955	$ 1,514,613	39%
2012-Mar	$ 3,892,195	$ 2,339,154	$ 1,553,041	40%
2012-Apr	$ 3,660,634	$ 2,241,036	$ 1,419,598	39%
2012-May	$ 3,191,648	$ 1,961,629	$ 1,230,019	39%
2012-Jun	$ 4,259,260	$ 2,540,976	$ 1,718,284	40%
2012-Jul	$ 2,519,873	$ 1,488,274	$ 1,031,598	41%
2012-Aug	$ 3,799,274	$ 2,312,303	$ 1,486,971	39%
2012-Sep	$ 3,739,098	$ 2,239,469	$ 1,499,629	40%
2012-Oct	$ 3,036,456	$ 1,897,354	$ 1,139,102	38%
2012-Nov	$ 3,528,099	$ 2,193,961	$ 1,334,138	38%
2012-Dec	$ 2,905,449	$ 1,693,359	$ 1,212,089	42%
2013-Jan	$ 4,574,043	$ 2,691,980	$ 1,882,063	41%
2013-Feb	$ 3,333,840	$ 1,925,155	$ 1,408,685	42%
2013-Mar	$ 4,266,053	$ 2,521,409	$ 1,744,645	41%
2013-Apr	$ 2,498,576	$ 1,417,551	$ 1,081,024	43%
2013-May	$ 3,533,538	$ 2,040,086	$ 1,493,452	42%
2013-Jun	$ 4,115,434	$ 2,386,136	$ 1,729,298	42%
2013-Jul	$ 2,696,222	$ 1,515,881	$ 1,180,341	44%
2013-Aug	$ 3,792,982	$ 2,165,853	$ 1,627,129	43%
2013-Sep	$ 4,087,106	$ 2,395,942	$ 1,691,164	41%
2013-Oct	$ 2,917,027	$ 1,699,705	$ 1,217,322	42%
2013-Nov	$ 3,647,346	$ 2,161,120	$ 1,486,225	41%
2013-Dec	$ 3,291,823	$ 1,925,886	$ 1,365,936	41%
2014-Jan	$ 4,114,861	$ 2,363,597	$ 1,751,264	43%
2014-Feb	$ 3,198,718	$ 1,732,256	$ 1,466,461	46%
2014-Mar	$ 3,789,271	$ 2,131,698	$ 1,657,573	44%
2014-Apr	$ 3,575,329	$ 2,035,458	$ 1,539,871	43%
2014-May	$ 3,541,237	$ 2,015,104	$ 1,526,133	43%
2014-Jun	$ 3,705,966	$ 2,063,802	$ 1,642,164	44%

*As a best practice, make sure that your data is formatted appropriately. In this case, in each column we will change the **Label** to represent the calculation. In columns with monetary values we will change the **Number formatting** to **Money**, and the **Format pattern** to $ #,##0;-$ #,##0. Set the **Number formatting** of the margin percentage to **Number**, and the **Formatting** to **Simple** and **12%**.*

You can see the calculated margin for each month based on the sales and the cost. You can also see what percentage of the sales makes up our margin.

In the app data, we already have data for the monthly margin. This is a good opportunity to make a comparison between our original data and our calculation.

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the *fx* symbol.
   The expression editor opens.

3. Enter the following : *Sum(Margin)*
4. Add another measure with the expression: *(Sum(Sales) - Sum(Cost)) - Sum(Margin)*
5. Click **Apply**.

The margin table with additional columns for monthly margin coming from the data set, and its difference to the calculated margin.

