# Tableau Interview Questions

## Table of Contents

# Tableau Interview Questions

## What is Tableau?

Tableau is a powerful data visualization tool used in the Business Intelligence Industry. It helps in simplifying raw data into a very easily understandable format.

- Tableau is a business intelligence software.
- It allows anyone to connect to the respective data.
- Visualizes and creates interactive, shareable dashboards.

## Explain dashboard lifecycle

Dashboard lifecycle in Tableau:

**Functional Knowledge:** Business Analysts give a current functional knowledge of the organization.

**Requirement Analysis:** Requirements that are kept in consideration are:

- The requirement of the dashboard.
- How is data flowing in the current system?
- Blueprint or layout of the system.
- Dashboard scope.
- The value that is added to the business
- required tools for the development of the project and its costs.

**Planning Phase:** It includes:

- Timeline and needed resources.
- Work and leave plan.
- Dependencies and future challenges.

**Methodologies to follow:** Scrum, Agile, Waterfall, etc.

**Technical Specs:** It includes:

- Technical details.
- SQL, relations, and Joins.
- Credentials for database access.
- Business logic.

**Development:** It includes:

- Query generation.
- Connecting databases and creating dimension model
- Publish it to the server.
- Unit testing.

**Q&A Testing:** It includes:

- Functionality and UI testing.
- SQL testing and data validation
- Security testing
- Testing of applied customization.

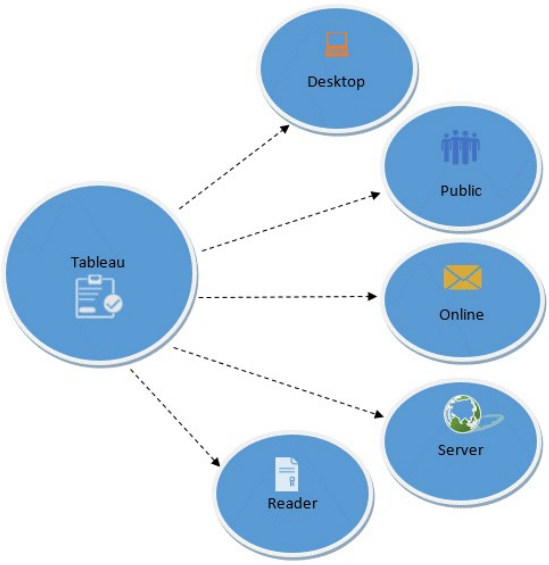
**Performance testing:** Report opening time, with or without any webpage.

**User Acceptance Testing (UAT):** User validates data and functionality.

**Production and Support:** System is produced, and support is given once it goes live.

## Explain Tableau Product Suite

The Tableau Product Suite consists of




For a clear understanding, data analytics in the tableau can be classified into two sections

**Tableau Desktop**

Tableau Desktop has a rich feature set and allows you to code and customize reports. It enables users to create charts, reports, and dashboards.

**Tableau Public**

It is the Tableau version specially build for cost-effective users. By the word “Public,” it means that the workbooks created cannot be saved locally. In turn, it should be saved to Tableau’s public cloud, which can be viewed and accessed by anyone.

**Tableau Server**

The software is specifically used to share the workbooks, visualizations that are created in the Tableau Desktop application across the organization.

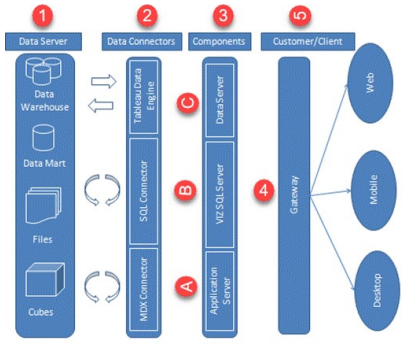
**Tableau Online**

As the name suggests, it is an online sharing tool for Tableau. Its functionalities are similar to Tableau Server, but the data is stored on servers hosted in the cloud, which are maintained by the Tableau group.

**Tableau Reader**

Tableau Reader is a free tool that enables the user to view the workbooks and visualizations created using Tableau Desktop or Tableau Public. The data can be filtered, but editing and modifications are restricted. The security level is zero in Tableau Reader as anyone who gets the workbook can view it using Tableau Reader.

## Explain the Tableau architecture



Different components of Tableau architecture are:

**Data server:** The primary component of Tableau Architecture is the Data sources it can connect to it.

**Data Connectors:** The Data Connectors provide an interface to connect external data sources to the Tableau Data Server.

**Components of Tableau Server:**

**1) Application Server:**

The application server is used to provide the authentications and authorizations. It handles the administration and permission for web and mobile interfaces.

**2) VizQL Server:**

VizQL server is used to convert the queries from the data source into visualizations. Once the client request is forwarded to VizQL process, it sends the query directly to the data source and retrieves information in the form of images.

**3) Gateway:**

The gateway channelizes the requests from users to Tableau components. When the client makes a request, it is forwarded to the external load balancer for processing. The gateway works as a distributor of processes to various components.

**4) Clients:**

The dashboards and visualizations in Tableau server can be viewed and edited using different clients. The Clients are Tableau Desktop, web browser, and mobile applications.

## Differentiate between Excel and Tableau?

The difference between Excel and Tableau is:

| Excel | Tableau |
|---|---|
| Excel is spreadsheet software that is used for data manipulation. | Tableau is a data visualization tool that is used for analysis. |
| It is ideal for statistical analysis of structured data. | It is ideal for the quick and easy representation of big data. |
| Macro and visual primary language are must to fully utilize excel. | It can be used with no programming experience. |
| The inbuilt security feature is not as good as compared to Tableau. | The inbuilt security feature is not as good as compared to Excel. |
| Best for preparing on-off reports with small data | Best while working with big data. |
| Excel integrates with around 60 applications. | Tableaus integrated with over 250 applications. |

## What are the popular features of Tableau?

The popular features of Tableau are:

- Data blending
- No need of technical knowledge
- Real-time analysis
- Data collaboration and data notifications

- DAX analysis function
- Patented technology from Stanford university
- Toggle view and drag-and-drop
- List of native data connectors
- Highlight and filter data
- Share dashboards
- Embed dashboards within
- Mobile-ready dashboards
- Tableau reader for data viewing
- Dashboard commenting
- Create “no-code” data queries
- Translate queries to visualizations
- Import all ranges and sizes of data

## What are the different Tableau files?

Different Tableau files include:

- **Workbooks:** Workbooks hold one or more worksheets and dashboards.
- **Bookmarks:** It contains a single spreadsheet, and it’s an easy way to quickly share your work.
- **Packaged workbooks:** It includes a workbook having supporting background images and local file data.
- **Data extraction files:** Data extract files are basically a local copy of the entire data source or a subset.
- **Data connection files:** It is a XML file containing various information related to connection.

## Mention what is the difference between published data sources and embedded data sources in Tableau?

The difference between published data source and embedded data source is that,

- Published data source: It contains connection information that is independent of any workbook and can be used by multiple workbooks.
- Embedded data source: It contains connection information and is associated with a workbook.

## What is TDE file?

TDE is a Tableau desktop file that contains a .tde extension. It refers to the file that contains data extracted from external sources like MS Excel, MS Access or CSV file.
There are two aspects of TDE design that make them ideal for supporting analytics and data discovery.

- Firstly, TDE is a columnar store.
- The second is how they are structured which impacts how they are loaded into memory and used by Tableau. This is an important aspect of how TDEs are “architecture aware”. Architecture-awareness means that TDEs use all parts of your computer memory, from RAM to hard disk, and put each part to work what best fits its characteristics.

## What is the difference between .twb and .twbx extension?

- A .twb is an xml document which contains all the selections and layout made you have made in your Tableau workbook. It does not contain any data.
- A .twbx is a ‘zipped’ archive containing a .twb and any external files such as extracts and background images.

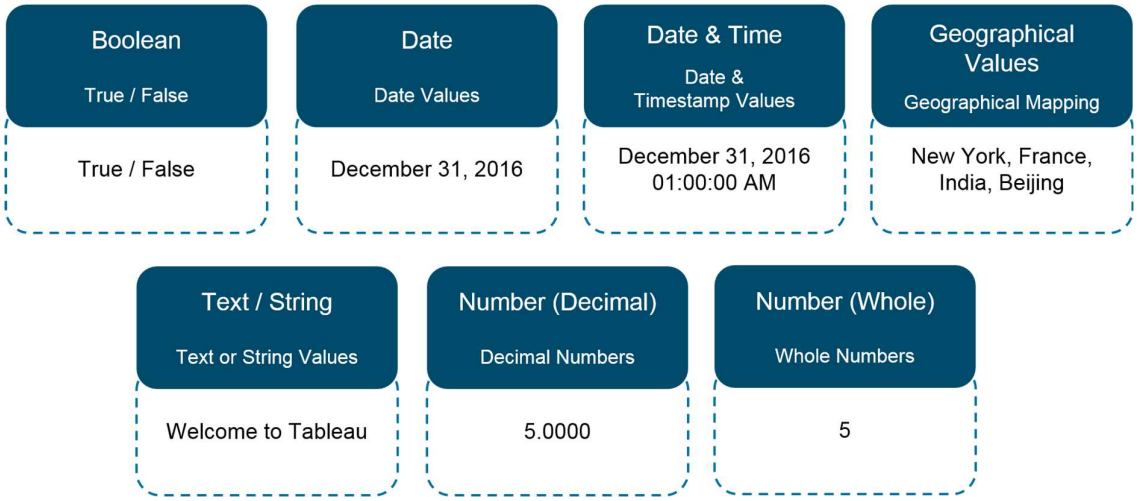
## What are Measures and Dimensions?

**Measures** are the numeric metrics or measurable quantities of the data, which can be analyzed by dimension table. Measures are stored in a table that contain foreign keys referring uniquely to the associated dimension tables. The table supports data storage at atomic level and thus, allows more number of records to be inserted at one time. For instance, a Sales table can have product key, customer key, promotion key, items sold, referring to a specific event.

**Dimensions** are the descriptive attribute values for multiple dimensions of each attribute, defining multiple characteristics. A dimension table ,having reference of a product key form the table, can consist of product name, product type, size, color, description, etc.

## What are the different datatypes in Tableau?

Tableau supports the following data-types:




## What are the different connections you can make with your dataset?

We can either connect live to our data set or extract data onto Tableau.

**Live:** Connecting live to a data set leverages its computational processing and storage. New queries will go to the database and will be reflected as new or updated within the data.

**Extract:** An extract will make a static snapshot of the data to be used by Tableau's data engine. The snapshot of the data can be refreshed on a recurring schedule as a whole or incrementally append data. One way to set up these schedules is via the Tableau server.

The benefit of Tableau extract over live connection is that extract can be used anywhere without any connection and you can build your own visualization without connecting to database.

## What is Tableau Data Server?

Tableau server acts a middle man between Tableau users and the data. Tableau Data Server allows you to upload and share data extracts, preserve database connections, as well as reuse calculations and field metadata. This means any changes you make to the data-set, calculated fields, parameters, aliases, or definitions, can be saved and shared with others, allowing for a secure, centrally managed and standardized dataset. Additionally, you can leverage your server's resources to run queries on extracts without having to first transfer them to your local machine.

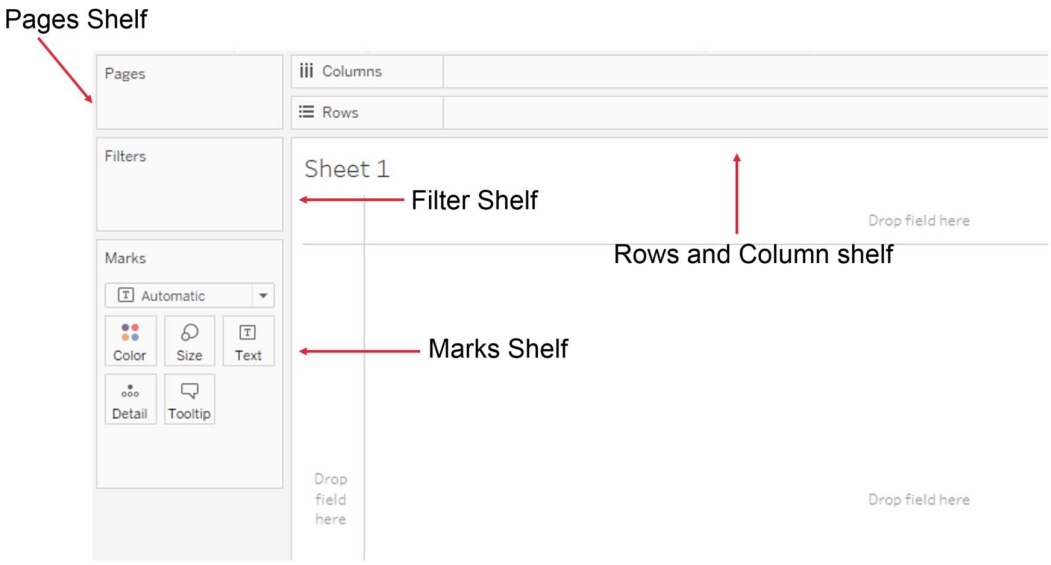
## What is Tableau Data Engine?

Tableau Data Engine is a really cool feature in Tableau. It's an analytical database designed to achieve instant query response, predictive performance, integrate seamlessly into existing data infrastructure and is not limited to load entire data sets into memory.

If you work with a large amount of data, it does takes some time to import, create indexes and sort data but after that everything speeds up. Tableau Data Engine is not really in-memory technology. The data is stored in disk after it is imported and the RAM is hardly utilized.

## What are shelves?

They are Named areas to the left and top of the view. You build views by placing fields onto the shelves. Some shelves are available only when you select certain mark types.




## Why use a hierarchical field in tableau?

A hierarchical field in tableau helps you to drill down data. It allows you to view your data in a more granular level.

## Define the term analytics pane concerning Tableau

The analytics pane offers quick and easy access to everyday analytic objects in Tableau. It allows you to drag forecasts, reference and trend lines, and other objects into your view from the Analytics pane.

## What are sets?

**Sets** are custom fields that define a subset of data based on some conditions. A **set** can be based on a computed condition, for example, a **set** may contain customers with sales over a certain threshold. Computed **sets** update as your data changes. Alternatively, a **set** can be based on specific data point in your view.

## What are groups?

A group is a combination of dimension members that make higher level categories. For example, if you are working with a view that shows average test scores by major, you may want to group certain majors together to create major categories.

## How to use group in calculated field?

By adding the same calculation to 'Group By' clause in SQL query or creating a Calculated Field in the Data Window and using that field whenever you want to group the fields.

- **Using groups in a calculation.** You cannot reference ad-hoc groups in a calculation.
- **Blend data using groups created in the secondary data source:** Only calculated groups can be used in data blending if the group was created in the secondary data source.
- **Use a group in another workbook.** You can easily replicate a group in another workbook by copy and pasting a calculation.

## What are the properties of Tableau combined sets?

Properties of Tableau combined sets are:

**Name:** It is used to specify the unique name of a tableau set.

**Sets:** Users can select the existing set from the menu. The first set in the menu acts as a left set. The second set act as the right set.

**All members in both sets:** This is an option to combined set that holds all the members from left as well as right set.

**Shared members in both sets:** This option holds matching members from both left and right sets. It means every record must match the condition present in these sets.

**Left set except shared members:** This Tableau set is used to hold all the members from the left set except matching members from the right set.

**Right set except shared members:** It holds all the members from the right set by matching members from the left set.

## What are the different Join types in Tableau

In general, there are four types of joins that you can use in Tableau: inner, left, right, and full outer. If you aren't sure what join type you want to use to combine data from multiple tables, you should use relationships.

| **Join Type** | **Result** |
|---|---|
| Inner  | When you use an inner join to combine tables, the result is a table that contains values that have matches in both tables.<br><br>When a value doesn't match across both tables, it is dropped entirely. |
| Left  | When you use a left join to combine tables, the result is a table that contains all values from the left table and corresponding matches from the right table.<br><br>When a value in the left table doesn't have a corresponding match in the right table, you see a null value in the data grid. |

| | |
|---|---|
| Right  | When you use a right join to combine tables, the result is a table that contains all values from the right table and corresponding matches from the left table.<br><br>When a value in the right table doesn't have a corresponding match in the left table, you see a null value in the data grid. |
| Full outer  | When you use a full outer join to combine tables, the result is a table that contains all values from both tables.<br><br>When a value from either table doesn't have a match with the other table, you see a null value in the data grid. |
| *Union*  | Though union is not a type of join, union is another method for combining two or more tables by appending rows of data from one table to another. Ideally, the tables that you union have the same number of fields, and those fields have matching names and data types. For more information about union, see Union Your Data. |

Not all databases support all join types. If an option is unavailable in the join dialog, it is likely due to a constraint from your data source.

## What is the difference between Relationships and Joins

The default method in Tableau Desktop is to use relationships. Relationships preserve the original tables' level of detail when combining information. Relationships also allow for context-based joins to be performed on a sheet-by-sheet basis, making each data source more flexible. Relationships are the recommended method of combining data in most instances.

However, there may be times when you want to directly establish a join, either for control or for desired aspects of a join compared to a relationship, such as deliberate filtering or duplication.

**Note**: Relationships eventually leverage joins (just behind the scenes). For example, a relationship across data sources will produce a cross-database join when the viz uses fields from tables in different data sources. As such, Improve Performance for Cross-Database Joins may be relevant.

Common issues

- To view, edit, or create joins, you must open a logical table in the relationship canvas—the area you see when you first open or create a data source—and access the join canvas.
- **Published Tableau data sources cannot be used in joins.** To combine published data sources, you must edit the original data sources to natively contain the join or use a data blend.
- When joining tables, the fields that you join on must be the same data type. If you change the data type after you join the tables, the join will break.
- Fields used in the join clause cannot be removed without breaking the join. To join data and be able to clean up duplicate fields, use Tableau Prep Builder instead of Desktop

**Tip**: While Tableau Desktop has the capability to create joins and do some basic data shaping, Tableau Prep Builder is designed for data preparation. If you need to do multiple joins, clean up field names, change data types, perform multiple pivots, or other sorts of involved data prep, consider using Tableau Prep Builder

## How Relationships Differ from Joins.

Relationships are a dynamic, flexible way to combine data from multiple tables for analysis. You don't define join types for relationships, so you won't see a Venn diagram when you create them.

Think of a relationship as a contract between two tables. When you are building a viz with fields from these tables, Tableau brings in data from these tables using that contract to build a query with the appropriate joins.

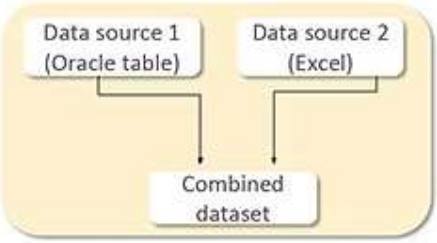
- **No up-front join type**. You only need to select matching fields to define a relationship (no join types). Tableau first attempts to create the relationship based on existing key constraints and matching field names. You can then check to ensure they are the fields you want to use, or add more field pairs to better define how the tables should be related.
- **Automatic and context-aware**. Relationships defer joins to the time and context of analysis. Tableau automatically selects join types based on the fields being used in the visualization. During analysis, Tableau adjusts join types intelligently and preserves the native level of detail in your data. You can see aggregations at the level of detail of the fields in your viz rather than having to think about the underlying joins. You don't need to use LOD expressions such as FIXED to deduplicate data in related tables.
- **Flexible**. Relationships can be many-to-many and support full outer joins. When you combine tables using relationships, it's like creating a custom, flexible data source for every viz, all in a single data source for the workbook. Because Tableau queries only tables that are needed based on fields and filters in a viz, you can build a data source that can be used for a variety of analytic flows.

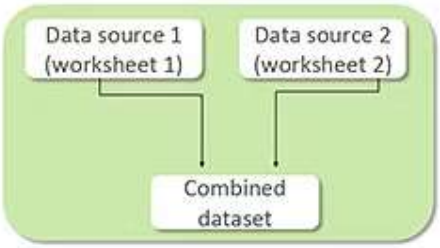
## Explain the primary differences between blending and joining in Tableau?

Joining terms is helpful when you are combining data from the same source. On the other hand, blending would require two completely defined data sources in your report.

## Explain when would you use Joins vs. Blending in Tableau?

If data resides in a single source, it is always desirable to use Joins. When your data is not in one place blending is the most viable way to create a left join like the connection between your primary and secondary data sources.




## What is default Data Blending Join?

Data blending is the ability to bring data from multiple data sources into one Tableau view, without the need for any special coding. A default blend is equivalent to a left outer join. However, by switching which data source is primary, or by filtering nulls, it is possible to emulate left, right and inner joins.

## Explain Blending

Data blending mixes the data from different data sources and allows users to perform the analysis in a single sheet. Blending means mixing, and when one is mixing the data sources, then it is called data blending.

**Rules to Perform Data Blending**

**The following are the rules of performing data blending:**

- If one is performing data blending on two data sources, then the two data sources should have at least one common dimension.
- In that common dimension, at least one value should be matching.

**In Tableau, one can perform data blending in two ways:**

- **Automatic Way:** Here, Tableau automatically defines the relationship between the two data sources based on the common dimensions and based on the matching values, and the relationship is indicated in orange.
- **Custom or Manual Way:** In the custom way, one needs to define the relationship manually.

**Data Blending Functionality**

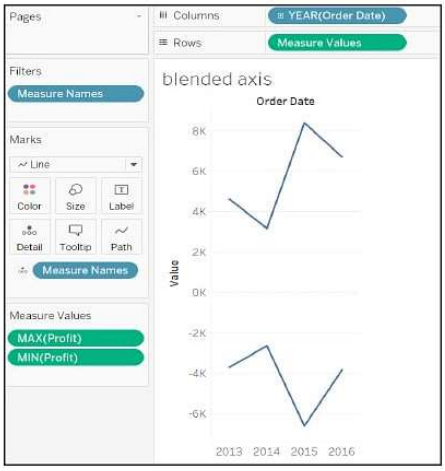
- All the primary and secondary data sources are linked by a specific relationship.
- While performing data blending, each worksheet has a primary connection, and optionally it might contain several secondary connections.
- All the primary connections are indicated in blue in the worksheet and all the secondary connections are indicated with an orange-colored tick mark.
- One sheet contains one primary data source and can contain n number of secondary data sources.

## What do you understand by blended axis?

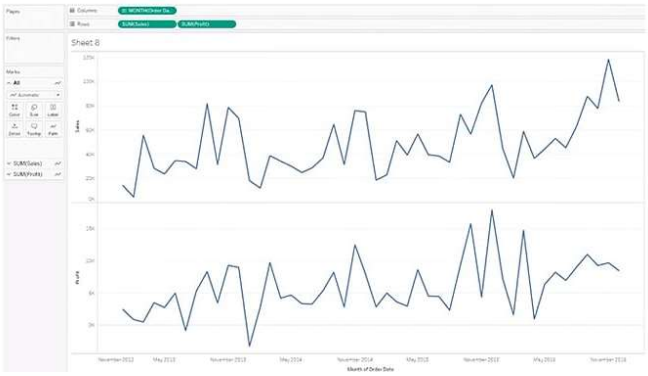
In Tableau, measures can share a single axis so that all the marks are shown in a single pane. Instead of adding rows and columns to the view, when you blend measures there is a single row or column and all of the values for each measure is shown along one continuous axis. We can blend multiple measures by simply dragging one measure or axis and dropping it onto an existing axis.

Blended Axis is used to blend two measures that share an axis when they have the same scale.

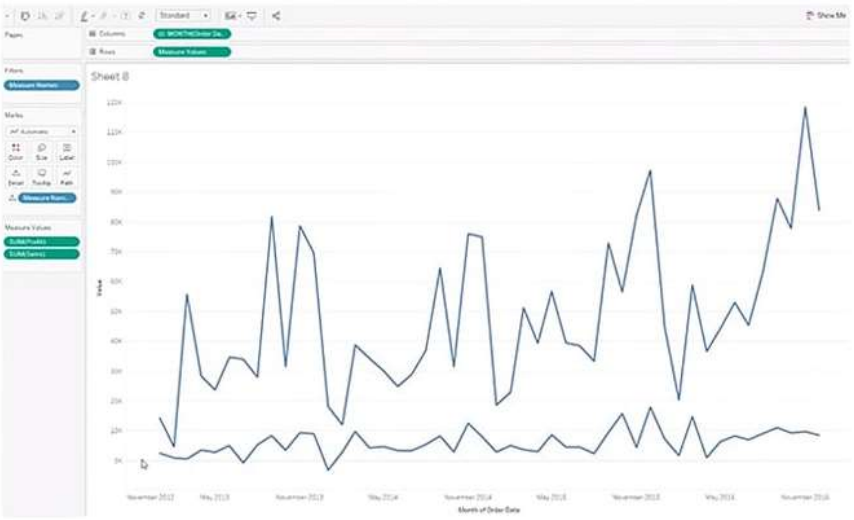
**Scenario:** Show Min and Max profit in the same pane and have a unified axis for both, so that it is quicker and easier to interpret the chart.




First, create a visualization that shows sales over time. Next, see profit along with sales over the same time. Here, you get two visualizations, one for sales over time and the other for-profit over time.



To see a visualization that has a blended axis for sales over time and profit over time, we bring in Measure Values and select the properties that we want to keep (Sales and Profit), removing all of the rest. You can now see profit and sales over one blended axis.




## How many maximum numbers of tables can be joined in Tableau?

The maximum number of tables that can be joined in Tableau is 32.

## What are the different filters in Tableau and how are they different from each other?

In Tableau, filters are used to restrict the data from database.

The different filters in Tableau are: Quick , Context and Normal/Traditional filter are:

- ***Normal Filter*** is used to restrict the data from database based on selected dimension or measure. A Traditional Filter can be created by simply dragging a field onto the 'Filters' shelf.
- ***Quick filter*** is used to view the filtering options and filter each worksheet on a dashboard while changing the values dynamically (within the range defined) during the run time.
- ***Context Filter*** is used to filter the data that is transferred to each individual worksheet. When a worksheet queries the data source, it creates a temporary, flat table that is uses to compute the chart. This temporary table includes all values that are not filtered out by either the Custom SQL or the Context Filter.

## Explain the context filter

A Context filter is an independent filter that can create a separate dataset out of the original data set and compute the selections made in the worksheet. One or more categorical filter that separates the dataset into major parts can be used as a context filter. All other filters used in the worksheet works based on the selection of context filter. The functions of context filters can be explained through an excel sheet.

## What are the advantages of Using Context Filters?

The advantages of Using Context Filters

- **Improve Performance:** When context filter is used in large data sources, it can improve the performance as it creates a temporary dataset part based on the context filter selection. The performance can be effectively improved through the selection of major categorical context filters.
- **Dependent Filter Conditions:** Context filters can be used to create dependent filter conditions based on the business requirement. When the data source size is large, context filters can be selected on the primary category, and other relevant filters can be executed.

## How is the Context Filter different from the other filters?

- Whenever one creates a Context Filter, Tableau will create a temporary table for that particular filter set, and the other filters will be applied on the Context Filter data such as cascade parameters.
- Suppose, one has created a Context Filter on USA and India, Tableau will create a temporary table for these two countries' data. If one does not have the Context Filter and has other filters, then they will be applied on these two countries' data; each record will check for all filters.

## What are the disadvantages of the Context Filter?

- The Context Filter is not frequently changed by the user—if it is changed, then the database must be recomputed and the temporary table must be rewritten.
- When one sets a dimension to context, Tableau creates a temporary table that requires a reload each time the view is initiated. For Excel, Access, and text data sources, the temporary table created is in an Access table format. For SQL Server, MySQL, and Oracle data sources, one must have permission to create a temporary table on their server. For a multidimensional data source or cubes, temporary tables are not created. The Context Filter defines which filters are independent and which are dependent.

## What is Dimension Filters?

When a dimension is used to filter the data in a worksheet, it is called a Dimension filter. It is a non-aggregated filter where a dimension, group, sets, and the bin can be added. A dimension filter can be applied through the top or bottom conditions, wildcard match, and formula.

## What is Dimension Filters?

A measure filter can filter the data based on the values present in a measure. The aggregated measure values can be used in measure filters to modify the data.

## Explain Measure filter

A measure filter can filter the data based on the values present in a measure. The aggregated values can be used in measure filters to modify the data.

## What is the user filter?

User filter secures the row-level data present in a dataset. It can be used when publishing the workbook on a server. Different filter conditions can be applied to different users.

## Where can a developer use global filters?

A developer can use global filters in sheets, dashboards, and stories.

## What are the various ways to use parameters in Tableau?

Various ways to use parameters in Tableau are: 1) filters, 2) calculated fields, 3) actions, 4) measure-swaps, 5) changing views, and 6) auto-updates

## What is a parameter in Tableau?

Parameters in Tableau are dynamic values that you can replace as constant values in calculations. These values serve as context filters.

The parameters in Tableau are the workbook variables like a number, date, or calculated field that allows users to replace a constant value in a calculation, filter, or reference line.

For example, the user can create a new calculated field that returns True if the aggregate of total marks is greater than 90% and returns False if it is less than 90%. Users can replace the constant value of "90%" in the formula with the parameters in Tableau as per the requirements. With the parameter control, users can dynamically vary the threshold values in their calculation.

## Distinguish between parameters and filters

- Parameters are dynamic values that can replace constant values in calculations. Parameters can serve as filters as well.
- Filters, on the other hand, are used to restrict the data based on a condition that is mentioned in the filters shelf.

## How to Create Parameters in Tableau?

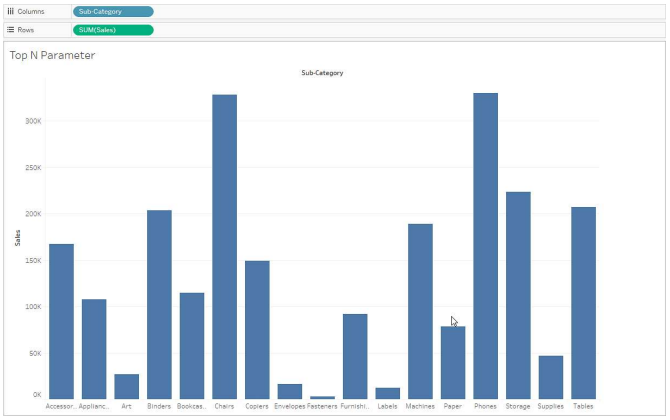
There are multiple possibilities of creating the Parameters in Tableau based on the user's requirements. In this practical section, you will see the most commonly used Parameters. Here, this tutorial will use the Sample Superstore Dataset available in Tableau Public.

- Top N Parameters in Tableau
- Date Field Parameters in Tableau
- Dynamic Measures
- Dynamic Dimensions

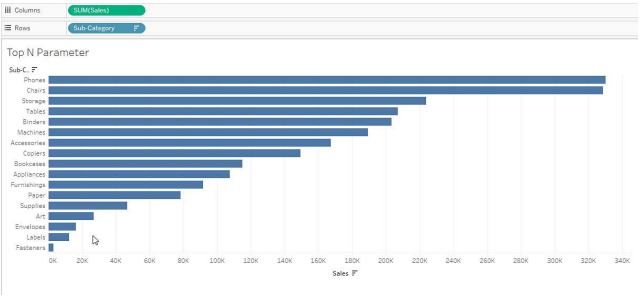
**Top N Parameters in Tableau**

The following are the steps to create Top N Parameters in Tableau.

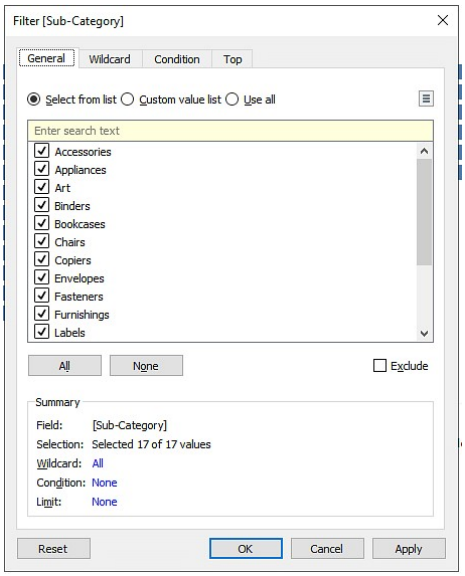
- In this Parameters demo, you will find the Top N Sub-Categories giving maximum sales.
- Start Tableau and import the sample superstore dataset.
- Please create a new sheet and rename it as Top N Parameters, for your reference.
- Now, select sales from the measures tab and drag it to rows.
- Select sub-category from dimensions and drag it to columns.
- Tableau will automatically generate a bar chart as shown below.



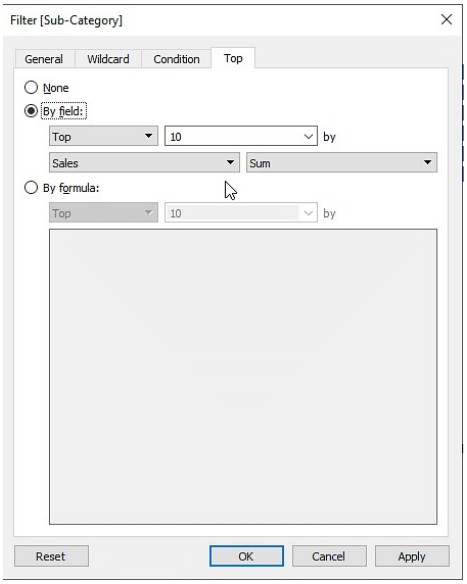
- The orientation can be changed if necessary.
- The next step is to organize the chart in descending order of the sum of sales.



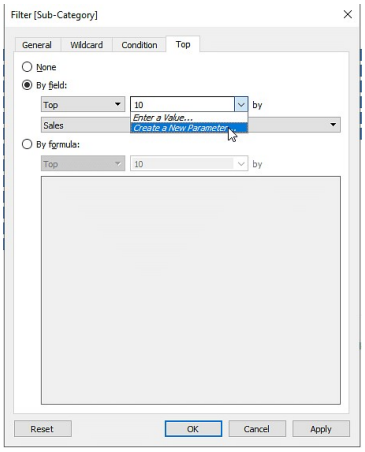

- To find the Top N Sub-Categories giving the maximum sales, drag the sub-category pill from dimensions onto the filter in the marks-sheet.



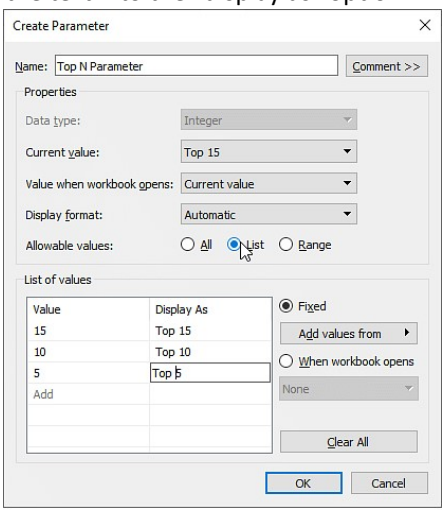

- A new "filter" dialogue box will pop-up on the screen.
- Select the top option in the menu bar.



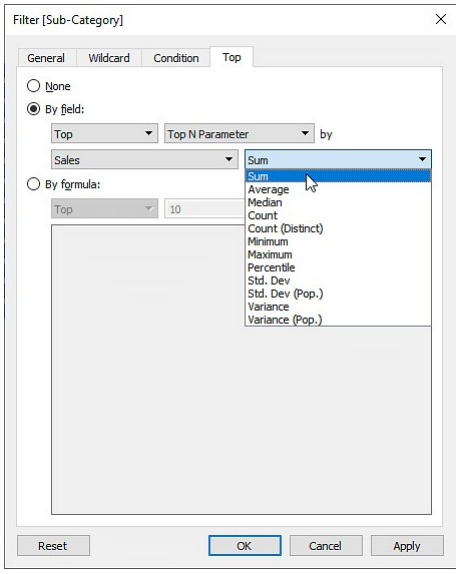

- Select the field option
- Select the new parameter option



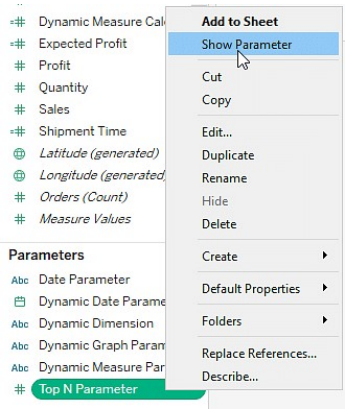

- Rename the parameter
- Select the list option
- Write in the values and type in the text into the "display as" option.



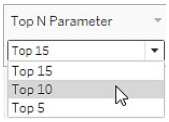

- Make sure the aggregation is SUM.
- Select the OK option.



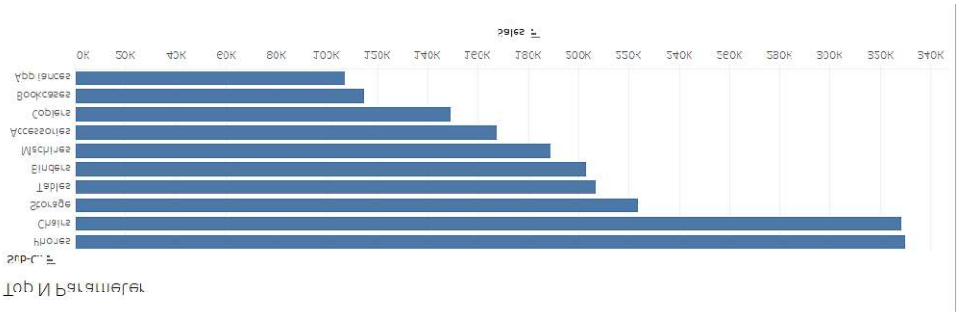

- Under the measures template, you can find the newly created parameter.
- Select the newly created parameter and right-click the pill.
- Select the show parameter option and select the top 10 option.




- The Tableau will provide the Top N Parameter list on the screen.




- Select Top 10 and Tableau will present the output.

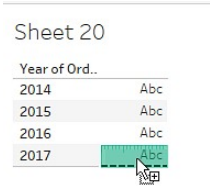


**Date Field Parameters in Tableau**

In the date parameters, you will find out the sales according to the dates. It could be in terms of a year, quarter, month, week, or day. The "Parameter options" will be available in the parameter.

Now, the following steps are needed to be followed to create a date parameter.

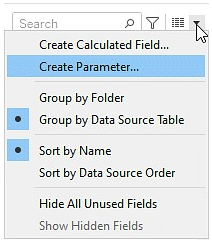
- Please create a new sheet and rename it as Date Parameter.
- Drag order date to rows and sales to the table as shown below.



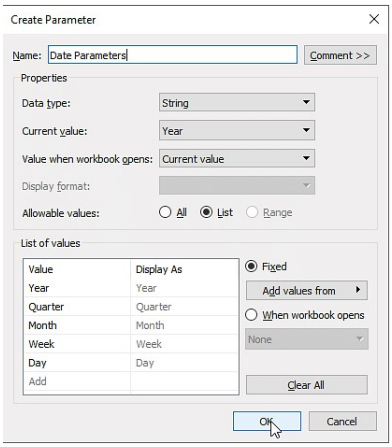

- The resultant text table will as follows.

| Year of Ord.. | |
|---|---|
| 2014 | 49,544 |
| 2015 | 61,619 |
| 2016 | 81,795 |
| 2017 | 93,439 |

- To create a new parameter, select the arrow icon in the top left corner and select the create parameter option as shown below.

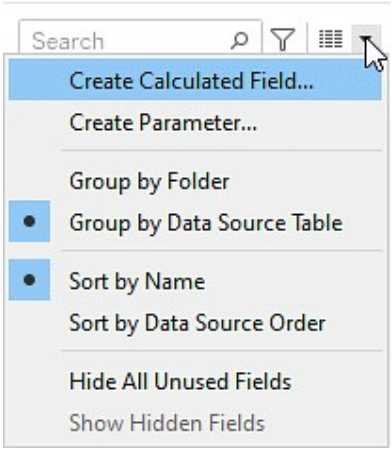



- The new parameter dialogue box will appear on the screen.
- Rename the parameter as date parameters
- Select the datatype as string.
- Select the list option
- Add the values and "display as" details and select OK as shown below.




According to the year, quarter, month, week, and day, you need a new calculated field to display the sales.

- Create a new calculated field named date calculation.
- Go to the arrow icon option in the top left corner.
- Select the arrow icon and click on the "create a calculated field" option, as shown below.




- A new window will appear on your screen. Here you can add your queries, as shown below.
- Write the following query in the date calculation window and select OK as shown below.

//Code

CASE [Date Parameter ]

WHEN "Year" THEN STR(YEAR([Order Date]))

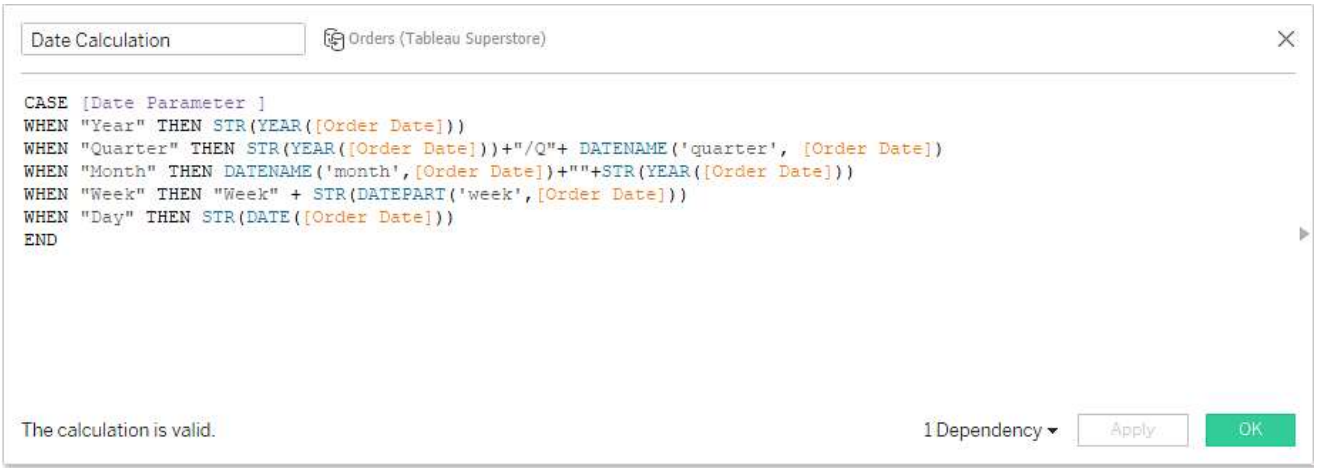
WHEN "Quarter" THEN STR(YEAR([Order Date]))+"/Q"+ DATENAME('quarter', [Order Date])

WHEN "Month" THEN DATENAME('month',[Order Date])+""+STR(YEAR([Order Date]))

WHEN "Week" THEN "Week" + STR(DATEPART('week',[Order Date]))

WHEN "Day" THEN STR(DATE([Order Date]))

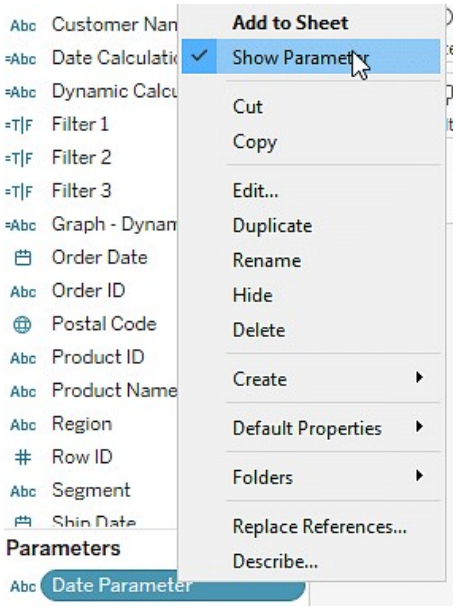
END



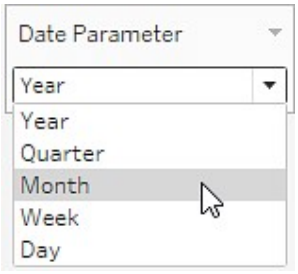

- Now, you have the date parameter in the parameters section and date calculation in the dimensions section.
- Next, change the text table to a bar graph. Select the 'Show me' tab and choose the option of a bar graph in the list.


- The text table will not get converted into a bar graph.
- Select the marks sheet's label option to give you the details related to sales in a readable format.

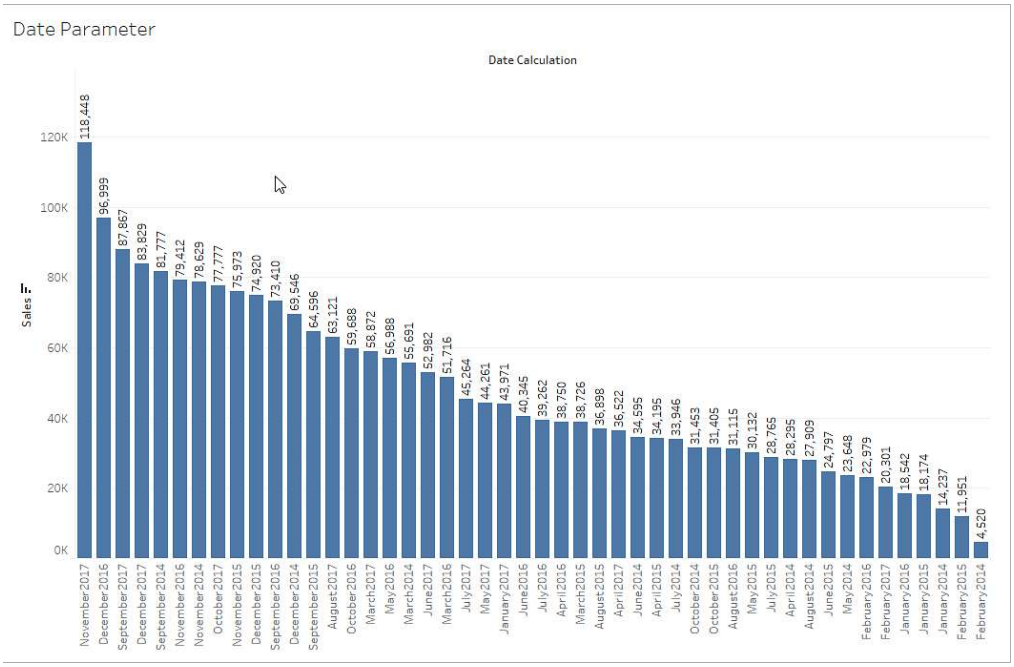
To visualize the sales in terms of the year, quarter, month, week, or day, you need to use the Date Parameters. Select the Date Parameters option from the parameters panel and right-click the pill, and select the show parameter option as shown below.




- The parameters option will be shown as follows.




Moving on, select the month option, and Tableau will provide you with sales information per month.

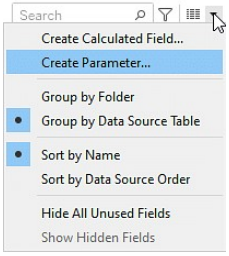



Advancing, we will look into Dynamic measures and Dynamic Dimensions.

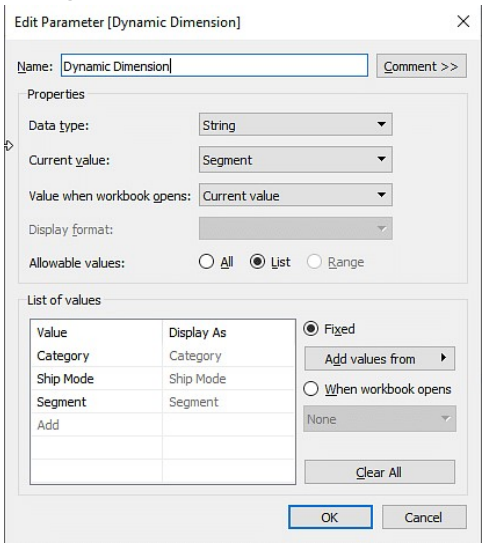
**Dynamic Dimensions**

To create the Dynamic Dimensions parameter, you need to execute the following steps.

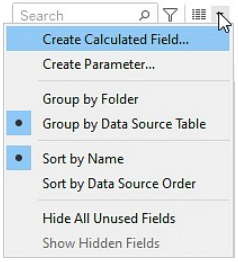
- Please create a new sheet and rename it as Dynamic Dimensions.
- Create a new parameter. Select the arrow icon and click on the new parameter option.




- Now, rename the parameter as Dynamic Dimension Parameter.
- Select the datatype as a string.
- Select category, ship-mode, and segment dimensions from the dimensions panel.

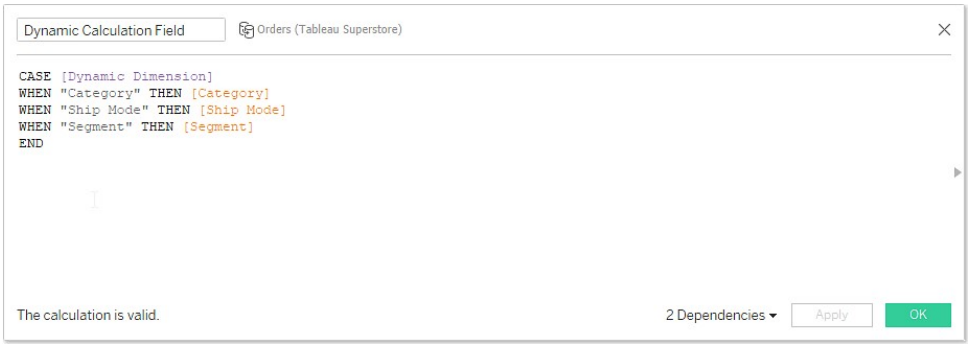



- Select the OK option and now, create a new calculated field.
- Hover over to the arrow icon and select "create a calculated field."



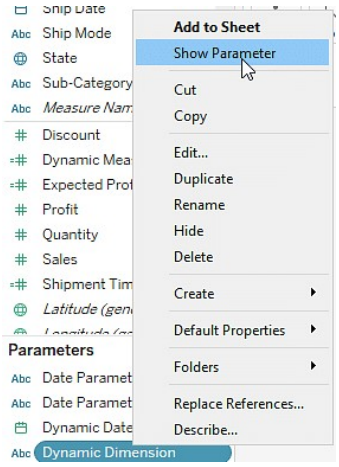

- Write the following code in the calculated field window and select OK as shown below.

```
CASE [Dynamic Dimension]

WHEN "Category" THEN [Category]

WHEN "Ship Mode" THEN [Ship Mode]

WHEN "Segment" THEN [Segment]

END
```

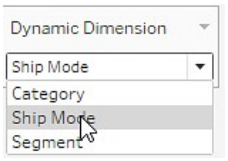



- Now, you can find the newly created dynamic dimension parameter and dynamic dimension calculated fields in the left panel.
- Drag the Dynamic Calculation field from dimensions panel to columns
- Drag category to columns
- Drag sales to rows


- Now right click on the Dynamic Dimensions parameter pill and select the show parameter option.



Now, take a look at the Parameter menu. The output is now capable of visualizing dynamic dimension data based on dimensions like Category, Ship-mode, and Segment, as shown in the following output image.





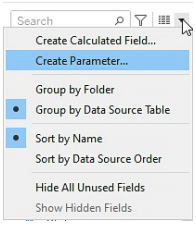
Up next, it's time to learn about Dynamic Measures.

**Dynamic Measures**

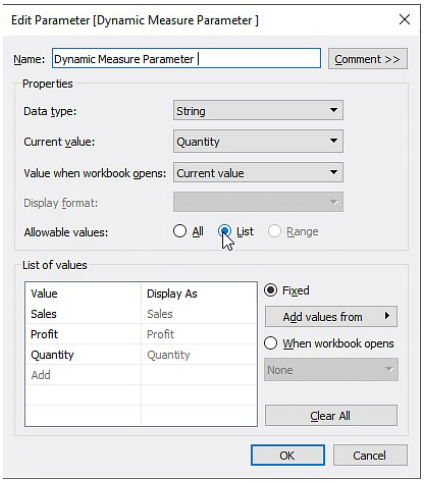
Now, Dynamic Measures are completely similar to Dynamic Dimensions, with only one difference. Instead of Dimension values, we use Measure values.

To create a Dynamic Measures Parameter, you need to follow the steps as explained below.

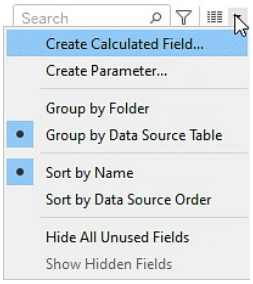
- Please create a new sheet and rename it as Dynamic Measures Parameter for reference.
- Create a new parameter. Select the arrow icon click on the new parameter option.




- Next up, rename the parameter as Dynamic measures parameter.
- Select the data type as a string.
- Moving on, select the list option and write the sales, profit, and quantity values as shown below.

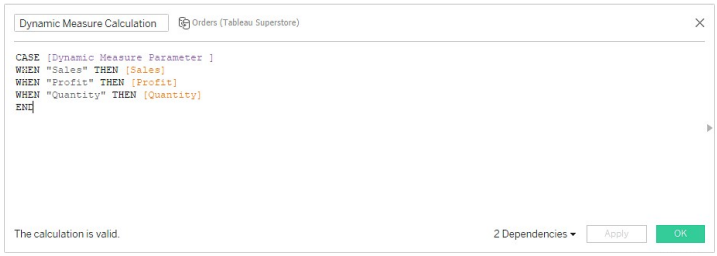


- Now you will find a newly created pill in the parameters panel.
- The next step is to create a new calculated field.
- Select the arrow icon, click on the create new calculated field option.

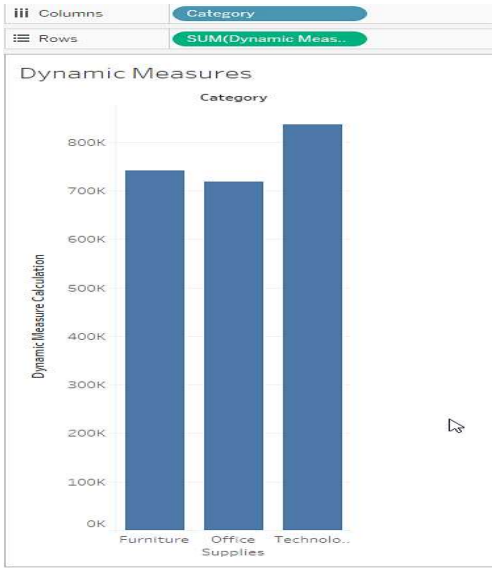



- A new calculated field window will appear on the screen.
- Type the following formula and select OK to create a new calculated field.

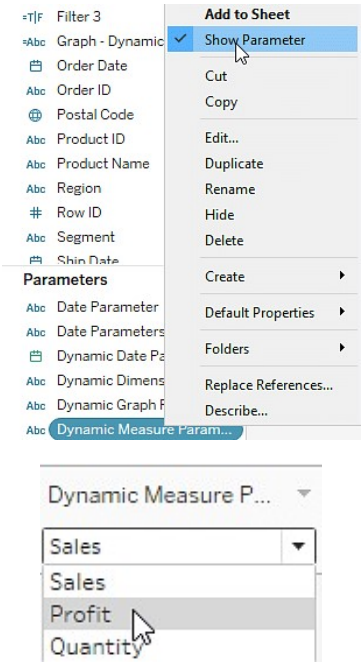
```
CASE [Dynamic Measure Parameter ]

WHEN "Sales" THEN [Sales]

WHEN "Profit" THEN [Profit]

WHEN "Quantity" THEN [Quantity]

END
```



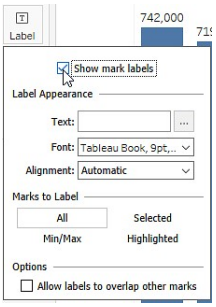

- Now, you can find a newly created calculated field in the measures panel.
- Drag the newly created calculated field in the measures panel to rows.
- Drag category from dimensions to columns.
- Tableau will automatically generate a bar chart.



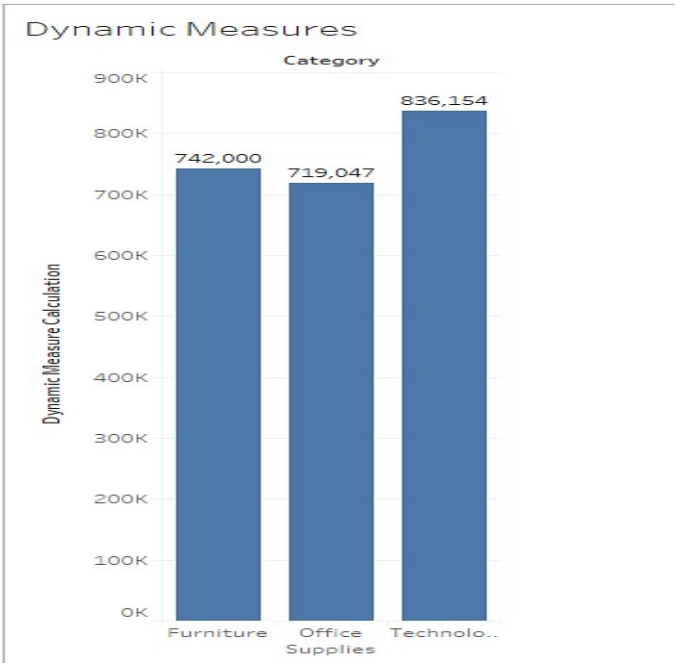

- Now right-click on the newly created parameter and select the show parameter option.




- Now, select the label icon from the marks card and tick the label box.




- The final result will look as follows.



## Parameters in Tableau vs. Filters in Tableau

| Tableau Parameters | Tableau Filters |
|---|---|
| Parameters are preferred for Static Data | Filters are used for Dynamic Data |
| Parameters are implemented to the entire workbook | Filters can be implemented to specific sheets only |
| Multiple values cannot be selected while implementing Parameters | Multiple values can be selected while implementing Filters |
| Parameters are implemented using Calculated fields | Filters do not use calculated fields |
| Parameters support multiple datatypes | Fields cannot support multiple datatypes |

## How to remove 'All' options from a Tableau auto-filter?

The auto-filter provides a feature of removing 'All' options by simply clicking the down arrow in the auto-filter heading. You can scroll down to 'Customize' in the dropdown and then uncheck the 'Show "All" Value' attribute. It can be activated by checking the field again.

## How to view underlying SQL Queries in Tableau?

Viewing underlying SQL Queries in Tableau provides two options:

- Create a Performance Recording to record performance information about the main events you interact with workbook. Users can view the performance metrics in a workbook created by Tableau.
    - Help -> Settings and Performance -> Start Performance Recording
    - Help -> Setting and Performance -> Stop Performance Recording.
- Reviewing the Tableau Desktop Logs located at C:Users/My Documents/My Tableau Repository. For live connection to data source, you can check log.txt and tabprotosrv.txt files. For an extract, check tdeserver.txt file.

## What are the ways to sort out data in Tableau?

The ways to sort out data in Tableau are:

- Computed sorting: It is a sort that can be applied on an axis using a sort button.
- Manual sorting: It can be used to rearrange the dimension field order by dragging them to each other in an ad hoc manner.

## How to do Performance Testing in Tableau?

Performance testing is again an important part of implementing tableau. This can be done by loading Testing Tableau Server with TabJolt, which is a "Point and Run" load generator created to perform QA. While TabJolt is not supported by tableau directly, it has to be installed using other open source products.

## Mention whether you can create relational joins in Tableau without creating a new table?

Yes, one can create relational joins in tableau without creating a new table.

## What is story in Tableau?

A story is a sheet that contains a sequence of worksheets or dashboards that work together to convey information. You can create stories to show how facts are connected, provide context, demonstrate how decisions relate to outcomes, or simply make a compelling case. Each individual sheet in a story is called a story point.

## How to create stories in Tableau?

There are many ways to create story in Tableau. Each story point can be based on a different view or dashboard, or the entire story can be based on the same visualization, just seen at different stages, with different marks filtered and annotations added. You can use stories to make a business case or to simply narrate a sequence of events.

- Click the New Story tab.
- In the lower-left corner of the screen, choose a size for your story. Choose from one of the predefined sizes, or set a custom size, in pixels.
- By default, your story gets its title from its sheet name. To edit it, double-click the title. You can also change your title's font, color, and alignment. Click Apply to view your changes.
- To start building your story, drag a sheet from the Story tab on the left and drop it into the center of the view
- Click Add a caption to summarize the story point.
- To highlight a key takeaway for your viewers, drag a text object over to the story worksheet and type your comment.
- To further highlight the main idea of this story point, you can change a filter or sort on a field in the view, then save your changes by clicking Update above the navigator box.

## How to embed views onto Webpages?

You can embed interactive Tableau views and dashboards into web pages, blogs, wiki pages, web applications, and intranet portals. Embedded views update as the underlying data changes, or as their workbooks are updated on Tableau Server. Embedded views follow the same licensing and permission restrictions used on Tableau Server. That is, to see a Tableau view that's embedded in a web page, the person accessing the view must also have an account on Tableau Server.

Alternatively, if your organization uses a core-based license on Tableau Server, a Guest account is available. This allows people in your organization to view and interact with Tableau views embedded in web pages without having to sign in to the server. Contact your server or site administrator to find out if the Guest user is enabled for the site you publish to.

You can do the following to embed views and adjust their default appearance:

- Get the embed code provided with a view: The Share button at the top of each view includes embed code that you can copy and paste into your webpage. (The Share button doesn't appear in embedded views if you change the showShareOptions parameter to false in the code.)
- Customize the embed code: You can customize the embed code using parameters that control the toolbar, tabs, and more. For more information, see Parameters for Embed Code.
- Use the Tableau JavaScript API: Web developers can use Tableau JavaScript objects in web applications. To get access to the API, documentation, code examples, and the Tableau developer community, see the Tableau Developer Portal.

## Think that I am using Tableau Desktop & have a live connection to Cloudera Hadoop data. I need to press F5 to refresh the visualization. Is there anyway to automatically refresh visualization every 'x' seconds instead of pressing F5?

Here is an example of refreshing the dashboard for every 5 seconds.

All you need to do is replace the api src and server url with yours.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Tableau JavaScript API </title>
<script type="text/javascript" src="http://servername/javascripts/api/tableau_v8.js"></script>
</head>
<div id="tableau Viz"></div>
<script type='text/javascript'>
var placeholderDiv = document.getElementById("tableau Viz");
var url = "http://servername/t/311/views/Mayorscreenv5/Mayorscreenv2";
var options={
hideTabs:True,
width:"100%",
height:"1000px"
};
var viz= new tableauSoftware.Viz(placeholderDiv,url,options);
setInterval (function() {viz.refreshDataAsync()},5000);
</script>
</body>
</html>
```

## What is the use of trend lines?

Trend lines are used to know the continuation of a trend of variables. It helps users to search the correlation between two or more variables. There is a wide range of mathematical models for establishing trend lines. These models are 1) Logarithmic, 2) Linear, 3) Exponential, and 4 ) Polynomial.

## Explain alias in Tableau

Alias in Tableau can refer as an alternative name that the user can assign to a dimension member a field.

## Can you get values from two different sources as a single input into parameters?

Tableau currently does not support multi-valued parameters.

**Case Study:** The dynamic-parameter-with-a-blend technique can be used to highlight a single value, but not multiple values because of the way it works. As Tableau parameters are not dynamic, one cannot filter the list of values at runtime.

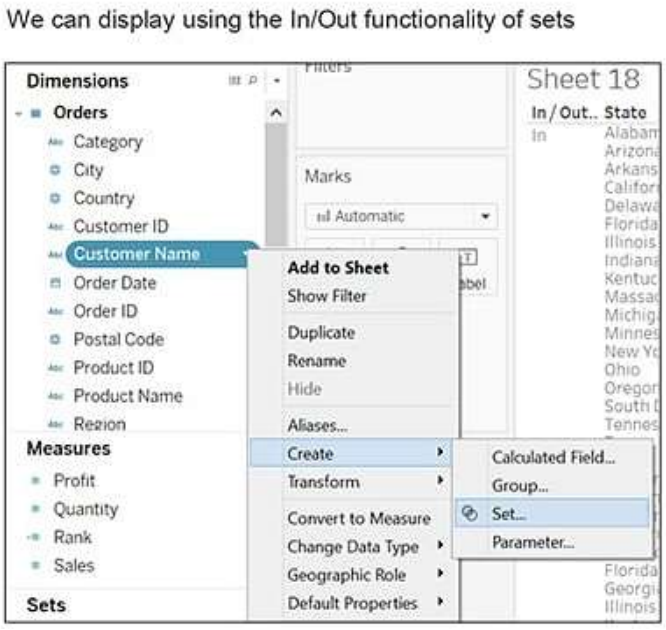
## How to create cascading filters without using the Context Filter?

- Suppose one has Filter1 and Filter2. Based on Filter1, one needs to use Filter2 on the data. Consider Filter1 as “Country” and Filter2 as “States”.
- Consider “India” and the “Country”; so, Filter2 should display only the states of India.
- Choose options of Filter2 states:
- Select the option of “Only relevant values”.

## How to display the top five and the last five sales in the same view?

The top five and the last five sales can be displayed in the same view by using filters or calculated fields.

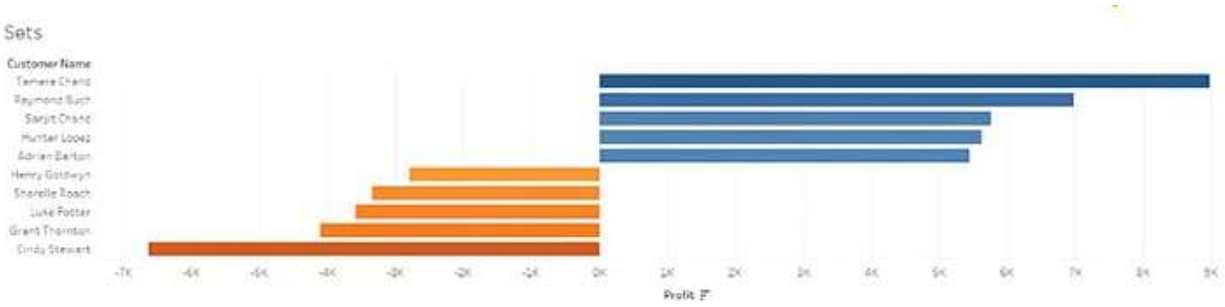
We can display it using the In/Out functionality of sets.




Follow these steps:

1. Drag the Customer Name field to Rows shelf and Profit field to Columns shelf to get the visualization.
2. Create a set by right-clicking on the Customer Name field. Choose to create an option and click on Set.
3. Provide the name ‘Top Customers’ to the set. Configure the set by clicking on Top tab, selecting By field, and filling the values as Top, 5, Profit, and Sum.
4. Similarly, create a second set called ‘Bottom Customers’ and fill the By Field values as Bottom, 5, Profit, and Sum.
5. Select these two sets and right-click on it. Use the option Create Combined Set. Name it ‘Top and Bottom Customers’ and include all members of both sets. Pull the Top and Bottom Customers onto Filters.

The top five and bottom five are displayed:



## How to view an SQL generated by Tableau Desktop?

Tableau Desktop log files are located in C:Users/MyDocuments/My Tableau Repository. If one has a live connection to the data source, then they need to check the log.txt and tabprotosrv.txt files. If one is using Extract, then they have to check the tdeserver.txt file. The tabprotosrv.txt file often shows detailed information about queries.

## How to rectify SQL performance for developed dashboards?

After the creation of dashboards in Tableau, if there is a problem from the SQL side, it means Custom SQL. Rather than using custom SQL connections in Tableau, it is better to use SQL statements creating a view inside of the database, then connecting that to Tableau. This will streamline the efforts without hunkering them down with the rest of the SQL, which are generated by Tableau without the custom SQL subquery.

## Is There a Difference Between Sets and Groups in Tableau?

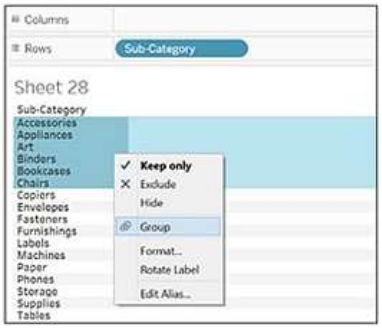
A Tableau group is one dimensional, used to create a higher level category by using lower-level category members. Tableau sets can have conditions and can be grouped across multiple dimensions/measures.

Example: Sub-category can be grouped by category.

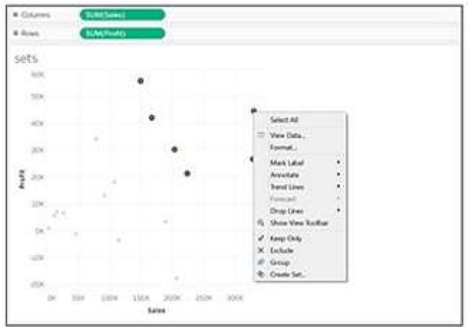
Top Sales and profit can be clubbed together for different categories by creating a set:

1. Continuing with the above example of Sets, select the Bottom Customers set where customer names are arranged based on profit.
2. Go to the 'Groups' tab and select the top five entries from the list.
3. Right-click and select create a group option.
4. Similarly, select the bottom five entries and create their group. Hide all the other entries.

Grouping




Sets




A key difference here is that the groups will consist of the same customers even if their profits change later. While for sets, if the profit changes, the top five and bottom five customers will change accordingly.

**Did You Know-** We can't use groups in calculated fields, but we can use sets.

## How Can You Optimize the Performance of a Dashboard?

There are multiple ways to optimize the performance of the dashboard like:

- Maximize the number of fields and records. You can exclude unused fields from your visualization or use extract filters.
- Limit the number of filters used, by avoiding quick filters and using action and parameter filters instead. These filters reduce query loads.
- use Min/Max instead of Average because average functions require more processing time than Min/Max
- Use boolean or numerical calculations more than string calculations. Computers can process integers and boolean much faster than strings.

**Boolean > int > float > date-time > string**

## Which Visualization Will Be Used in the given Scenarios?

1. To show aggregated sales totals across a range of product categories and subcategories
2. To show the duration of events or activities
3. To show quarter wise profit growth

We would use the following visualizations for the given scenarios:

1. Treemap
2. Gantt chart
3. Waterfall chart

# Number Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces number functions and their uses in Tableau. It also demonstrates how to create a number calculation using an example.

**Why use number functions**

Number functions allow you to perform computations on the data values in your fields. Number functions can only be used with fields that contain numerical values. For more information, see Data Types.

For example, you might have a field that contains values for the variance in your budget, titled Budget Variance. One of those values might be -7. You can use the ABS function to return the absolute value of that number, and all the other numbers in that field.

The calculation might look something like this:

ABS[Budget Variance]

Therefore, ABS(-7) = 7.

Number functions available in Tableau

| Function | Syntax | Description |
|---|---|---|
| **ABS** | ABS(number) | Returns the absolute value of the given number.<br>Examples:<br>ABS(-7) = 7<br>ABS([Budget Variance])<br>The second example returns the absolute value for all the numbers contained in the Budget Variance field. |
| **ACOS** | ACOS(number) | Returns the arc cosine of the given number. The result is in radians.<br>Example:<br>ACOS(-1) = 3.14159265358979 |
| **ASIN** | ASIN(number) | Returns the arc sine of a given number. The result is in radians.<br>Example:<br>ASIN(1) = 1.5707963267949 |
| **ATAN** | ATAN(number) | Returns the arc tangent of a given number. The result is in radians.<br>Example:<br>ATAN(180) = 1.5652408283942 |
| **ATAN2** | ATAN2(y number, x number) | Returns the arc tangent of two given numbers (x and y). The result is in radians.<br>Example:<br>ATAN2(2, 1) = 1.10714871779409 |

<table>
<tr>
<td>CEILING</td>
<td>CEILING (number)</td>
<td>Rounds a number to the nearest integer of equal or greater value.<br><br>Example:<br><br>CEILING (3.1415) = 4<br><br>Availability by data source:<br><br>
<table>
<tr><th>Data Source</th><th>Support</th></tr>
<tr><td>Microsoft Access</td><td>Not supported</td></tr>
<tr><td>Microsoft Excel</td><td>Supported</td></tr>
<tr><td>Text File</td><td>Supported</td></tr>
<tr><td>Statistical File</td><td>Supported</td></tr>
<tr><td>Tableau Server</td><td>Supported</td></tr>
<tr><td>Actian Vector</td><td>Not supported</td></tr>
<tr><td>Amazon Aurora for MySQL</td><td>Not supported</td></tr>
<tr><td>Amazon EMR Hadoop Hive</td><td>Supported</td></tr>
<tr><td>Amazon Redshift</td><td>Supported</td></tr>
<tr><td>Aster Database</td><td>Not supported</td></tr>
<tr><td>Cloudera Hadoop</td><td>Supported</td></tr>
<tr><td>DataStax Enterprise</td><td>Supported</td></tr>
<tr><td>EXASOL</td><td>Not supported</td></tr>
<tr><td>Firebird</td><td>Not supported</td></tr>
<tr><td>Google Analytics</td><td>Supported</td></tr>
<tr><td>Google BigQuery</td><td>Supported</td></tr>
<tr><td>Google Cloud SQL</td><td>Not supported</td></tr>
<tr><td>Google Sheets</td><td>Not supported</td></tr>
<tr><td>Hortonworks Hadoop Hive</td><td>Supported</td></tr>
<tr><td>IBM BigInsights</td><td>Not supported</td></tr>
<tr><td>IBM DB2</td><td>Not supported</td></tr>
<tr><td>IBM PDA (Netezza)</td><td>Not supported</td></tr>
<tr><td>MapR Hadoop Hive</td><td>Supported</td></tr>
<tr><td>MarkLogic</td><td>Not supported</td></tr>
</table>
</td>
</tr>
</table>

| | | |
|---|---|---|
| | | <table><tr><td>Microsoft Analysis Services</td><td>Not supported</td></tr><tr><td>Microsoft PowerPivot</td><td>Not supported</td></tr><tr><td>Microsoft SQL Server</td><td>Supported</td></tr><tr><td>MySQL</td><td>Not supported</td></tr><tr><td>Oracle</td><td>Not supported</td></tr><tr><td>Oracle Essbase</td><td>Not supported</td></tr><tr><td>Actian Matrix (ParAccel)</td><td>Not supported</td></tr><tr><td>Pivotal Greenplum</td><td>Not supported</td></tr><tr><td>PostgreSQL</td><td>Not supported</td></tr><tr><td>Progress OpenEdge</td><td>Not supported</td></tr><tr><td>Salesforce</td><td>Supported</td></tr><tr><td>SAP HANA</td><td>Not supported</td></tr><tr><td>SAP Sybase ASE</td><td>Not supported</td></tr><tr><td>SAP Sybase IQ</td><td>Not supported</td></tr><tr><td>Spark SQL</td><td>Supported</td></tr><tr><td>Splunk</td><td>Not supported</td></tr><tr><td>Teradata</td><td>Not supported</td></tr><tr><td>Teradata OLAP Connector</td><td>Not supported</td></tr><tr><td>Vertica</td><td>Not supported</td></tr></table> |
| **COS** | COS(number) | Returns the cosine of an angle. Specify the angle in radians.<br>Example:<br>COS(PI( ) /4) = 0.707106781186548 |
| **COT** | COT(number) | Returns the cotangent of an angle. Specify the angle in radians.<br>Example:<br>COT(PI( ) /4) = 1 |
| **DEGREES** | DEGREES(number) | Converts a given number in radians to degrees.<br>Example:<br>DEGREES (PI( )/4) = 45.0 |
| **DIV** | DIV(integer1, integer2) | Returns the integer part of a division operation, in which integer1 is divided by integer2.<br>Example: |

<table>
<tr><td></td><td></td><td>DIV(11,2) = 5</td></tr>
<tr><td>EXP</td><td>EXP(number)</td><td>Returns e raised to the power of the given number.<br><br>Examples:<br><br>EXP(2) = 7.389<br>EXP(-[Growth Rate]*[Time])</td></tr>
<tr><td>FLOOR</td><td>FLOOR(number)</td><td>Rounds a number to the nearest integer of equal or lesser value.<br><br>Example:<br><br>FLOOR(3.1415) = 3<br><br>Availability by data source:<br><br>
<table>
<tr><th>Data Source</th><th>Support</th></tr>
<tr><td>Microsoft Access</td><td>Not supported</td></tr>
<tr><td>Microsoft Excel</td><td>Supported</td></tr>
<tr><td>Text File</td><td>Supported</td></tr>
<tr><td>Statistical File</td><td>Supported</td></tr>
<tr><td>Tableau Server</td><td>Supported</td></tr>
<tr><td>Actian Vector</td><td>Not supported</td></tr>
<tr><td>Amazon Aurora for MySQL</td><td>Not supported</td></tr>
<tr><td>Amazon EMR Hadoop Hive</td><td>Supported</td></tr>
<tr><td>Amazon Redshift</td><td>Not supported</td></tr>
<tr><td>Aster Database</td><td>Not supported</td></tr>
<tr><td>Cloudera Hadoop</td><td>Supported</td></tr>
<tr><td>DataStax Enterprise</td><td>Supported</td></tr>
<tr><td>EXASOL</td><td>Not supported</td></tr>
<tr><td>Firebird</td><td>Not supported</td></tr>
<tr><td>Google Analytics</td><td>Supported</td></tr>
</table>
</td></tr>
</table>

| | |
|---|---|
| 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 | 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 |
| **HEXBINX** | 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.<br><br>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.<br><br>Example:<br><br>HEXBINX([Longitude], [Latitude]) | |
| **HEXBINY** | 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.<br><br>Example:<br><br>HEXBINY([Longitude], [Latitude]) | |
| **LN** | LN(number) | Returns the natural logarithm of a number. Returns Null if number is less than or equal to 0. | |
| **LOG** | LOG(number [, base]) | Returns the logarithm of a number for the given base. If the base value is omitted, base 10 is used. | |
| **MAX** | 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.<br><br>Examples:<br><br>MAX(4,7)<br>MAX(Sales,Profit)<br>MAX([First Name],[Last Name]) | |

| | | |
|---|---|---|
| **MIN** | 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.<br><br>Examples:<br><br>MIN(4,7)<br>MIN(Sales,Profit)<br>MIN([First Name],[Last Name]) |
| **PI** | PI( ) | Returns the numeric constant pi: 3.14159. |
| **POWER** | POWER(number, power) | Raises the number to the specified power.<br><br>Examples:<br><br>POWER(5,2) = 52 = 25<br>POWER(Temperature, 2)<br><br>You can also use the ^ symbol:<br><br>5^2 = POWER(5,2) = 25 |
| **RADIANS** | RADIANS(number) | Converts the given number from degrees to radians.<br><br>Example:<br><br>RADIANS(180) = 3.14159 |
| **ROUND** | 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.<br><br>Example:<br><br>This example rounds every Sales value to an integer:<br><br>ROUND(Sales)<br><br>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.<br><br>**Note**: because ROUND may run into issues due to the underlying floating point representation of numbers—such as 9.405 rounding to 9.40—it may be preferable to format the number to the desired number of decimal points rather than rounding. Formatting 9.405 to two decimal places will yield the expected 9.41. |
| **SIGN** | 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.<br><br>Example:<br><br>If the average of the profit field is negative, then<br><br>SIGN(AVG(Profit)) = -1 |
| **SIN** | SIN(number) | Returns the sine of an angle. Specify the angle in radians. |

| | | |
|---|---|---|
| | | Examples:<br><br>SIN(0) = 1.0<br>SIN(PI( )/4) = 0.707106781186548 |
| **SQRT** | SQRT(number) | Returns the square root of a number.<br><br>Example:<br><br>SQRT(25) = 5 |
| **SQUARE** | SQUARE(number) | Returns the square of a number.<br><br>Example:<br><br>SQUARE(5) = 25 |
| **TAN** | TAN(number) | Returns the tangent of an angle. Specify the angle in radians..<br><br>Example:<br><br>TAN(PI ( )/4) = 1.0 |
| **ZN** | ZN(expression) | Returns the expression if it is not null, otherwise returns zero. Use this function to use zero values instead of null values.<br><br>Example:<br><br>ZN([Profit]) = [Profit] |

# String Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces string functions and their uses in Tableau. It also demonstrates how to create a string calculation using an example.

Why use string functions

String functions allow you to manipulate string data (i.e. data made of text). Tableau uses the current International Components for Unicode (ICU) library when comparing strings. The way strings are sorted and compared is based both on language and locale, and it's possible for vizzes to change as the ICU is continuously updated for better language support.

For example, you might have a field that contains all of your customers' first and last names. One member might be: Jane Johnson. You can pull the last names from all your customers into a new field using a string function.

The calculation might look something like this:

SPLIT([Customer Name], ' ', 2)

Therefore, SPLIT('Jane Johnson' , ' ', 2) = 'Johnson'.

String functions available in Tableau:

| Function | Syntax | Definition |
|---|---|---|
| **ASCII** | ASCII(string) | Returns the ASCII code for the first character of string.<br>Example:<br>ASCII('A') = 65 |
| **CHAR** | CHAR(number) | Returns the character encoded by the ASCII code number.<br>Example:<br>CHAR(65) = 'A' |
| **CONTAINS** | CONTAINS(string, substring) | Returns true if the given string contains the specified substring.<br>Example:<br>CONTAINS("Calculation", "alcu") = true |
| **ENDSWITH** | ENDSWITH(string, substring) | Returns true if the given string ends with the specified substring. Trailing white spaces are ignored.<br>Example:<br>ENDSWITH("Tableau", "leau") = true |
| **FIND** | 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.<br>Examples: |

| | | |
|---|---|---|
| | | FIND("Calculation", "alcu") = 2<br>FIND("Calculation", "Computer") = 0<br>FIND("Calculation", "a", 3) = 7<br>FIND("Calculation", "a", 2) = 2<br>FIND("Calculation", "a", 8) = 0<br>FIND("Calculation", "a", 3) = 7<br>FIND("Calculation", "a", 2) = 2<br>FIND("Calculation", "a", 8) = 0 |
| **FINDNTH** | 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.<br>**Note**: FINDNTH is not available for all data sources.<br>Example:<br>FINDNTH("Calculation", "a", 2) = 7 |
| **LEFT** | LEFT(string, number) | Returns the left-most number of characters in the string.<br>Example:<br>LEFT("Matador", 4) = "Mata" |
| **LEN** | LEN(string) | Returns the length of the string.<br>Example:<br>LEN("Matador") = 7 |
| **LOWER** | LOWER(string) | Returns string, with all characters lowercase.<br>Example:<br>LOWER("ProductVersion") = "productversion" |
| **LTRIM** | LTRIM(string) | Returns the string with any leading spaces removed.<br>Example:<br>LTRIM(" Matador ") = "Matador " |
| **MAX** | 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.<br>Example:<br>MAX ("Apple","Banana") = "Banana" |
| **MID** | (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.<br>Examples:<br>MID("Calculation", 2) = "alculation"<br>MID("Calculation", 2, 5) ="alcul" |

| | | |
|---|---|---|
| **MIN** | 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.<br><br>Example:<br><br>MIN ("Apple","Banana") = "Apple" |
| **PROPER** | PROPER(string) | Converts a text string so the first letter of each word is capitalized and the remaining letters are in lowercase. Spaces and non-alphanumeric characters such as punctuation also act as separators.<br><br>Example:<br><br>PROPER("PRODUCT name") = "Product Name"<br><br>PROPER("darcy-mae") = "Darcy-Mae" |
| **REPLACE** | REPLACE(string, substring, replacement) | Searches string for substring and replaces it with replacement. If substring is not found, the string is not changed.<br><br>Example:<br><br>REPLACE("Version8.5", "8.5", "9.0") = "Version9.0" |
| **RIGHT** | RIGHT(string, number) | Returns the right-most number of characters in string.<br><br>Example:<br><br>RIGHT("Calculation", 4) = "tion" |
| **RTRIM** | RTRIM(string) | Returns string with any trailing spaces removed.<br><br>Example:<br><br>RTRIM(" Calculation ") = " Calculation" |
| **SPACE** | SPACE(number) | Returns a string that is composed of the specified number of repeated spaces.<br><br>Example:<br><br>SPACE(1) = " " |
| **SPLIT** | SPLIT(string, delimiter, token number) | Returns a substring from a string, using a delimiter character to divide the string into a sequence of tokens.<br><br>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.<br><br>Examples:<br><br>SPLIT ('a-b-c-d', '-', 2) = 'b'<br>SPLIT ('a\|b\|c\|d', '\|', -2) = 'c' |

**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 allow 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 support 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 |
| **STARTSWITH** | STARTSWITH(string, substring) | Returns true if string starts with substring. Leading white spaces are ignored.<br>Example:<br>STARTSWITH("Joker", "Jo") = true | | | |
| **TRIM** | TRIM(string) | Returns the string with leading and trailing spaces removed.<br>Example:<br>TRIM(" Calculation ") = "Calculation" | | | |
| **UPPER** | UPPER(string) | Returns string, with all characters uppercase.<br>Example:<br>UPPER("Calculation") = "CALCULATION" | | | |

# Date Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

Dates are a common element in many data sources. If a field contains recognizable dates, it will have a **date** or **date time** data type. When date fields are used in the viz they get a special set of functionality, including an automatic date hierarchy drill down, date-specific filter options, and specialized date formatting options.

Date functions allow you to manipulate dates in your data source.

Date Functions

Date functions sometimes reference date-specific elements, including the date_part argument, the optional [start_of_week] parameter, and date literals (#). These are covered in more detail at the end of this topic.

There are several other topics that might be of interest but aren't part of date functions:

- Format how a date is displayed in a viz: Custom Date Formats
- Set default date properties: Date Properties for a Data Source
- Work with fiscal dates: Fiscal Dates
- Use the ISO-8601 calendar: ISO-8601 Week-Based Calendar

## DATE

Type conversion function that changes string and number expressions into dates, as long as they are in a recognizable format.

| | |
|---|---|
| Syntax | DATE(expression) |
| Output | Date |
| Definition | Returns a date given a number, string, or date expression. |
| Example | DATE([Employee Start Date])<br>DATE("September 22, 2018")<br>DATE("9/22/2018")<br>DATE(#2018-09-22 14:52#) |
| Notes | Unlike DATEPARSE, there is no need to provide a pattern as DATE automatically recognizes many standard date formats. If DATE does not recognize the input, however, try using DATEPARSE and specifying the format.<br>MAKEDATE is another similar function, but MAKEDATE requires the input of numeric values for year, month, and day. |

## DATEADD

Adds a specified number of date parts (months, days, etc) to the starting date.

| | |
|---|---|
| Syntax | DATEADD (date_part, interval, date) |
| Output | Date |

| | |
|---|---|
| Definition | Returns the specified date with the specified number interval added to the specified date_part of that date. For example, adding three months or 12 days to a starting date. |
| Example | Push out all due dates by one week<br>DATEADD('week', 1, [due date])<br><br>Add 280 days to the date February 20, 2021<br>DATEADD('day', 280, #2/20/21#) = #November 27, 2021# |
| Notes | Supports ISO 8601 dates. |

## DATEDIFF

Returns the number of date parts (weeks, years, etc) between two dates.

| | |
|---|---|
| Syntax | DATEDIFF(date_part, date1, date2, [start_of_week]) |
| Output | Integer |
| Definition | Returns the difference between date1 and date2 expressed in units of date_part. For example, subtracting the dates someone entered and left a band to see how long they were in the band. |
| Example | Number of days between March 25, 1986 and February 20, 2021<br>DATEDIFF('day', #3/25/1986#, #2/20/2021#) = 12,751<br><br>How many months someone was in a band<br>DATEDIFF('month', [date joined band], [date left band]) |
| Notes | Supports ISO 8601 dates. |

## DATENAME

Returns the name of the specified date part as a discrete string.

| | |
|---|---|
| Syntax | DATENAME(date_part, date, [start_of_week]) |
| Output | String |
| Definition | Returns date_part of date as a string. |
| Example | DATENAME('year', #3/25/1986#) = "1986"<br>DATENAME('month', #1986-03-25#) = "March" |
| Notes | Supports ISO 8601 dates.<br>A very similar calculation is DATEPART, which returns the value of the specified date part as a continuous integer. DATEPART can be faster because it is a numerical operation. |

| | |
|---|---|
| | By changing the attributes of the calculation's result (dimension or measure, continuous or discrete) and the date formatting, the results of DATEPART and DATENAME can be formatted to be identical.<br><br>An inverse function is DATEPARSE, which takes a string value and formats it as a date. |

## DATEPARSE

Returns specifically formatted strings as dates.

| | |
|---|---|
| Syntax | DATEPARSE(date_format, [date_string]) |
| Output | Date |
| Definition | The date_format argument will describe how the [string] field is arranged. Because of the variety of ways the string field can be ordered, the date_format must match exactly. For a full explanation, see Convert a Field to a Date Field. |
| Example | DATEPARSE('yyyy-MM-dd', "1986-03-25") = #March 25, 1986# |
| Notes | DATE is a similar function that automatically recognizes many standard date formats. DATEPARSE may be a better option if DATE does not recognize the input pattern.<br><br>MAKEDATE is another similar function, but MAKEDATE requires the input of numeric values for year, month, and day.<br><br>Inverse functions, which take dates apart and return the value of their parts, are DATEPART (integer output) and DATENAME (string output). |
| Database limitations | DATEPARSE is available through the following connectors: non-legacy Excel and text file connections, Amazon EMR Hadoop Hive, Cloudera Hadoop, Google Sheets, Hortonworks Hadoop Hive, MapR Hadoop Hive, MySQL, Oracle, PostgreSQL, and Tableau extracts. Some formats may not be available for all connections.<br><br>DATEPARSE is not supported on Hive variants. Only Denodo, Drill, and Snowflake are supported. |

## DATEPART

Returns the name of the specified date part as an integer.

| | |
|---|---|
| Syntax | DATEPART(date_part, date, [start_of_week]) |
| Output | Integer |
| Definition | Returns date_part of date as an integer. |
| Example | DATEPART('year', #1986-03-25#) = 1986<br><br>DATEPART('month', #1986-03-25#) = 3 |
| Notes | Supports ISO 8601 dates.<br><br>A very similar calculation is DATENAME, which returns the name of the specified date part as a discrete string. DATEPART can be faster because it is a numerical operation. By changing |

| | |
|---|---|
| | the attributes of the field (dimension or measure, continuous or discrete) and the date formatting, the results of DATEPART and DATENAME can be formatted to be identical.<br><br>An inverse function is DATEPARSE, which takes a string value and formats it as a date. |

## DATETRUNC

This function can be thought of as date rounding. It takes a specific date and returns a version of that date at the desired specificity. Because every date must have a value for day, month, quarter, and year, DATETRUNC sets the values as the lowest value for each date part up to the date part specified. Refer to the example for more information.

| | |
|---|---|
| Syntax | DATETRUNC(date_part, date, [start_of_week]) |
| Output | Date |
| Definition | Truncates the 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. |
| Example | DATETRUNC('day', #9/22/2018#) = #9/22/2018#<br><br>DATETRUNC('week', #9/22/2018#) = #9/16/2018#<br><br>(the sunday of the week containing 9/22/2018)<br><br>DATETRUNC('iso-week', #9/22/2018#) = #9/17/2018#<br><br>(the monday of the week containing 9/22/2018)<br><br>DATETRUNC(month, #9/22/2018#) = #9/1/2018#<br><br>(the first day of the month containing 9/22/2018)<br><br>DATETRUNC(quarter, #9/22/2018#) = #7/1/2018#<br><br>(the first day of the quarter containing 9/22/2018)<br><br>DATETRUNC('week', #9/22/2018#) = #1/1/2018#<br><br>(the first day of the year containing 9/22/2018)<br><br>Note: For week and iso-week, the start_of_week comes into play. ISO-weeks always start on Monday. For the locale of this example, an unspecified start_of_week means the week starts on Sunday. |
| Notes | Supports ISO 8601 dates.<br><br>You would not use DATETRUNC to, for example, stop showing the time for a datetime field in a viz. If you want to truncate the *display* of a date rather than round its accuracy, adjust the formatting.<br><br>For example, DATETRUNC('day', #5/17/2022 3:12:48 PM#), if shown in the viz to the second, would display as 5/17/2022 12:00:00 AM. |

## DAY

Returns the day of the month (1-31) as an integer.

| | |
|---|---|
| Syntax | DAY(date) |
| Output | Integer |
| Definition | Returns the day of the given date as an integer. |
| Example | Day(#September 22, 2018#) = 22 |
| Notes | See also WEEK, MONTH, QUARTER, YEAR, and the ISO equivalents |

## ISDATE

Checks if the string is a valid date format.

| | |
|---|---|
| Syntax | ISDATE(string) |
| Output | Boolean |
| Definition | Returns true if a given string is a valid date. |
| Example | ISDATE(09/22/2018) = true<br><br>ISDATE(22SEP18) = false |
| Notes | The required argument must be a string. ISDATE cannot be used for a field with a date data type—the calculation will return an error. |

## MAKEDATE

| | |
|---|---|
| Syntax | MAKEDATE(year, month, day) |
| Output | Date |
| Definition | Returns a date value constructed from the specified year, month, and date. |
| Example | MAKEDATE(1986,3,25) = #1986-03-25#<br><br>Note that incorrectly entered values will be adjusted into a date, such as MAKEDATE(2020,4,31) = May 1, 2020 rather than returning an error that there is no 31st day of April. |
| Notes | Available for Tableau Data Extracts. Check for availability in other data sources.<br><br>MAKEDATE requires numerical inputs for the parts of a date. If your data is a string that should be a date, try the DATE function. DATE automatically recognizes many standard date formats. If DATE does not recognize the input try using DATEPARSE. |

## MAKEDATETIME

| | |
|---|---|
| Syntax | MAKEDATETIME(date, time) |
| Output | Datetime |
| Definition | 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. |

| | |
|---|---|
| Example | MAKEDATETIME("1899-12-30", #07:59:00#) = #12/30/1899 7:59:00 AM#<br><br>MAKEDATETIME([Date], [Time]) = #1/1/2001 6:00:00 AM# |
| Notes | This function is available only for MySQL-compatible connections (which for Tableau are MySQL and Amazon Aurora).<br><br>MAKETIME is a similar function available for Tableau Data Extracts and some other data sources. |

## MAKETIME

| | |
|---|---|
| Syntax | MAKETIME(hour, minute, second) |
| Output | Datetime |
| Definition | Returns a date value constructed from the specified hour, minute, and second. |
| Example | MAKETIME(14, 52, 40) = #1/1/1899 14:52:40# |
| Notes | Because Tableau does not support a time data type, only date time, the output is a a datetime. The date portion of the field will be 1/1/1899.<br><br>Similar function to MAKEDATETIME, which is only available for MYSQL-compatible connections. |

## MAX

| | |
|---|---|
| Syntax | MAX(expression) or MAX(expr1, expr2) |
| Output | Date (see notes) |
| Definition | MAX is usually applied to numbers but also works on dates. Returns the maximum (most recent) of a date field or two dates. |
| Example | MAX(#Sept 22, 2018#, #Feb 20, 2021#) = #Feb 20, 2021#<br><br>MAX([Ship date]) |
| Notes | MAX(expression) is treated as an aggregate function and returns a single aggregated result. This will display as AGG([calculation name]) in the viz and will not have a date hierarchy.<br><br>MAX(expr1, expr2) compares the two values and returns a row-level value. For dates, that value will be a date, and the results will retain the date hierarchy.<br><br>Returns Null if any argument is Null. |

## MIN

| | |
|---|---|
| Syntax | MIN(expression) or MIN(expr1, expr2) |
| Output | Date (see notes) |
| Definition | MIN is usually applied to numbers but also works on dates. Returns the minimum (earliest) of a date field or two dates. |
| Example | MIN(#Sept 22, 2018#, #Feb 20, 2021#) = #Sept 22, 2018# |

| | |
|---|---|
| | MIN(Ship date]) |
| Notes | MIN(expression) is treated as an aggregate function and returns a single aggregated result. This will display as AGG([calculation name]) in the viz and will not have a date hierarchy.<br><br>MIN(expr1, expr2) compares the two values and returns a row-level value. For dates, that value will be a date, and the results will retain the date hierarchy.<br><br>Returns Null if any argument is Null. |

## MONTH

| | |
|---|---|
| Syntax | MONTH(date) |
| Output | Integer |
| Definition | Returns the month of the given date as an integer. |
| Example | MONTH(#1986-03-25#) = 3 |
| Notes | See also DAY, WEEK, QUARTER, YEAR, and the ISO equivalents. |

## NOW

| | |
|---|---|
| Syntax | NOW() |
| Output | Datetime |
| Definition | Returns the current local system date and time. |
| Example | NOW() = 1986-03-25 1:08:21 PM |
| Notes | NOW does not take an argument.<br><br>See also TODAY, a similar calculation that returns a date instead of a datetime.<br><br>If the data source is a live connection, the system date and time could be in another timezone. For more information on how to address this, see the Knowledge Base. |

## QUARTER

| | |
|---|---|
| Syntax | QUARTER(date) |
| Output | Integer |
| Definition | Returns the quarter of the given date as an integer. |
| Example | QUARTER(#1986-03-25#) = 1 |
| Notes | See also DAY, WEEK, MONTH, YEAR, and the ISO equivalents. |

## TODAY

| | |
|---|---|
| Syntax | TODAY() |

| | |
|---|---|
| Output | Date |
| Definition | Returns the current local system date. |
| Example | TODAY() = 1986-03-25 |
| Notes | TODAY does not take an argument.<br><br>See also NOW, a similar calculation that returns a datetime instead of a date.<br><br>If the data source is a live connection, the system date could be in another timezone. For more information on how to address this, see the Knowledge Base. |

## WEEK

| | |
|---|---|
| Syntax | WEEK(date) |
| Output | Integer |
| Definition | Returns the week of the given date as an integer. |
| Example | WEEK(#1986-03-25#) = 13 |
| Notes | See also DAY, MONTH, QUARTER, YEAR, and the ISO equivalents. |

## YEAR

| | |
|---|---|
| Syntax | YEAR(date) |
| Output | Integer |
| Definition | Returns the year of the given date as an integer. |
| Example | YEAR(#1986-03-25#) = 1,986 |
| Notes | See also DAY, WEEK, MONTH, QUARTER, and the ISO equivalents. |

## ISOQUARTER

| | |
|---|---|
| Syntax | ISOQUARTER(date) |
| Output | Integer |
| Definition | Returns the ISO8601 week-based quarter of a given date as an integer. |
| Example | ISOQUARTER(#1986-03-25#) = 1 |
| Notes | See also ISOWEEK, ISOWEEKDAY, ISOYEAR, and the non-ISO equivalents. |

## ISOWEEK

| | |
|---|---|
| Syntax | ISOWEEK(date) |
| Output | Integer |

| | |
|---|---|
| Definition | Returns the ISO8601 week-based week of a given date as an integer. |
| Example | ISOWEEK(#1986-03-25#) = 13 |
| Notes | See also ISOWEEKDAY, ISOQUARTER, ISOYEAR, and the non-ISO equivalents. |

## ISOWEEKDAY

| | |
|---|---|
| Syntax | ISOWEEKDAY(date) |
| Output | Integer |
| Definition | Returns the ISO8601 week-based weekday of a given date as an integer. |
| Example | ISOWEEKDAY(#1986-03-25#) = 2 |
| Notes | See also ISOWEEK, ISOQUARTER, ISOYEAR, and the non-ISO equivalents |

## ISOYEAR

| | |
|---|---|
| Syntax | ISOYEAR(date) |
| Output | Integer |
| Definition | Returns the ISO8601 week-based year of a given date as an integer. |
| Example | ISOYEAR(#1986-03-25#) = 1,986 |
| Notes | See also ISOWEEK, ISOWEEKDAY, ISOQUARTER, and the non-ISO equivalents. |

## The date_part argument

Many date functions in Tableau take the argument date_part, which is a string constant that tells the function what part of a date to consider, such as day, week, quarter, etc.

The valid date_part values that you can use are:

| **DATE_PART** | **VALUES** |
|---|---|
| 'year' | Four-digit year |
| 'quarter' | 1-4 |
| 'month' | 1-12 or "January", "February", and so on |
| 'dayofyear' | Day of the year; Jan 1 is 1, Feb 1 is 32, and so on |
| 'day' | 1-31 |
| 'weekday' | 1-7 or "Sunday", "Monday", and so on |
| 'week' | 1-52 |
| 'hour' | 0-23 |

| DATE_PART | VALUES |
|---|---|
| 'minute' | 0-59 |
| 'second' | 0-60 |
| 'iso-year' | Four-digit ISO 8601 year |
| 'iso-quarter' | 1-4 |
| 'iso-week' | 1-52, start of week is always Monday |
| 'iso-weekday' | 1-7, start of week is always Monday |

## The [start_of_week] parameter

Some functions have the optional parameter [start_of_week]. The start_of_week parameter can be used to specify what day is considered the first day of the week, such as "Sunday" or "Monday". If it is omitted, the start of week is determined by the data source. See Date Properties for a Data Source.

For the examples below, 22 September is a Sunday and 24 September is a Tuesday. The DATEDIFF function is being used to calculate the weeks between these dates.

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

- Because start_of_week is 'monday', these dates are in different weeks.

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

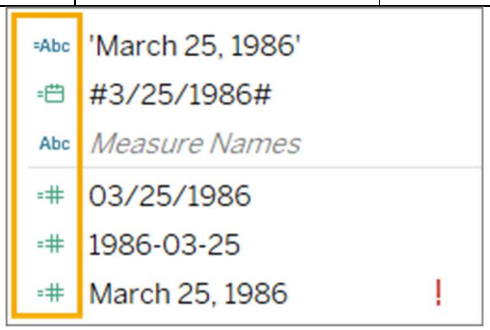
- Because start_of_week is 'sunday', these dates are in the same week.

The date literal (#)

Examples often use the pound symbol (#) with date expressions. This is the date literal, similar to using quotes for text strings, and it tells Tableau that the value inside the symbols is a date.

Without the date literals, dates may be interpreted as various other data types. For example:

| Format | Data Type | Value |
|---|---|---|
| 'March 25, 1986' | String | 'March 25, 1986' |
| #3/25/1986# | Date | #3/25/1986# |
| 03/25/1986 | Floating decimal | 0.00006042 |
| 1986-03-25 | Integer | 1,958 |
| March 25, 1986 | | invalid |



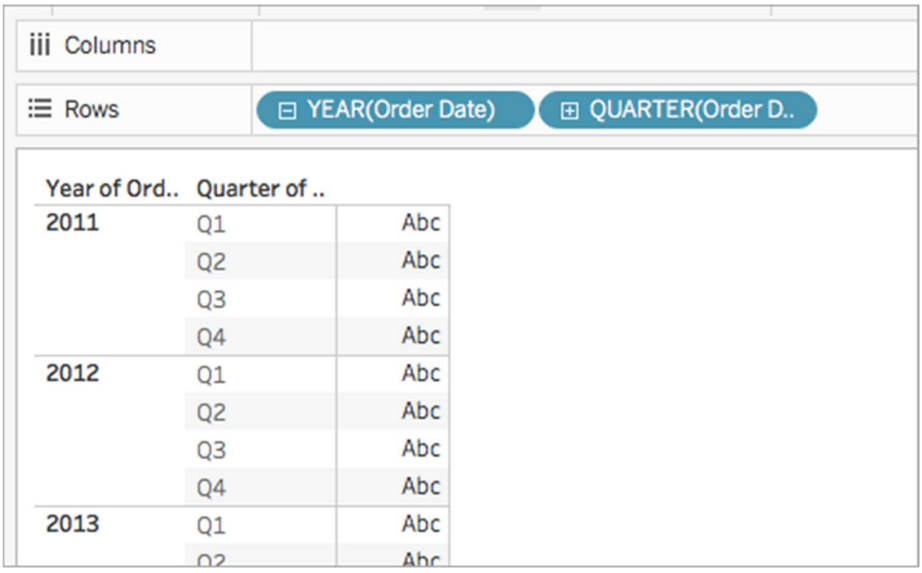

For more information, see Literal expression syntax

Create a date calculation.

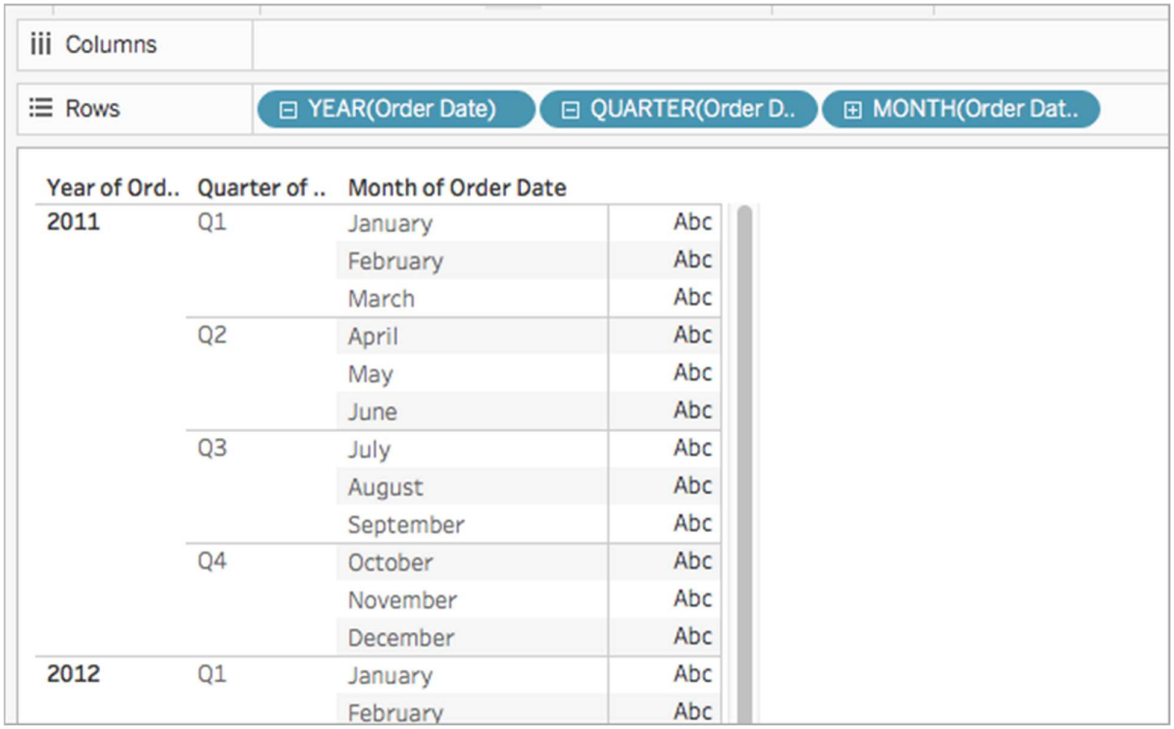
Practice creating a date calculation using the Superstore sample data source.

1. In Tableau Desktop, connect to the **Sample-Superstore** saved data source, which comes with Tableau.
2. Open a worksheet.
3. From the **Data** pane, under Dimensions, drag **Order Date** to the **Rows** shelf.
4. On the **Rows** shelf, click the plus icon (**+**) on the **YEAR(Order Date)** field.

QUARTER(Order Date) is added to the Rows shelf and the view updates.




5. On the **Rows** shelf, click the plus icon (**+**) on the **QUARTER(Order Date)** field to drill down to MONTH(Order Date).



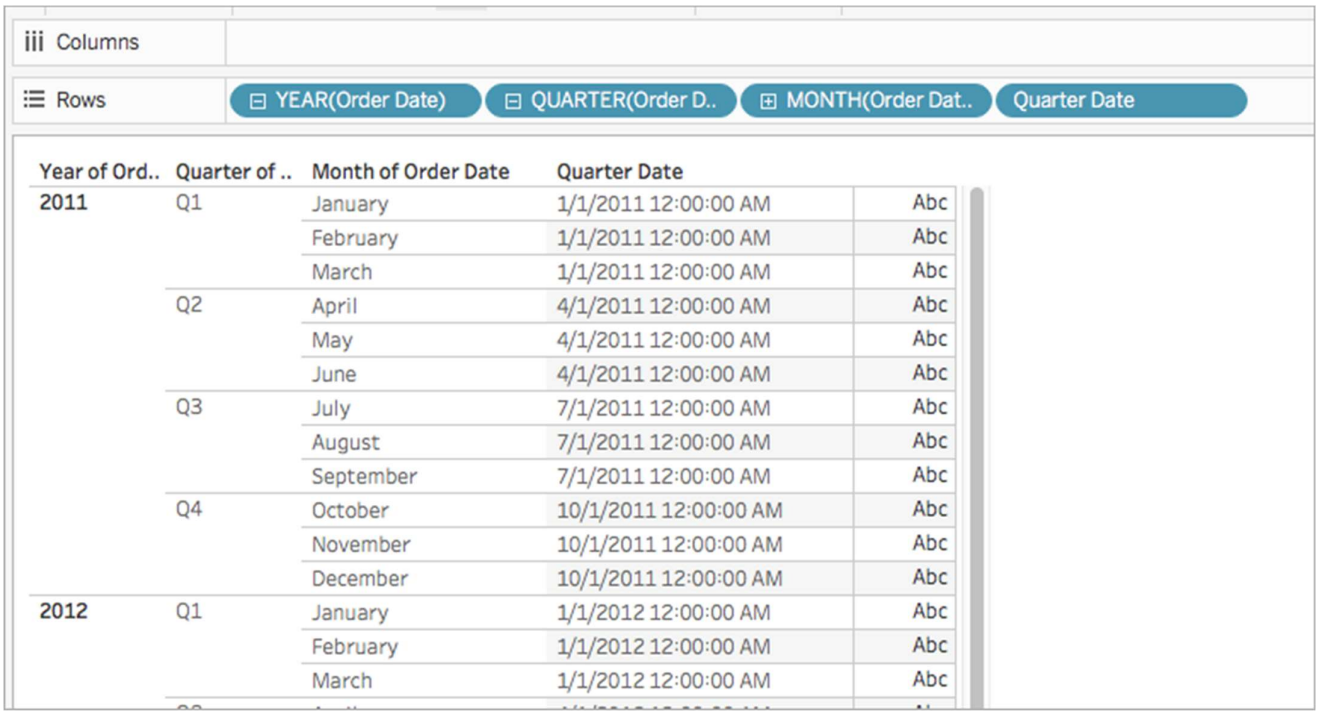

6. Select **Analysis** > **Create Calculated Field**.
7. In the calculation editor that opens, do the following:

- Name the calculated field, Quarter Date.
- Enter the following formula: DATETRUNC('quarter', [Order Date])
- When finished, click **OK**.

The new date calculated field appears under **Dimensions** in the **Data** pane. Just like your other fields, you can use it in one or more visualizations.

8. From the **Data** pane, under Dimensions, drag **Quarter Date** to the **Rows** shelf and place it to the right of MONTH(Order Date). The visualization updates with year values. This is because Tableau rolls date data up to the highest level of detail.
9. On the Rows shelf, right-click **YEAR(Quarter Date)** and select **Exact Date**.
10. On the Rows shelf, right-click **YEAR(Quarter Date)** again and select **Discrete**.

The visualization updates with the exact quarter date for each row in the table.




| Year of Ord.. | Quarter of .. | Month of Order Date | Quarter Date | |
|---|---|---|---|---|
| 2011 | Q1 | January | 1/1/2011 12:00:00 AM | Abc |
| | | February | 1/1/2011 12:00:00 AM | Abc |
| | | March | 1/1/2011 12:00:00 AM | Abc |
| | Q2 | April | 4/1/2011 12:00:00 AM | Abc |
| | | May | 4/1/2011 12:00:00 AM | Abc |
| | | June | 4/1/2011 12:00:00 AM | Abc |
| | Q3 | July | 7/1/2011 12:00:00 AM | Abc |
| | | August | 7/1/2011 12:00:00 AM | Abc |
| | | September | 7/1/2011 12:00:00 AM | Abc |
| | Q4 | October | 10/1/2011 12:00:00 AM | Abc |
| | | November | 10/1/2011 12:00:00 AM | Abc |
| | | December | 10/1/2011 12:00:00 AM | Abc |
| 2012 | Q1 | January | 1/1/2012 12:00:00 AM | Abc |
| | | February | 1/1/2012 12:00:00 AM | Abc |
| | | March | 1/1/2012 12:00:00 AM | Abc |

# Type Conversion

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces type conversion functions and their uses in Tableau. It also demonstrates how to create a type conversion calculation using an example.

Why use type conversion functions

Type conversion functions allow you to convert fields from one data type to another. For example, you can convert numbers to strings, such as age values (numbers) to string values so that Tableau does not try to aggregate them.

The calculation for such a task might look similar to the following:

STR([Age])

Type conversion functions available in Tableau:

The result of any expression in a calculation can be converted to a specific data type. The conversion functions are STR(), DATE(), DATETIME(), INT(), and FLOAT(). For example, if you want to cast a floating point number like 3.14 as an integer, you could write INT(3.14). The result would be 3, which is an integer. The casting functions are described below.

A boolean can be cast to an integer, float, or string, It cannot be cast to a date. True is 1, 1.0, or “1”, while False is 0, 0.0 or “0”. Unknown maps to Null.

| Function | Syntax | Description |
|---|---|---|
| **DATE** | DATE(expression) | Returns a date given a number, string, or date expression.<br><br>Examples:<br><br>DATE([Employee Start Date])<br>DATE("April 15, 2004") = #April 15, 2004#<br>DATE("4/15/2004")<br>DATE(#2006-06-15 14:52#) = #2006-06-15#<br><br>Quotation marks are required in the second and third examples. |
| **DATETIME** | DATETIME(expression) | Returns a datetime given a number, string, or date expression.<br><br>Example:<br><br>DATETIME(“April 15, 2005 07:59:00”) = April 15, 2005 07:59:00 |
| **DATEPARSE** | 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.<br><br>Examples:<br><br>DATEPARSE ("dd.MMMM.yyyy", "15.April.2004") = #April 15, 2004#<br>DATEPARSE ("h'h' m'm' s's'", "10h 5m 3s") = #10:05:03# |
| **FLOAT** | FLOAT(expression) | Casts its argument as a floating point number.<br><br>Examples:<br><br>FLOAT(3) = 3.000<br>FLOAT([Age]) converts every value in the Age field to a floating point number. |
| **INT** | INT(expression) | Casts its argument as an integer. For expressions, this function truncates results to the closest integer toward zero.<br><br>Examples:<br><br>INT(8.0/3.0) = 2<br>INT(4.0/1.5) = 2<br>INT(0.50/1.0) = 0<br>INT(-9.7) = -9<br><br>When a string is converted to an integer it is first converted to a float and then rounded. |
| **STR** | STR(expression) | Casts its argument as a string.<br><br>Example:<br><br>STR([Age])<br><br>This expression takes all of the values in the measure called Age and converts them to strings. |

Create a type conversion calculation

Follow along with the steps below to learn how to create a type conversion calculation.

1. In Tableau Desktop, connect to the **Sample - Superstore** saved data source, which comes with Tableau.
2. Navigate to a worksheet.
3. Select **Analysis** > **Create Calculated Field**.
4. In the calculation editor that opens, do the following:
    - Name the calculated field, Postal Code String.
    - Enter the following formula:

## STR([Postal Code])

This calculation converts the Postal Code field from a number to a string.

- When finished, click **OK**.

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

Converting this field from a number to a string ensures that Tableau treats it as a string and not a number (i.e. Tableau does not aggregate it).

# Logical Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Server*

This article introduces logical functions and their uses in Tableau. It also demonstrates how to create a logical calculation using an example.

Why use logical calculations

Logical calculations allow you to determine if a certain condition is true or false (boolean logic). For example, you might want to quickly see if sales for each country you distribute your merchandise to were above or below a certain threshold.

The logical calculation might look something like this:

SUM(Sales) > 1,000,000

Logical functions available in Tableau:

| Function | Syntax | Description |
|---|---|---|
| **IN** | <expr1> IN <expr2> | Returns TRUE if any value in <expr1> matches any value in <expr2>.<br><br>The values in <expr2> can be a set, list of literal values, or combined field.<br><br>Examples:<br><br>SUM([Cost]) IN (1000, 15, 200)<br><br>[Field] IN [SET] |
| **AND** | IF <expr1> AND <expr2> THEN <then> END | Performs a logical conjunction on two expressions.<br><br>Example:<br><br>IF (ATTR([Market]) = "New Business" AND SUM([Sales]) > [Emerging Threshold] )THEN "Well Performing" |
| **CASE** | CASE <expression> WHEN <value1> THEN <return1> WHEN <value2> THEN <return2> ... ELSE <default return> END | Performs logical tests and returns appropriate values. 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.<br><br>CASE also supports WHEN IN construction, such as<br><br>CASE <expression> WHEN IN <set1> THEN <return1> WHEN IN <combinedfield> THEN <return2> ... ELSE <default> END<br><br>The values that WHEN IN compare to must be a set, list of literal values, or combined field. |

| | | |
|---|---|---|
| | | **Additional notes**<br>• *CASE versus IF*: CASE is often easier to use than IIF or IF THEN ELSE. Typically, an IF function performs a sequence of arbitrary tests, and a CASE function searches for a match to an expression. But a CASE function can always be rewritten as an IF function, although the CASE function will generally be more concise.<br>• *CASE versus groups*: Many times you can use a group to get the same results as a complicated CASE function. You may want to see which is more performant for your scenario.<br>Examples:<br>CASE [Region] WHEN 'West' THEN 1 WHEN 'East' THEN 2 ELSE 3 END<br>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 |
| **ELSE** | IF <expr> THEN <then> ELSE <else> END | Tests a series of expressions returning the <then> value for the first true <expr>.<br>Example:<br>If [Profit] > 0 THEN 'Profitable' ELSE 'Loss' END |
| **ELSEIF** | IF <expr> THEN <then> [ELSEIF <expr2> THEN <then2>...] [ELSE <else>] END | Tests a series of expressions returning the <then> value for the first true <expr>.<br>Example:<br>IF [Profit] > 0 THEN 'Profitable' ELSEIF [Profit] = 0 THEN 'Breakeven' ELSE 'Loss' END |
| **END** | IF <expr> THEN <then> [ELSEIF <expr2> THEN <then2>...] [ELSE <else>] END | Tests a series of expressions returning the <then> value for the first true <expr>. Must be placed at the end of an expression.<br>Example:<br>IF [Profit] > 0 THEN 'Profitable' ELSEIF [Profit] = 0 THEN 'Breakeven' ELSE 'Loss' END |
| **IF** | IF <expr> THEN <then> [ELSEIF <expr2> THEN <then2>...] [ELSE <else>] END | Tests a series of expressions returning the <then> value for the first true <expr>.<br>Example: |

| | | |
|---|---|---|
| | | IF [Profit] > 0 THEN 'Profitable' ELSEIF [Profit] = 0 THEN 'Breakeven' ELSE 'Loss' END |
| **IFNULL** | IFNULL(expr1, expr2) | Returns <expr1> if it is not null, otherwise returns <expr2>.<br>Example:<br>IFNULL([Profit], 0) |
| **IIF** | IIF(test, then, else, [unknown]) | Checks whether a condition is met, and returns one value if TRUE, another value if FALSE, and an optional third value or NULL if unknown.<br>Example:<br>IIF([Profit] > 0, 'Profit', 'Loss') |
| **ISDATE** | ISDATE(string) | Returns true if a given string is a valid date.<br>Example:<br>ISDATE("2004-04-15") = True |
| **ISNULL** | ISNULL(expression) | Returns true if the expression is NULL (does not contain valid data).<br>Example:<br>ISNULL([Profit]) |
| **MAX** | MAX(expression) or Max(expr1, expr2) | Returns the maximum of a single expression across all records or the maximum of two expressions for each record.<br>Example:<br>MAX([Sales]) |
| **MIN** | MIN(expression) or MIN(expr1, expr2) | Returns the minimum of an expression across all records or the minimum of two expressions for each record.<br>Example:<br>MIN([Profit]) |
| **NOT** | IF NOT <expr> THEN <then> END | Performs logical negation on an expression.<br>Example:<br>IF NOT [Profit] > 0 THEN "Unprofitable" END |
| **OR** | IF <expr1> OR <expr2> THEN <then> END | Performs a logical disjunction on two expressions.<br>Example:<br>IF [Profit] < 0 OR [Profit] = 0 THEN "Needs Improvement" END |

| | | |
|---|---|---|
| **THEN** | IF <expre> THEN <then> [ELSEIF ,expr2> THEN <then2>...] [ELSE <else>] END | Tests a series of expressions returning the <then> value for the first true <expr>.<br>Example:<br>IF [Profit] > 0 THEN 'Profitable' ELSEIF [Profit] = 0 THEN 'Break even' ELSE 'unprofitable' END |
| **WHEN** | CASE <expr> WHEN <Value1> THEN <return1> ... [ELSE <else>] END | Finds the first <value> that matches <expr> and returns the corresponding <return>.<br>Example:<br>CASE [RomanNumberal] WHEN 'I' THEN 1 WHEN 'II' THEN 2 ELSE 3 END |
| **ZN** | ZN(expression) | Returns <expression> if it is not null, otherwise returns zero.<br>Example:<br>ZN([Profit]) |

**Note**: some of these are actually logical operators and appear in black, not blue. For more information, see Operator syntax.

Create a logical calculation

Follow along with the steps below to learn how to create a logical calculation.

1. In Tableau Desktop, connect to the **Sample - Superstore** saved data source, which comes with Tableau.
2. Navigate to a worksheet.
3. From the **Data** pane, drag **State** to the **Rows** shelf.
4. From the **Data** pane, drag **Category** to the **Rows** shelf and place it to the right of State.
5. From the **Data** pane, drag **Sales** to the **Columns** shelf.
6. Select **Analysis** > **Create Calculated Field**.
7. In the calculation editor that opens, do the following:
    - Name the calculated field, KPI.
    - Enter the following formula:

SUM([Profit]) > 0

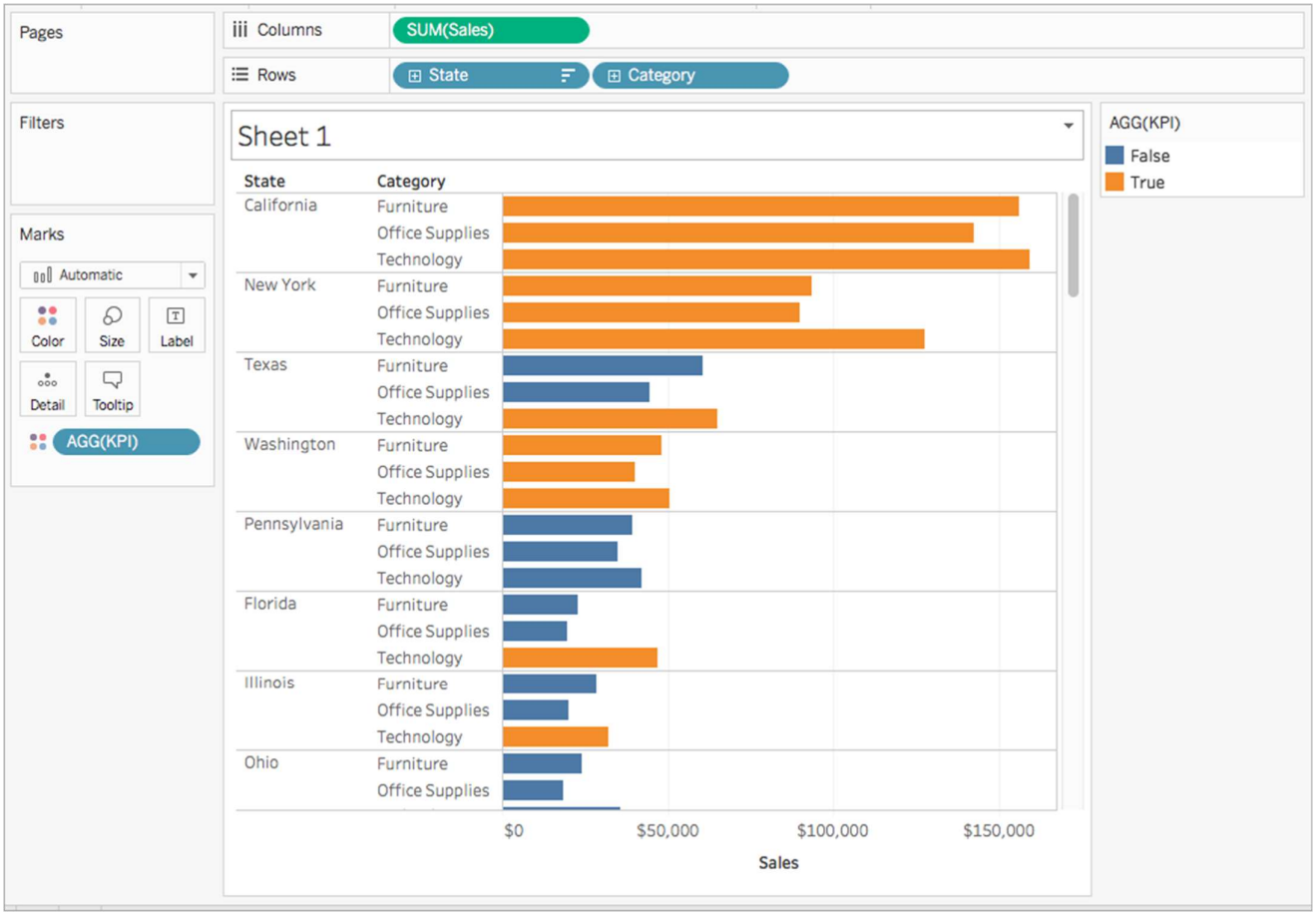
This calculation quickly checks if a member is great than zero. If so, it returns true; if not, it returns false.

    - When finished, click **OK**.

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

8. From the **Data** pane, drag **KPI** to **Color** on the Marks card.

You can now see which categories are losing money in each state.



# Aggregate Functions in Tableau

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces aggregate functions and their uses in Tableau. It also demonstrates how to create an aggregate calculation using an example.

Why use aggregate functions

Aggregate functions allow you to summarize or change the granularity of your data.

For example, you might want to know exactly how many orders your store had for a particular year. You can use the COUNTD function to summarize the exact number of orders your company had, and then break the visualization down by year.

The calculation might look something like this:

COUNTD (Order ID)

The visualization might look something like this:


Aggregate functions available in Tableau

**Aggregations and floating-point arithmetic**: The results of some aggregations may not always be exactly as expected. For example, you may find that the Sum function returns a value such as -1.42e-14 for a column of numbers that you know should sum to exactly 0. This happens because the Institute of Electrical and Electronics Engineers (IEEE) 754 floating-point standard requires that numbers be stored in binary format, which means that numbers are sometimes rounded at extremely fine levels of precision. You can eliminate this potential distraction by using the ROUND function (see Number Functions) or by formatting the number to show fewer decimal places.

| Function | Syntax | Definition |
|---|---|---|
| **ATTR** | 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** | AVG(expression) | Returns the average of all the values in the expression. AVG can be used with numeric fields only. Null values are ignored. |
| **COLLECT** | COLLECT (spatial) | An aggregate calculation that combines the values in the argument field. Null values are ignored.<br><br>**Note**: The COLLECT function can only be used with spatial fields.<br><br>Example: |

| | | |
|---|---|---|
| | | COLLECT ([Geometry]) |
| **CORR** | CORR(expression 1, expression2) | Returns the Pearson correlation coefficient of two expressions.<br><br>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.<br><br>CORR is available with the following data sources:<br><br>• Tableau data extracts (you can create an extract from any data source)<br>• Cloudera Hive<br>• EXASolution<br>• Firebird (version 3.0 and later)<br>• Google BigQuery<br>• Hortonworks Hadoop Hive<br>• IBM PDA (Netezza)<br>• Oracle<br>• PostgreSQL<br>• Presto<br>• SybaseIQ<br>• Teradata<br>• Vertica<br><br>For other data sources, consider either extracting the data or using WINDOW_CORR. See Table Calculation Functions.<br><br>**Note**: The square of a CORR result is equivalent to the R-Squared value for a linear trend line model. See Trend Line Model Terms.<br><br>Example:<br><br>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:<br><br>{CORR(Sales, Profit)}<br><br>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. |

| | | |
|---|---|---|
| **COUNT** | COUNT(expression) | Returns the number of items in a group. Null values are not counted. |
| **COUNTD** | 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. |
| **COVAR** | COVAR(expression 1, expression2) | Returns the *sample covariance* of two expressions.<br><br>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.<br><br>COVAR is available with the following data sources:<br>• Tableau data extracts (you can create an extract from any data source)<br>• Cloudera Hive<br>• EXASolution<br>• Firebird (version 3.0 and later)<br>• Google BigQuery<br>• Hortonworks Hadoop Hive<br>• IBM PDA (Netezza)<br>• Oracle<br>• PostgreSQL<br>• Presto<br>• SybaseIQ<br>• Teradata<br>• Vertica<br><br>For other data sources, consider either extracting the data or using WINDOW_COVAR. See Table Calculation Functions.<br><br>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.<br><br>Example:<br><br>The following formula returns the sample covariance of **Sales** and **Profit**.<br><br>COVAR([Sales], [Profit]) |
| **COVARP** | COVARP(expression 1, expression2) | Returns the *population covariance* of two expressions.<br><br>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.<br><br>COVARP is available with the following data sources:<br><br>• Tableau data extracts (you can create an extract from any data source)<br>• Cloudera Hive<br>• EXASolution<br>• Firebird (version 3.0 and later)<br>• Google BigQuery<br>• Hortonworks Hadoop Hive<br>• IBM PDA (Netezza)<br>• Oracle<br>• PostgreSQL<br>• Presto<br>• SybaseIQ<br>• Teradata<br>• Vertica<br><br>For other data sources, consider either extracting the data or using WINDOW_COVARP. See Table Calculation Functions.<br><br>If expression1 and expression2 are the same—for example, COVARP([profit], [profit])—COVARP returns a value that indicates how widely values are distributed.<br><br>**Note**: The value of COVARP(X, X) is equivalent to the value of VARP(X) and also to the value of STDEVP(X)^2. |

| | | |
|---|---|---|
| | | Example:<br><br>The following formula returns the population covariance of **Sales** and **Profit**.<br><br>COVARP([Sales], [Profit]) |
| **MAX** | MAX(expression) | Returns the maximum of an expression across all records. If the expression is a string value, this function returns the last value where last is defined by alphabetical order. |
| **MEDIAN** | 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. It is also not available for connections using any of the following data sources:<br><br>• Access<br>• Amazon Redshift<br>• Cloudera Hadoop<br>• HP Vertica<br>• IBM DB2<br>• IBM PDA (Netezza)<br>• Microsoft SQL Server<br>• MySQL<br>• SAP HANA<br>• Teradata<br><br>For other data source types, you can extract your data into an extract file to use this function. See Extract Your Data. |
| **MIN** | MIN(expression) | Returns the minimum of an expression across all records. If the expression is a string value, this function returns the first value where first is defined by alphabetical order. |
| **PERCENTILE** | 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.<br><br>This function is available for the following data sources.<br><br>• Non-legacy Microsoft Excel and Text File connections.<br>• Extracts and extract-only data source types (for example, Google Analytics, OData, or Salesforce).<br>• Sybase IQ 15.1 and later data sources. |

| | | |
|---|---|---|
| | | • Oracle 10 and later data sources.<br>• Cloudera Hive and Hortonworks Hadoop Hive data sources.<br>• EXASolution 4.2 and later data sources.<br>For other data source types, you can extract your data into an extract file to use this function. See Extract Your Data. |
| **STDEV** | STDEV(expression) | Returns the statistical standard deviation of all values in the given expression based on a sample of the population. |
| **STDEVP** | STDEVP(expression) | Returns the statistical standard deviation of all values in the given expression based on a biased population. |
| **SUM** | SUM(expression) | Returns the sum of all values in the expression. SUM can be used with numeric fields only. Null values are ignored. |
| **VAR** | VAR(expression) | Returns the statistical variance of all values in the given expression based on a sample of the population. |
| **VARP** | VARP(expression) | Returns the statistical variance of all values in the given expression on the entire population. |

## Create an aggregate calculation

Follow along with the steps below to learn how to create an aggregate calculation.

1. In Tableau Desktop, connect to the **Sample - Superstore** saved data source, which comes with Tableau.
2. Navigate to a worksheet and select **Analysis** > **Create Calculated Field**.
3. In the calculation editor that opens, do the following:
    - Name the calculated field **Margin**.
    - Enter the following formula:

IIF(SUM([Sales]) !=0, SUM([Profit])/SUM([Sales]), 0)

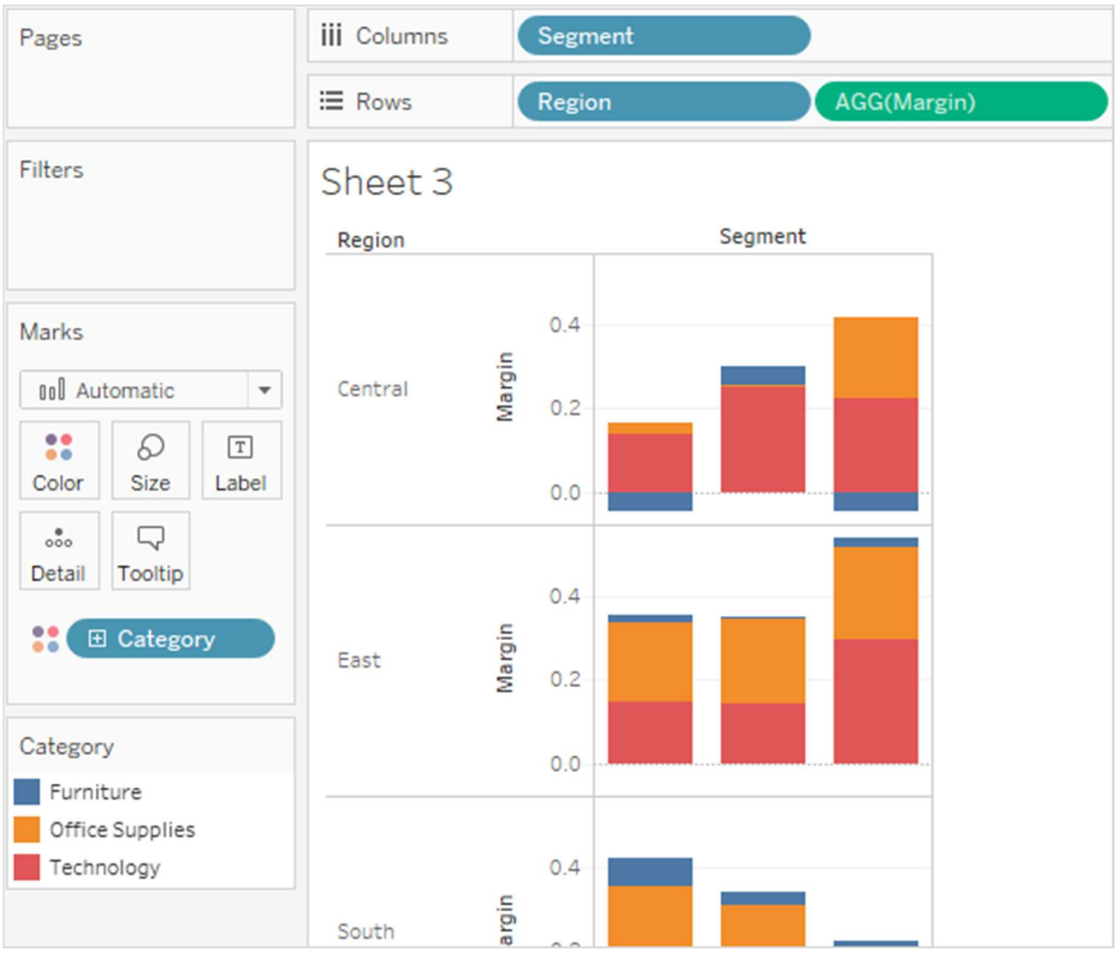
**Note**: You can use the function reference to find and add aggregate functions and other functions (like the logical IIF function in this example) to the calculation formula. For more information, see Use the functions reference in the calculation editor.

    - When finished, click **OK**.

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

**Note**: Aggregation calculations are always measures.

When **Margin** is placed on a shelf or card in the worksheet, its name is changed to **AGG(Margin)**, which indicates that it is an aggregate calculation and cannot be aggregated any further.




Rules for aggregate calculations

The rules that apply to aggregate calculations are as follows:

- For any aggregate calculation, you cannot combine an aggregated value and a disaggregated value. For example, SUM(Price)*[Items] is not a valid expression because SUM(Price) is aggregated and Items is not. However, SUM(Price*Items) and SUM(Price)*SUM(Items) are both valid.
- Constant terms in an expression act as aggregated or disaggregated values as appropriate. For example: SUM(Price*7) and SUM(Price)*7 are both valid expressions.
- All of the functions can be evaluated on aggregated values. However, the arguments to any given function must either all be aggregated or all disaggregated. For example: MAX(SUM(Sales),Profit) is not a valid expression because Sales is aggregated and Profit is not. However, MAX(SUM(Sales),SUM(Profit)) is a valid expression.
- The result of an aggregate calculation is always a measure.
- Like predefined aggregations, aggregate calculations are computed correctly for grand totals. Refer to Grand Totals for more information.

# Pass-Through Functions (RAWSQL)

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

These RAWSQL pass-through functions can be used to send SQL expressions directly to the database, without first being interpreted by Tableau. If you have custom database functions that Tableau doesn't know about, you can use these pass-through functions to call these custom functions.

Your database usually will not understand the field names that are shown in Tableau. Because Tableau does not interpret the SQL expressions you include in the pass-through functions, using the Tableau field names in your expression may cause errors. You can use a substitution syntax to insert the correct field name or expression for a Tableau calculation into pass-through SQL. For example, if you had a function that computed the median of a set of values, you could call that function on the Tableau column [Sales] like this:

## RAWSQLAGG_REAL("MEDIAN(%1)", [Sales])

Because Tableau does not interpret the expression, you must define the aggregation. You can use the RAWSQLAGG functions described below when you are using aggregated expressions.

RAWSQL pass-through functions will not work with published data sources or with Tableau extracts.

These functions may return different results starting in Tableau Desktop 8.2 than they did in earlier versions of Tableau Desktop. This is because Tableau now uses ODBC for pass-through functions instead of OLE DB. ODBC truncates when returning real values as integer; OLE DB rounds when returning real values as integer.

## RAWSQL Functions

The following RAWSQL functions are available in Tableau.

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("%1 > %2", [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("%1", [Order 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

RAWSQLAGG_DATE("MAX(%1)", [Order Date])

## RAWSQLAGG_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

RAWSQLAGG_DATETIME("MIN(%1)", [Delivery Date])

## RAWSQLAGG_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

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

## RAWSQLAGG_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])

# User Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces user functions and their uses in Tableau. It also demonstrates how to create a user calculation using an example.

**Why use user functions**

User functions can be used to create user filters or row-level security filters that affect visualizations published to Tableau Server or Tableau Cloud, so that only certain people can see your visualization.

For example, if you have a visualization that shows the sales performance for each employee in your department published on Tableau Server or Tableau Cloud, you might want to only allow employees to see their own sales numbers when they access that visualization.

In this case, you can use the ISMEMBEROF function to create a field that returns true if the username of the person signed in to the server is a member of a specified group (on the server), such as the "Managers" group, for example. Then when you filter the view using this calculated field, only a person who is part of that group can see the data.

The calculation in this case might look something like the following:

ISMEMBEROF('Managers')

**Note**: If your group or user names contain certain non-alphanumeric you must use HTML URL Encoding for the special characters when using the functions below.

Some special characters are permitted without HTML URL encoding, such as underscores, parentheses, and exclamation points. _ ( ) ! Many other characters must be encoded.

For example, the function ISMEMBEROF("USERS+") needs to be written as ISMEMBEROF("USERS%2B"), because the '%2B' is the HTML URL Encoding for the '+' symbol. For information on HTML URL encoding, please see HTML URL Encoding Reference(Link opens in a new window) at the W3schools Web Developer site.

For embedding workflows in Tableau Cloud only

Among the user functions, a subset of user attribute functions can be used with Tableau Connected apps(Link opens in a new window). The user attribute functions allow user attributes to be captured by Tableau at runtime as part of the authentication workflow. When user attributes are passed from JSON Web Tokens (JWTs), embedded content authored with these functions can control and customize data displayed to users.

**Note:** Preview of the content with these functions is not available when authoring in Tableau Desktop or Tableau Cloud. User attribute functions will return NULL or FALSE values. To ensure user attribute functions work as expected, we recommend you review the content after embedding in your external application. For more information about embedding workflows that include these user functions, see the Embedding API v3(Link opens in a new window) Help.

User functions available in Tableau:

| Function | Syntax | Description |
|---|---|---|
| **FULLNAME** | 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.<br><br>Examples:<br><br>FULLNAME( )<br><br>This returns the full name of the signed in user, Dave Hallsten.<br><br>[Manager]=FULLNAME( )<br><br>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. |
| **ISFULLNAME** | 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.<br><br>Example:<br><br>ISFULLNAME("Dave Hallsten")<br><br>This example returns true if Dave Hallsten is the current user, otherwise it returns false. |
| **ISMEMBEROF** | ISMEMBEROF(string) | Returns true if the person currently using Tableau is a member of a group that matches the given string. If the person currently using Tableau is signed in, the group membership is determined by groups on Tableau Server or Tableau Cloud. If the person is not signed in, this function returns NULL.<br><br>**Note**: The function will return a "True" value if the given string is "All Users", whether signed in to Tableau Server or Tableau Cloud. |

| | | |
|---|---|---|
| | | The ISMEMBEROF() function will also accept Active Directory domains. The Active Directory domain must be declared in the calculation with the group name.<br><br>Example:<br><br>IF ISMEMBEROF('domain.lan\Sales') THEN "Sales" ELSE "Other" END |
| **ISUSERNAME** | 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.<br><br>Example:<br><br>ISUSERNAME("dhallsten")<br><br>This example returns true if dhallsten is the current user; otherwise it returns false.<br><br>Note: "All Users" will always return as true. |
| **USERDOMAIN** | 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.<br><br>Example:<br><br>[Manager]=USERNAME() AND [Domain]=USERDOMAIN() |
| **USERNAME** | 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.<br><br>Examples:<br><br>USERNAME( ) |

| | | |
|---|---|---|
| | | This returns the username of the signed in user, dhallsten.<br><br>[Manager]=USERNAME( )<br><br>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. |
| **USERATTRIBUTE** | USERATTRIBUTE('attribute_name') | Returns a string. If 'attribute_name' is part of the JWT passed to Tableau, the calculation returns the first value of 'attribute_name'. Returns null if 'attribute_name' does not exist.<br><br>**Note:** You can use the USERATTRIBUTEINCLUDES function if you expect 'attribute_name' to return multiple values.<br><br>Example:<br><br>Suppose 'Region' is the user attribute that is included in the JWT and passed to Tableau using the connected app already configured by your site admin. As the workbook author, you can set up your visualization to filter data based on a specified region. In that filter, you can reference the following calculation.<br><br>[Region]=USERATTRIBUTE('Region')<br><br>When Alan Wang from the West region views the embedded visualization, Tableau shows the appropriate data for the West region only. |
| **USERATTRIBUTEINCLUDES** | USERATTRIBUTEINCLUDES ('attribute_name', 'expected_value') | Returns a boolean. Returns "true" if the following are true:<br>1) 'attribute_name' is part of the JWT passed to Tableau and 2) one of 'attribute_name' values equals 'expected_value' . Returns "false" otherwise. |

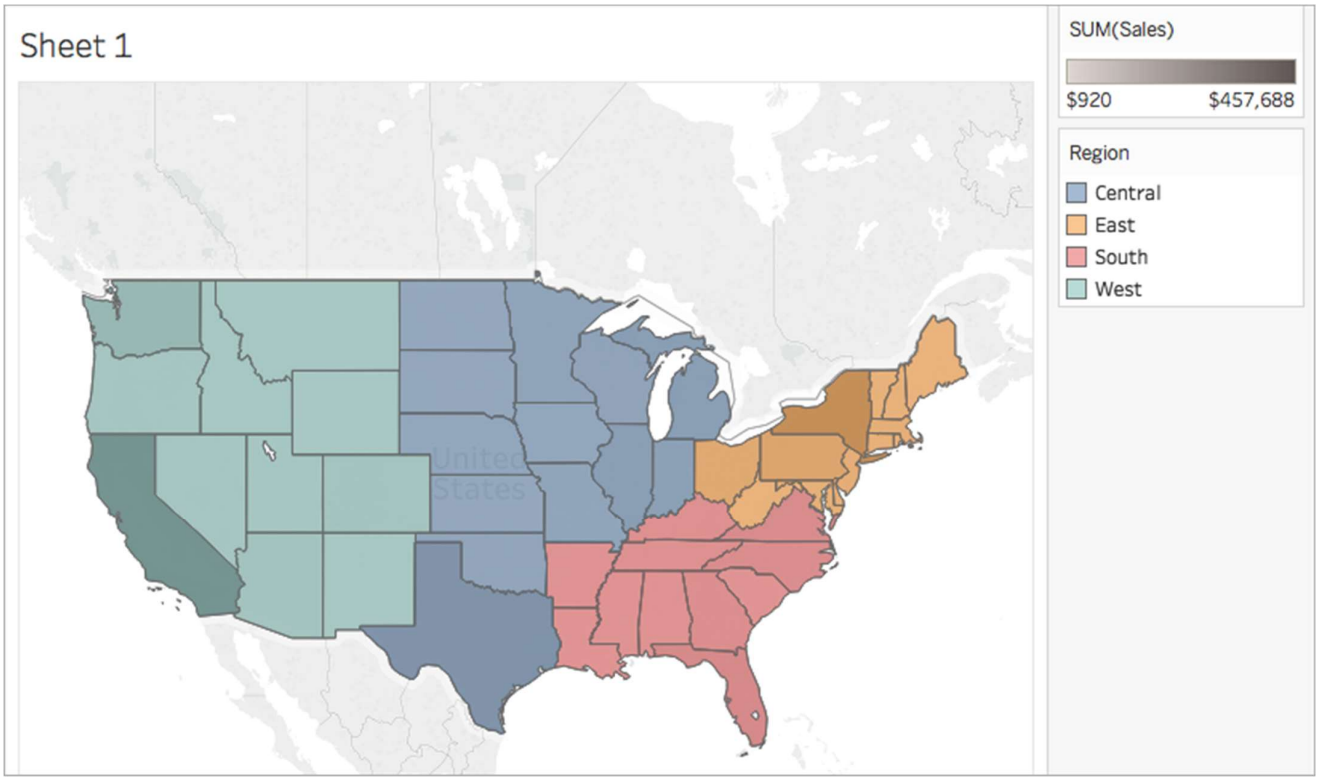
| | | |
|---|---|---|
| | | **Note:** You can use the USERATTRIBUTE function if you expect 'attribute_name' to return a single string value.<br><br>Example:<br><br>Suppose ‘Region’ is the user attribute that is defined in the JWT and passed to Tableau using the connected app already configured by your site admin. As the workbook author, you can set up your visualization to filter data based on the [Region]. In that filter, you can reference the following calculation.<br><br>USERATTRIBUTEINCLUDES('Region', [Region])<br><br>If Alan Wang from the West region accesses the embedded visualization, Tableau checks if the Region user attribute matches one of [Region] field values. When true, the visualization shows the appropriate data. When another user, Michele Kim from the North region accesses the same visualization, she’s unable to see any data because there’s no match with [Region] field values. |

**Create a user calculation**

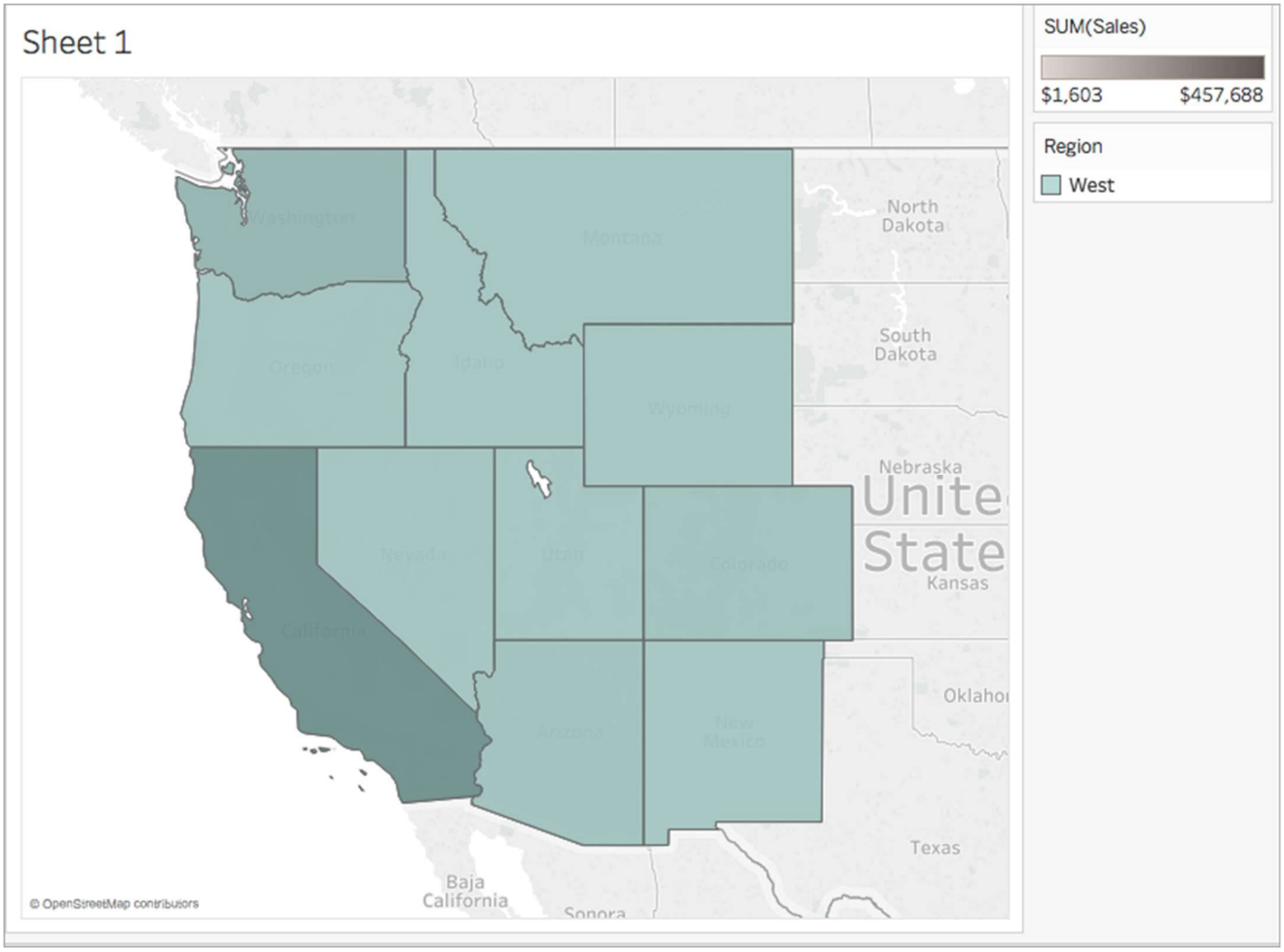
User calculations work directly with the users and groups you have set up on Tableau Server or Tableau Cloud. You can create user calculations to use as filters so users only see the data that is relevant to them.

For example, if you have a map viz similar to the following, which shows sales data for 48 US states, you can create a user calculation to show only parts of the map that are relevant to each user, such as data relevant for a regional manager versus a national manager. (A national manager should be able to see data for the entire country, while a regional manager should only be able to see data for the region they manage).

When the national manger is signed in, they see the following visualization:




When the western regional manager is signed in, they see only sales for their region:



To create a user function that performs similarly to this example, follow the steps below.

Before you begin

To follow along with this example you must have access to Tableau Server or Tableau Cloud. You must also be a Server or Site Administrator.

Step 1: Create the users and groups

1. Sign in to Tableau Server or Tableau Cloud.
2. In Tableau Server or Tableau Cloud, add the following users:
    - Sadie Pawthorne
    - Chuck Magee
    - Fred Suzuki
    - Roxanne Rodriguez

For more information, see Add Users to a Site(Link opens in a new window) in the Tableau Server Help.

3. Create a new group called **National Managers**.

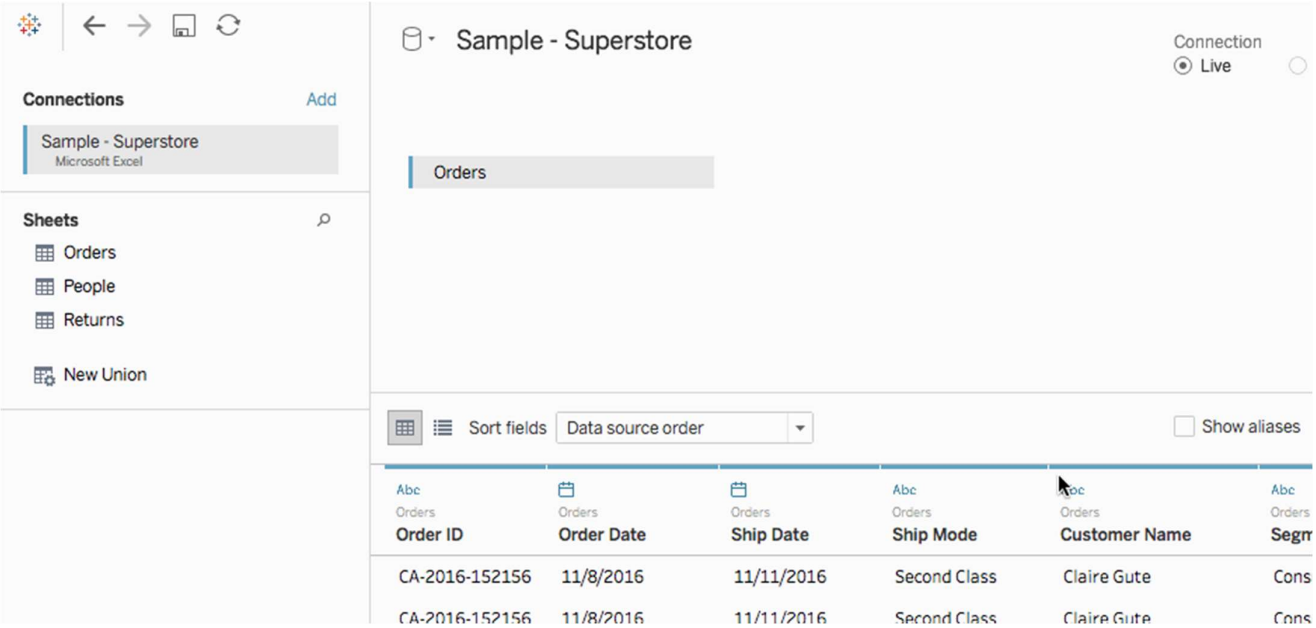
For more information, see Create a Local Group(Link opens in a new window) in the Tableau Server Help.

4. Add yourself to the National Managers group.

For more information, see Add Users to a Group(Link opens in a new window) in the Tableau Server Help.

Step 2: Create the visualization

1. Open Tableau Desktop, and connect to the **Sample-Superstore** data source, which comes with Tableau.
2. In the bottom left corner of the workspace, click the Data Source tab.
3. On the Data Source page, from the Connections pane on the left, drag the People sheet to the join area.
4. Click the join icon and select **Left**.

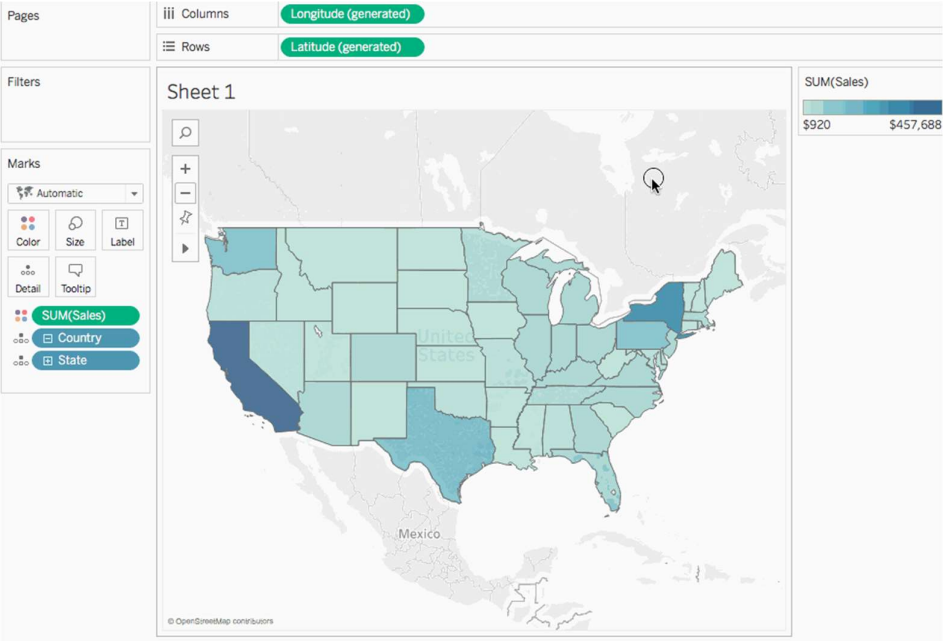



5. Navigate to a new worksheet.

6. In the **Data** pane, under Dimensions, double-click **State**.

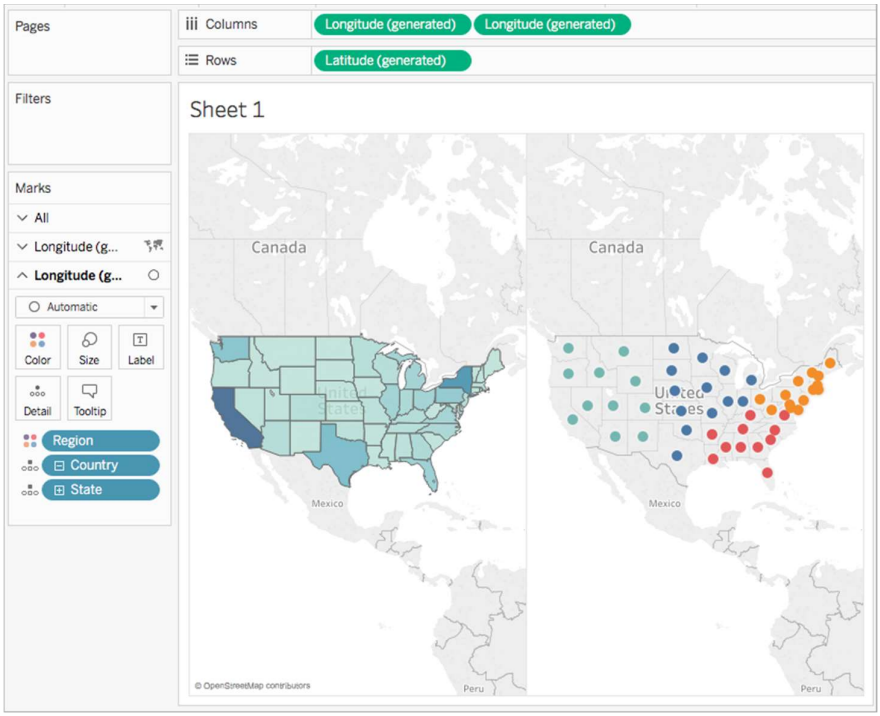
A map view is created.

7. From the **Data** pane, under Measures, drag **Sales** to **Color** on the Marks card.
8. On the Columns shelf, select the Longitude field and hold down Control (Comman on Mac) on your keyboard to copy it. Drag the copy to the right of the original on the Columns shelf.

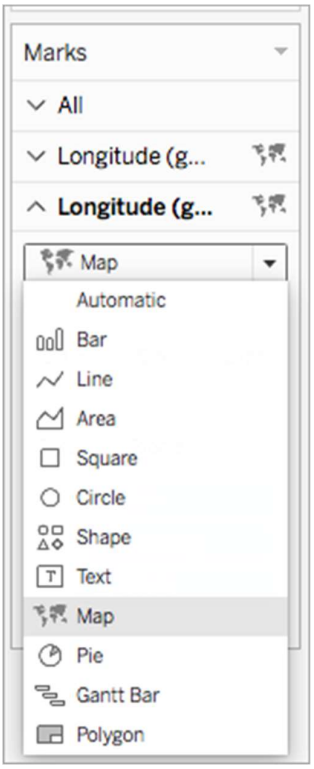



9. On the Marks card, click the second (bottom) **Longitude** tab.
10. From the **Data** pane, drag **Region** to **Color** on the Marks card.

The map view on the right updates with new colors.

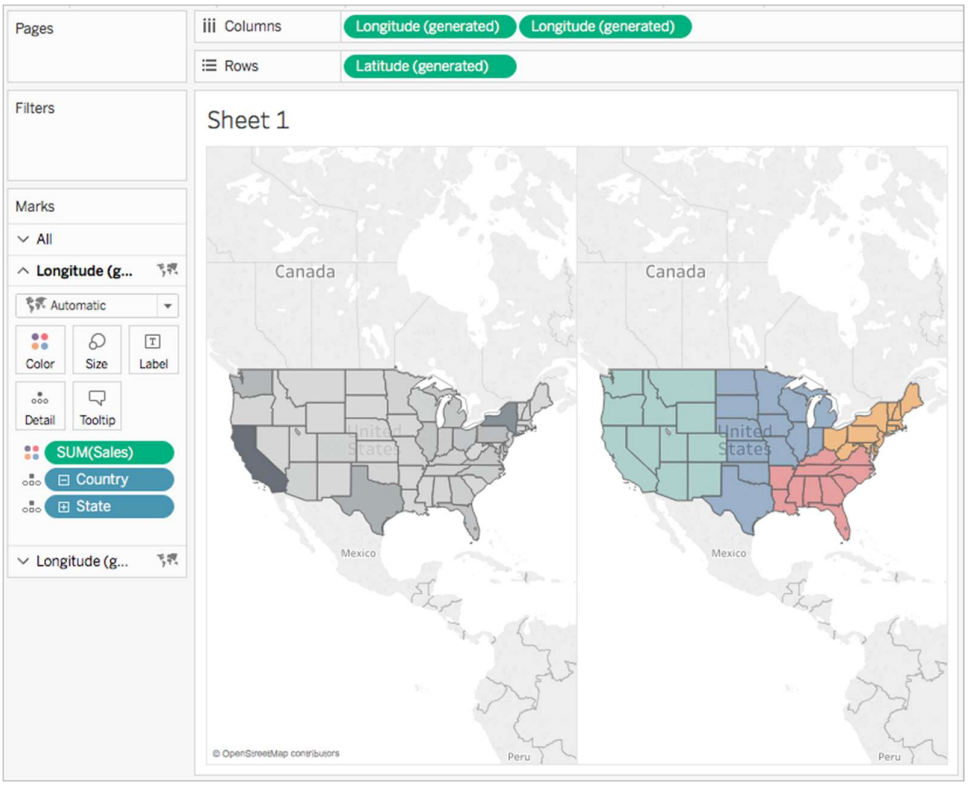



11. On the Marks card, click the mark type drop-down and select **Map**.

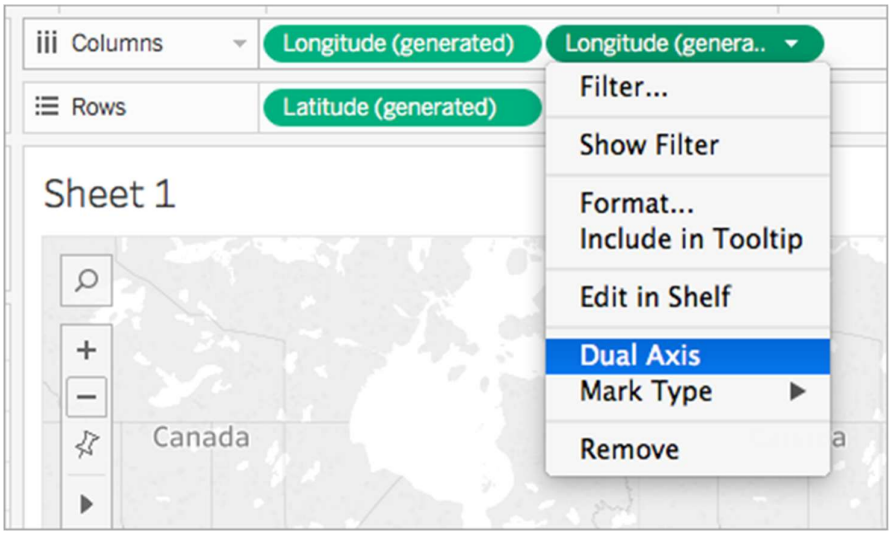



12. On the Marks card, click **Color**, and, under Opacity, adjust the slider to **50%**.
13. On the Marks card, click the first **Longitude** tab.
14. On the Marks card, click **Color** > **Edit Colors**, and then select **Gray** from the color palette drop-down list.

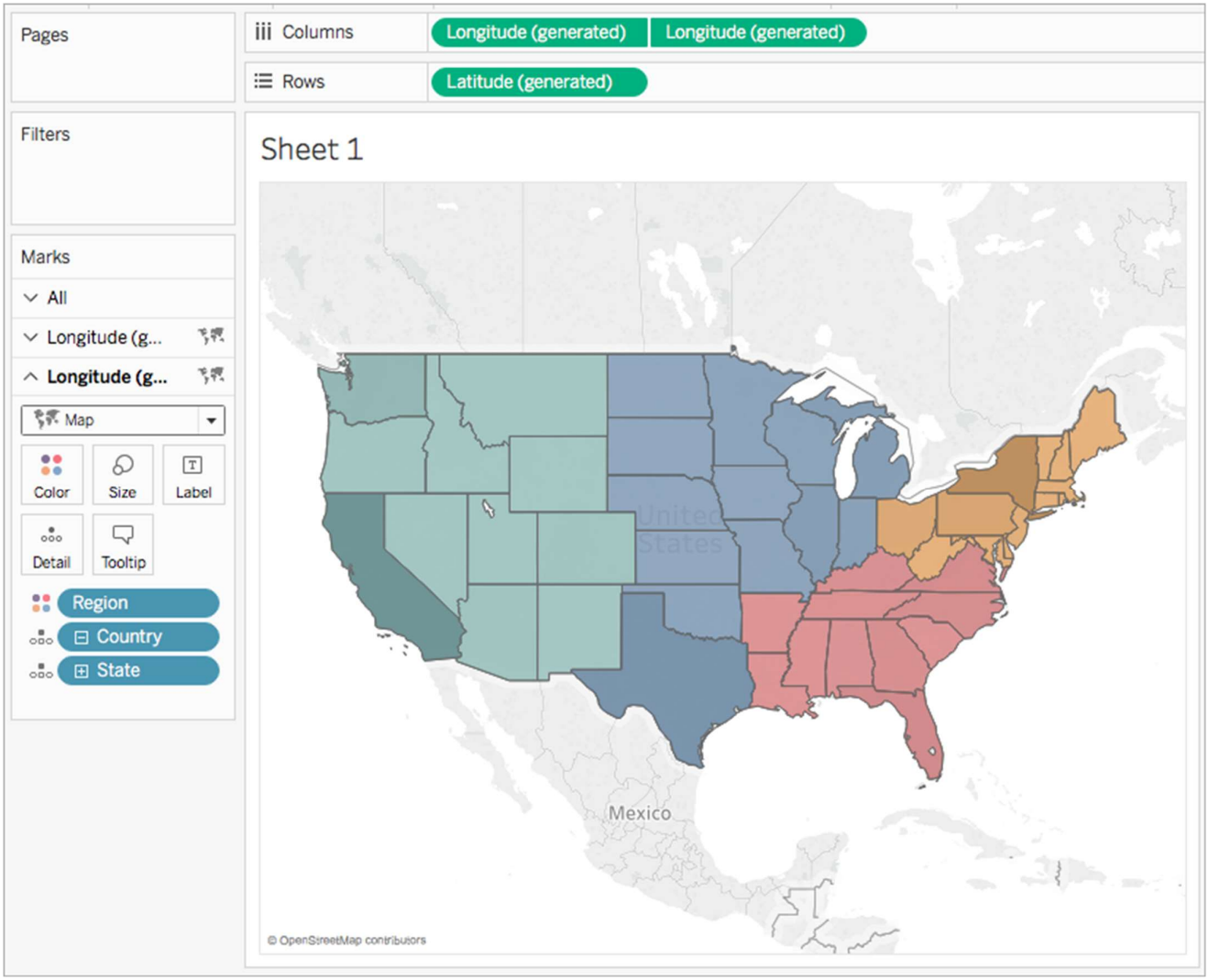
The map view on the left updates.




15. On the Columns shelf, right-click the **Longitude** field on the right and select **Dual Axis**.




The map looks like the following:




Step 3: Create the User Calculation

1. Select **Analysis** > **Create Calculated Field**.
2. In the calculation editor that opens, do the following:

- Name the calculated field, User Filter.
- Enter the following formula:

[Regional Manager] = USERNAME() OR ISMEMBEROF("National Managers")

This calculation checks if a person is included in the Region (People) field, or if a person is included in the National Managers group. If so, it returns true.

- When finished, click **OK**.

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

Step 4: Add the user calculation to the Filters shelf

1. From the Data pane, under Dimensions, drag User Filter to the Filters shelf.
2. In the Filter dialog box that opens, select **True**, and then click **OK**.

**Note**: If you are not signed in to Tableau Server or Tableau Cloud, the True option is not visible. In Tableau Desktop, sign in to Tableau Server or Tableau Cloud to select it. See Sign in to Tableau Server or Tableau Cloud(Link opens in a new window) for more information.

Step 5: Test the calculation

1. In Tableau Desktop, in the bottom-right corner of the workspace, click the Filter as User drop-down and change the user to **Sadie Pawthorne**.

The map updates to show only the West region of the United States because Sadie is assigned to the West region in the People sheet.

2. Select the Filter as User drop-down again and change the user to Roxanne Rodriguez.

The map updates to show only the Central region of the United States because Roxanne is assigned to the Central region in the People sheet.

3. Select the Filter as User drop-down again and change the user to Chuck Magee.

The map updates to show only the East region of the United States because Chuck is assigned to the East region in the People sheet.

4. Select the Filter as User drop-down again and change the user to Fred Suzuki.

The map updates to show only the South region of the United States because Fred is assigned to the South region in the People sheet.

5. Select the Filter as User drop-down one more time and change the user back to yourself.

The map updates to show all data because you are a part of the National Managers group on the server.

This behavior persists when you publish the view to Tableau Server or Tableau Cloud. Users not listed in the National Managers group, or in the People sheet in the Sample Superstore data source see only a blank visualization.

# Table Calculation Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

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

**Why use table calculation functions**

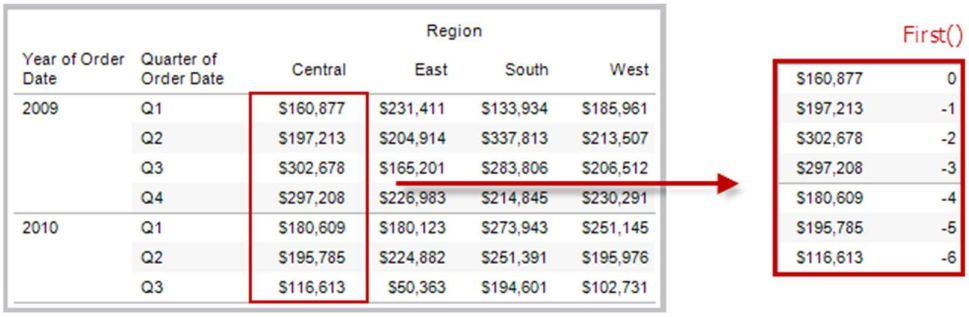
Table calculation functions allow you to perform computations on values in a table.

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

Table calculation functions available in Tableau

## FIRST( )

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



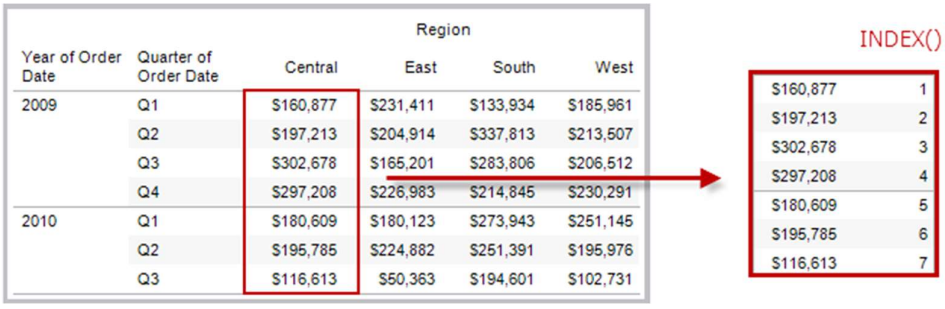
| Year of Order Date | Quarter of Order Date | Region: Central | East | South | West | First() |
|---|---|---|---|---|---|---|
| 2009 | Q1 | $160,877 | $231,411 | $133,934 | $185,961 | 0 |
| | Q2 | $197,213 | $204,914 | $337,813 | $213,507 | -1 |
| | Q3 | $302,678 | $165,201 | $283,806 | $206,512 | -2 |
| | Q4 | $297,208 | $226,983 | $214,845 | $230,291 | -3 |
| 2010 | Q1 | $180,609 | $180,123 | $273,943 | $251,145 | -4 |
| | Q2 | $195,785 | $224,882 | $251,391 | $195,976 | -5 |
| | Q3 | $116,613 | $50,363 | $194,601 | $102,731 | -6 |

Example

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

## INDEX( )

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



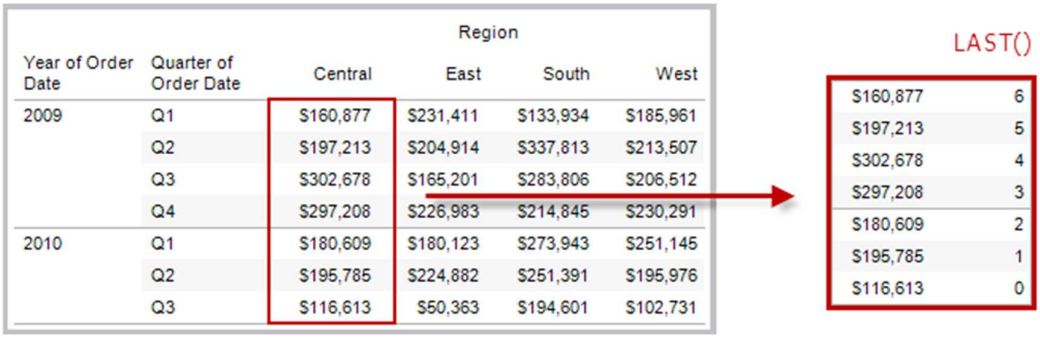
| Year of Order Date | Quarter of Order Date | Region: Central | East | South | West | INDEX() |
|---|---|---|---|---|---|---|
| 2009 | Q1 | $160,877 | $231,411 | $133,934 | $185,961 | 1 |
| | Q2 | $197,213 | $204,914 | $337,813 | $213,507 | 2 |
| | Q3 | $302,678 | $165,201 | $283,806 | $206,512 | 3 |
| | Q4 | $297,208 | $226,983 | $214,845 | $230,291 | 4 |
| 2010 | Q1 | $180,609 | $180,123 | $273,943 | $251,145 | 5 |
| | Q2 | $195,785 | $224,882 | $251,391 | $195,976 | 6 |
| | Q3 | $116,613 | $50,363 | $194,601 | $102,731 | 7 |

Example

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

## LAST( )

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



| | | Region | | | | |
|---|---|---|---|---|---|---|
| Year of Order Date | Quarter of Order Date | Central | East | South | West | LAST() |
| 2009 | Q1 | $160,877 | $231,411 | $133,934 | $185,961 | 6 |
| | Q2 | $197,213 | $204,914 | $337,813 | $213,507 | 5 |
| | Q3 | $302,678 | $165,201 | $283,806 | $206,512 | 4 |
| | Q4 | $297,208 | $226,983 | $214,845 | $230,291 | 3 |
| 2010 | Q1 | $180,609 | $180,123 | $273,943 | $251,145 | 2 |
| | Q2 | $195,785 | $224,882 | $251,391 | $195,976 | 1 |
| | Q3 | $116,613 | $50,363 | $194,601 | $102,731 | 0 |

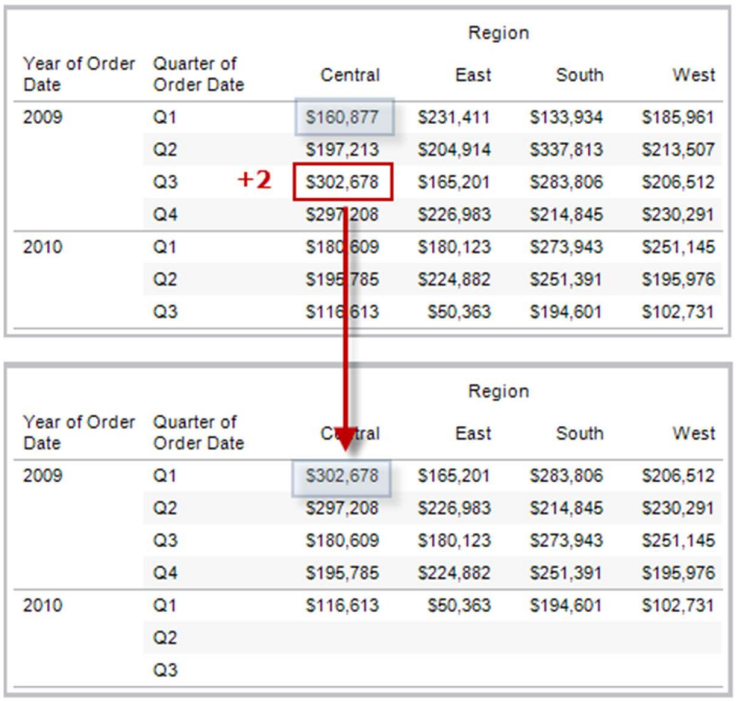
Example

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

## LOOKUP(expression, [offset])

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

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



| | | | Region | | | |
|---|---|---|---|---|---|---|
| Year of Order Date | Quarter of Order Date | | Central | East | South | West |
| 2009 | Q1 | | $160,877 | $231,411 | $133,934 | $185,961 |
| | Q2 | | $197,213 | $204,914 | $337,813 | $213,507 |
| | Q3 | +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 |

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

Example

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

## MODEL_EXTENSION_BOOL (model_name, arguments, expression)

Returns the boolean result of an expression as calculated by a named model deployed on a TabPy external service.

Model_name is the name of the deployed analytics model you want to use.

Each argument is a single string that sets the input values that the deployed model accepts, and is defined by the analytics model.

Use expressions to define the values that are sent from Tableau to the analytics model. Be sure to use aggregation functions (SUM, AVG, etc.) to aggregate the results.

When using the function, the data types and order of the expressions must match that of the input arguments.

Example

MODEL_EXTENSION_BOOL ("isProfitable","inputSales", "inputCosts", SUM([Sales]), SUM([Costs]))

## MODEL_EXTENSION_INT (model_name, arguments, expression)

Returns an integer result of an expression as calculated by a named model deployed on a TabPy external service.

Model_name is the name of the deployed analytics model you want to use.

Each argument is a single string that sets the input values that the deployed model accepts, and is defined by the analytics model.

Use expressions to define the values that are sent from Tableau to the analytics model. Be sure to use aggregation functions (SUM, AVG, etc.) to aggregate the results.

When using the function, the data types and order of the expressions must match that of the input arguments.

Example

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

## MODEL_EXTENSION_REAL (model_name, arguments, expression)

Returns a real result of an expression as calculated by a named model deployed on a TabPy external service.

Model_name is the name of the deployed analytics model you want to use.

Each argument is a single string that sets the input values that the deployed model accepts, and is defined by the analytics model.

Use expressions to define the values that are sent from Tableau to the analytics model. Be sure to use aggregation functions (SUM, AVG, etc.) to aggregate the results.

When using the function, the data types and order of the expressions must match that of the input arguments.

Example

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

## MODEL_EXTENSION_STRING (model_name, arguments, expression)

Returns the string result of an expression as calculated by a named model deployed on a TabPy external service.

Model_name is the name of the deployed analytics model you want to use.

Each argument is a single string that sets the input values that the deployed model accepts, and is defined by the analytics model.

Use expressions to define the values that are sent from Tableau to the analytics model. Be sure to use aggregation functions (SUM, AVG, etc.) to aggregate the results.

When using the function, the data types and order of the expressions must match that of the input arguments.

Example

MODEL_EXTENSION_STR ("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).

This function is the inverse of MODEL_QUANTILE. For information on predictive modeling functions, see How Predictive Modeling Functions Work in Tableau.

Example

The following formula returns the quantile of the mark for sum of sales, adjusted for count of orders.

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.

This function is the inverse of MODEL_PERCENTILE. For information on predictive modeling functions, see How Predictive Modeling Functions Work in Tableau.

Example

The following formula returns the median (0.5) predicted sum of sales, adjusted for count of orders.

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

## PREVIOUS_VALUE(expression)

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

Example

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

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

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

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

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

For information on different ranking options, see Rank calculation.

Example

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

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

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

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

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

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

For information on different ranking options, see Rank calculation.

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

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

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

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

For information on different ranking options, see Rank calculation.

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

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

With this function, the set of values (6, 9, 9, 14) would be ranked (0.00, 0.67, 0.67, 1.00).

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

For information on different ranking options, see Rank calculation.

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

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

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

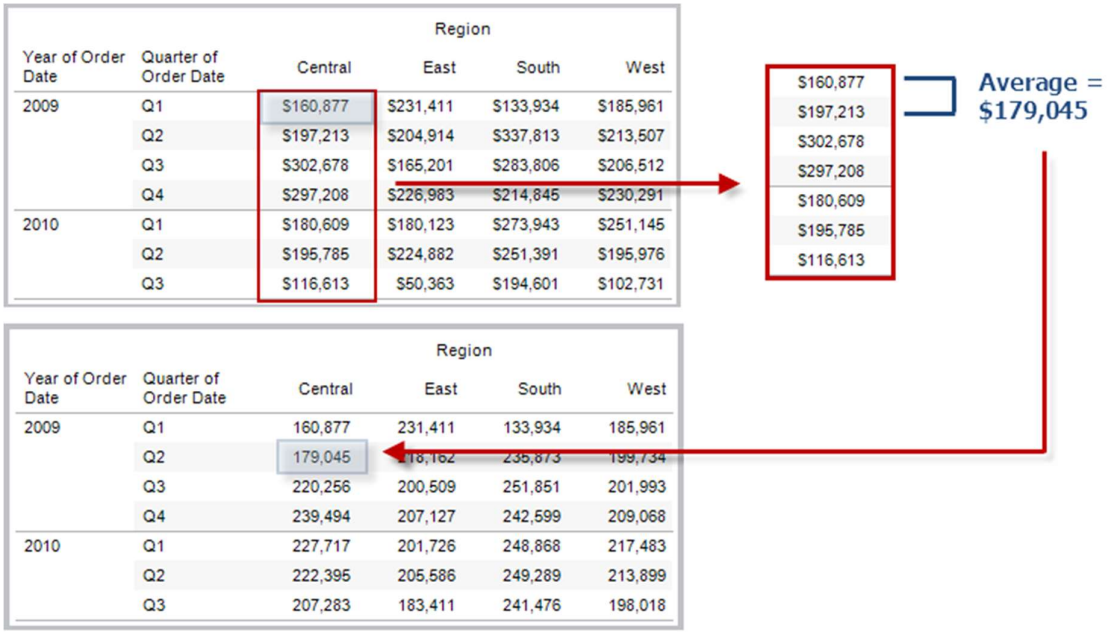
Nulls are ignored in ranking functions. They are not numbered and they do not count against the total number of records in percentile rank calculations.

For information on different ranking options, see Rank calculation.

## 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.




Example

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

## RUNNING_COUNT(expression)

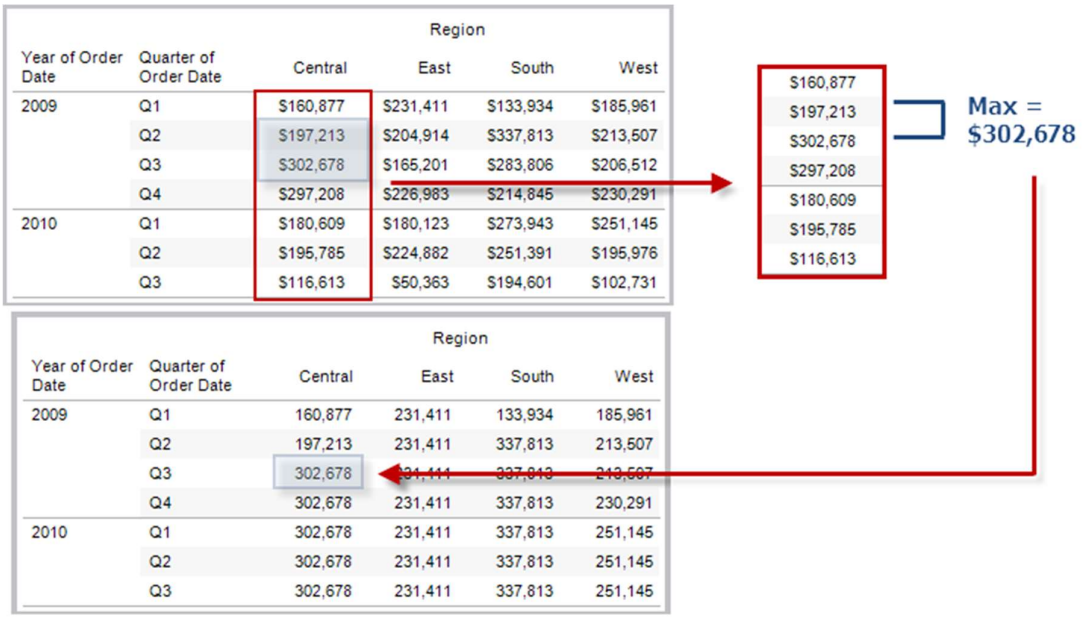
Returns the running count of the given expression, from the first row in the partition to the current row.

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.

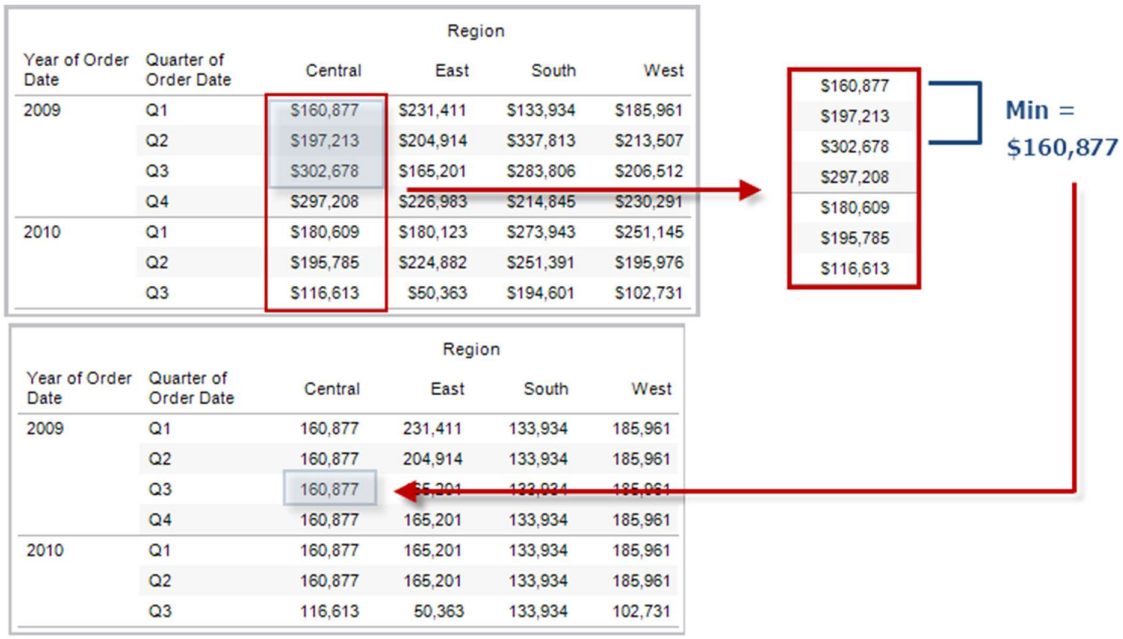



Example

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.

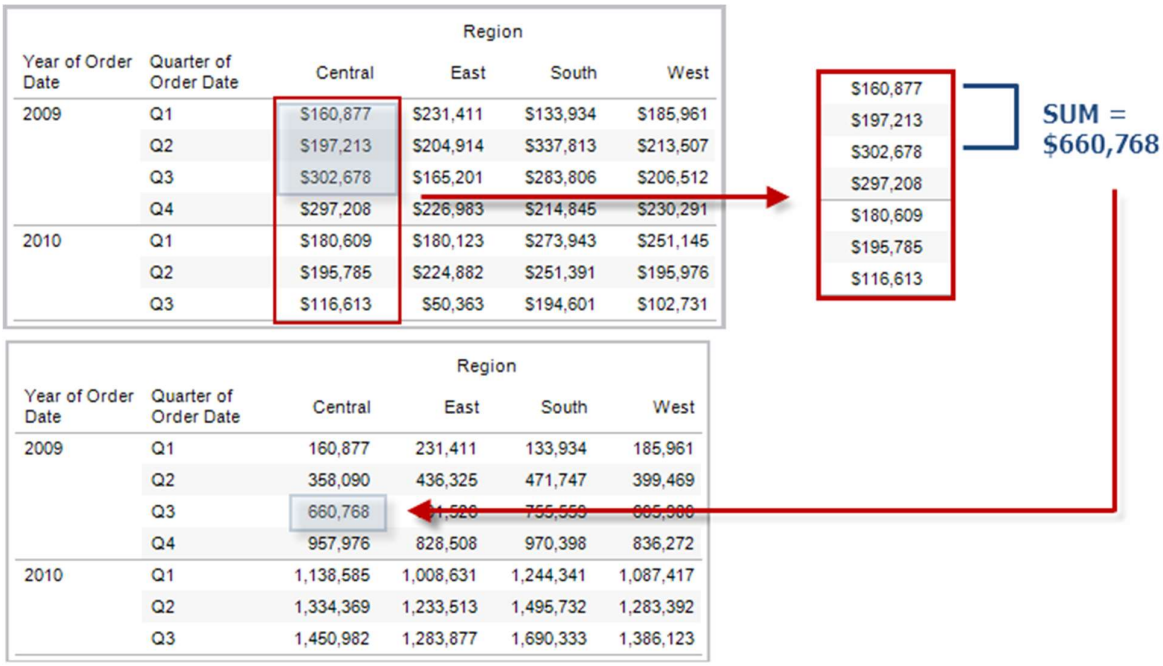



Example

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.

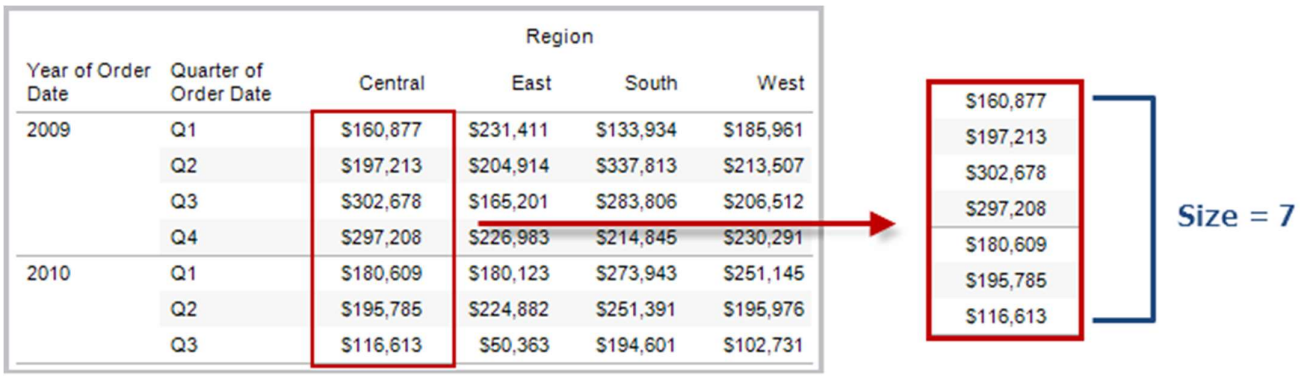



Example

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

## SIZE()

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




Example

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

## SCRIPT_BOOL

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

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

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

Examples

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

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

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

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

A command for Python would take this form:

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

## SCRIPT_INT

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

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

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

Examples

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

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

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

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

A command for Python would take this form:

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

## SCRIPT_REAL

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

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

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

Examples

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

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

The next example converts temperature values from Celsius to Fahrenheit.

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

A command for Python would take this form:

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

## SCRIPT_STR

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

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

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

Examples

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

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

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

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

A command for Python would take this form:

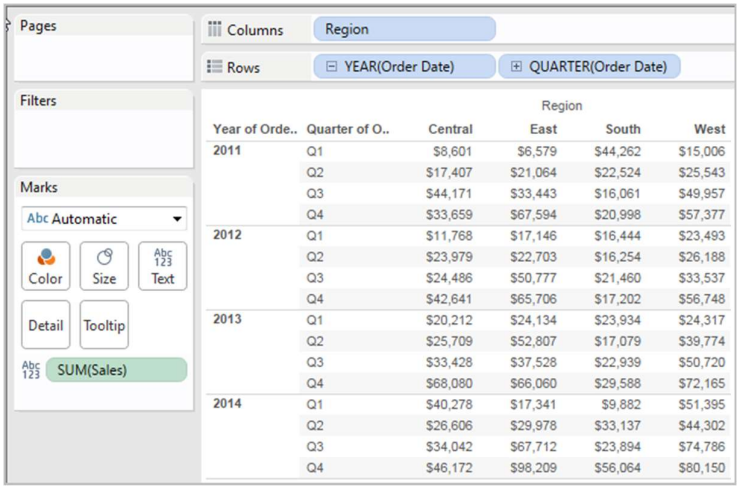
**SCRIPT_STR("return map(lambda x : x[:2], _arg1)", ATTR([Region]))**

## TOTAL(expression)

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

Example

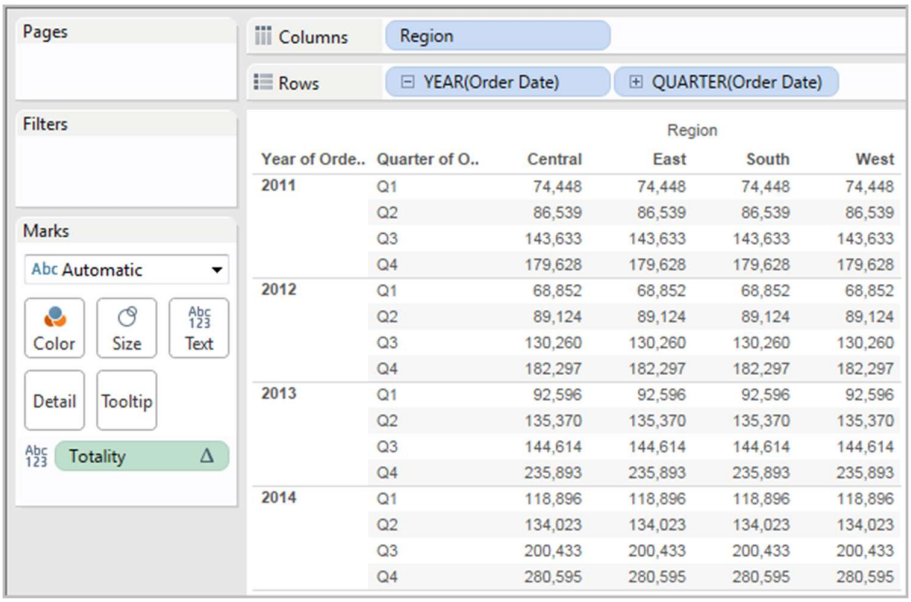
Assume you are starting with this view:




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

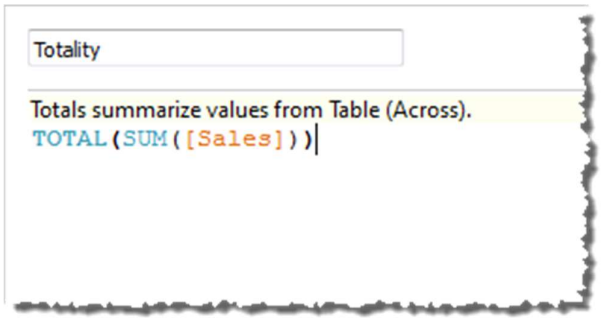
You open the calculation editor and create a new field which you name **Totality**:


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:




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

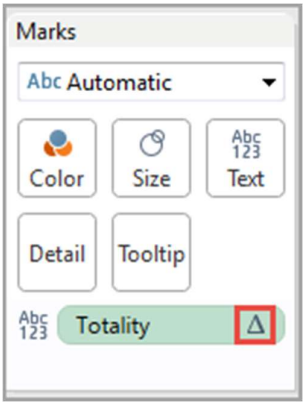
This raises the question, What is the default **Compute Using** value? If you right-click (Control-click on a Mac) **Totality** in the Data pane and choose **Edit**, there is now an additional bit of information available:



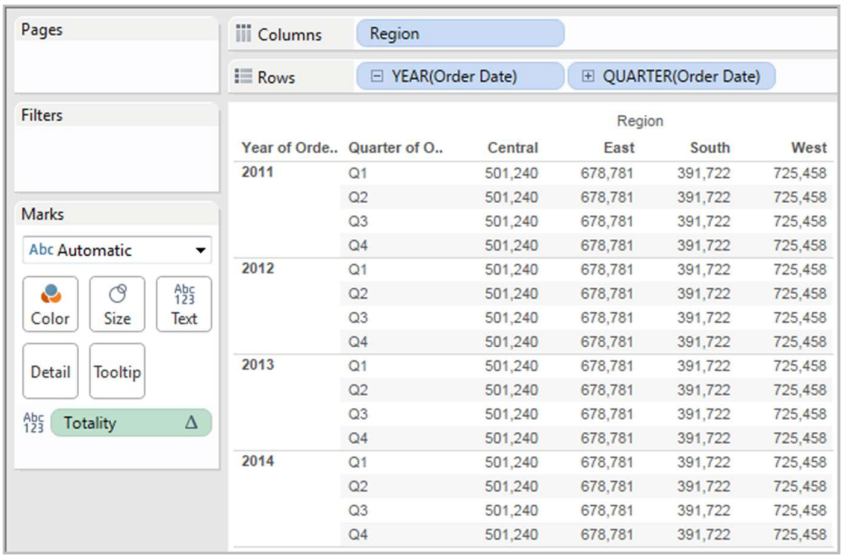

The default **Compute Using** value is **Table (Across)**. The result is that **Totality** is summing the values across each row of your table. Thus, the value that you see across each row is the sum of the values from the original version of the table.

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

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




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

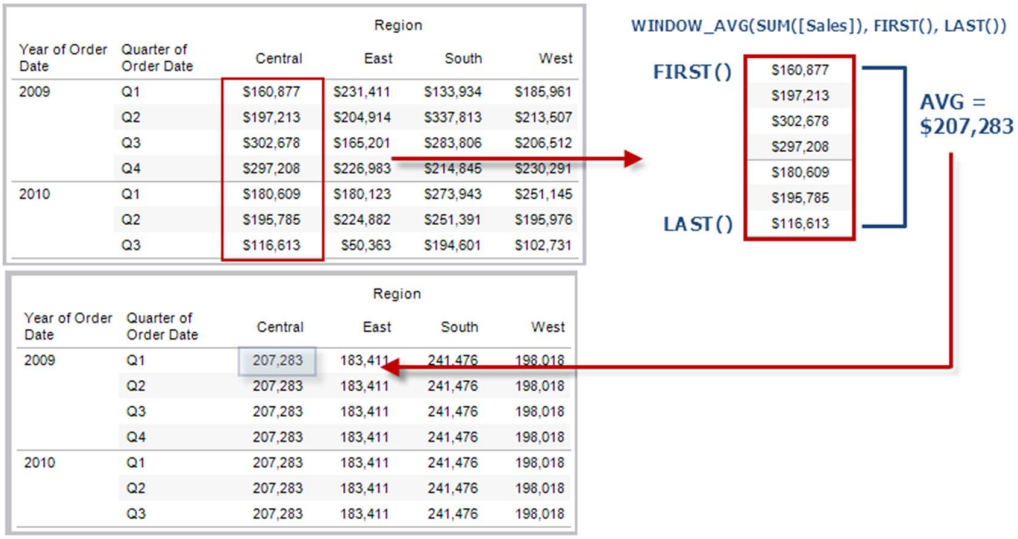



| | | Region | | | |
|---|---|---|---|---|---|
| Year of Orde.. | Quarter of O.. | Central | East | South | West |
| 2011 | Q1 | 501,240 | 678,781 | 391,722 | 725,458 |
| | Q2 | 501,240 | 678,781 | 391,722 | 725,458 |
| | Q3 | 501,240 | 678,781 | 391,722 | 725,458 |
| | Q4 | 501,240 | 678,781 | 391,722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

## WINDOW_AVG(expression, [start, end])

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

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




| | | 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 fuction: CORR. See Tableau Functions (Alphabetical)(Link opens in a new window).

Example

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

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

## WINDOW_COUNT(expression, [start, end])

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

Example

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

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

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

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

There is an equivalent aggregation fuction: COVAR. See Tableau Functions (Alphabetical)(Link opens in a new window).

Example

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

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

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

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

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

There is an equivalent aggregation fuction: COVARP. Tableau Functions (Alphabetical)(Link opens in a new window).

Example

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

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

## WINDOW_MEDIAN(expression, [start, end])

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

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


| Year of Order Date | Quarter of Order Date | Region: 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 |

| Year of Order Date | Quarter of Order Date | Region: 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 |

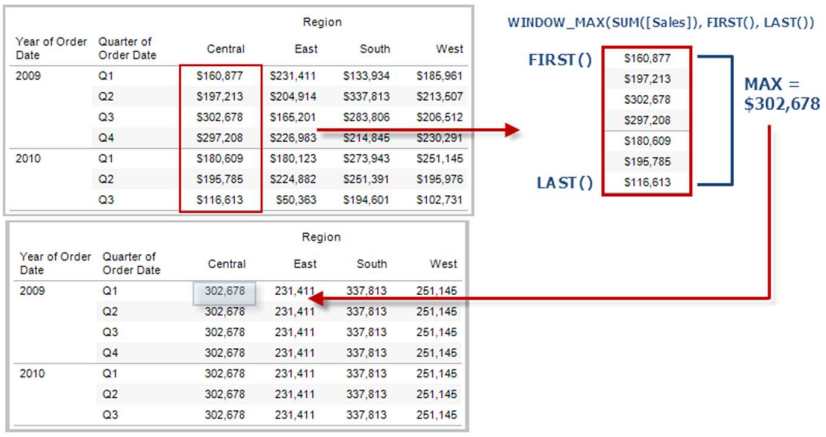
Example

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

## WINDOW_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.



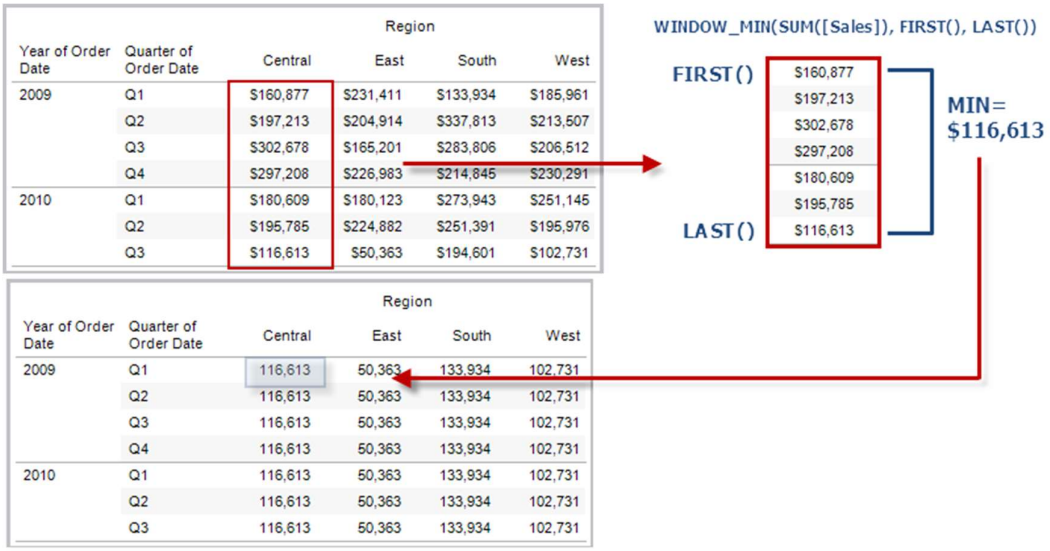

Example

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

## WINDOW_MIN(expression, [start, end])

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

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




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.

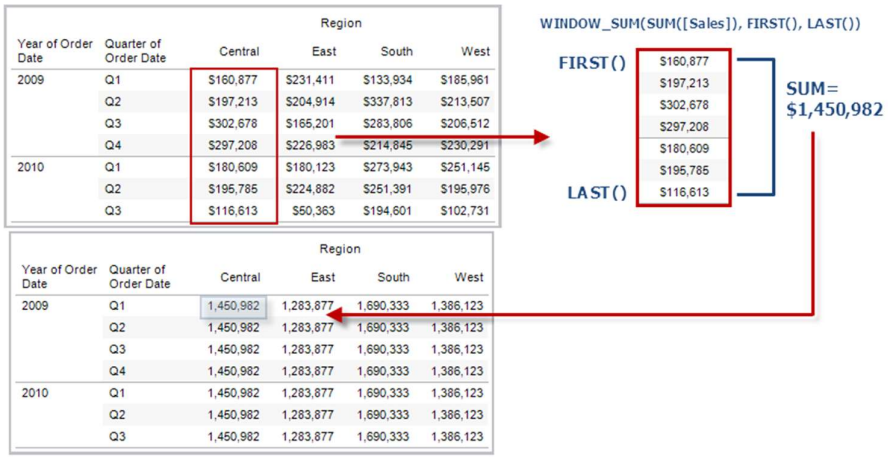
Example

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.




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

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

Example

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

## WINDOW_VAR(expression, [start, end])

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

Example

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

## WINDOW_VARP(expression, [start, end])

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

Example

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

**Create a table calculation using the calculation editor**

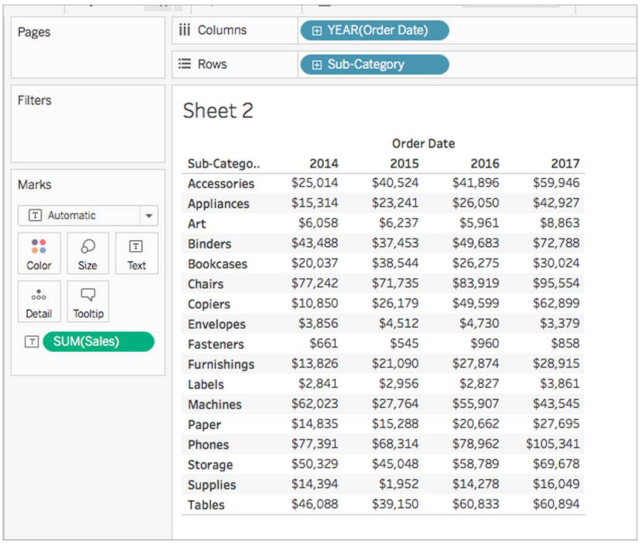
Follow along with the steps below to learn how to create a table calculation using the calculation editor.

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

Step 1: Create the visualization

1. In Tableau Desktop, connect to the **Sample-Superstore** saved data source, which comes with Tableau.
2. Navigate to a worksheet.
3. From the **Data** pane, under Dimensions, drag **Order Date** to the **Columns** shelf.
4. From the **Data** pane, under Dimensions, drag **Sub-Category** to the **Rows** shelf.
5. From the **Data** pane, under Measures, drag **Sales** to **Text** on the Marks card.

Your visualization updates to a text table.




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

Step 2: Create the table calculation

1. Select **Analysis** > **Create Calculated Field**.
2. In the calculation editor that opens, do the following:
    - Name the calculated field, Running Sum of Profit.
    - Enter the following formula:

RUNNING_SUM(SUM([Profit]))

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

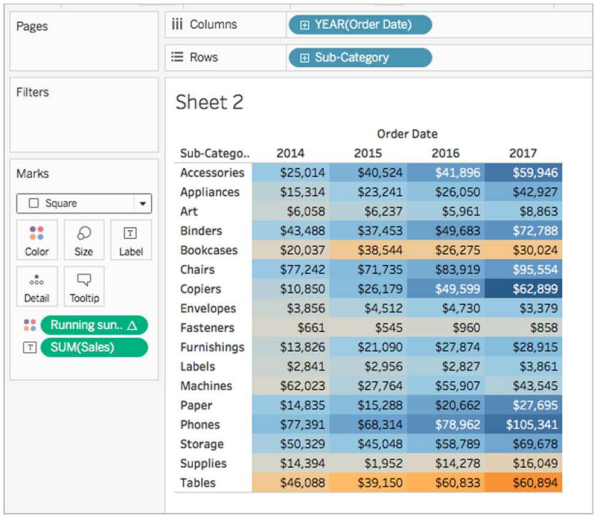
- When finished, click **OK**.

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

Step 3: Use the table calculation in the visualization

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

The visualization updates to a highlight table:

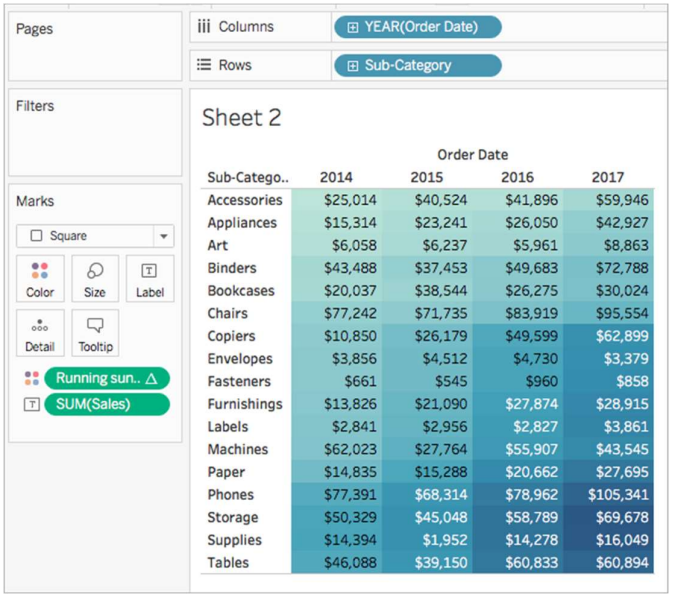



| | Order Date | | | |
|---|---|---|---|---|
| Sub-Catego.. | 2014 | 2015 | 2016 | 2017 |
| Accessories | $25,014 | $40,524 | $41,896 | $59,946 |
| Appliances | $15,314 | $23,241 | $26,050 | $42,927 |
| Art | $6,058 | $6,237 | $5,961 | $8,863 |
| Binders | $43,488 | $37,453 | $49,683 | $72,788 |
| Bookcases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

Step 4: Edit the table calculation

1. On the Marks card, right-click **Running Sum of Profit** and select **Edit Table Calculation**.
2. In the Table Calculation dialog box that opens, under Compute Using, select **Table (down)**.

The visualization updates to the following:



# Spatial Functions

*Applies to: Tableau Desktop*

Spatial functions allow you to perform advanced spatial analysis and combine spatial files with data in other formats like text files or spreadsheets. For example, you might have a spatial file of city council districts, and a text file containing latitude and longitude coordinates of reported potholes. You can use a spatial calculation when creating your data source to join these files and analyze which district takes the longest to repair potholes.

You can also create a line that connects two data points for origin-destination maps. For example, you might have a spreadsheet of public transportation data that tells you where commuters began and ended their trips. You can use a spatial calculation to see what paths commuters are taking.

List of spatial functions in Tableau

| Function | Syntax | Description |
|---|---|---|
| **AREA** | AREA(Spatial Polygon, "units") | Returns the total surface area of a spatial polygon.<br>Supported unit names: meters ("meters," "metres" "m"), kilometers ("kilometers," "kilometres," "km"), miles ("miles" or "mi"), feet ("feet," "ft"). |
| **BUFFER** | BUFFER(Spatial Point, distance, "units") | Returns a polygon shape with a radius determined by the distance and unit values defined in the calculation.<br>**Note:** The Buffer calculation will only work with a Point spatial object.<br>BUFFER supports the same unit names as the DISTANCE function. |
| **DISTANCE** | DISTANCE(<Spatial Point1>,<Spatial Point2>,"<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").<br>This function can only be created with a live connection and will continue to work when a data source is converted to an extract.<br>Example:<br>DISTANCE([Origin MakePoint],[Destination MakePoint], "km") |
| **INTERSECTS** | INTERSECTS (<geometry1>,<geometry2>) | Returns a Boolean (True/False) indicating if two geometries overlap in space.<br>Supported combinations: point/polygon, line/polygon, and polygon/polygon. |
| **MAKELINE** | MAKELINE(<Spatial Point1>,<Spatial Point2>) | Generates a line mark between two points; useful for building origin-destination maps.<br>Examples: |

| | | |
|---|---|---|
| | | MAKELINE(OriginPoint, DestinationPoint)<br><br>MAKELINE(MAKEPOINT(OriginLat],[OriginLong]),MAKEPOINT([DestinationLat],[DestinationLong]) |
| **MAKEPOINT** | MAKEPOINT(<latitude>, <longitude>) | Converts data from latitude and longitude columns into spatial objects.<br><br>You can use MAKEPOINT to spatially-enable a data source so that it can be joined with a spatial file using a spatial join. For more information, see Join Spatial Files in Tableau .<br><br>To use MAKEPOINT, your data must contain latitude and longitude coordinates.<br><br>Example:<br><br>MAKEPOINT([AirportLatitude],[AirportLongitude]) |
| **MAKEPOINT (X,Y, SRID)** | MAKEPOINT(<coordinatesX>, <coordinatesY>, <SRID> | Converts data from projected geographic coordinates into spatial objects. SRID is a spatial reference identifier that uses ESPG reference system codes to specify coordinate systems. If SRID is not specified, WGS84 is assumed and parameters are treated as latitude/longitude in degrees.<br><br>This function can only be created with a live connection and will continue to work when a data source is converted to an extract.<br><br>Example:<br><br>MAKEPOINT([Xcoord],[Ycoord],3493) |

Use a spatial calculation

**Create a spatial data source using MAKEPOINT**

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.

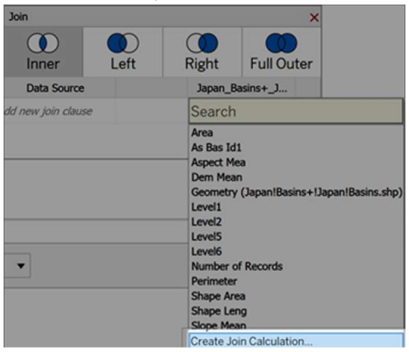
1. Open Tableau and connect to a spatial data source.
2. Under Connections, click Add to add a second, non-spatial data source.

The two data sources are added to the canvas.

Tip: To get the Join dialog box to appear, double-click (control click on Mac) a data source on the canvas.

3. Drag the non-spatial data source onto the Join dialog box.
4. Click the Join icon.
5. In the Join dialog box that appears, do the following:
    - Select a join type.
    - Under Data Source, select a spatial field from your spatial file to join by. Spatial fields have a globe icon next to them.

6. For the non-spatial data source, select Create Join Calculation as the join clause.




The calculation should look something like this:

MAKEPOINT(Latitude,Longitude)

7. Seleck **OK.**
8. Select the Intersects join clause operator to create a data source for spatial analysis.
9. When finished, close the Join dialog box.

For more information on spatial joins, see Join Spatial Files in Tableau .

Create a visualization using MAKELINE

In Tableau Desktop, download the Flight Path workbook from Tableau Public, available here(Link opens in a new window).

1. Navigate to a new worksheet.
2. Select Analysis > Create Calculated Field.
3. In the calculation that opens, do the following:
    - Name the calculated field Flight Paths
    - Enter the following formula

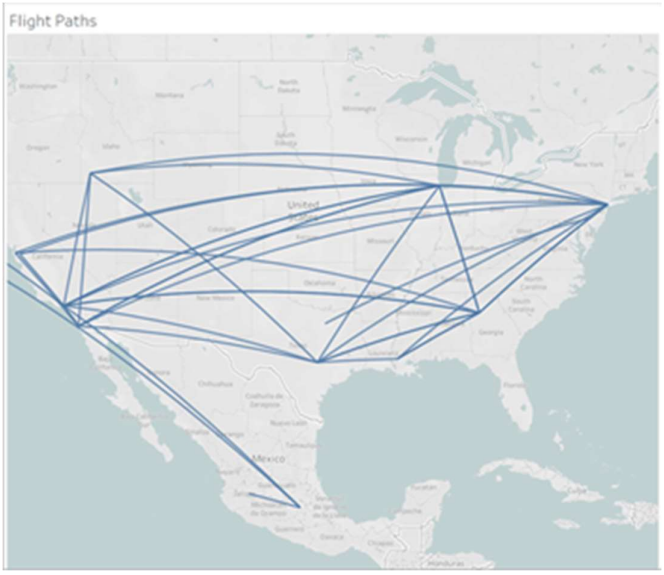
MAKELINE(MAKEPOINT([Lat],[Lng]),MAKEPOINT([Dest Lat],[Dest Lng]))

This formula takes latitude and longitude coordinates from your origin and destination cities and turns them into geographic points for spatial analysis. Those coordinates are used to build to two-point lines between origin and destination.

4. When finished, click **OK**.

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

5. From the Data pane, double-click Flight Paths to add it to your visualization, which should automatically render as a map.

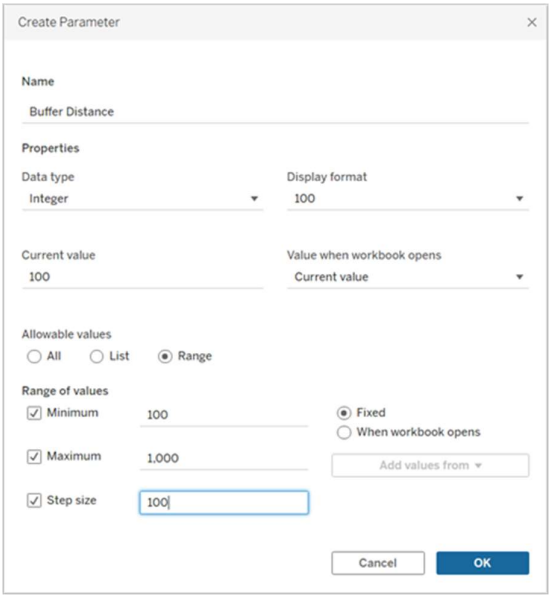



The calculation automatically produces curved geodesic lines when the lines span longer expanses of the globe.

Visualize an area with BUFFER

In Tableau Desktop, download the Flight Path workbook from Tableau Public, available here(Link opens in a new window).

1. Navigate to a new worksheet.
2. Right-click the Data pane and select **Create Parameter**.
3. In the Parameter dialog that opens, set the options below:
    - Name the parameter Buffer Distance
    - Set the Data Type to Integer
    - Set Allowable values to Range
    - Set the Minimum range to 100, the Maximum range to 1000, and the step size to 100.




4. When finished, click **OK.**

This parameter allows us to customize the radius of our buffer, ranging from 100 to 1000 miles. Right-click the parameter and select **Show Parameter.**

5. Select **Analysis** > **Create Calculated Field**.
6. In the calculation that opens, do the following:
    - Name the calculated field Buffer
    - Enter the following formula

BUFFER(MAKEPOINT([Dest Lat],[Dest Lng]),[Buffer Distance],"miles")

The BUFFER calculation takes point spatial data and converts it into shapes with a radius in miles determined by the Buffer Distance parameter.

**Note**: Because BUFFER can only be used with point spatial data, we're converting the latitude and longitude data into a point with Makepoint, as demonstrated in the previous example.

7. When finished, click **OK**.

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

8. From the Data pane, double-click **Buffer** to add it to your visualization, which should automatically render as a map.
9. Drag **Destination** to the Color panel on the Marks card to complete the visualization.


**Note:** If your view does not look like the image above, make sure that the Mark type is set to **Map** and not **Circle**. For more information on Marks type, see Change the Type of Mark in the View.

# Predictive Modeling Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

This article introduces predictive modeling functions and their uses in Tableau. It also demonstrates with an example how to create table calculations using the predictive modeling functions.

**Why use predictive modeling functions**

Predictive modeling functions can help you quickly generate predictions that can be manipulated, visualized, and exported like data using table calculations.

Before, you may have had to integrate Tableau with R and Python in order to perform advanced statistical calculations and visualize them in Tableau. Now, you can select targets and predictors by updating the variables and visualizing multiple models with different combinations of predictors. The data can be filtered, aggregated, and transformed at all levels of detail, with inputs and predictions automatically recalculated to match the data in the view.

For more information about predictive modeling functions in Tableau, see How Predictive Modeling Functions Work in Tableau

Predictive modeling functions available in Tableau:

| Function | Syntax | Description |
|---|---|---|
| **MODEL_QUANTILE** | MODEL_QUANTILE( model_specification (optional), 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.<br>Example:<br>MODEL_QUANTILE(0.5, SUM([Sales]),COUNT([Orders])) |
| **MODEL_PERCENTILE** | MODEL_PERCENTILE( model_specification (optional), 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).<br>Example:<br>MODEL_PERCENTILE( SUM([Sales]),COUNT([Orders])) |

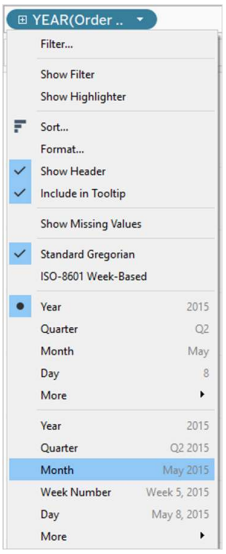
**Create a prediction calculation**

Follow along with the steps below to learn how to create a simple prediction calculation using the MODEL_QUANTILE function. For a more detailed example, see Example - Explore Female Life Expectancy with Predictive Modeling Functions

Step 1: Create a Visualization

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

3. From the **Data** pane, drag the **Order Date** dimension to the Columns shelf.
4. Open the measure's context menu to change its list level to Month and Year:




5. Drag **Sales** to the Rows shelf.

Step 2: Create the Calculated Field

1. Click to open the **Analysis** menu at the top, and then select **Create Calculated Field**.
2. In the Calculation Editor, do the following:
    - