Table Calculation Functions

Version: 2018.3 Applies to: Tableau Desktop, Tableau Online, Tableau Public, Tableau Server

This article introduces table calculation functions and their uses in Tableau. It also demonstrates how to create a table calculation using the calculation editor.

In this article

Why use table calculation functions

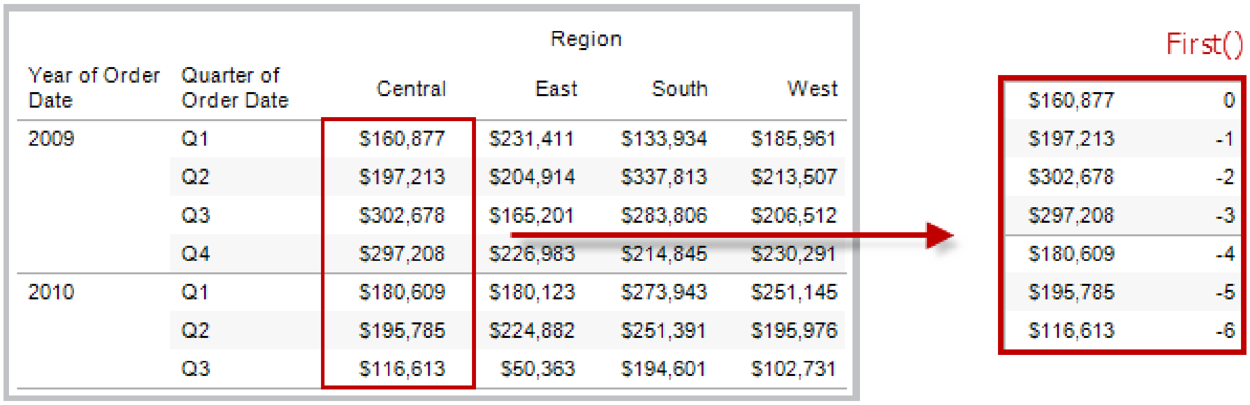
Table calculation functions allow you to perform computations on values in a table.

For example, you can calculate the percent of total an individual sale is for the year, or for several years.

Table calculation functions available in Tableau

FIRST( )

Returns the number of rows from the current row to the first row in the partition. For example, the view below shows quarterly sales. When FIRST() is computed within the Date partition, the offset of the first row from the second row is -1.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

First()

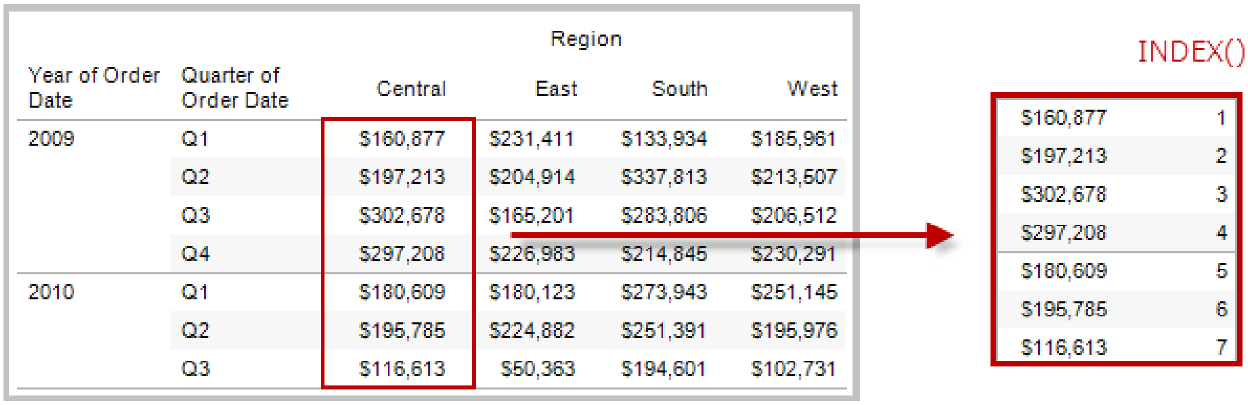
$160,877	0
$197,213	-1
$302,678	-2
$297,208	-3
$180,609	-4
$195,785	-5
$116,613	-6

Example

When the current row index is 3, `FIRST() = -2`.

INDEX( )

Returns the index of the current row in the partition, without any sorting with regard to value. The first row index starts at 1. For example, the table below shows quarterly sales. When INDEX() is computed within the Date partition, the index of each row is 1, 2, 3, 4..., etc.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

INDEX()

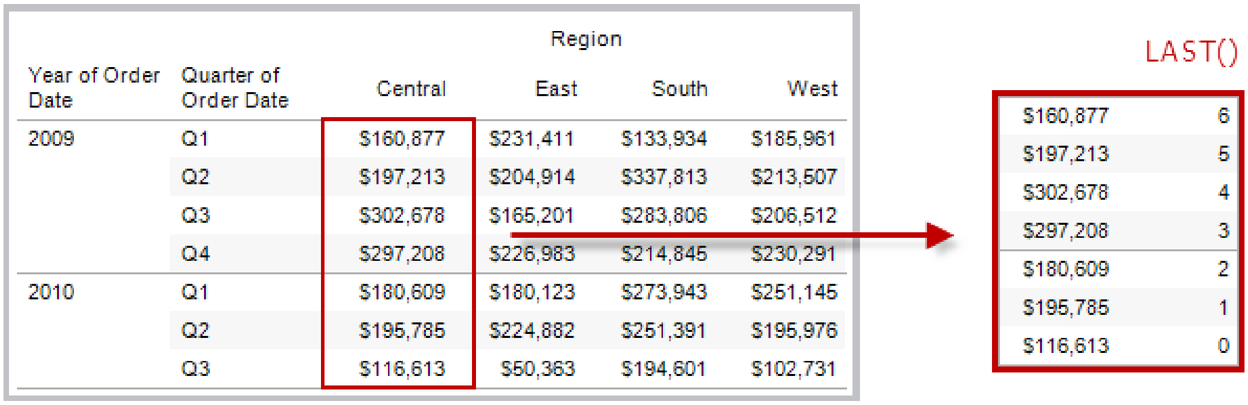
$160,877	1
$197,213	2
$302,678	3
$297,208	4
$180,609	5
$195,785	6
$116,613	7

Example

For the third row in the partition, `INDEX() = 3`.

LAST( )

Returns the number of rows from the current row to the last row in the partition. For example, the table below shows quarterly sales. When LAST() is computed within the Date partition, the offset of the last row from the second row is 5.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

LAST()

$160,877	6
$197,213	5
$302,678	4
$297,208	3
$180,609	2
$195,785	1
$116,613	0

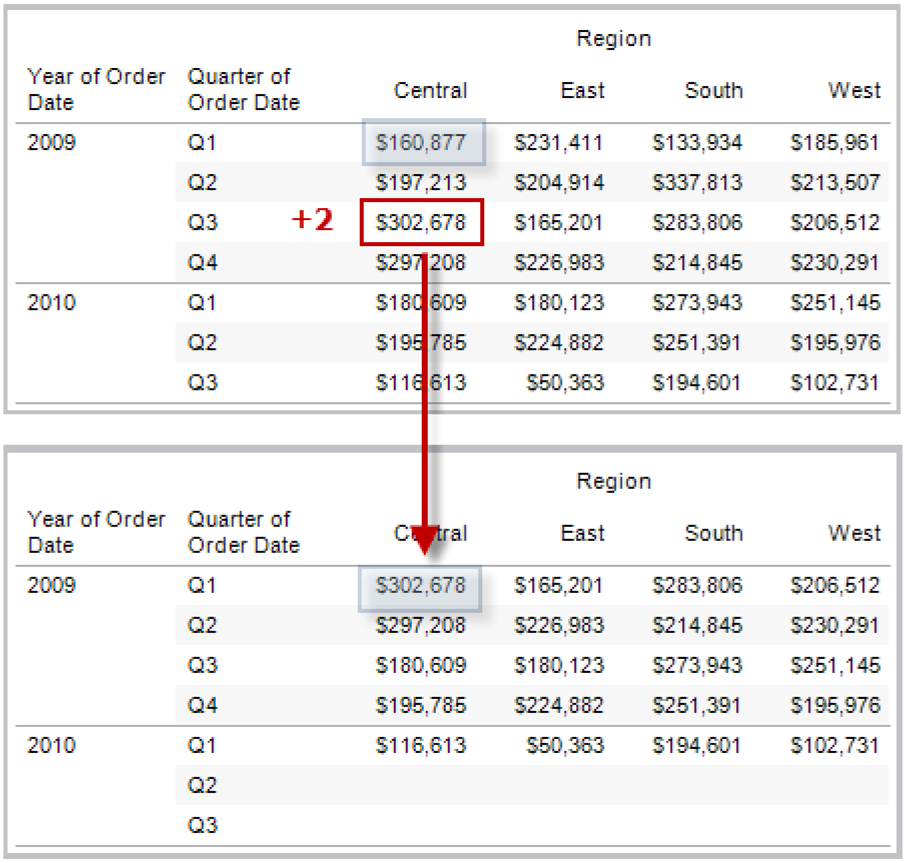
Example

When the current row index is 3 of 7, `LAST() = 4`.

LOOKUP(expression, [offset])

Returns the value of the expression in a target row, specified as a relative offset from the current row. Use FIRST() + n and LAST() - n as part of your offset definition for a target relative to the first/last rows in the partition. If `offset` is omitted, the row to compare to can be set on the field menu. This function returns NULL if the target row cannot be determined.

The view below shows quarterly sales. When `LOOKUP (SUM(Sales), 2)` is computed within the Date partition, each row shows the sales value from 2 quarters into the future.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$302,678	$165,201	$283,806	$206,512
	Q2	$297,208	$226,983	$214,845	$230,291
	Q3	$180,609	$180,123	$273,943	$251,145
	Q4	$195,785	$224,882	$251,391	$195,976
2010	Q1	$116,613	$50,363	$194,601	$102,731
	Q2				
	Q3				

Example

`LOOKUP(SUM([Profit]), FIRST()+2)` computes the SUM(Profit) in the third row of the partition.

PREVIOUS_VALUE(expression)

Returns the value of this calculation in the previous row. Returns the given expression if the current row is the first row of the partition.

Example

`SUM([Profit]) * PREVIOUS_VALUE(1)` computes the running product of SUM(Profit).

RANK(expression, ['asc' | 'desc'])

Returns the standard competition rank for the current row in the partition. Identical values are assigned an identical rank. Use the optional `'asc' | 'desc'` argument to specify ascending or descending order. The default is descending.