Margin

Month	Sum(Sales)	Sum(Cost)	Calculated Margin	Margin %	Sum(Margin)	Margin Discrepancy
Totals	**$ 104,852,675**	**$ 61,571,565**	**$ 43,281,110**	**41%**	**$ 43,253,189**	**$ 27,921**
2012-Jan	$ 1,773,750	$ 1,122,474	$ 651,276	37%	$ 651,276	-$ 0
2012-Feb	$ 3,867,568	$ 2,352,955	$ 1,514,613	39%	$ 1,514,613	-$ 0
2012-Mar	$ 3,892,195	$ 2,339,154	$ 1,553,041	40%	$ 1,553,041	-$ 0
2012-Apr	$ 3,660,634	$ 2,241,036	$ 1,419,598	39%	$ 1,419,598	-$ 0
2012-May	$ 3,191,648	$ 1,961,629	$ 1,230,019	39%	$ 1,230,019	-$ 0
2012-Jun	$ 4,259,260	$ 2,540,976	$ 1,718,284	40%	$ 1,718,284	$ 0
2012-Jul	$ 2,519,873	$ 1,488,274	$ 1,031,598	41%	$ 1,031,598	-$ 0
2012-Aug	$ 3,799,274	$ 2,312,303	$ 1,486,971	39%	$ 1,486,971	$ 0
2012-Sep	$ 3,739,098	$ 2,239,469	$ 1,499,629	40%	$ 1,499,629	-$ 0
2012-Oct	$ 3,036,456	$ 1,897,354	$ 1,139,102	38%	$ 1,139,102	-$ 0
2012-Nov	$ 3,528,099	$ 2,193,961	$ 1,334,138	38%	$ 1,334,138	-$ 0
2012-Dec	$ 2,905,449	$ 1,693,359	$ 1,212,089	42%	$ 1,212,089	-$ 0
2013-Jan	$ 4,574,043	$ 2,691,980	$ 1,882,063	41%	$ 1,882,063	$ 0
2013-Feb	$ 3,333,840	$ 1,925,155	$ 1,408,685	42%	$ 1,408,685	$ 0
2013-Mar	$ 4,266,053	$ 2,521,409	$ 1,744,645	41%	$ 1,744,645	$ 0
2013-Apr	$ 2,498,576	$ 1,417,551	$ 1,081,024	43%	$ 1,081,024	$ 0
2013-May	$ 3,533,538	$ 2,040,086	$ 1,493,452	42%	$ 1,493,452	$ 0
2013-Jun	$ 4,115,434	$ 2,386,136	$ 1,729,298	42%	$ 1,729,298	-$ 0
2013-Jul	$ 2,696,222	$ 1,515,881	$ 1,180,341	44%	$ 1,180,341	-$ 0
2013-Aug	$ 3,792,982	$ 2,165,853	$ 1,627,129	43%	$ 1,627,129	$ 0
2013-Sep	$ 4,087,106	$ 2,395,942	$ 1,691,164	41%	$ 1,691,164	-$ 0
2013-Oct	$ 2,917,027	$ 1,699,705	$ 1,217,322	42%	$ 1,217,322	$ 0
2013-Nov	$ 3,647,346	$ 2,161,120	$ 1,486,225	41%	$ 1,486,225	-$ 0
2013-Dec	$ 3,291,823	$ 1,925,886	$ 1,365,936	41%	$ 1,365,936	-$ 0
2014-Jan	$ 4,114,861	$ 2,363,597	$ 1,751,264	43%	$ 1,731,437	$ 19,827
2014-Feb	$ 3,198,718	$ 1,732,256	$ 1,466,461	46%	$ 1,463,099	$ 3,363
2014-Mar	$ 3,789,271	$ 2,131,698	$ 1,657,573	44%	$ 1,657,573	-$ 0
2014-Apr	$ 3,575,329	$ 2,035,458	$ 1,539,871	43%	$ 1,537,112	$ 2,759
2014-May	$ 3,541,237	$ 2,015,104	$ 1,526,133	43%	$ 1,526,133	-$ 0
2014-Jun	$ 3,705,966	$ 2,063,802	$ 1,642,164	44%	$ 1,640,192	$ 1,972

Some values in the calculated margin column differ from the values from the margin column coming directly from our data. The margin discrepancy column clearly shows that this takes place in a months during 2014. The difference between the calculated margin and the margin coming from the data set is small, but the fact that it takes place in a specific year creates some questions. What changed during that year? Looking into the data and asking the right questions might prove to be important for your business.

