

Tableau Functions (Alphabetical)

Applies to: Tableau Cloud, Tableau Desktop, Tableau Server

The Tableau functions in this reference are organized alphabetically. Click a letter to see functions that start with it. If no functions start with that letter, the functions that start with the next letter in the alphabet are shown. You can also press Ctrl+F (Command-F on a Mac) to open a search box that you can use to search the page for a specific function.

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#)
[Z](#)

ABS(number)

Returns the absolute value of the given number.

Examples

`ABS(-7) = 7`

`ABS([Budget Variance])`

The second example returns the absolute value for all the numbers contained in the `Budget Variance` field.

ACOS(number)

Returns the arc cosine of the given number. The result is in radians.

Example

```
ACOS(-1) = 3.14159265358979
```

AREA(geometry, 'units')

Returns the total surface area of a spatial polygon.

Supported unit names: meters ("meters," "metres" "m"), kilometers ("kilometers," "kilometres," "km"), miles ("miles" or "mi"), feet ("feet," "ft").

Example

```
AREA([Geometry], 'km')
```

ASCII(string)

Return the ASCII code for the first character of `string`.

Example

```
ASCII('A') = 65
```

ASIN(number)

Returns the arc sine of a given number. The result is in radians.

Example

ASIN(1) = 1.5707963267949

ATAN(number)

Returns the arc tangent of a given number. The result is in radians.

Example

ATAN(180) = 1.5652408283942

ATAN2(y number, x number)

Returns the arc tangent of two given numbers (x and y). The result is in radians.

Example

ATAN2(2, 1) = 1.10714871779409

ATTR(expression)

Returns the value of the expression if it has a single value for all rows. Otherwise returns an asterisk. Null values are ignored.

AVG(expression)

Returns the average of all the values in the expression. AVG can be used with

numeric fields only. Null values are ignored.

BUFFER(geometry, number, 'units')

Returns distance measurement between two points in a specified unit.

Supported unit names: meters ("meters," "metres" "m"), kilometers ("kilometers," "kilometres," "km"), miles ("miles" or "mi"), feet ("feet," "ft").

This function can only be created with a live connection and will continue to work when a data source is converted to an extract.

Example

```
BUFFER(MAKEPOINT(47.59, -122.32), 5, 'km')
```

CASE

CASE <expression> WHEN <value1> THEN <return1> WHEN <value2> THEN <return2> ... ELSE <default return> END

Use the CASE function to perform logical tests and return appropriate values. CASE is often easier to use than IIF or IF THEN ELSE. The CASE function evaluates `expression`, compares it to a sequence of values, `value1`, `value2`, etc., and returns a result. When a value that matches `expression` is encountered, CASE returns the corresponding return value. If no match is found, the default return expression is used. If there is no default return and no values match, then Null is returned.

Examples

```
CASE [Region] WHEN 'West' THEN 1 WHEN 'East' THEN 2 ELSE 3 END
```

```
CASE LEFT(DATENAME('weekday',[Order Date]),3) WHEN 'Sun' THEN 0 WHEN 'Mon'  
THEN 1 WHEN 'Tue' THEN 2 WHEN 'Wed' THEN 3 WHEN 'Thu' THEN 4 WHEN 'Fri'  
THEN 5 WHEN 'Sat' THEN 6 END
```

CEILING(number)

Rounds a number to the nearest integer of equal or greater value.

Example

```
CEILING(3.1415) = 4
```

Availability by data source

- Microsoft Excel
- Text File
- Statistical File
- Tableau Server
- Amazon EMR Hadoop Hive
- Cloudera Hadoop
- DataStax Enterprise
- Google Analytics
- Google BigQuery
- Hortonworks Hadoop HIVE
- MapR Hadoop Hive

- Salesforce
 - Spark SQL
-

CHAR(number)

Returns the character encoded by the ASCII code `number`.

Example

```
CHAR(65) = 'A'
```

COLLECT (spatial)

An aggregate calculation that combines the values in the argument field. Null values are ignored.

Note: The COLLECT function can only be used with spatial fields.

Example

```
COLLECT ([Geometry])
```

CONTAINS(string, substring)

Returns true if the given string contains the specified substring.

Example

```
CONTAINS("Calculation", "alcu") = true
```

CORR(expression1, expression2)

Returns the Pearson correlation coefficient of two expressions.

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.

Availability by data source

- Tableau data extracts (you can create an extract from any data source)
- Cloudera Hive
- EXASOL
- Firebird (version 3.0 and later)
- Google BigQuery
- Hortonworks Hadoop Hive
- Oracle
- PostgreSQL
- Presto
- SybaseIQ
- Teradata
- Vertica

For other data sources, consider either extracting the data or using `WINDOW_CORR`. See [Table Calculation Functions \(functions_functions_tablecalculation.htm\)](#).

Example

You can use CORR to visualize correlation in a disaggregated scatter plot. The way to do this is to use a table-scoped level of detail expression. For example:

```
{CORR(Sales, Profit)}
```

With a level of detail expression, the correlation is run over all rows. If you used a formula like `CORR(Sales, Profit)` (without the surrounding brackets to make it a level of detail expression), the view would show the correlation of

each individual point in the scatter plot with each other point, which is undefined.

See [Table-Spaced \(calculations_calculatedfields_lod.htm#Table\)](#)

COS(number)

Returns the cosine of an angle. Specify the angle in radians.

Example

```
COS(PI() / 4) = 0.707106781186548
```

COT(number)

Returns the cotangent of an angle. Specify the angle in radians.

Example

```
COT(PI() / 4) = 1
```

COUNT(expression)

Returns the number of items in a group. Null values are not counted.

COUNTD(expression)

Returns the number of distinct items in a group. Null values are not counted.

This function is not available in the following cases: workbooks created before Tableau Desktop 8.2 that use Microsoft Excel or text file data sources,

workbooks that use the legacy connection, and workbooks that use Microsoft Access data sources. Extract your data into an extract file to use this function. See [Extract Your Data \(extracting_data.htm\)](#).

COVAR(expression 1, expression2)

Returns the *sample covariance* of two expressions.

Covariance quantifies how two variables change together. A positive covariance indicates that the variables tend to move in the same direction, as when larger values of one variable tend to correspond to larger values of the other variable, on average. 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 (available with the 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.

Availability by data source

- Tableau data extracts (you can create an extract from any data source)
- Cloudera Hive
- EXASOL
- Firebird (version 3.0 and later)
- Google BigQuery
- Hortonworks Hadoop Hive
- IBM PDA (Netezza)
- Oracle
- PostgreSQL

- Presto
- SybaseIQ
- Teradata
- Vertica

For other data sources, consider either extracting the data or using **WINDOW_COVAR**. See [Table Calculation Functions \(functions_functions_tablecalculation.htm\)](#).

If expression1 and expression2 are the same—for example, COVAR([profit], [profit])—COVAR returns a value that indicates how widely values are distributed.

Note: The value of COVAR(X, X) is equivalent to the value of VAR(X) and also to the value of STDEV(X)².

Example

The following formula returns the sample covariance of **Sales** and **Profit**.

```
COVAR([Sales], [Profit])
```

COVARP(expression1, expression2)

Returns the *population covariance* of two expressions.

Covariance quantifies how two variables change together. A positive covariance indicates that the variables tend to move in the same direction, as when larger values of one variable tend to correspond to larger values of the other variable, on average. 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 COVAR function) is appropriate.

Availability by data source

- Tableau data extracts (you can create an extract from any data source)
- Cloudera Hive
- EXASOL
- Firebird (version 3.0 and later)
- Google BigQuery
- Hortonworks Hadoop Hive
- IBM PDA (Netezza)
- Oracle
- PostgreSQL
- Presto
- SybaseIQ
- Teradata
- Vertica

For other data sources, consider either extracting the data or using `WINDOW_COVARP`. See [Table Calculation Functions \(functions_functions_tablecalculation.htm\)](#).

If `expression1` and `expression2` are the same—for example, `COVARP([profit], [profit])`—`COVARP` returns a value that indicates how widely values are distributed.

Note: The value of COVARP(X, X) is equivalent to the value of VARP(X) and also to the value of STDEV(X)².

Example

The following formula returns the population covariance of **Sales** and **Profit**.

```
COVARP([Sales], [Profit])
```

DATE(expression)

Returns a date given a number, string, or date expression.