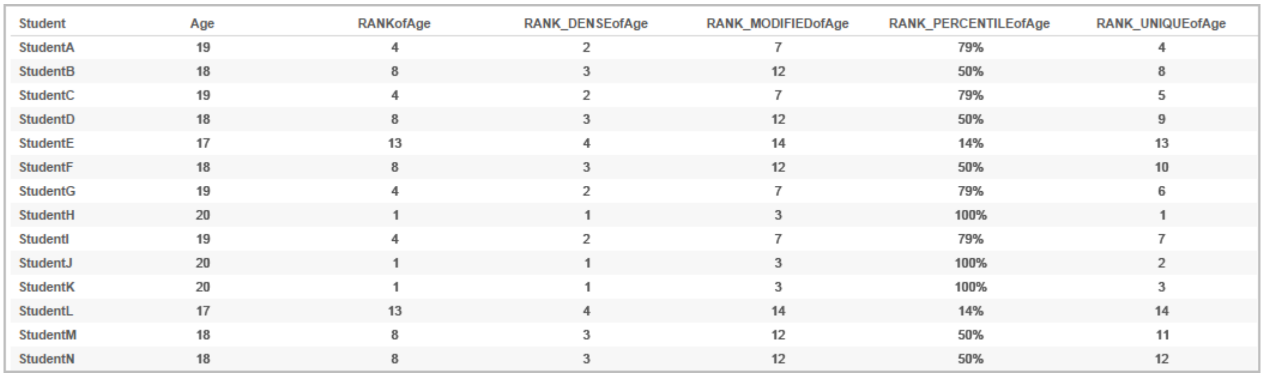
With this function, the set of values (6, 9, 9, 14) would be ranked (4, 2, 2, 1).

Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation (calculations_tablecalculations_definebasic_runningtotal.htm#Rank).

Example

The following image shows the effect of the various ranking functions (RANK, RANK_DENSE, RANK_MODIFIED, RANK_PERCENTILE, and RANK_UNIQUE) on a set of values. The data set contains information on 14 students (StudentA through StudentN); the **Age** column shows the current age of each student (all students are between 17 and 20 years of age). The remaining columns show the effect of each rank function on the set of age values, always assuming the default order (ascending or descending) for the function.



Student	Age	RANKofAge	RANK_DENSEofAge	RANK_MODIFIEDofAge	RANK_PERCENTILEofAge	RANK_UNIQUEofAge
StudentA	19	4	2	7	79%	4
StudentB	18	8	3	12	50%	8
StudentC	19	4	2	7	79%	5
StudentD	18	8	3	12	50%	9
StudentE	17	13	4	14	14%	13
StudentF	18	8	3	12	50%	10
StudentG	19	4	2	7	79%	6
StudentH	20	1	1	3	100%	1
StudentI	19	4	2	7	79%	7
StudentJ	20	1	1	3	100%	2
StudentK	20	1	1	3	100%	3
StudentL	17	13	4	14	14%	14
StudentM	18	8	3	12	50%	11
StudentN	18	8	3	12	50%	12

RANK_DENSE(expression, ['asc' | 'desc'])

Returns the dense rank for the current row in the partition. Identical values are assigned an identical rank, but no gaps are inserted into the number sequence. Use the optional `'asc' | 'desc'` argument to specify ascending or descending order. The default is descending.

With this function, the set of values (6, 9, 9, 14) would be ranked (3, 2, 2, 1).

Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation (calculations_tablecalculations_definebasic_runningtotal.htm#Rank).

RANK_MODIFIED(expression, ['asc' | 'desc'])

Returns the modified competition rank for the current row in the partition. Identical values are assigned an identical rank. Use the optional `'asc' | 'desc'` argument to specify ascending or descending order. The default is descending.

With this function, the set of values (6, 9, 9, 14) would be ranked (4, 3, 3, 1).

Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation (calculations_tablecalculations_definebasic_runningtotal.htm#Rank).

RANK_PERCENTILE(expression, ['asc' | 'desc'])

Returns the percentile rank for the current row in the partition. Use the optional `'asc' | 'desc'` argument to specify ascending or descending order. The default is ascending.

With this function, the set of values (6, 9, 9, 14) would be ranked (0.25, 0.75, 0.75, 1.00).

Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation (calculations_tablecalculations_definebasic_runningtotal.htm#Rank).

RANK_UNIQUE(expression, ['asc' | 'desc'])

Returns the unique rank for the current row in the partition. Identical values are assigned different ranks. Use the optional `'asc' | 'desc'` argument to specify ascending or descending order. The default is descending.

With this function, the set of values (6, 9, 9, 14) would be ranked (4, 2, 3, 1).

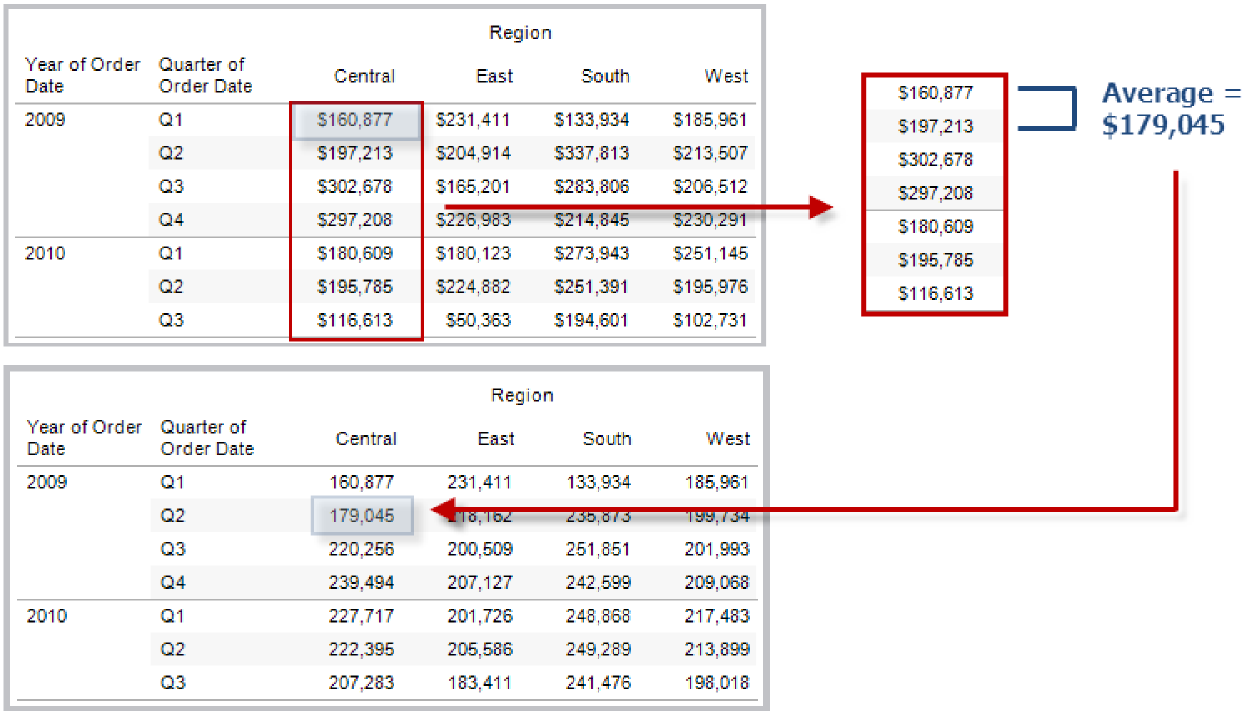
Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation (calculations_tablecalculations_definebasic_runningtotal.htm#Rank).

RUNNING_AVG(expression)

Returns the running average of the given expression, from the first row in the partition to the current row.

The view below shows quarterly sales. When `RUNNING_AVG(SUM([Sales])` is computed within the Date partition, the result is a running average of the sales values for each quarter.



Year of Order Date	Quarter of Order Date	Region Central	 East	 South	 West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

$160,877
$197,213
$302,678
$297,208
$180,609
$195,785
$116,613

Average = $179,045

Year of Order Date	Quarter of Order Date	Region Central	 East	 South	 West
2009	Q1	160,877	231,411	133,934	185,961
	Q2	179,045	218,162	235,873	199,734
	Q3	220,256	200,509	251,851	201,993
	Q4	239,494	207,127	242,599	209,068
2010	Q1	227,717	201,726	248,868	217,483
	Q2	222,395	205,586	249,289	213,899
	Q3	207,283	183,411	241,476	198,018

Example

`RUNNING_AVG(SUM([Profit]))` computes the running average of SUM(Profit).

RUNNING_COUNT(expression)

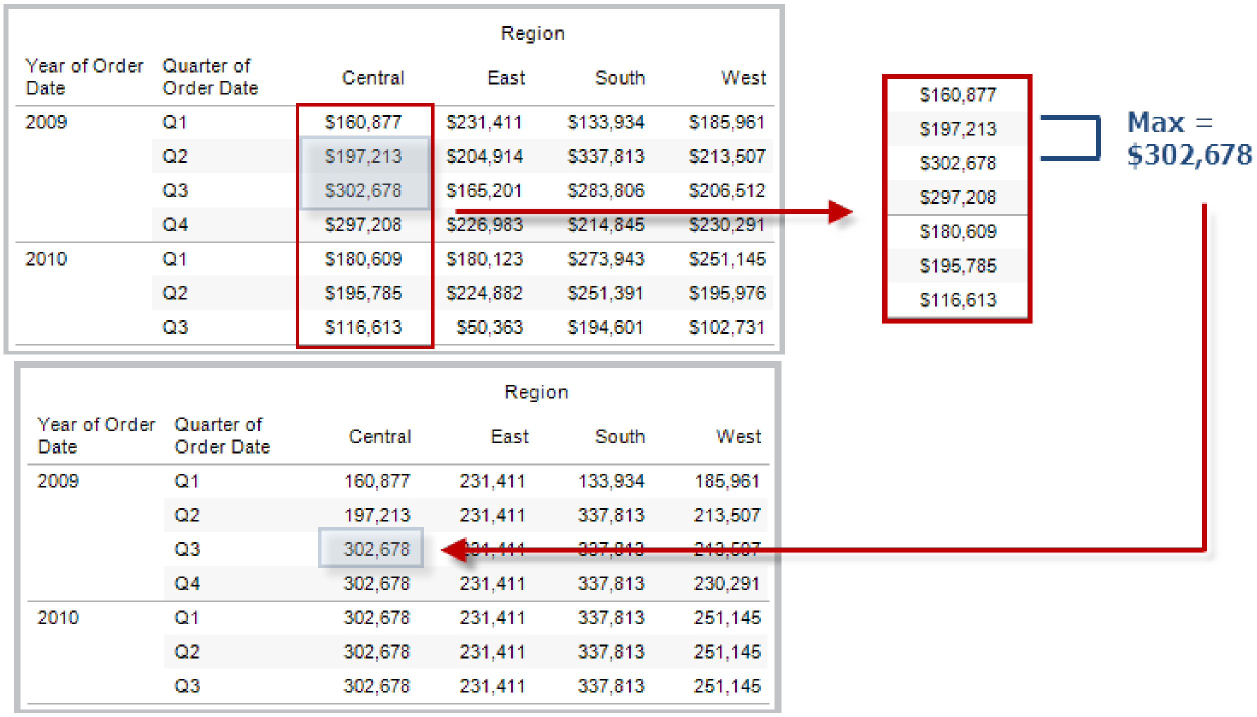
Returns the running count of the given expression, from the first row in the partition to the current row.

Example

`RUNNING_COUNT(SUM([Profit]))` computes the running count of SUM(Profit).

RUNNING_MAX(expression)

Returns the running maximum of the given expression, from the first row in the partition to the current row.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

$160,877
$197,213
$302,678
$297,208
$180,609
$195,785
$116,613
Max = $302,678

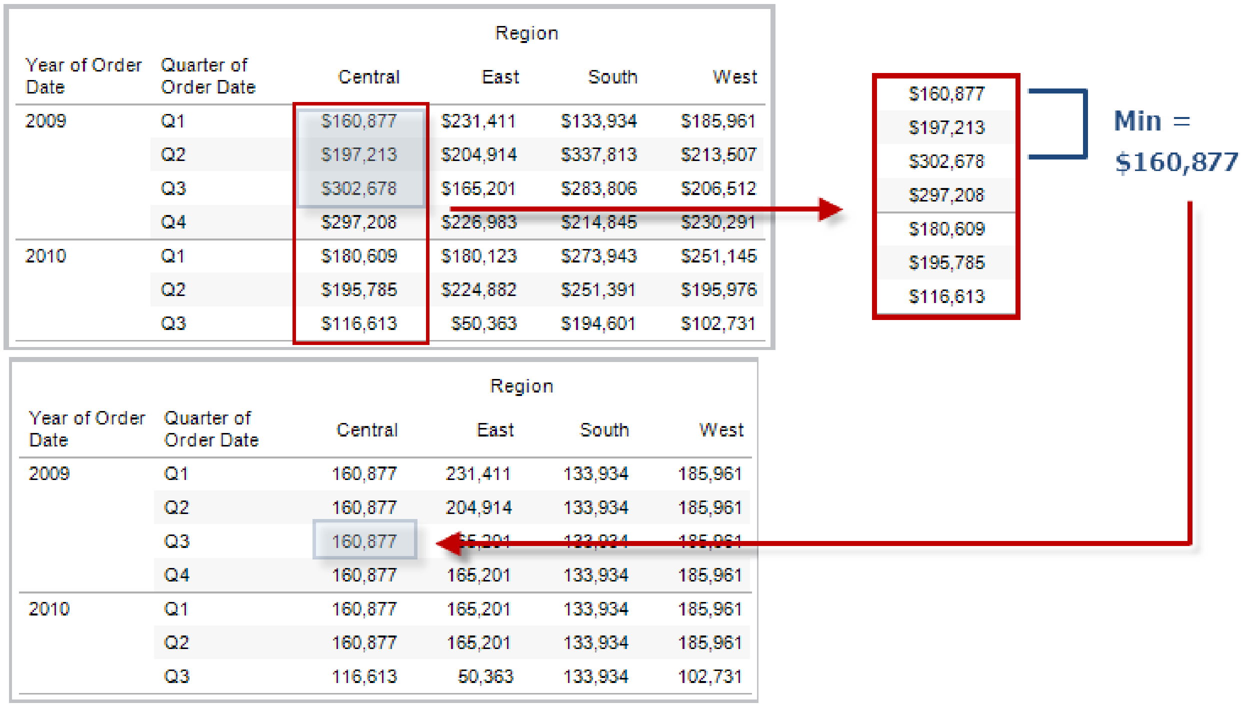
		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	160,877	231,411	133,934	185,961
	Q2	197,213	231,411	337,813	213,507
	Q3	302,678	231,411	337,813	213,507
	Q4	302,678	231,411	337,813	230,291
2010	Q1	302,678	231,411	337,813	251,145
	Q2	302,678	231,411	337,813	251,145
	Q3	302,678	231,411	337,813	251,145

Example

`RUNNING_MAX(SUM([Profit]))` computes the running maximum of SUM(Profit).

RUNNING_MIN(expression)

Returns the running minimum of the given expression, from the first row in the partition to the current row.



Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

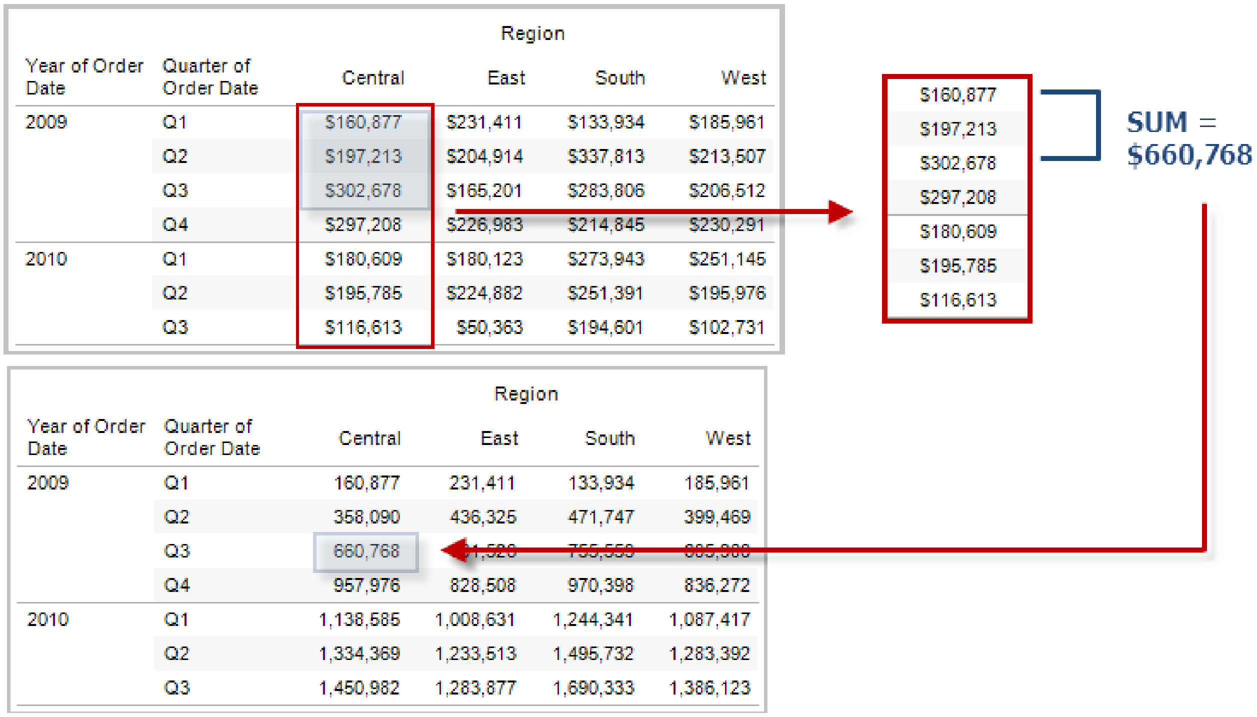
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	160,877	231,411	133,934	185,961
	Q2	160,877	204,914	133,934	185,961
	Q3	160,877	165,201	133,934	185,961
	Q4	160,877	165,201	133,934	185,961
2010	Q1	160,877	165,201	133,934	185,961
	Q2	160,877	165,201	133,934	185,961
	Q3	116,613	50,363	133,934	102,731

Example

`RUNNING_MIN(SUM([Profit]))` computes the running minimum of SUM(Profit).

RUNNING_SUM(expression)

Returns the running sum of the given expression, from the first row in the partition to the current row.



Year of Order Date	Quarter of Order Date	Central	East	South	West
		Region			
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

$160,877
$197,213
$302,678
$297,208
$180,609
$195,785
$116,613

SUM = $660,768

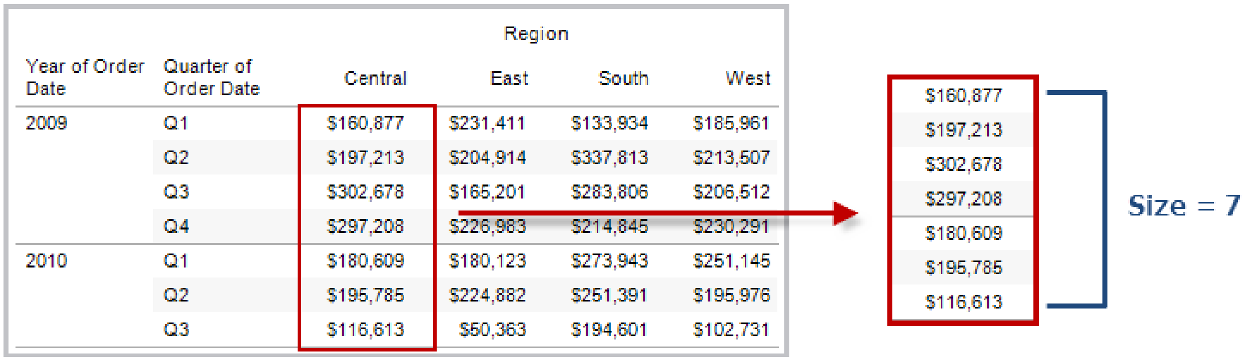
Year of Order Date	Quarter of Order Date	Central	East	South	West
		Region			
2009	Q1	160,877	231,411	133,934	185,961
	Q2	358,090	436,325	471,747	399,469
	Q3	660,768	601,526	755,553	605,981
	Q4	957,976	828,508	970,398	836,272
2010	Q1	1,138,585	1,008,631	1,244,341	1,087,417
	Q2	1,334,369	1,233,513	1,495,732	1,283,392
	Q3	1,450,982	1,283,877	1,690,333	1,386,123

Example

`RUNNING_SUM(SUM([Profit]))` computes the running sum of SUM(Profit)

SIZE()

Returns the number of rows in the partition. For example, the view below shows quarterly sales. Within the Date partition, there are seven rows so the Size() of the Date partition is 7.



Year of Order Date	Quarter of Order Date	Region Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

Example

`SIZE() = 5` when the current partition contains five rows.

SCRIPT_BOOL

Returns a Boolean result from the specified expression. The expression is passed directly to a running external service instance.

In R expressions, use .arg*n* (with a leading period) to reference parameters (.arg1, .arg2, etc.).

In Python expressions, use _arg*n* (with a leading underscore).

Examples

In this R example, .arg1 is equal to SUM([Profit]):

```
SCRIPT_BOOL("is.finite(.arg1)", SUM([Profit]))
```

The next example returns True for store IDs in Washington state, and False otherwise. This example could be the definition for a calculated field titled IsStoreInWA.

```
SCRIPT_BOOL('grepl(".*_WA", .arg1, perl=TRUE)',ATTR([Store ID]))
```

A command for Python would take this form:

```
SCRIPT_BOOL("return map(lambda x : x > 0, _arg1)", SUM([Profit]))
```

SCRIPT_INT

Returns an integer result from the specified expression. The expression is passed directly to a running external service instance.

In R expressions, use .arg*n* (with a leading period) to reference parameters (.arg1, .arg2, etc.)

In Python expressions, use _arg*n* (with a leading underscore).

Examples

In this R example, .arg1 is equal to SUM([Profit]):

```
SCRIPT_INT("is.finite(.arg1)", SUM([Profit]))
```

In the next example, k-means clustering is used to create three clusters:

```
SCRIPT_INT('result <- kmeans(data.frame(.arg1,.arg2,.arg3,.arg4),
3);result$cluster;', SUM([Petal length]), SUM([Petal width]),SUM([Sepal
length]),SUM([Sepal width]))
```

A command for Python would take this form:

```
SCRIPT_INT("return map(lambda x : int(x * 5), _arg1)", SUM([Profit]))
```

SCRIPT_REAL

Returns a real result from the specified expression. The expression is passed directly to a running external service instance. In

R expressions, use .arg*n* (with a leading period) to reference parameters (.arg1, .arg2, etc.)

In Python expressions, use _arg*n* (with a leading underscore).

Examples

In this R example, .arg1 is equal to SUM([Profit]):

```
SCRIPT_REAL("is.finite(.arg1)", SUM([Profit]))
```

The next example converts temperature values from Celsius to Fahrenheit.

```
SCRIPT_REAL('library(udunits2);ud.convert(.arg1, "celsius",
"degree_fahrenheit")',AVG([Temperature]))
```

A command for Python would take this form:

```
SCRIPT_REAL("return map(lambda x : x * 0.5, _arg1)", SUM([Profit]))
```

SCRIPT_STR

Returns a string result from the specified expression. The expression is passed directly to a running external service instance.

In R expressions, use .arg*n* (with a leading period) to reference parameters (.arg1, .arg2, etc.)

In Python expressions, use _arg*n* (with a leading underscore).

Examples

In this R example, .arg1 is equal to SUM([Profit]):

```
SCRIPT_STR("is.finite(.arg1)", SUM([Profit]))
```

The next example extracts a state abbreviation from a more complicated string (in the original form 13XSL_CA, A13_WA):

```
SCRIPT_STR('gsub(".*_", "", .arg1)',ATTR([Store ID]))
```

A command for Python would take this form:

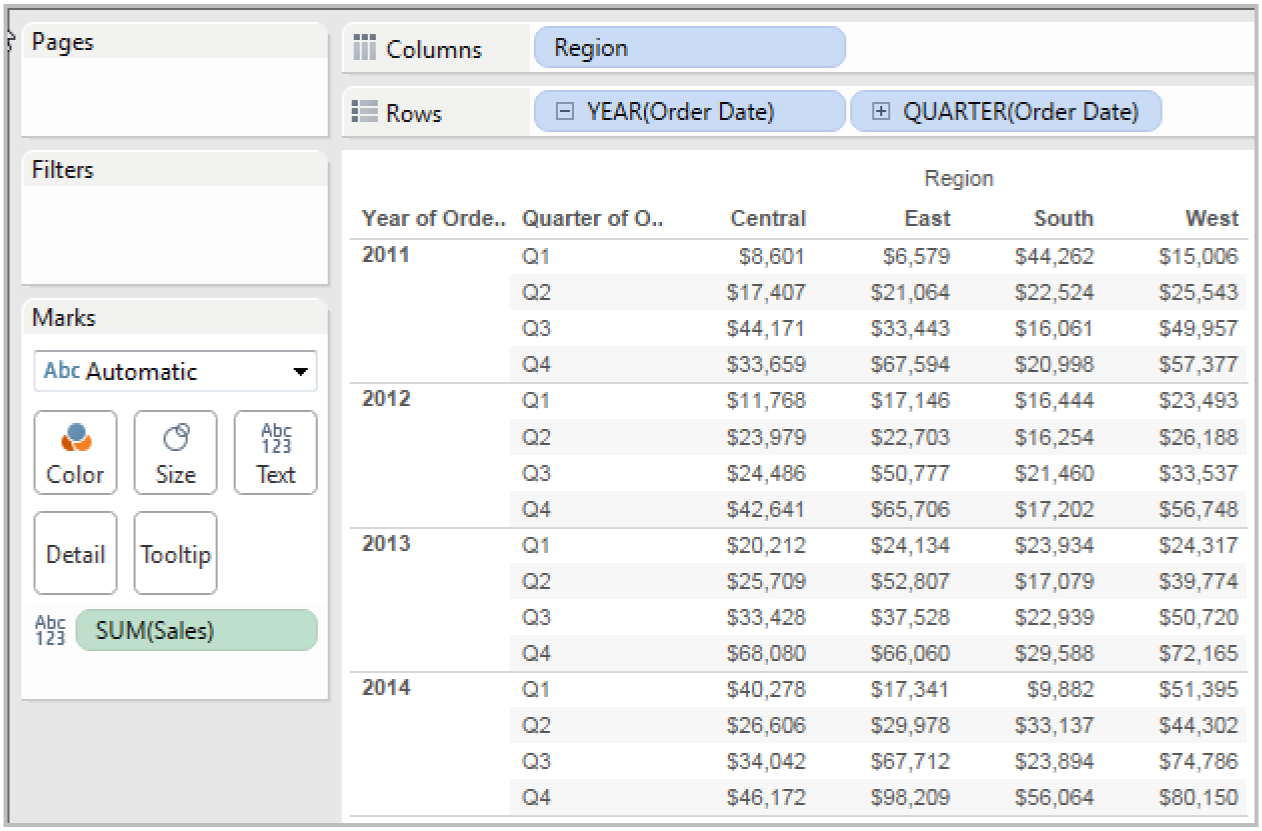
```
SCRIPT_STR("return map(lambda x : x[:2], _arg1)", ATTR([Region]))
```

TOTAL(expression)

Returns the total for the given expression in a table calculation partition.

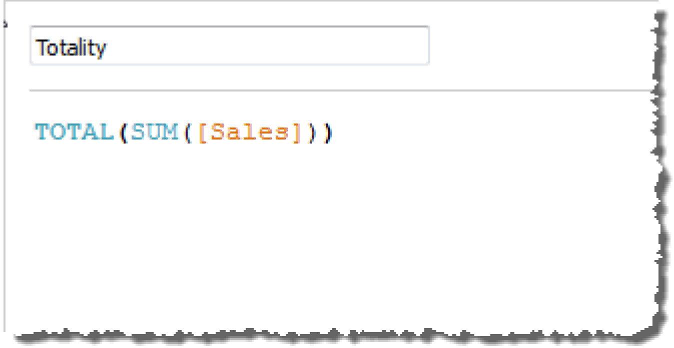
Example

Assume you are starting with this view:



		Region			
Year of Orde..	Quarter of O..	Central	East	South	West
2011	Q1	$8,601	$6,579	$44,262	$15,006
	Q2	$17,407	$21,064	$22,524	$25,543
	Q3	$44,171	$33,443	$16,061	$49,957
	Q4	$33,659	$67,594	$20,998	$57,377
2012	Q1	$11,768	$17,146	$16,444	$23,493
	Q2	$23,979	$22,703	$16,254	$26,188
	Q3	$24,486	$50,777	$21,460	$33,537
	Q4	$42,641	$65,706	$17,202	$56,748
2013	Q1	$20,212	$24,134	$23,934	$24,317
	Q2	$25,709	$52,807	$17,079	$39,774
	Q3	$33,428	$37,528	$22,939	$50,720
	Q4	$68,080	$66,060	$29,588	$72,165
2014	Q1	$40,278	$17,341	$9,882	$51,395
	Q2	$26,606	$29,978	$33,137	$44,302
	Q3	$34,042	$67,712	$23,894	$74,786
	Q4	$46,172	$98,209	$56,064	$80,150

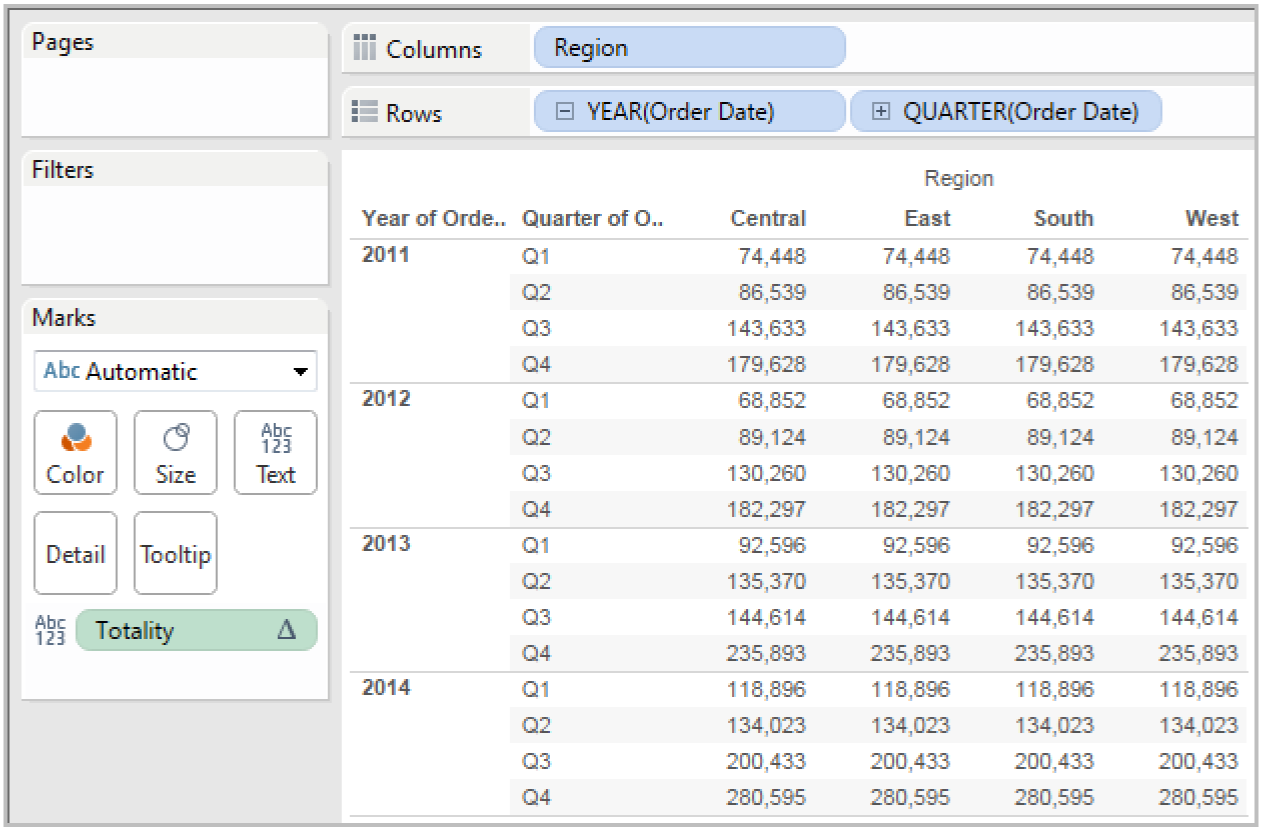
You open the calculation editor and create a new field which you name **Totality**:



Totality

```
TOTAL(SUM([Sales]))
```

You then drop **Totality** on Text, to replace **SUM(Sales)**. Your view changes such that it sums values based on the default **Compute Using** value:

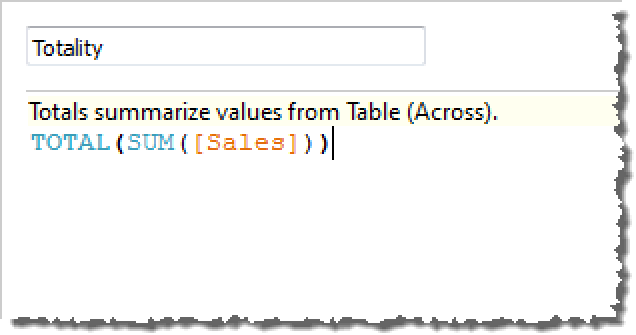


Columns: Region

Rows: YEAR(Order Date), QUARTER(Order Date)

		Region			
Year of Orde..	Quarter of O..	Central	East	South	West
2011	Q1	74,448	74,448	74,448	74,448
	Q2	86,539	86,539	86,539	86,539
	Q3	143,633	143,633	143,633	143,633
	Q4	179,628	179,628	179,628	179,628
2012	Q1	68,852	68,852	68,852	68,852
	Q2	89,124	89,124	89,124	89,124
	Q3	130,260	130,260	130,260	130,260
	Q4	182,297	182,297	182,297	182,297
2013	Q1	92,596	92,596	92,596	92,596
	Q2	135,370	135,370	135,370	135,370
	Q3	144,614	144,614	144,614	144,614
	Q4	235,893	235,893	235,893	235,893
2014	Q1	118,896	118,896	118,896	118,896
	Q2	134,023	134,023	134,023	134,023
	Q3	200,433	200,433	200,433	200,433
	Q4	280,595	280,595	280,595	280,595

This raises the question, What is the default **Compute Using** value? If you right-click (Control-click on a Mac) **Totality** in the Data pane and choose **Edit**, there is now an additional bit of information available:



Totality

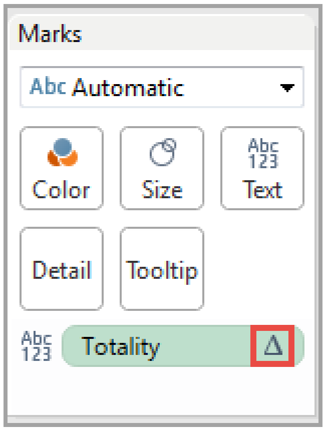
Totals summarize values from Table (Across).

```
TOTAL(SUM([Sales]))
```

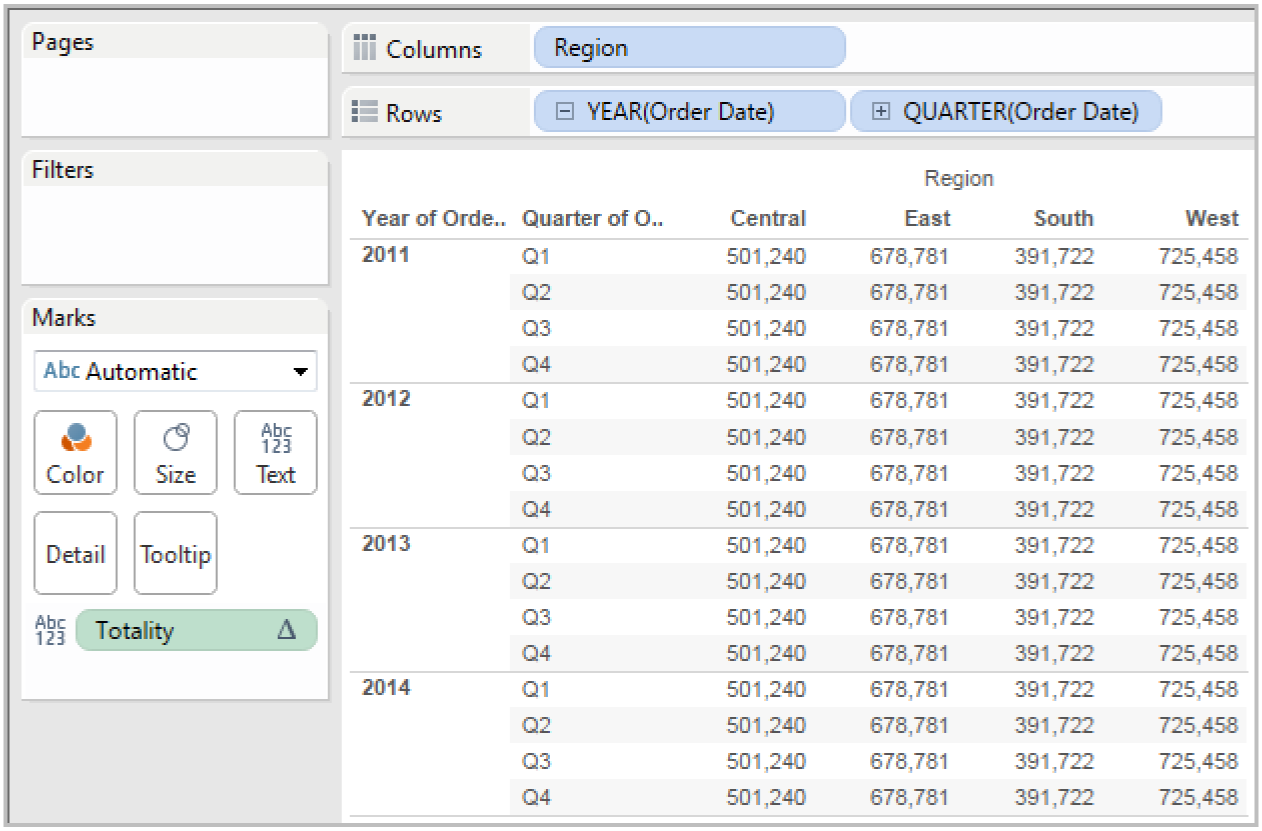
The default **Compute Using** value is **Table (Across)**. The result is that **Totality** is summing the values across each row of your table. Thus, the value that you see across each row is the sum of the values from the original version of the table.

The values in the 2011/Q1 row in the original table were $8601, $6579, $44262, and $15006. The values in the table after **Totality** replaces **SUM(Sales)** are all $74,448, which is the sum of the four original values.

Notice the triangle next to Totality after you drop it on Text:



This indicates that this field is using a table calculation. You can right-click the field and choose **Edit Table Calculation** to redirect your function to a different **Compute Using** value. For example, you could set it to **Table (Down)**. In that case, your table would look like this:

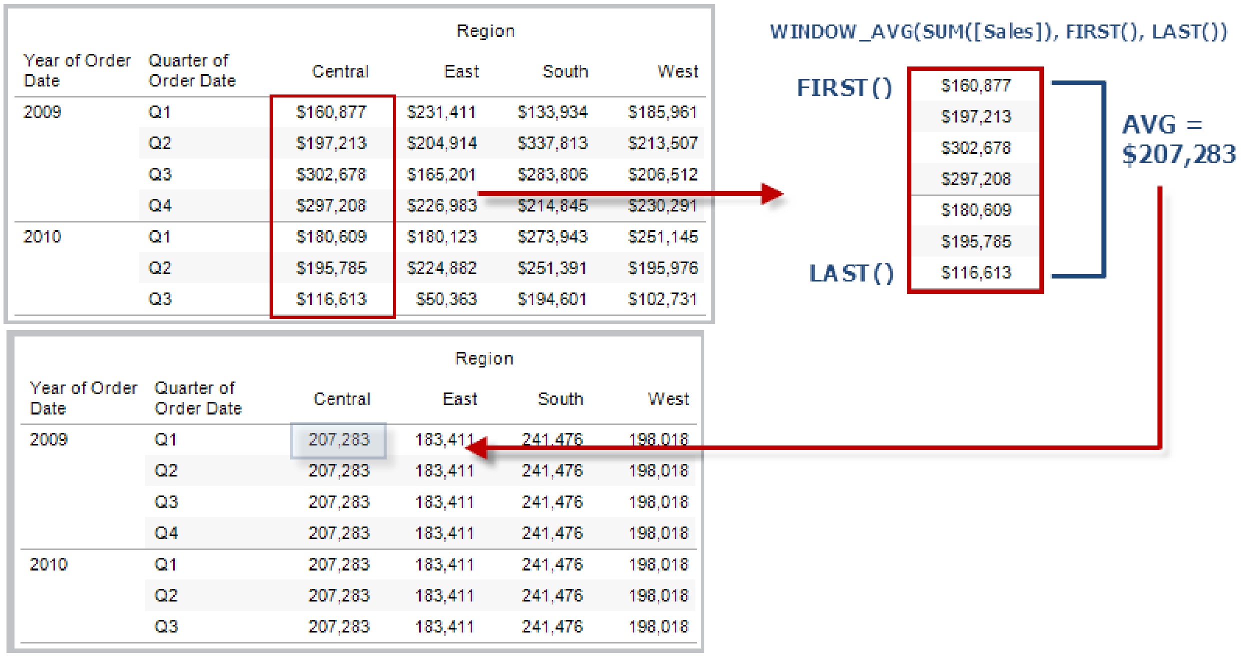


		Region			
Year of Orde..	Quarter of O..	Central	East	South	West
2011	Q1	501,240	678,781	391,722	725,458
	Q2	501,240	678,781	391,722	725,458
	Q3	501,240	678,781	391,722	725,458
	Q4	501,240	678,781	391,722	725,458
2012	Q1	501,240	678,781	391,722	725,458
	Q2	501,240	678,781	391,722	725,458
	Q3	501,240	678,781	391,722	725,458
	Q4	501,240	678,781	391,722	725,458
2013	Q1	501,240	678,781	391,722	725,458
	Q2	501,240	678,781	391,722	725,458
	Q3	501,240	678,781	391,722	725,458
	Q4	501,240	678,781	391,722	725,458
2014	Q1	501,240	678,781	391,722	725,458
	Q2	501,240	678,781	391,722	725,458
	Q3	501,240	678,781	391,722	725,458
	Q4	501,240	678,781	391,722	725,458

WINDOW_AVG(expression, [start, end])

Returns the average of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

For example, the view below shows quarterly sales. A window average within the Date partition returns the average sales across all dates.



Year of Order Date	Quarter of Order Date	Region Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

Year of Order Date	Quarter of Order Date	Region Central	East	South	West
2009	Q1	207,283	183,411	241,476	198,018
	Q2	207,283	183,411	241,476	198,018
	Q3	207,283	183,411	241,476	198,018
	Q4	207,283	183,411	241,476	198,018
2010	Q1	207,283	183,411	241,476	198,018
	Q2	207,283	183,411	241,476	198,018
	Q3	207,283	183,411	241,476	198,018

Example

`WINDOW_AVG(SUM([Profit]), FIRST()+1, 0)` computes the average of SUM(Profit) from the second row to the current row.

WINDOW_CORR(expression1, expression2, [start, end])

Returns the Pearson correlation coefficient of two expressions within the window. The window is defined as offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If start and end are omitted, the entire partition is used.

The Pearson correlation measures the linear relationship between two variables. Results range from -1 to +1 inclusive, where 1 denotes an exact positive linear relationship, as when a positive change in one variable implies a positive change of corresponding magnitude in the other, 0 denotes no linear relationship between the variance, and -1 is an exact negative relationship.

There is an equivalent aggregation fuction: CORR. See Tableau Functions (Alphabetical) (functions_all_alphabetical.htm).

Example

The following formula returns the Pearson correlation of **SUM(Profit)** and **SUM(Sales)** from the five previous rows to the current row.

```
WINDOW_CORR(SUM[Profit]), SUM([Sales]), -5, 0)
```

WINDOW_COUNT(expression, [start, end])

Returns the count of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

Example

`WINDOW_COUNT(SUM([Profit]), FIRST()+1, 0)` computes the count of SUM(Profit) from the second row to the current row

WINDOW_COVAR(expression1, expression2, [start, end])

Returns the *sample covariance* of two expressions within the window. The window is defined as offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end arguments are omitted, the window is the entire partition.

Sample covariance uses the number of non-null data points n - 1 to normalize the covariance calculation, rather than n, which is used by the population covariance (with the WINDOW_COVARP function). Sample covariance is the appropriate choice when the data is a random sample that is being used to estimate the covariance for a larger population.

There is an equivalent aggregation fuction: COVAR. See Tableau Functions (Alphabetical) (functions_all_alphabetical.htm).

Example

The following formula returns the sample covariance of **SUM(Profit)** and **SUM(Sales)** from the two previous rows to the current row.

```
WINDOW_COVAR(SUM([Profit]), SUM([Sales]), -2, 0)
```

WINDOW_COVARP(expression1, expression2, [start, end])

Returns the *population covariance* of two expressions within the window. The window is defined as offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If start and end are omitted, the entire partition is used.

Population covariance is sample covariance multiplied by (n-1)/n, where n is the total number of non-null data points. Population covariance is the appropriate choice when there is data available for all items of interest as opposed to when there is only a random subset of items, in which case sample covariance (with the WINDOW_COVAR function) is appropriate.

There is an equivalent aggregation fuction: COVARP. Tableau Functions (Alphabetical) (functions_all_alphabetical.htm).

Example

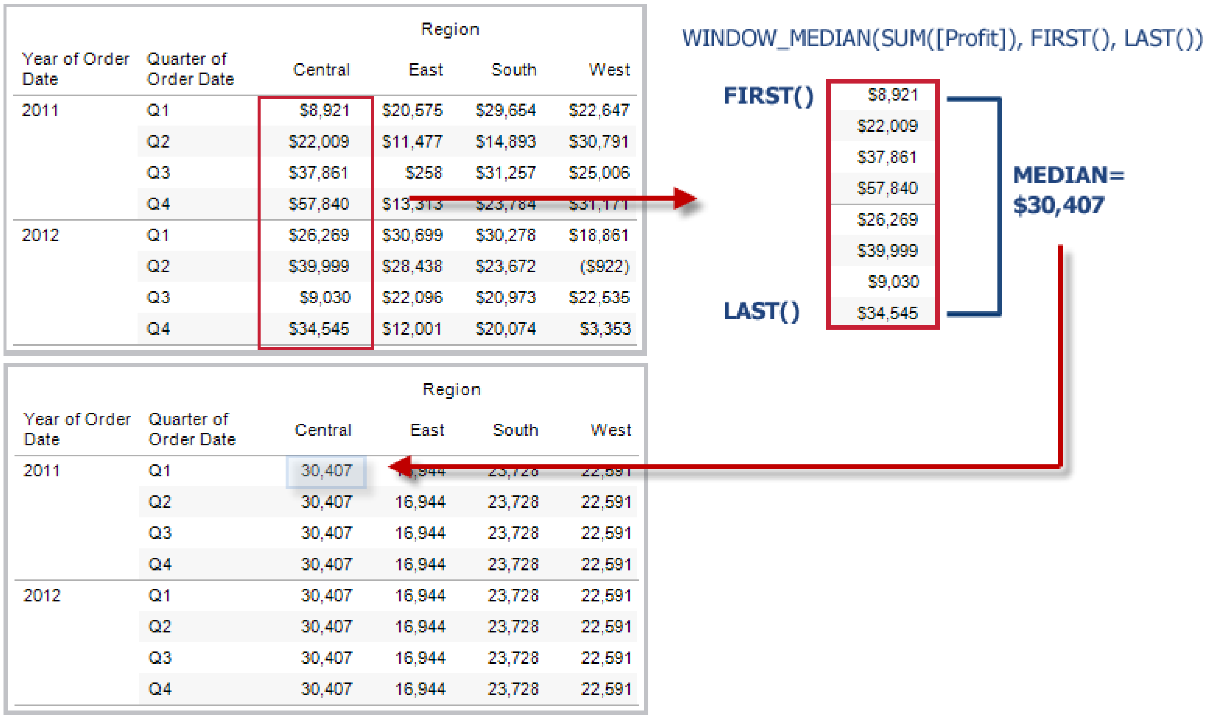
The following formula returns the population covariance of **SUM(Profit)** and **SUM(Sales)** from the two previous rows to the current row.

```
WINDOW_COVARP(SUM([Profit]), SUM([Sales]), -2, 0)
```

WINDOW_MEDIAN(expression, [start, end])

Returns the median of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

For example, the view below shows quarterly profit. A window median within the Date partition returns the median profit across all dates.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2011	Q1	$8,921	$20,575	$29,654	$22,647
	Q2	$22,009	$11,477	$14,893	$30,791
	Q3	$37,861	$258	$31,257	$25,006
	Q4	$57,840	$13,313	$23,784	$31,171
2012	Q1	$26,269	$30,699	$30,278	$18,861
	Q2	$39,999	$28,438	$23,672	($922)
	Q3	$9,030	$22,096	$20,973	$22,535
	Q4	$34,545	$12,001	$20,074	$3,353

		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2011	Q1	30,407	16,944	23,728	22,591
	Q2	30,407	16,944	23,728	22,591
	Q3	30,407	16,944	23,728	22,591
	Q4	30,407	16,944	23,728	22,591
2012	Q1	30,407	16,944	23,728	22,591
	Q2	30,407	16,944	23,728	22,591
	Q3	30,407	16,944	23,728	22,591
	Q4	30,407	16,944	23,728	22,591

WINDOW_MEDIAN(SUM([Profit]), FIRST(), LAST())

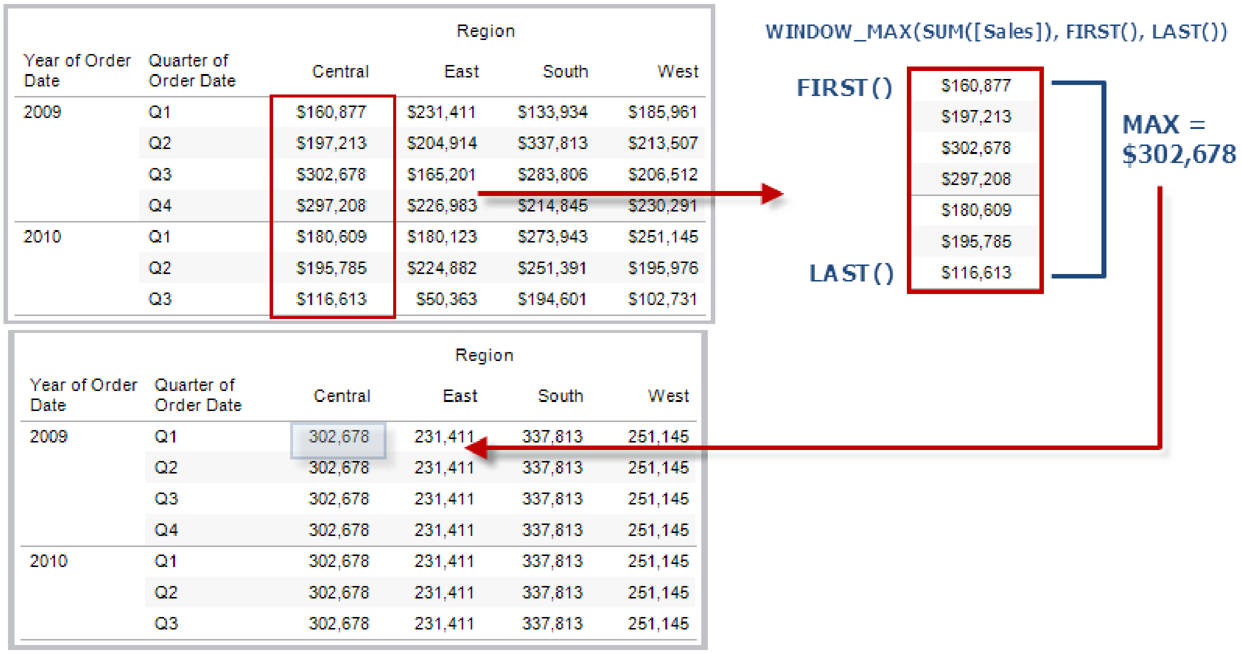
Example

`WINDOW_MEDIAN(SUM([Profit]), FIRST()+1, 0)` computes the median of SUM(Profit) from the second row to the current row.

WINDOW_MAX(expression, [start, end])

Returns the maximum of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

For example, the view below shows quarterly sales. A window maximum within the Date partition returns the maximum sales across all dates.



Year of Order Date	Quarter of Order Date	Region Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

Year of Order Date	Quarter of Order Date	Region Central	East	South	West
2009	Q1	302,678	231,411	337,813	251,145
	Q2	302,678	231,411	337,813	251,145
	Q3	302,678	231,411	337,813	251,145
	Q4	302,678	231,411	337,813	251,145
2010	Q1	302,678	231,411	337,813	251,145
	Q2	302,678	231,411	337,813	251,145
	Q3	302,678	231,411	337,813	251,145

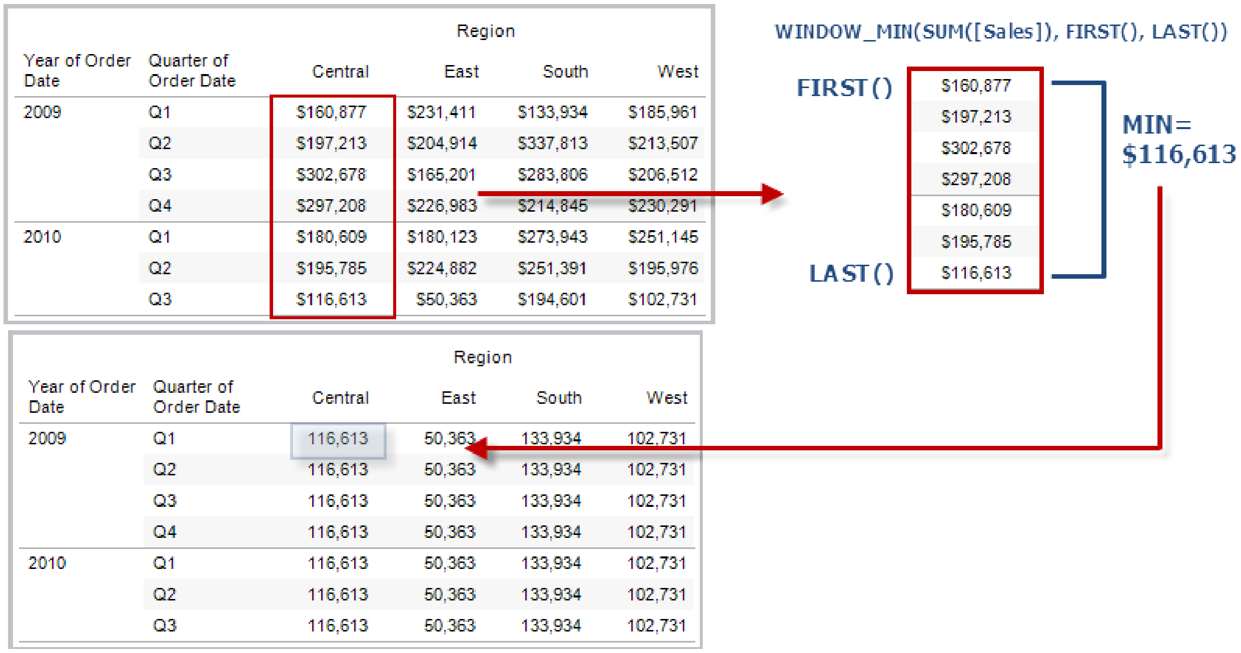
Example

`WINDOW_MAX(SUM([Profit]), FIRST()+1, 0)` computes the maximum of SUM(Profit) from the second row to the current row.

WINDOW_MIN(expression, [start, end])

Returns the minimum of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

For example, the view below shows quarterly sales. A window minimum within the Date partition returns the minimum sales across all dates.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	116,613	50,363	133,934	102,731
	Q2	116,613	50,363	133,934	102,731
	Q3	116,613	50,363	133,934	102,731
	Q4	116,613	50,363	133,934	102,731
2010	Q1	116,613	50,363	133,934	102,731
	Q2	116,613	50,363	133,934	102,731
	Q3	116,613	50,363	133,934	102,731

Example

`WINDOW_MIN(SUM([Profit]), FIRST()+1, 0)` computes the minimum of SUM(Profit) from the second row to the current row.

WINDOW_PERCENTILE(expression, number, [start, end])

Returns the value corresponding to the specified percentile within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

Example

`WINDOW_PERCENTILE(SUM([Profit]), 0.75, -2, 0)` returns the 75th percentile for SUM(Profit) from the two previous rows to the current row.

WINDOW_STDEV(expression, [start, end])

Returns the sample standard deviation of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

Example

`WINDOW_STDEV(SUM([Profit]), FIRST()+1, 0)` computes the standard deviation of SUM(Profit) from the second row to the current row.

WINDOW_STDEVP(expression, [start, end])

Returns the biased standard deviation of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

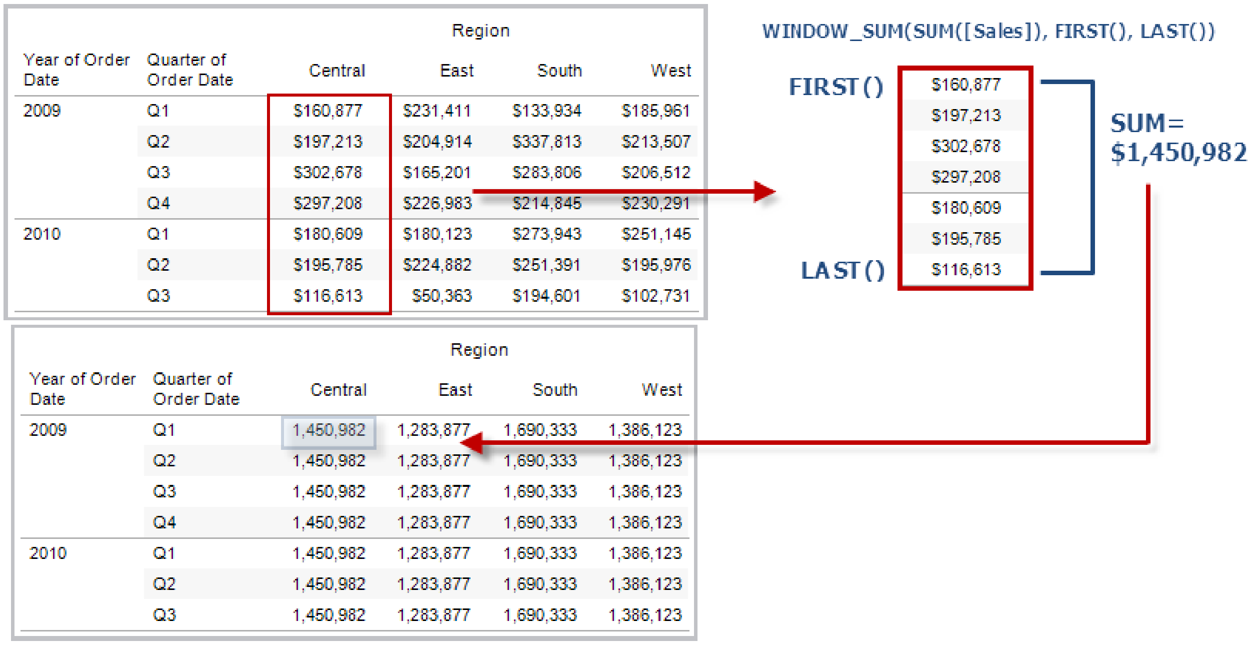
Example

`WINDOW_STDEVP(SUM([Profit]), FIRST()+1, 0)` computes the standard deviation of SUM(Profit) from the second row to the current row.

WINDOW_SUM(expression, [start, end])

Returns the sum of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

For example, the view below shows quarterly sales. A window sum computed within the Date partition returns the summation of sales across all quarters.



		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	$160,877	$231,411	$133,934	$185,961
	Q2	$197,213	$204,914	$337,813	$213,507
	Q3	$302,678	$165,201	$283,806	$206,512
	Q4	$297,208	$226,983	$214,845	$230,291
2010	Q1	$180,609	$180,123	$273,943	$251,145
	Q2	$195,785	$224,882	$251,391	$195,976
	Q3	$116,613	$50,363	$194,601	$102,731

		Region			
Year of Order Date	Quarter of Order Date	Central	East	South	West
2009	Q1	1,450,982	1,283,877	1,690,333	1,386,123
	Q2	1,450,982	1,283,877	1,690,333	1,386,123
	Q3	1,450,982	1,283,877	1,690,333	1,386,123
	Q4	1,450,982	1,283,877	1,690,333	1,386,123
2010	Q1	1,450,982	1,283,877	1,690,333	1,386,123
	Q2	1,450,982	1,283,877	1,690,333	1,386,123
	Q3	1,450,982	1,283,877	1,690,333	1,386,123

Example

`WINDOW_SUM(SUM([Profit]), FIRST()+1, 0)` computes the sum of SUM(Profit) from the second row to the current row.

WINDOW_VAR(expression, [start, end])

Returns the sample variance of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

Example

`WINDOW_VAR((SUM([Profit])), FIRST()+1, 0)` computes the variance of SUM(Profit) from the second row to the current row.

WINDOW_VARP(expression, [start, end])

Returns the biased variance of the expression within the window. The window is defined by means of offsets from the current row. Use FIRST()+n and LAST()-n for offsets from the first or last row in the partition. If the start and end are omitted, the entire partition is used.

Example

`WINDOW_VARP(SUM([Profit]), FIRST()+1, 0)` computes the variance of SUM(Profit) from the second row to the current row.

Create a table calculation using the calculation editor

Follow along with the steps below to learn how to create a table calculation using the calculation editor.

Note: There are several ways to create table calculations in Tableau. This example demonstrates only one of those ways. For more information, see Transform Values with Table Calculations (calculations_tablecalculations.htm).

Step 1: Create the visualization

1. In Tableau Desktop, connect to the **Sample-Superstore** saved data source, which comes with Tableau.

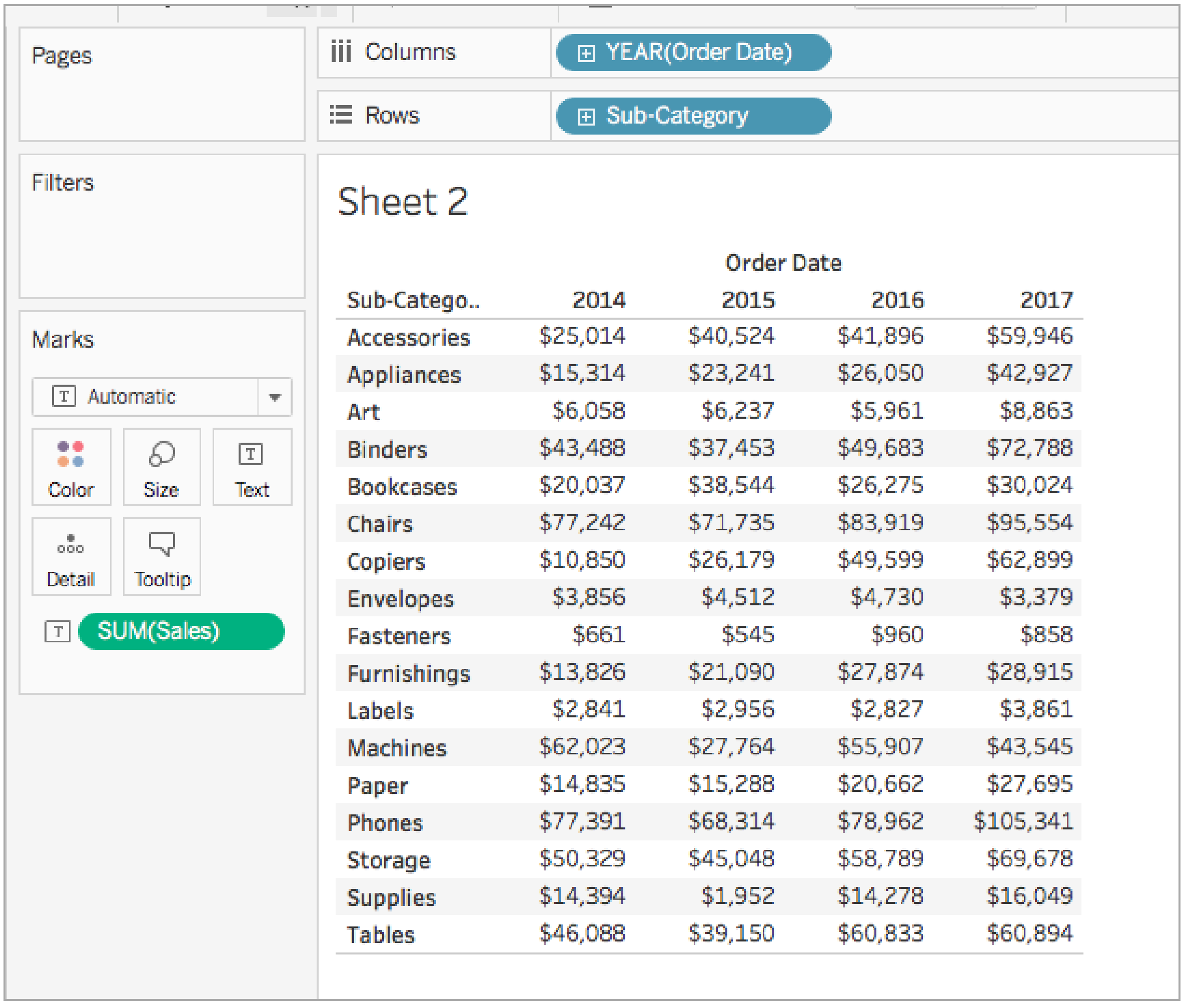
2. Navigate to a worksheet.

3. From the **Data** pane, under Dimensions, drag **Order Date** to the **Columns** shelf.

4. From the **Data** pane, under Dimensions, drag **Sub-Category** to the **Rows** shelf.

5. From the **Data** pane, under Measures, drag **Sales** to **Text** on the Marks card.

   Your visualization updates to a text table.



Sheet 2

	Order Date			
Sub-Catego..	2014	2015	2016	2017
Accessories	$25,014	$40,524	$41,896	$59,946
Appliances	$15,314	$23,241	$26,050	$42,927
Art	$6,058	$6,237	$5,961	$8,863
Binders	$43,488	$37,453	$49,683	$72,788
Bookcases	$20,037	$38,544	$26,275	$30,024
Chairs	$77,242	$71,735	$83,919	$95,554
Copiers	$10,850	$26,179	$49,599	$62,899
Envelopes	$3,856	$4,512	$4,730	$3,379
Fasteners	$661	$545	$960	$858
Furnishings	$13,826	$21,090	$27,874	$28,915
Labels	$2,841	$2,956	$2,827	$3,861
Machines	$62,023	$27,764	$55,907	$43,545
Paper	$14,835	$15,288	$20,662	$27,695
Phones	$77,391	$68,314	$78,962	$105,341
Storage	$50,329	$45,048	$58,789	$69,678
Supplies	$14,394	$1,952	$14,278	$16,049
Tables	$46,088	$39,150	$60,833	$60,894

Step 2: Create the table calculation

1. Select **Analysis** > **Create Calculated Field**.

2. In the calculation editor that opens, do the following:

   - Name the calculated field, Running Sum of Profit.

   - Enter the following formula:

```
RUNNING_SUM(SUM([Profit]))
```

This formula calculates the running sum of profit sales. It is computed across the entire table.

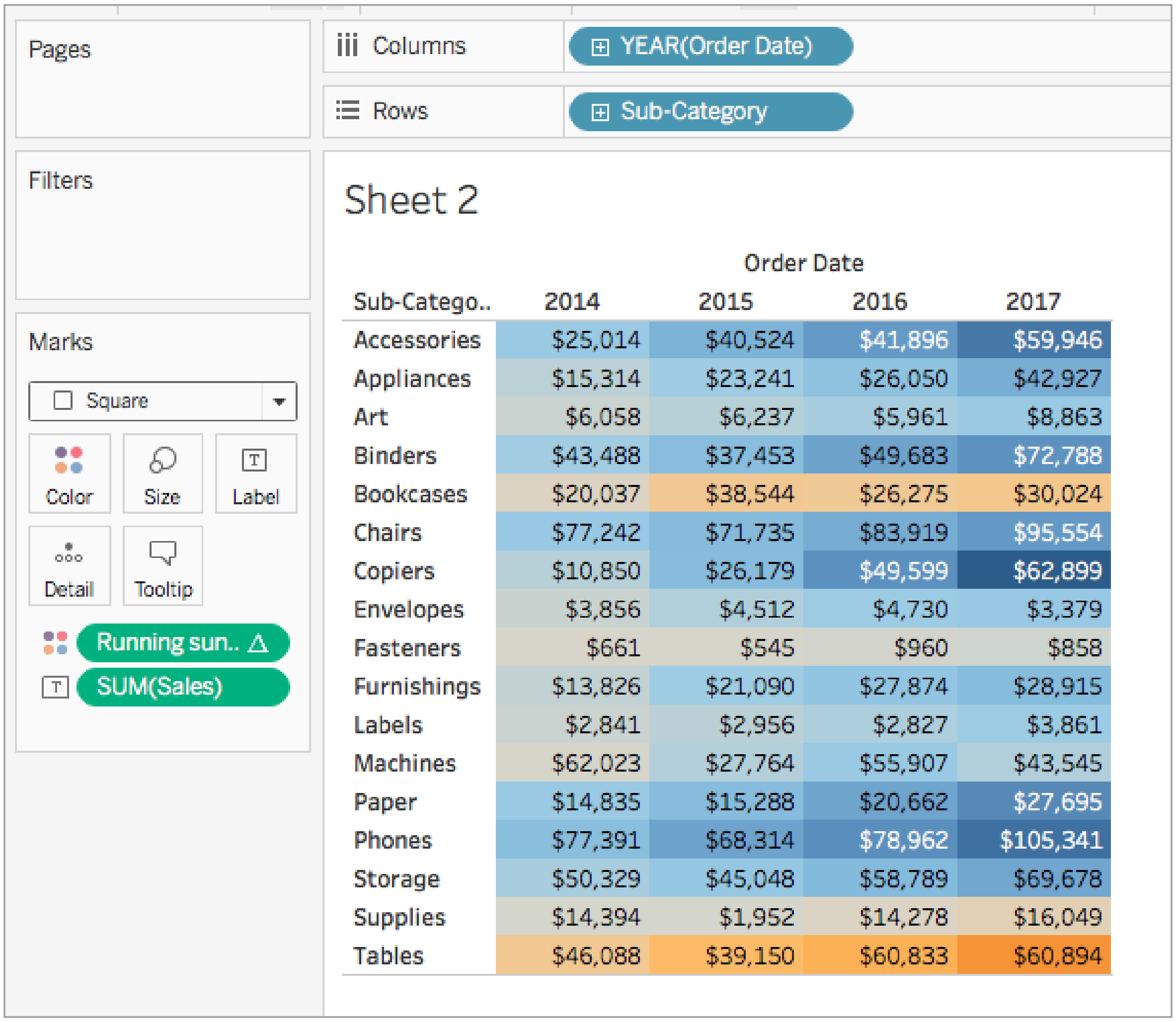
- When finished, click **OK**.

  The new table calculation field appears under Measures in the Data pane. Just like your other fields, you can use it in one or more visualizations.

Step 3: Use the table calculation in the visualization

1. From the Data pane, under Measures, drag **Running Sum of Profit** to **Color** on the Marks card.
2. On the Marks card, click the Mark Type drop-down and select **Square**.

   The visualization updates to a highlight table:



Columns: YEAR(Order Date)

Rows: Sub-Category

Marks: Square

Running sun.. △

SUM(Sales)

Sheet 2

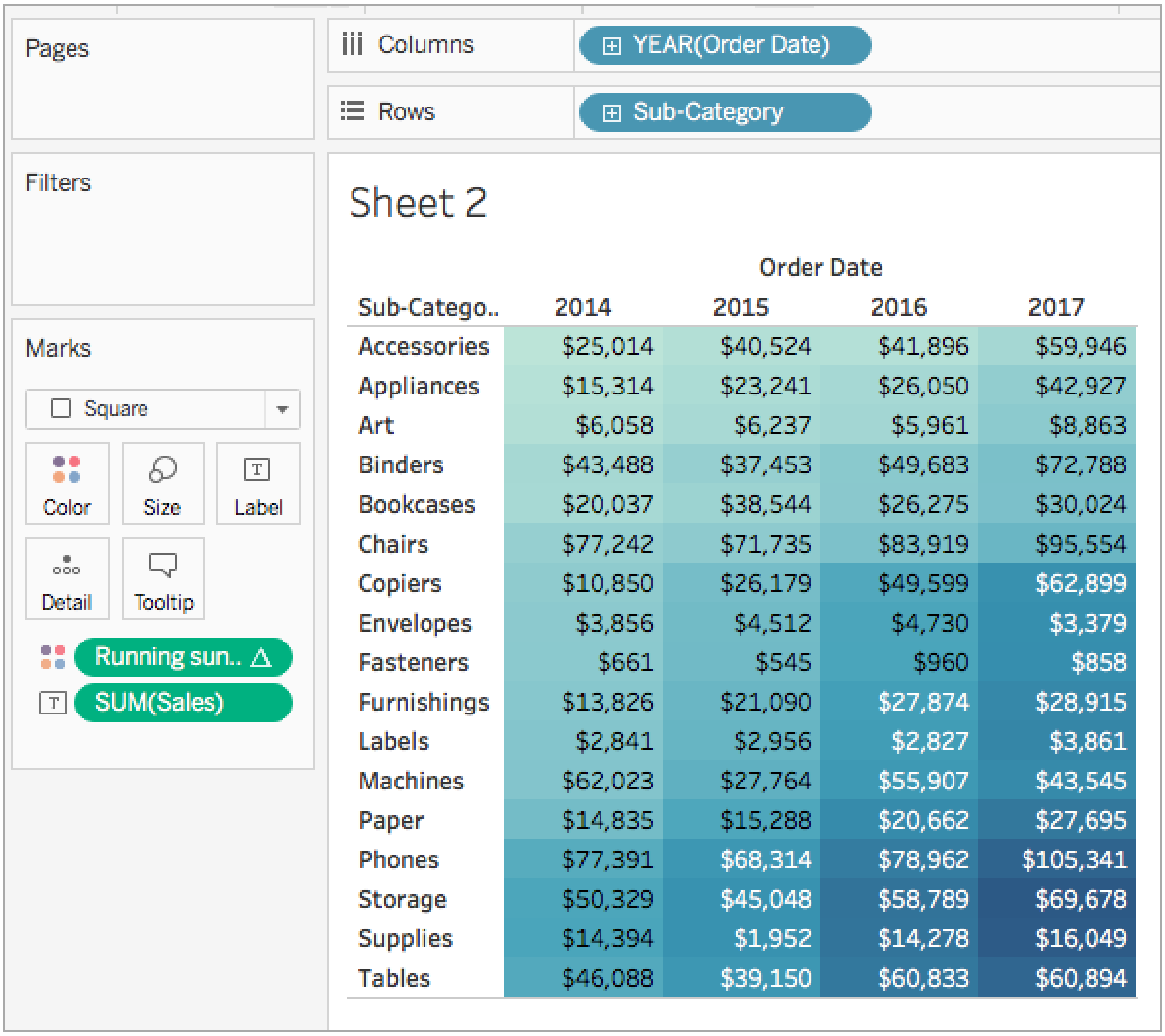
	Order Date			
Sub-Catego..	2014	2015	2016	2017
Accessories	$25,014	$40,524	$41,896	$59,946
Appliances	$15,314	$23,241	$26,050	$42,927
Art	$6,058	$6,237	$5,961	$8,863
Binders	$43,488	$37,453	$49,683	$72,788
Bookcases	$20,037	$38,544	$26,275	$30,024
Chairs	$77,242	$71,735	$83,919	$95,554
Copiers	$10,850	$26,179	$49,599	$62,899
Envelopes	$3,856	$4,512	$4,730	$3,379
Fasteners	$661	$545	$960	$858
Furnishings	$13,826	$21,090	$27,874	$28,915
Labels	$2,841	$2,956	$2,827	$3,861
Machines	$62,023	$27,764	$55,907	$43,545
Paper	$14,835	$15,288	$20,662	$27,695
Phones	$77,391	$68,314	$78,962	$105,341
Storage	$50,329	$45,048	$58,789	$69,678
Supplies	$14,394	$1,952	$14,278	$16,049
Tables	$46,088	$39,150	$60,833	$60,894

Step 4: Edit the table calculation

1. On the Marks card, right-click **Running Sum of Profit** and select **Edit Table Calculation**.

2. In the Table Calculation dialog box that opens, under Compute Using, select **Table (down)**.

   The visualization updates to the following:



Columns: YEAR(Order Date)

Rows: Sub-Category

Sheet 2

	Order Date			
Sub-Catego..	2014	2015	2016	2017
Accessories	$25,014	$40,524	$41,896	$59,946
Appliances	$15,314	$23,241	$26,050	$42,927
Art	$6,058	$6,237	$5,961	$8,863
Binders	$43,488	$37,453	$49,683	$72,788
Bookcases	$20,037	$38,544	$26,275	$30,024
Chairs	$77,242	$71,735	$83,919	$95,554
Copiers	$10,850	$26,179	$49,599	$62,899
Envelopes	$3,856	$4,512	$4,730	$3,379
Fasteners	$661	$545	$960	$858
Furnishings	$13,826	$21,090	$27,874	$28,915
Labels	$2,841	$2,956	$2,827	$3,861
Machines	$62,023	$27,764	$55,907	$43,545
Paper	$14,835	$15,288	$20,662	$27,695
Phones	$77,391	$68,314	$78,962	$105,341
Storage	$50,329	$45,048	$58,789	$69,678
Supplies	$14,394	$1,952	$14,278	$16,049
Tables	$46,088	$39,150	$60,833	$60,894

See Also

Create a table calculation (calculations_tablecalculations.htm#Create)

Table Calculation Types (calculations_tablecalculations_definebasic_runningtotal.htm)

Customize Table Calculations (calculations_tablecalculations_custom.htm)

Quick Table Calculations (calculations_tablecalculations_quick.htm)

Functions in Tableau (functions.htm)

Tableau Functions (by Category) (functions_all_categories.htm)

Tableau Functions (Alphabetical) (functions_all_alphabetical.htm)