7.2 Invoicing delays

For this example we will be using data based on a company that collects dates both for the creation of invoices and the promised delivery of the goods they produce. The two dates are not always the same. Additionally some invoices might have two promised delivery dates. The shortest date is always the same as

the invoice date, as it is automatically created by the invoicing system used by the company. The largest promised delivery date is the date when a delivery was agreed to be made between the company and the client.

Let us start by adding these dates on a table.

On the *Examples from real life* sheet you will find a table titled *Invoicing delays*.

Do the following:

1. Select the available table titled Invoicing delays.
   The properties panel opens.
2. Click **Add column** and select **Measure**.
3. Click on the fx symbol.
   The expression editor opens.
4. Enter the following : *Only([Invoice Date])*
5. Add another measure with the expression: *Max([Promised Delivery Date])*
6. Click **Apply**.

Table showing promised delivery date and invoice date for each invoice

Invoicing delays

Invoice Number	Invoice date	Promised delivery date
Totals	-	**31 Dec 2014**
100001	30 Apr 2013	29 Apr 2013
100002	30 Apr 2013	30 Apr 2013
100005	30 Apr 2013	30 Apr 2013
100006	30 Apr 2013	30 Apr 2013
100007	30 Apr 2013	30 Apr 2013
100008	30 Apr 2013	30 Apr 2013
100009	30 Apr 2013	30 Apr 2013
100010	30 Apr 2013	30 Apr 2013
100011	01 May 2013	01 May 2013
100013	01 May 2013	01 May 2013
100018	02 May 2013	02 May 2013
100021	02 May 2013	02 May 2013
100023	02 May 2013	02 May 2013
100027	03 May 2013	03 May 2013
100028	03 May 2013	03 May 2013
100029	03 May 2013	03 May 2013
100030	03 May 2013	03 May 2013
100034	06 May 2013	06 May 2013
100036	06 May 2013	06 May 2013

*As a best practice, make sure that your data is formatted appropriately. In columns that show dates, set the **Number formatting** to **Date**, and set the **Formatting** to **Simple** and **17 Feb 2014**.*

You can see that the invoice date and the promised delivery date are not always the same. When there are two promised delivery dates we need to use the largest one for our calculation.

Let us calculate the difference between the invoice date and the promised delivery date. We will use the following expression:

```
Max([Promised Delivery Date])-[Invoice Date]
```

There are three scenarios:

- The two dates are the same, and the result of the expression is 0.
- The products were promised after the invoice was created, and the result is a positive integer.
- The invoice was created after the products were promised to be delivered, and the result is a negative integer.

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the *fx* symbol.
   The expression editor opens.
3. Enter the following : *Max([Promised Delivery Date])-[Invoice Date]*
4. Click **Apply**.

Table showing promised delivery date and invoice date for each invoice, as well as the number of days from invoicing to promised delivery

Invoicing delays

Invoice Number	Invoice date	Promised delivery date	Days from invoicing to delivery
Totals	-	**31 Dec 2014**	-
307258	21 Jul 2012	22 Feb 2012	-150
108707	30 Jul 2013	29 Apr 2013	-92
109851	09 Aug 2013	14 May 2013	-87
111190	26 Aug 2013	31 May 2013	-87
112112	05 Sep 2013	10 Jun 2013	-87
116817	28 Oct 2013	16 Aug 2013	-73
109998	12 Aug 2013	05 Jun 2013	-68
113609	23 Sep 2013	22 Jul 2013	-63
115559	14 Oct 2013	12 Aug 2013	-63
108081	22 Jul 2013	21 May 2013	-62
109357	05 Aug 2013	05 Jun 2013	-61
310525	26 Aug 2012	26 Jun 2012	-61
315709	25 Oct 2012	25 Aug 2012	-61
329238	27 Dec 2012	27 Oct 2012	-61
103809	03 Jun 2013	08 Apr 2013	-56
112368	09 Sep 2013	16 Jul 2013	-55
118091	11 Nov 2013	18 Sep 2013	-54
112120	05 Sep 2013	15 Jul 2013	-52
112121	05 Sep 2013	18 Jul 2013	-49

Sort the table based on the last column, named Days from invoicing to delivery.