Examples

```
DATE([Employee Start Date])
```

```
DATE("April 15, 2004") = #April 15, 2004#
```

```
DATE("4/15/2004")
```

```
DATE(#2006-06-15 14:52#) = #2006-06-15#
```

Quotation marks are required in the second and third examples.

DATEADD(date_part, interval, date)

Returns the specified date with the specified number `interval` added to the specified `date_part` of that date.

Example

```
DATEADD('month', 3, #2004-04-15#) = 2004-07-15 12:00:00 AM
```

This expression adds three months to the date #2004-04-15#.

DATEDIFF(date_part, date1, date2, [start_of_week])

Returns the difference between `date1` and `date2` expressed in units of `date_part`.

The `start_of_week` parameter, which you can use to specify which day is to be considered the first day of the week, is optional. Possible values are 'monday', 'tuesday', etc. If it is omitted, the start of week is determined by the data source. See [Date Properties for a Data Source \(date_properties.htm\)](#).

Example

```
DATEDIFF('week', #2013-09-22#, #2013-09-24#, 'monday')= 1
```

```
DATEDIFF('week', #2013-09-22#, #2013-09-24#, 'sunday')= 0
```

The first expression returns 1 because when `start_of_week` is 'monday', then 22 September (a Sunday) and 24 September (a Tuesday) are in different weeks. The second expression returns 0 because when `start_of_week` is 'sunday' then 22 September (a Sunday) and 24 September (a Tuesday) are in the same week.

DATENAME(date_part, date, [start_of_week])

Returns `date_part` of `date` as a string. The `start_of_week` parameter, which you can use to specify which day is to be considered the first day or the week, is optional. Possible values are 'monday', 'tuesday', etc. If `start_of_week` is

omitted, the start of week is determined by the data source. See [Date Properties for a Data Source \(date_properties.htm\)](#).

Examples

```
DATENAME ('year', #2004-04-15#) = "2004"
```

```
DATENAME ('month', #2004-04-15#) = "April"
```

DATEPARSE(format, string)

Converts a string to a datetime in the specified format. Support for some locale-specific formats is determined by the computer's system settings. Letters that appear in the data and do not need to be parsed should be surrounded by single quotes (' '). For formats that do not have delimiters between values (for example, MMddyy), verify that they are parsed as expected. The format must be a constant string, not a field value. This function returns `Null` if the data does not match the format.

This function is available for several connectors. For more information, see [Convert a Field to a Date Field \(data_dateparse.htm\)](#).

Examples

```
DATEPARSE ("dd.MMMM.yyyy", "15.April.2004") = #April 15, 2004#
```

```
DATEPARSE ("h'h' m'm' s's'", "10h 5m 3s") = #10:05:03#
```

DATEPART(date_part, date, [start_of_week])

Returns `date_part` of `date` as an integer.

The `start_of_week` parameter, which you can use to specify which day is to be considered the first day of the week, is optional. Possible values are 'monday', 'tuesday', etc. If `start_of_week` is omitted, the start of week is determined by the data source. See [Date Properties for a Data Source \(date_properties.htm\)](#).

Note: When the `date_part` is `weekday`, the `start_of_week` parameter is ignored. This is because Tableau relies on a fixed weekday ordering to apply offsets.

Examples

```
DATEPART('year', #2004-04-15#) = 2004
```

```
DATEPART('month', #2004-04-15#) = 4
```

DATETIME(expression)

Returns a datetime given a number, string, or date expression.

Example

```
DATETIME("April 15, 2005 07:59:00") = April 15, 2005 07:59:00
```

DATETRUNC(date_part, date, [start_of_week])

Truncates the specified date to the accuracy specified by the `date_part`. This function returns a new date. For example, when you truncate a date that is in the middle of the month at the month level, this function returns the first day of the month. The `start_of_week` parameter, which you can use to specify

which day is to be considered the first day of the week, is optional. Possible values are 'monday', 'tuesday', etc. If `start_of_week` is omitted, the start of week is determined by the data source. See [Date Properties for a Data Source \(date_properties.htm\)](#).

Examples

```
DATETRUNC('quarter', #2004-08-15#) = 2004-07-01 12:00:00 AM
```

```
DATETRUNC('month', #2004-04-15#) = 2004-04-01 12:00:00 AM
```

DAY(date)

Returns the day of the given date as an integer.

Example

```
DAY(#2004-04-12#) = 12
```

DEGREES(number)

Converts a given number in radians to degrees.

Example

```
DEGREES(PI() / 4) = 45.0
```

DISTANCE(Geometry1, Geometry2, "Units")

Returns distance measurement between two points in a specified unit.

Supported unit names: meters ("meters," "metres" "m), kilometers ("kilometers," "kilometres," "km"), miles ("miles" or "miles"), feet ("feet," "ft").

This function can only be created with a live connection and will continue to work when a data source is converted to an extract.

Examples

```
DISTANCE ({ EXCLUDE [Branch Name] : COLLECT([Selected Point]) },  
[unselected point], 'km')
```

```
DISTANCE([Origin MakePoint], [Destination MakePoint], "miles")
```

DIV(integer1, integer2)

Returns the integer part of a division operation, in which integer1 is divided by integer2.

Example

```
DIV(11,2) = 5
```

DOMAIN(string_url)

Note : Supported only when connected to Google BigQuery

Given a URL string, returns the domain as a string.

Example

DOMAIN('http://www.google.com:80/index.html') = 'google.com'

ELSE

See [IF THEN ELSE](#).

ELSEIF

See [IF THEN ELSE](#).

END

Used with functions like IF and CASE to indicate the end of the series of expressions.

ENDSWITH(string, substring)

Returns true if the given string ends with the specified substring. Trailing white spaces are ignored.

Example

```
ENDSWITH("Tableau", "leau") = true
```

EXP(number)

Returns e raised to the power of the given number.

Examples

```
EXP(2) = 7.389
```

```
EXP(-[Growth Rate]*[Time])
```

FIND(string, substring, [start])

Returns the index position of `substring` in `string`, or 0 if the `substring` isn't found. If the optional argument `start` is added, the function ignores any instances of `substring` that appear before the index position `start`. The first character in the string is position 1.

Examples

```
FIND("Calculation", "alcu") = 2
```

```
FIND("Calculation", "Computer") = 0
```

```
FIND("Calculation", "a", 3) = 7
```

```
FIND("Calculation", "a", 2) = 2
```

```
FIND("Calculation", "a", 8) = 0
```

FINDNTH(string, substring, occurrence)

Returns the position of the nth occurrence of `substring` within the specified `string`, where n is defined by the `occurrence` argument.

Note: FINDNTH is not available for all data sources.

Example

```
FINDNTH("Calculation", "a", 2) = 7
```

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.

The diagram illustrates the computation of the FIRST() function. On the left, a Tableau view displays quarterly sales data for the years 2009 and 2010 across four regions: Central, East, South, and West. The 'Central' column is highlighted with a red border. On the right, a separate table titled 'First()' shows the offset of each row from the first row in the partition. The first row has an offset of 0, and the second row has an offset of -1, indicated by a red arrow pointing from the highlighted cell in the view to the second row of the calculation table.

Region						First()
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	

Example

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

FLOAT(expression)

Casts its argument as a floating point number.

Examples

```
FLOAT(3) = 3.000
```

FLOAT([Age]) converts every value in the Age field to a floating point number.

FLOOR(number)

Rounds a number to the nearest integer of equal or lesser value.

Example

FLOOR(3.1415) = 3

Availability by data source

Data Source	Support
Microsoft Access	Not supported
Microsoft Excel	Supported
Text File	Supported
Statistical File	Supported
Tableau Server	Supported
Actian Vector	Not supported
Amazon Aurora	Not supported
Amazon EMR Hadoop Hive	Supported
Amazon Redshift	Not supported
Aster Database	Not supported
Cloudera Hadoop	Supported

DataStax Enterprise	Supported
EXASOL	Not supported
Firebird	Not supported
Google Analytics	Supported
Google BigQuery	Supported
Google Cloud SQL	Not supported
Hortonworks Hadoop Hive	Supported
IBM BigInsights	Not supported
IBM DB2	Not supported
IBM Netezza	Not supported
MapR Hadoop Hive	Supported
MarkLogic	Not supported
Microsoft Analysis Services	Not supported
Microsoft PowerPivot	Not supported
Microsoft SQL Server	Not supported
MySQL	Not supported
Oracle	Not supported
Oracle Essbase	Not supported
ParAccel	Not supported
Pivotal Greenplum	Not supported

PostgreSQL	Not supported
Progress OpenEdge	Not supported
Salesforce	Supported
SAP HANA	Not supported
SAP Sybase ASE	Not supported
SAP Sybase IQ	Not supported
Spark SQL	Supported
Splunk	Not supported
Teradata	Not supported
Teradata OLAP Connector	Not supported
Vertica	Not supported

FULLNAME()

Returns the full name for the current user. This is the Tableau Server or Tableau Cloud full name when the user is signed in; otherwise the local or network full name for the Tableau Desktop user.

Example

```
[Manager]=FULLNAME( )
```

If manager Dave Hallsten is signed in, this example returns True only if the Manager field in the view contained Dave Hallsten. When used as a filter, this calculated field can be used to create a user filter that only shows data that is

relevant to the person signed in to the server.

GET_JSON_OBJECT(JSON string, JSON path)

Note: Supported only when connected to Hadoop Hive.

Returns the JSON object within the JSON string based on the JSON path.

GROUP_CONCAT(expression)

Note : Supported only when connected to Google BigQuery

Concatenates values from each record into a single comma-delimited string.

This function acts like a SUM() for strings.

Example

GROUP_CONCAT(Region) = "Central,East,West"

HEXBINX(number, number)

Maps an x, y coordinate to the x-coordinate of the nearest hexagonal bin. The bins have side length 1, so the inputs may need to be scaled appropriately.

HEXBINX and HEXBINY are binning and plotting functions for hexagonal bins. Hexagonal bins are an efficient and elegant option for visualizing data in an x/y plane such as a map. Because the bins are hexagonal, each bin closely approximates a circle and minimizes variation in the distance from the data point to the center of the bin. This makes the clustering both more accurate and informative.

Example

```
HEXBINX( [Longitude] , [Latitude] )
```

HEXBINY(number, number)

Maps an x, y coordinate to the y-coordinate of the nearest hexagonal bin. The bins have side length 1, so the inputs may need to be scaled appropriately.

Example

```
HEXBINY( [Longitude] , [Latitude] )
```

HOST(string_url)

Note : Supported only when connected to Google BigQuery

Given a URL string, returns the host name as a string.

Example

```
HOST('http://www.google.com:80/index.html') = 'www.google.com:80'
```

IF THEN ELSE

IF test THEN value END / IF test THEN value ELSE else END

Use the IF THEN ELSE function to perform logical tests and return appropriate values. The IF THEN ELSE function evaluates a sequence of test conditions and returns the value for the first condition that is true. If no condition is true, the

ELSE value is returned. Each test must be a boolean: either be a boolean field in the data source or the result of a logical expression. The final ELSE is optional, but if it is not provided and there is no true test expression, then the function returns Null. All of the value expressions must be of the same type.

Examples

```
IF [Cost]>[Budget Cost] THEN 'Over Budget' ELSE 'Under Budget' END
```

```
IF [Budget Sales] !=0 THEN [Sales]/[Budget Sales] END
```

IF THEN ELSEIF

```
IF test1 THEN value1 ELSEIF test2 THEN value2 ELSE else END
```

Use this version of the IF function to perform logical tests recursively. There is no built-in limit to the number of ELSEIF values you can use with an IF function, though individual databases may impose a limit on IF function complexity. While an IF function can be rewritten as a series of nested IIF statements, there are differences in how the expressions will be evaluated. In particular, an IIF statement distinguishes TRUE, FALSE and UNKNOWN, whereas an IF statement only worries about TRUE and not true (which includes both FALSE and UNKNOWN).

Example

When you create bins from a measure, Tableau creates bins of equal size by default. For example, say you have a measure that represents age.

When you create bins from that measure, Tableau makes all the bins of equal size. You can specify how big you want the bins to be, but you

cannot specify a separate range of values for each bin. A way around this constraint is to create a calculated field to define bins. Then you can create one bin for ages 0 – 20, another for ages 21 – 32, and so on. The following procedure shows how you could do that.

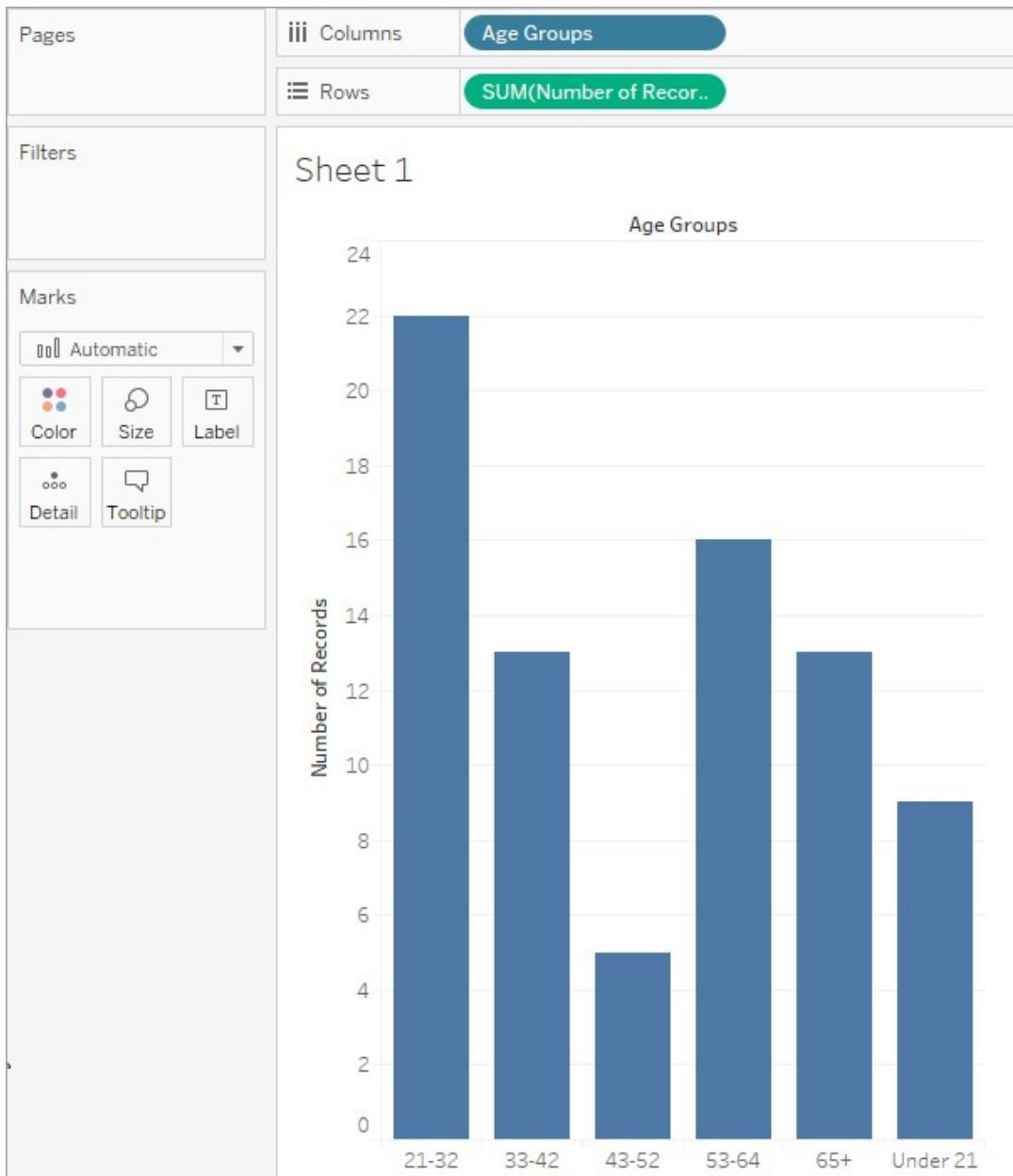
1. Create a new calculated field by choosing **Analysis > Create Calculated Field**.
2. Name the field **Age Groups** and type the following in the definition area

```
IF  
[Age] < 21 THEN 'Under 21'  
ELSEIF  
[Age] <= 32 THEN '21-32'  
ELSEIF  
[Age] <= 42 THEN '33-42'  
ELSEIF  
[Age] <= 52 THEN '43-52'  
ELSEIF  
[Age] <= 64 THEN '53-64'  
ELSE '65+'  
END
```

Confirm that the status message indicates that the formula is valid, and then click **OK**.

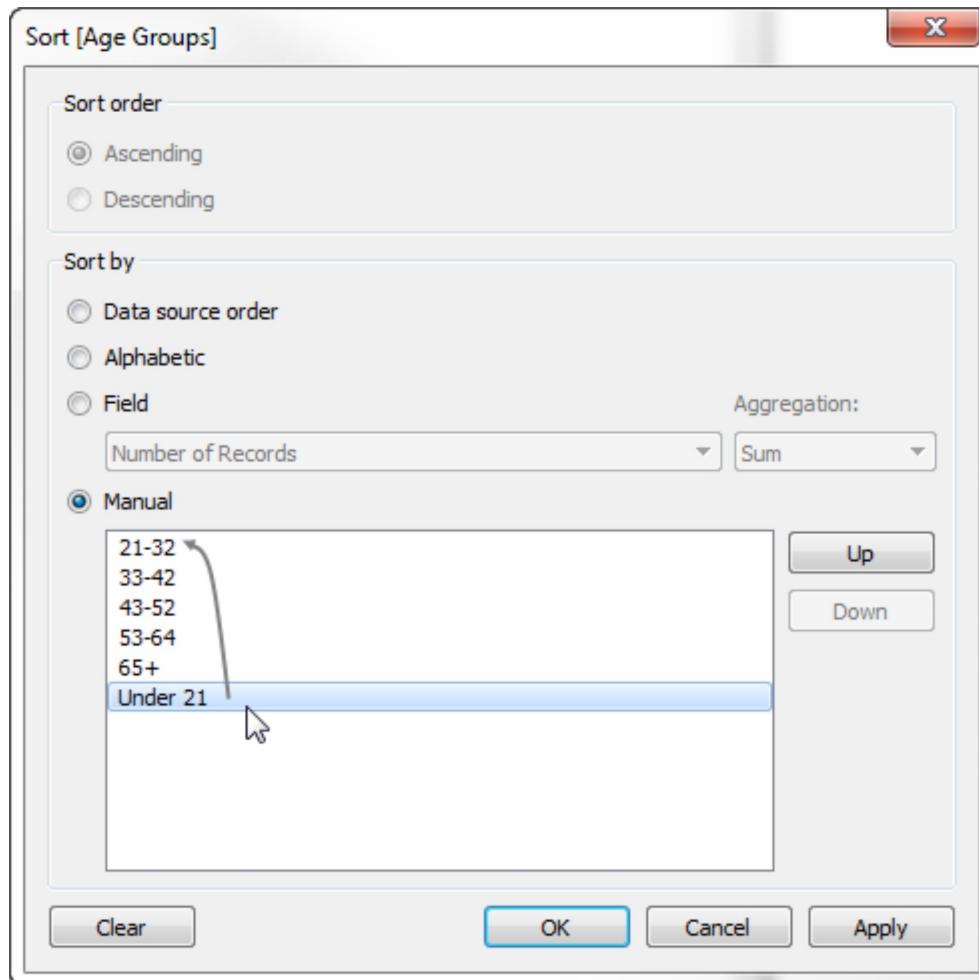
3. From the Measures area of the **Data** pane, drag **Number of Records** to **Rows**.
4. From the Dimensions area of the **Data** pane, drag **Age Groups** to **Columns**.

The records are now divided among the six bins that you defined:



Unfortunately, the **Under 21** bin is at far right, when you would expect it to be at far left. Tableau's smart enough to put the bins with entirely numerical names in the right order, but it can't guess that the bin name beginning with 'Under' belongs at the left. Fix the problem with a manual sort.

5. Click the down arrow at the right side of the Age Groups field on Columns and then click Sort. Choose Manual and then move the Under 21 bin up to the top of the list:



Your view is now complete.

IIF(test, then, else, [unknown])

Use the IIF function to perform logical tests and return appropriate values. The first argument, `test`, must be a boolean: either a boolean field in the data source, or the result of a logical expression using operators (or a logical comparison of AND, OR, or NOT). If `test` evaluates to TRUE, then IIF returns

the `then` value. If `test` evaluates to FALSE, then IIF returns the `else` value.

A boolean comparison may also yield the value UNKNOWN (neither TRUE nor FALSE), usually due to the presence of Null values in `test`. The final argument to IIF is returned in the event of an UNKNOWN result for the comparison. If this argument is left out, Null is returned.

Examples

```
IIF(7>5, 'Seven is greater than five', 'Seven is less than five')
```

```
IIF([Cost]>[Budget Cost], 'Over Budget', 'Under Budget')
```

```
IIF([Budget Sales]!=0, [Sales]/[Budget Sales], 0)
```

```
IIF(Sales>=[Budget Sales], 'Over Cost Budget and Over Sales Budget', 'Over Cost Budget and Under Sales Budget', 'Under Cost Budget')
```

IFNULL(expression1, expression2)

The IFNULL function returns the first expression if the result is not null, and returns the second expression if it is null.

Example

```
IFNULL([Profit], 0) = [Profit]
```

<expression1> IN <expression2>

Returns TRUE if <expression1> matches any value in <expression2>.

Example

[Name] IN [Set of attendees]

SUM([Cost]) IN (19.99, 20.99, 21.99)

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.

The diagram illustrates the use of the INDEX() function. On the left, a table shows quarterly sales data from 2009 to 2010 across four regions: Central, East, South, and West. The third row of the table (2009 Q3) is highlighted with a red box. An arrow points from this row to a vertical list on the right, also enclosed in a red box, which is labeled 'INDEX()'. The list contains the values \$160,877, \$197,213, \$302,678, \$297,208, \$180,609, \$195,785, and \$116,613, corresponding to the index numbers 1 through 7 respectively.

		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.

INT(expression)

Casts its argument as an integer. For expressions, this function truncates results to the closest integer toward zero.

Examples

`INT(8.0/3.0) = 2`

`INT(4.0/1.5) = 2`

`INT(0.50/1.0) = 0`

`INT(-9.7) = -9`

When a string is converted to an integer it is first converted to a float and then rounded.

ISDATE(string)

The ISDATE function returns `TRUE` if the string argument can be converted to a date and `FALSE` if it cannot.

Examples

`ISDATE('January 1, 2003') = TRUE`

`ISDATE('Jan 1 2003') = TRUE`

`ISDATE('1/1/03') = TRUE`

`ISDATE('Janxx 1 2003') = FALSE`

ISFULLNAME(string)

Returns true if the current user's full name matches the specified full name, or false if it does not match. This function uses the Tableau Server or Tableau

Cloud full name when the user is signed in; otherwise it uses the local or network full name for the Tableau Desktop user.

Example

```
ISFULLNAME("Dave Hallsten")
```

This example returns true if Dave Hallsten is the current user, otherwise it returns false.

ISMEMBEROF(string)

Returns TRUE if the current user is a member of the given group. This uses the logged in Tableau Server or Tableau Cloud site to resolve group membership, otherwise it always returns false.

Example

```
ISMEMBEROF("All Users")
```

ISNULL(expression)

The ISNULL function returns TRUE if the expression is Null and FALSE if it is not.

Example

The following example uses ISNULL in combination with IIF to replace null values with 0's.

```
IIF(ISNULL([Sales]), 0, [Sales] )
```

ISOQUARTER (date)

Returns the ISO8601 week-based quarter of a given date as an integer.

Example

```
ISOQUARTER (#2022-03-29#) = 1
```

ISOWEEK (date)

Returns the ISO8601 week-based week of a given date as an integer.

Example

```
ISOWEEK (#2022-03-29#) = 13
```

ISOWEEKDAY (date)

Returns the ISO8601 week-based weekday of a given date as an integer.

Example

```
ISOWEEKDAY (#2022-03-29#) = 2
```

ISOYEAR (date)

Returns the ISO8601 week-based year of a given date as an integer.

Example

ISOYEAR(#2019-12-31#) = 2020

ISUSERNAME(string)

Returns true if the current user's username matches the specified username, or false if it does not match. This function uses the Tableau Server or Tableau Cloud username when the user is signed in; otherwise it uses the local or network username for the Tableau Desktop user.

Example

```
ISUSERNAME("dhallsten")
```

This example returns true if dhallsten is the current user; otherwise it returns false.

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				LAST()
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	

Example

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

LEFT(string, number)

Returns the left-most number of characters in the string.

Example

LEFT("Matador", 4) = "Mata"

LEN(string)

Returns the length of the string.

Example

LEN("Matador") = 7

LN(number)

Returns the natural logarithm of a number. Returns `Null` if number is less than or equal to 0.

LOG(number [, base])

Returns the logarithm of a number for the given base. If the base value is omitted, base 10 is used.

LOG2(number)

Note : Supported only when connected to Google BigQuery

Returns the logarithm base 2 of a number.

Example

`LOG2(16) = '4.00'`

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.

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	+2 \$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	\$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.

LOWER(string)

Returns `string`, with all characters lowercase.

Example

`LOWER("ProductVersion") = "productversion"`

LTRIM(string)

Returns the string with any leading spaces removed.

Example

```
LTRIM(" Matador ") = "Matador "
```

LTRIM_THIS(string, string)

Note : Supported only when connected to Google BigQuery

Returns the first string with any leading occurrence of the second string removed.

Example

```
LTRIM_THIS('[-Sales-]', '[-') = 'Sales-'
```

MAKEDATE(year, month, day)

Returns a date value constructed from the specified year, month, and date.

Available for Tableau Data Extracts. Check for availability in other data sources.

Example

```
MAKEDATE(2004, 4, 15) = #April 15, 2004#
```

MAKEDATETIME(date, time)

Returns a datetime that combines a date and a time. The date can be a date, datetime, or a string type. The time must be a datetime. This function is available only for MySQL-compatible connections (which for Tableau are, in addition to MySQL, Amazon Aurora and Amazon Aurora).

Examples

```
MAKEDATETIME ("1899-12-30", #07:59:00#) = #12/30/1899 7:59:00 AM#
```

```
MAKEDATETIME ([Date], [Time]) = #1/1/2001 6:00:00 AM#
```

```
MAKELINE(geometry1,geometry2)
```

```
MAKELINE(Geometry1,Geometry2)
```

Generates a line mark between two spatial points; useful for building origin-destination maps.

Example

```
MAKELINE(MAKEPOINT([OriginLat],[OriginLong]),MAKEPOINT([DestinationLat],  
[DestinationLong]))
```

```
MAKELINE(Geometry1, Geometry2)
```

```
MAKEPOINT(Latitude,Longitude)
```

Converts data from latitude and longitude columns into spatial objects. You can use MAKEPOINT to spatially-enable a data source so that it can be joined with a spatial file using a spatial join. To use MAKEPOINT, your data must contain latitude and longitude coordinates.

Example

```
MAKEPOINT([AirportLatitude],[AirportLongitude])
```

MAKEPOINT(Xcoordinate,Ycoordinate,SRID)

Converts data from projected geographic coordinates into spatial objects. SRID is a spatial reference identifier that uses [ESPG reference system](#) (<https://epsg.io/>) codes to specify coordinate systems. If SRID is not specified, WGS84 is assumed and parameters are treated as latitude/longitude in degrees.

This function can only be created with a live connection and will continue to work when a data source is converted to an extract.

Example

```
MAKEPOINT([Xcoord],[Ycoord],3493)
```

MAKETIME(hour, minute, second)

Returns a date value constructed from the specified hour, minute, and second.

Available for Tableau Data Extracts. Check for availability in other data sources.

Example

```
MAKETIME(14, 52, 40) = #14:52:40#
```

MAX(a, b)

Returns the maximum of a and b (which must be of the same type). This function is usually used to compare numbers, but also works on strings. With strings, `MAX` finds the value that is highest in the sort sequence defined by the database for that column. It returns `Null` if either argument is `Null`.

Example

```
MAX ("Apple", "Banana") = "Banana"
```

MAX(expression) or MAX(expr1, expr2)

Usually applied to numbers but also works on dates. Returns the maximum of a and b (a and b must be of the same type). Returns `Null` if either argument is `Null`.

Examples

```
MAX(#2004-01-01# , #2004-03-01#) = 2004-03-01 12:00:00 AM
```

```
MAX([ShipDate1], [ShipDate2])
```

MAX(number, number)

Returns the maximum of the two arguments, which must be of the same type. Returns `Null` if either argument is `Null`. `MAX` can also be applied to a single field in an aggregate calculation.

Examples

```
MAX(4, 7)
```

MAX(Sales, Profit)

MAX([First Name], [Last Name])

MEDIAN(expression)

Returns the median of an expression across all records. Median can only be used with numeric fields. Null values are ignored. This function is not available for workbooks created before Tableau Desktop 8.2 or that use legacy connections.

Availability by data source

It is also **not** available for connections using any of the following data sources:

- Access
- Amazon Redshift
- Cloudera Hadoop
- IBM DB2
- IBM PDA (Netezza)
- Microsoft SQL Server
- MySQL
- SAP HANA
- Teradata
- Vertica

For these data source types, you can extract your data into an extract file to use this function. See [Extract Your Data \(extracting_data.htm\)](#).

MID(string, start, [length])

Returns the string starting at index position `start`. The first character in the string is position 1. If the optional argument `length` is added, the returned string includes only that number of characters.

Examples

```
MID("Calculation", 2) = "alculation"
```

```
MID("Calculation", 2, 5) ="alcul"
```

MIN(a, b)

Returns the minimum of `a` and `b` (which must be of the same type). This function is usually used to compare numbers, but also works on strings. With strings, `MIN` finds the value that is lowest in the sort sequence. It returns `NULL` if either argument is `NULL`.

Example

```
MIN ("Apple", "Banana") = "Apple"
```

MIN(expression) or MIN(expr1, expr2)

Usually applied to numbers but also works on dates. Returns the minimum of `a` and `b` (`a` and `b` must be of the same type). Returns `NULL` if either argument is `NULL`.

Examples

```
MIN(#2004-01-01# ,#2004-03-01#) = 2004-01-01 12:00:00 AM
```

```
MIN([ShipDate1], [ShipDate2])
```

MIN(number, number)

Returns the minimum of the two arguments, which must be of the same type.

Returns `Null` if either argument is `Null`. `MIN` can also be applied to a single field in an aggregate calculation.

Examples

```
MIN(4, 7)
```

```
MIN(Sales, Profit)
```

```
MIN([First Name], [Last Name])
```

MODEL_EXTENSION_BOOL(model_name, arguments, expression)

Returns the Boolean result of `model_name`, a deployed analytics extension model. Each argument is a single string that defines the elements you use. Use `expression` to define the input fields that are sent to the model, and use aggregation functions (SUM, AVG, etc.) to aggregate their results. Define and order each input field as its own argument.

Example

```
MODEL_EXTENSION_BOOL("model_isProfitable", "[inputSales]", "[inputCosts]",
```

```
SUM([Sales]), SUM([Costs]))
```

MODEL_EXTENSION_INT(model_name, arguments, expression)

Returns the integer result of model_name, a deployed analytics extension model. Each argument is a single string that defines the elements you use. Use expression to define the input fields that are sent to the model, and use aggregation functions (SUM, AVG, etc.) to aggregate their results. Define and order each input field as its own argument.

Example

```
MODEL_EXTENSION_INT("model_getPopulation", "[inputCity]", "[inputState]",  
MAX([City]), MAX([State]))
```

MODEL_EXTENSION_REAL(model_name, arguments, expression)

Returns the numeric result of model_name, a deployed analytics extension model. Each argument is a single string that defines the elements you use. Use expression to define the input fields that are sent to the model, and use aggregation functions (SUM, AVG, etc.) to aggregate their results. Define and order each input field as its own argument.

Example

```
MODEL_EXTENSION_REAL("model_ProfitRatio", "[inputSales]", "[inputCosts]",  
SUM([Sales]), SUM([Costs]))
```

MODEL_EXTENSION_STR(model_name, arguments, expression)

Returns the string result of model_name, a deployed analytics extension model. Each argument is a single string that defines the elements you use. Use expression to define the input fields that are sent to the model, and use aggregation functions (SUM, AVG, etc.) to aggregate their results. Define and order each input field as its own argument.

Example

```
MODEL_EXTENSION_STR("model_mostPopulatedCity", "[inputCountry]",  
"[inputYear]", MAX([Country]), MAX([Year]))
```

MODEL_PERCENTILE(target_expression, predictor_expression(s))

Returns the probability (between 0 and 1) of the expected value being less than or equal to the observed mark, defined by the target expression and other predictors. This is the Posterior Predictive Distribution Function, also known as the Cumulative Distribution Function (CDF).

Example

```
MODEL_PERCENTILE( SUM([Sales]), COUNT([Orders]))
```

MODEL_QUANTILE(quantile, target_expression, predictor_expression(s))

Returns a target numeric value within the probable range defined by the target expression and other predictors, at a specified quantile. This is the Posterior Predictive Quantile.

Example

```
MODEL_QUANTILE(0.5, SUM([Sales]), COUNT([Orders]))
```

MONTH(date)

Returns the month of the given date as an integer.

Example

```
MONTH(#2004-04-15#) = 4
```

NOT

Performs logical negation on an expression

Example

```
IF NOT <exp> THEN <then> END
```

NOW()

Returns the current date and time.

The return varies depending on the nature of the connection:

- For a live, unpublished connection, NOW returns the data source server time.
- For a live, published connection, NOW returns the data source server time.

- For an unpublished extract, NOW returns the local system time.
- For a published extract, NOW returns the local time of the Tableau Server Data Engine. When there are multiple worker machines in different time zones, this can produce inconsistent results.

Example

```
NOW( ) = 2004-04-15 1:08:21 PM
```

OR

Performs logical disjunction on two expressions

Example

```
IF <expr1> or <expr2> THEN <then> END
```

PARSE_URL(string, url_part)

Note: Supported only when connected to Hadoop Hive and Cloudera Impala.

Returns a component of the given URL string where the component is defined by url_part. Valid url_part values include: 'HOST', 'PATH', 'QUERY', 'REF', 'PROTOCOL', 'AUTHORITY', 'FILE' and 'USERINFO'.

Example

```
PARSE_URL('http://www.tableau.com', 'HOST') = 'www.tableau.com'
```

PARSE_URL_QUERY(string, key)

Note: Supported only when connected to Hadoop Hive and Cloudera Impala.

Returns the value of the specified query parameter in the given URL string. The query parameter is defined by the key.

Example

```
PARSE_URL_QUERY('http://www.tableau.com?page=1&cat=4', 'page') = '1'
```

PERCENTILE(expression, number)

Returns the percentile value from the given expression corresponding to the specified number. The number must be between 0 and 1 (inclusive)—for example, 0.66, and must be a numeric constant.

Availability by data source

This function is available for the following data sources.

- Non-legacy Microsoft Excel and Text File connections.
- Extracts and extract-only data source types (for example, Google Analytics, OData, or Salesforce).
- Sybase IQ 15.1 and later data sources.
- Oracle 10 and later data sources.
- Cloudera Hive and Hortonworks Hadoop Hive data sources.
- EXASOL 4.2 and later data sources.

For other data source types, you can extract your data into an extract file

to use this function. See [Extract Your Data \(extracting_data.htm\)](#).

PI()

Returns the numeric constant pi: 3.14159.

POWER(number, power)

Raises the number to the specified power.

Examples

POWER(5,2) = 5² = 25

POWER(Temperature, 2)

You can also use the \wedge symbol:

$5^2 = \text{POWER}(5,2) = 25$

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)`.

Quarter

Returns the quarter of a given date as an integer.

Example

```
QUARTER(#2021-02-20#) = 1
```

RADIANS(number)

Converts the given number from degrees to radians.

Example

```
RADIANS(180) = 3.14159
```

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.

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

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

Returns the modified competition rank for the current row in the partition.

Identical values are assigned an identical rank.

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

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

Returns the percentile rank for the current row in the partition.

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

Note: Unlike the other rank options, the default is ascending.

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

Returns the unique rank for the current row in the partition.

Identical values are assigned different ranks.

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

RAWSQL_BOOL("sql_expr", [arg1], ...[argN])

Returns a Boolean result from a given SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values.

Example

In the example, %1 is equal to [Sales] and %2 is equal to [Profit].

```
RAWSQL_BOOL("IIF(%1 > %2, True, False)", [Sales], [Profit])
```

RAWSQL_DATE("sql_expr", [arg1], ...[argN])

Returns a Date result from a given SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values.

Example

In this example, %1 is equal to [Order Date].

```
RAWSQL_DATE("%1", [Order Date])
```

RAWSQL_DATETIME("sql_expr", [arg1], ...[argN])

Returns a Date and Time result from a given SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Delivery Date].

Example

```
RAWSQL_DATETIME("MIN(%1)", [Delivery Date])
```

RAWSQL_INT("sql_expr", [arg1], ...[argN])

Returns an integer result from a given SQL expression. The SQL expression is

passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Sales].

Example

```
RAWSQL_INT("500 + %1", [Sales])
```

RAWSQL_REAL("sql_expr", [arg1], ...[argN])

Returns a numeric result from a given SQL expression that is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Sales]

Example

```
RAWSQL_REAL("-123.98 * %1", [Sales])
```

RAWSQL_SPATIAL

Returns a Spatial from a given SQL expression that is passed directly to the underlying data source. Use %n in the SQL expression as a substitution syntax for database values.

Example

In this example, %1 is equal to [Geometry].

```
RAWSQL_SPATIAL("%1", [Geometry])
```

RAWSQL_STR("sql_expr", [arg1], ...[argN])

Returns a string from a given SQL expression that is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Customer Name].

Example

```
RAWSQL_STR("%1", [Customer Name])
```

RAWSQLAGG_BOOL("sql_expr", [arg1], ...[argN])

Returns a Boolean result from a given aggregate SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values.

Example

In the example, %1 is equal to [Sales] and %2 is equal to [Profit].

```
RAWSQLAGG_BOOL("SUM( %1 ) > SUM( %2 )", [Sales], [Profit])
```

RAWSQLAGG_DATE("sql_expr", [arg1], ...[argN])

Returns a Date result from a given aggregate SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Order Date].

Example

```
RAWSQLLAGG_DATE ("MAX(%1)", [Order Date])
```

RAWSQLLAGG_DATETIME("sql_expr", [arg1], ...[argN])

Returns a Date and Time result from a given aggregate SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Delivery Date].

Example

```
RAWSQLLAGG_DATETIME ("MIN(%1)", [Delivery Date])
```

RAWSQLLAGG_INT("sql_expr", [arg1,] ...[argN])

Returns an integer result from a given aggregate SQL expression. The SQL expression is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Sales].

Example

```
RAWSQLLAGG_INT ("500 + SUM(%1)", [Sales])
```

RAWSQLLAGG_REAL("sql_expr", [arg1,] ...[argN])

Returns a numeric result from a given aggregate SQL expression that is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Sales]

Example

```
RAWSQLAGG_REAL("SUM( %1 )", [Sales])
```

RAWSQLAGG_STR("sql_expr", [arg1], ...[argN])

Returns a string from a given aggregate SQL expression that is passed directly to the underlying database. Use %n in the SQL expression as a substitution syntax for database values. In this example, %1 is equal to [Discount].

Example

```
RAWSQLAGG_STR("AVG(%1)", [Discount])
```

REGEXP_EXTRACT(string, pattern)

Returns the portion of the string that matches the regular expression pattern. This function is available for Text File, Hadoop Hive, Google BigQuery, PostgreSQL, Tableau Data Extract, Microsoft Excel, Salesforce, Vertica, Pivotal Greenplum, Teradata (version 14.1 and above), Snowflake, and Oracle data sources.

For Tableau data extracts, the pattern must be a constant.

For information on regular expression syntax, see your data source's

documentation. For Tableau extracts, regular expression syntax conforms to the standards of the ICU (International Components for Unicode), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the [Regular Expressions](#) page in the online ICU User Guide.

Example

```
REGEXP_EXTRACT('abc 123', '[a-z]+\s+(\d+)') = '123'
```

REGEXP_EXTRACT_NTH(string, pattern, index)

Returns the portion of the string that matches the regular expression pattern. The substring is matched to the nth capturing group, where n is the given index. If index is 0, the entire string is returned. This function is available for Text File, PostgreSQL, Tableau Data Extract, Microsoft Excel, Salesforce, Vertica, Pivotal Greenplum, Teradata (version 14.1 and above), and Oracle data sources.

For Tableau data extracts, the pattern must be a constant.

For information on regular expression syntax, see your data source's documentation. For Tableau extracts, regular expression syntax conforms to the standards of the ICU (International Components for Unicode), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the [Regular Expressions](#) page in the online ICU User Guide.

Example

```
REGEXP_EXTRACT_NTH('abc 123', '([a-z]+)\s+(\d+)', 2) = '123'
```

REGEXP_MATCH(string, pattern)

Returns TRUE if a substring of the specified string matches the regular expression pattern. This function is available for Text File, Google BigQuery, PostgreSQL, Tableau Data Extract, Microsoft Excel, Salesforce, Vertica, Pivotal Greenplum, Teradata (version 14.1 and above), Impala 2.3.0 (through Cloudera Hadoop data sources), Snowflake, and Oracle data sources.

For Tableau data extracts, the pattern must be a constant.

For information on regular expression syntax, see your data source's documentation. For Tableau extracts, regular expression syntax conforms to the standards of the ICU (International Components for Unicode), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the [Regular Expressions](#) page in the online ICU User Guide.

Example

```
REGEXP_MATCH('-[1234].[The.Market]-', '\[\s*(\w*\.).(\w*\s*\])')=true
```

REGEXP_REPLACE(string, pattern, replacement)

Returns a copy of the given string where the regular expression pattern is replaced by the replacement string. This function is available for Text File, Hadoop Hive, Google BigQuery, PostgreSQL, Tableau Data Extract, Microsoft Excel, Salesforce, Vertica, Pivotal Greenplum, Teradata (version 14.1 and

above), Snowflake, and Oracle data sources.

For Tableau data extracts, the pattern and the replacement must be constants.

For information on regular expression syntax, see your data source's documentation. For Tableau extracts, regular expression syntax conforms to the standards of the ICU (International Components for Unicode), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the [Regular Expressions](#) page in the online ICU User Guide.

Example

```
REGEXP_REPLACE('abc 123', '\s', '-') = 'abc-123'
```

REPLACE(string, substring, replacement)

Searches `string` for `substring` and replaces it with `replacement`. If `substring` is not found, the string is not changed.

Example

```
REPLACE("Version8.5", "8.5", "9.0") = "Version9.0"
```

RIGHT(string, number)

Returns the right-most number of characters in `string`.

Example

```
RIGHT("Calculation", 4) = "tion"
```

ROUND(number, [decimals])

Rounds numbers to a specified number of digits. The `decimals` argument specifies how many decimal points of precision to include in the final result. If `decimals` is omitted, `number` is rounded to the nearest integer.

Example

This example rounds every `Sales` value to an integer:

```
ROUND(Sales)
```

Some databases, such as SQL Server, allow specification of a negative `length`, where `-1` rounds `number` to 10's, `-2` rounds to 100's, and so on. This is not true of all databases. For example, it is not true of Excel or Access.

RTRIM(string)

Returns `string` with any trailing spaces removed.

Example

```
RTRIM(" Calculation ") = " Calculation"
```

RTRIM_THIS(string, string)

Note : Supported only when connected to Google BigQuery

Returns the first string with any trailing occurrence of the second string removed.

Example

`RTRIM_THIS('[-Market-]', '-') = '[-Market'`

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.

The diagram illustrates the calculation of a running average for quarterly sales. It shows two tables: one for raw data and one for the calculated running average.

Raw Data Table:

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

Calculated Table:

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	235,873	199,734	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

A red box highlights the 'Central' column of the raw data table, and another red box highlights the 'Central' column of the calculated table. A red arrow points from the raw data table to a vertical stack of numbers representing the running total for the Central region. To the right of this stack is a box labeled 'Average = \$179,045'.

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.

The diagram shows two tables and a vertical list of values. A red arrow points from the third row of the top table to the third value in the list. Another red arrow points from the third row of the bottom table to the same value in the list. The list is labeled "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	\$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	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

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

Max = \$302,678

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.

The diagram illustrates the calculation of `RUNNING_MIN(SUM([Profit]))`. It shows two tables: the original data and a calculated table where the running minimum is applied.

Original Data:

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

Calculated Data:

Year of Order Date	Quarter of Order Date	Region			
		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

Result: Min = \$160,877

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.

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	160,877	231,411	133,934	185,961
	Q2	358,090	436,325	471,747	399,469
	Q3	660,768	711,520	755,559	685,900
	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)`

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 `.argn` (with a leading period) to reference parameters (`.arg1`, `.arg2`, etc.).

In Python expressions, use `_argn` (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 `.argn` (with a leading period) to reference parameters (`.arg1`, `.arg2`, etc.)

In Python expressions, use `_argn` (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 `.argn` (with a leading period) to reference parameters (`.arg1`, `.arg2`, etc.)

In Python expressions, use `_argn` (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 `.argn` (with a leading period) to reference parameters (`.arg1`, `.arg2`, etc.)

In Python expressions, use `_argn` (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]))
```

SIGN(number)

Returns the sign of a number: The possible return values are `-1` if the number is negative, `0` if the number is zero, or `1` if the number is positive.

Example

If the average of the profit field is negative, then

```
SIGN(AVG(Profit)) = -1
```

SIN(number)

Returns the sine of an angle. Specify the angle in radians.

Example

$\text{SIN}(0) = 1.0$

$\text{SIN}(\text{PI}(\))/4) = 0.707106781186548$

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.

The diagram illustrates the calculation of the SIZE() function. On the left, a Tableau view displays quarterly sales data for two years, 2009 and 2010. The view has three main columns: Year of Order Date, Quarter of Order Date, and Region (Central, East, South, West). A red box highlights the Central column of the Date partition (rows 1-4). An arrow points from this red box to a vertical stack of seven numbers (160,877, 197,213, 302,678, 297,208, 226,983, 180,609, 195,785), which are also enclosed in a red box. To the right of this stack, the text "Size = 7" is displayed.

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	

Example

$\text{SIZE}() = 5$ when the current partition contains five rows.

SPACE(number)

Returns a string that is composed of the specified `number` of repeated spaces.

Example

```
SPACE(1) = " "
```

SPLIT(string, delimiter, token number)

Returns a substring from a string, using a delimiter character to divide the string into a sequence of tokens.

The string is interpreted as an alternating sequence of delimiters and tokens.

So for the string `abc-defgh-i-jkl`, where the delimiter character is '`-`', the tokens are abc, defgh, i, and jlk. Think of these as tokens 1 through 4. `SPLIT` returns the token corresponding to the token number. When the token number is positive, tokens are counted starting from the left end of the string; when the token number is negative, tokens are counted starting from the right.

Examples

```
SPLIT ('a-b-c-d', '-', 2) = 'b'
```

```
SPLIT ('a|b|c|d', '|', -2) = 'c'
```

Availability by data source

Note: The split and custom split commands are available for the following data sources types: Tableau data extracts, Microsoft Excel, Text File, PDF File, Salesforce, OData, Microsoft Azure Market Place, Google Analytics, Vertica, Oracle, MySQL, PostgreSQL, Teradata, Amazon

Redshift, Aster Data, Google Big Query, Cloudera Hadoop Hive, Hortonworks Hive, and Microsoft SQL Server.

Some data sources impose limits on splitting string. The following table shows which data sources support negative token numbers (splitting from the right) and whether there is a limit on the number of splits allowed per data source. A SPLIT function that specifies a negative token number and would be legal with other data sources will return this error with these data sources: “Splitting from right is not supported by the data source.”

Data Source	Left/Right Constraints	Maximum Number of Splits	Version Limitations
Tableau Data Extract	Both	Infinite	
Microsoft Excel	Both	Infinite	
Text file	Both	Infinite	
Salesforce	Both	Infinite	
OData	Both	Infinite	
Google Analytics	Both	Infinite	
Tableau Data Server	Both	Infinite	Supported in version 9.0.
Vertica	Left only	10	
Oracle	Left only	10	

MySQL	Both	10	
PostgreSQL	Left only prior to version 9.0; both for version 9.0 and above	10	
Teradata	Left only	10	Version 14 and later
Amazon Redshift	Left only	10	
Aster Database	Left only	10	
Google BigQuery	Left only	10	
Hortonworks Hadoop Hive	Left only	10	
Cloudera Hadoop	Left only	10	Impala supported starting in version 2.3.0.
Microsoft SQL Server	Both	10	2008 and later

SQRT(number)

Returns the square root of a number.

Example

SQRT(25) = 5

SQUARE(number)

Returns the square of a number.

Example

SQUARE(5) = 25

STARTSWITH(string, substring)

Returns true if `string` starts with `substring`. Leading white spaces are ignored.

Example

STARTSWITH("Joker", "Jo") = true

STDEV(expression)

Returns the statistical standard deviation of all values in the given expression based on a sample of the population.

STDEVP(expression)

Returns the statistical standard deviation of all values in the given expression based on a biased population.

STR(expression)

Casts its argument as a string.

Example

`STR([Age])` takes all of the values in the measure called `Age` and converts them to strings.

SUM(expression)

Returns the sum of all values in the expression. SUM can be used with numeric fields only. Null values are ignored.

TAN(number)

Returns the tangent of an angle. Specify the angle in radians..

Example

`TAN(PI() / 4) = 1.0`

THEN

See [IF THEN ELSE](#) and [CASE](#).

TIMESTAMP_TO_USEC(expression)

Note : Supported only when connected to Google BigQuery

Converts a TIMESTAMP data type to a UNIX timestamp in microseconds.

Example

```
TIMESTAMP_TO_USEC(#2012-10-01 01:02:03#)=1349053323000000
```

TLD(string_url)

Note : Supported only when connected to Google BigQuery

Given a URL string, returns the top level domain plus any country domain in the URL.

Example

```
TLD('http://www.google.com:80/index.html') = '.com'
```

```
TLD('http://www.google.co.uk:80/index.html') = '.co.uk'
```

TODAY()

Returns the current date.

Example

```
TODAY() = 2004-04-15
```

TOTAL(expression)

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

Detailed example

Assume you are starting with this view:

Year of Order Date	Quarter of Order Date	Region			
		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:

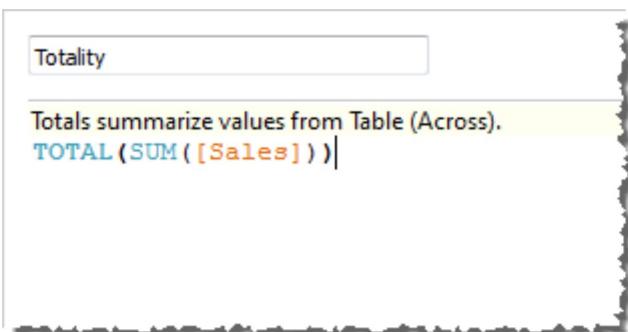
```
    Totality
TOTAL(SUM([Sales]))
```

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:

Year of Order..	Quarter of O..	Region			
		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:

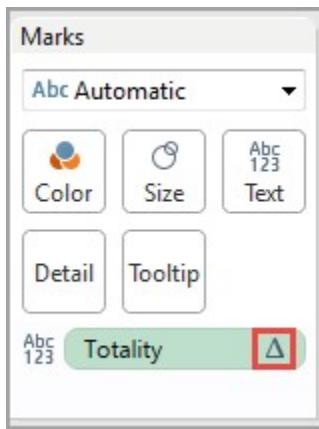


The default **Compute Using** value is **Table (Across)**. The result is that **Totality** is summing the values across each row of your table. Thus, the value that you see across each row is the sum of the values from the

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:

The screenshot shows a Tableau desktop interface with a data view. The columns are labeled: Year of Order Date, Quarter of Order Date, Central, East, South, and West. The 'Region' column is highlighted with a blue background. The data shows sales figures for each quarter from 2011 to 2014 across four regions. The 'Totality' button in the Marks shelf is also highlighted.

Year of Order Date	Quarter of Order Date	Region			
		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

TRIM(string)

Returns the string with leading and trailing spaces removed. For example,

```
TRIM(" Calculation ") = "Calculation"
```

UPPER(string)

Returns string, with all characters uppercase.

Example

```
UPPER("Calculation") = "CALCULATION"
```

USEC_TO_TIMESTAMP(expression)

Note : Supported only when connected to Google BigQuery

Converts a UNIX timestamp in microseconds to a TIMESTAMP data type.

Example

```
USEC_TO_TIMESTAMP(1349053323000000) = #2012-10-01 01:02:03#
```

USERDOMAIN()

Returns the domain for the current user when the user is signed on to Tableau Server. Returns the Windows domain if the Tableau Desktop user is on a domain. Otherwise this function returns a null string.

Example

```
[Manager]=USERNAME() AND [Domain]=USERDOMAIN()
```

USERNAME()

Returns the username for the current user. This is the Tableau Server or Tableau Cloud username when the user is signed in; otherwise it is the local or network username for the Tableau Desktop user.

Example

```
[Manager]=USERNAME()
```

If the manager dhallsten was signed in, this function would only return True when the Manager field in the view is dhallsten. When used as a filter this calculated field can be used to create a user filter that only shows data that is

relevant to the person signed in to the server.

VAR(expression)

Returns the statistical variance of all values in the given expression based on a sample of the population.

VARP(expression)

Returns the statistical variance of all values in the given expression on the entire population.

WEEK(date)

Returns the week of a given date as an integer.

Example

```
WEEK (#2022-03-29#) = 14
```

WHEN

See [CASE](#)

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.

WINDOW_AVG(SUM([Sales]), FIRST(), LAST())

		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	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 function: CORR.

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.

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 function: COVARP. See [Tableau Functions \(Alphabetical\)](#).

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_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.

WINDOW_MAX(SUM([Sales]), FIRST(), LAST())

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	\$206,612
	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_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.

The diagram shows two tables. The top table is a summary view with columns: Year of Order Date, Quarter of Order Date, and Region (Central, East, South, West). The bottom table is a detailed view with the same structure. A red box highlights the Central region in the top table, and a red arrow points from it to the Central region in the bottom table. A vertical red line connects the two tables. To the right, a vertical stack of numbers represents the profit values for the Central region: \$8,921, \$22,009, \$37,861, \$57,840, \$26,269, \$39,999, \$9,030, and \$34,545. The word 'FIRST()' is at the top of the stack, and 'LAST()' is at the bottom. To the right of the stack is the formula **MEDIAN= \$30,407**.

		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

Example

`WINDOW_MEDIAN(SUM([Profit]), FIRST()+1, 0)` computes the median 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.

WINDOW_MIN(SUM([Sales]), FIRST(), LAST())

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	\$166,201	\$283,806	\$206,612
	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	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.

`XPATH_BOOLEAN(XML string, XPath expression string)`

Note: Supported only when connected to Hadoop Hive

Returns true if the XPath expression matches a node or evaluates to true.

Example

```
XPATH_BOOLEAN('<values> <value id="0">1</value><value id="1">5</value>',  
'values/value[@id="1"] = 5') = true
```

`XPATH_DOUBLE(XML string, XPath expression string)`

Note: Supported only when connected to Hadoop Hive

Returns the floating-point value of the XPath expression.

Example

```
XPATH_DOUBLE ('<values><value>1.0</value><value>5.5</value> </values>',  
'sum(value/*)' ) = 6.5
```

XPATH_FLOAT(XML string, XPath expression string)

Note: Supported only when connected to Hadoop Hive

Returns the floating-point value of the XPath expression.

Example

```
XPATH_FLOAT ('<values><value>1.0</value><value>5.5</value>  
</values>', 'sum(value/*)' ) = 6.5
```

XPATH_INT(XML string, XPath expression string)

Note: Supported only when connected to Hadoop Hive

Returns the numerical value of the XPath expression, or zero if the XPath expression cannot evaluate to a number.

Example

```
XPATH_INT ('<values><value>1</value><value>5</value>  
</values>', 'sum(value/*)' ) = 6
```

XPATH_LONG(XML string, XPath expression string)

Note: Supported only when connected to Hadoop Hive

Returns the numerical value of the XPath expression, or zero if the XPath expression cannot evaluate to a number.

Example

```
XPATH_LONG ('<values><value>1</value><value>5</value>
</values>', 'sum(value/*)') = 6
```

XPATH_SHORT(XML string, XPath expression string)

Note: Supported only when connected to Hadoop Hive

Returns the numerical value of the XPath expression, or zero if the XPath expression cannot evaluate to a number.

Example

```
XPATH_SHORT ('<values><value>1</value><value>5</value>
</values>', 'sum(value/*)') = 6
```

XPATH_STRING(XML string, XPath expression string)

Note: Supported only when connected to Hadoop Hive

Returns the text of the first matching node.

Example

```
XPATH_STRING('<sites ><url domain="org">http://www.w3.org</url> <url  
domain="com">http://www.tableau.com</url></sites>',  
'sites/url[@domain="com"]') = 'http://www.tableau.com'
```

YEAR (date)

Returns the year of the given date as an integer.

Example

```
YEAR (#2004-04-15#) = 2004
```

ZN(expression)

Returns the expression if it is not null, otherwise returns zero. Use this function to use zero values instead of null values.

Example

```
ZN([Profit]) = [Profit]
```

Want to learn more about functions?

Read the [functions topics](#) .

See also

[Tableau Functions \(by Category\)](#)

[Functions in Tableau](#)