There is a range of differences between the dates. Negative values indicate that the invoice was delayed. Positive numbers indicate that the promised delivery was done after the invoice was created.

Let us calculate the number of invoices that were made after the promised delivery date.

Do the following:

1. Click **Add column** and select **Measure**.
2. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *Count(Distinct If(Aggr(Max([Promised Delivery Date])<[Invoice Date],[Invoice Number]),[Invoice Number]))*
4. Click **Apply**.

Alternatively we could use Sum(Aggr(If(Max([Promised Delivery Date])-[Invoice Date]< 0, 1, 0), [Invoice Number])).

The invoicing delays table with additional column showing the number of delayed invoices.

Invoicing delays

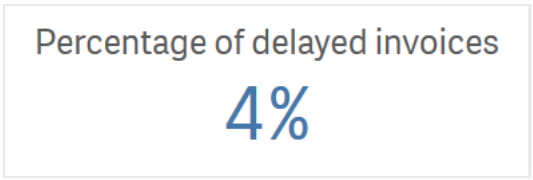
Invoice Number	Invoice date	Promised delivery date	Days from invoicing to delivery	Invoice delayed (T/F)
Totals	-	**31 Dec 2014**	-	**3421**
307258	21 Jul 2012	22 Feb 2012	-150	1
108707	30 Jul 2013	29 Apr 2013	-92	1
109851	09 Aug 2013	14 May 2013	-87	1
111190	26 Aug 2013	31 May 2013	-87	1
112112	05 Sep 2013	10 Jun 2013	-87	1
116817	28 Oct 2013	16 Aug 2013	-73	1
109998	12 Aug 2013	05 Jun 2013	-68	1
113609	23 Sep 2013	22 Jul 2013	-63	1
115559	14 Oct 2013	12 Aug 2013	-63	1
108081	22 Jul 2013	21 May 2013	-62	1
109357	05 Aug 2013	05 Jun 2013	-61	1
310525	26 Aug 2012	26 Jun 2012	-61	1
315709	25 Oct 2012	25 Aug 2012	-61	1
329238	27 Dec 2012	27 Oct 2012	-61	1
103809	03 Jun 2013	08 Apr 2013	-56	1
112368	09 Sep 2013	16 Jul 2013	-55	1
118091	11 Nov 2013	18 Sep 2013	-54	1
112120	05 Sep 2013	15 Jul 2013	-52	1
112121	05 Sep 2013	18 Jul 2013	-49	1

The last column makes more sense as a KPI as a percentage of the total number of invoices.

Do the following:

1. Create a KPI.
2. Click **Add measure**. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *Count(Distinct If(Aggr(Max([Promised Delivery Date])<[Invoice Date],[Invoice Number]),[Invoice Number]))/Count([Invoice Number])*
4. Click **Apply**.

A KPI showing the percentage of delayed invoices.



Let us calculate the average delay in invoicing.

Do the following:

1. Create a new KPI.
2. Click **Add measure**. Click on the fx symbol.
   The expression editor opens.
3. Enter the following: *Avg(Aggr(If(Max([Promised Delivery Date])<[Invoice Date],(Max([Promised Delivery Date])-[Invoice Date])), [Invoice Number]))*
4. Click **Apply**.

A KPI showing the average delay in invoicing

7.3 Thank you!

Now you have finished this tutorial, and hopefully you have gained some basic knowledge about chart expressions in Qlik Sense. Please visit our website for more inspiration for your apps.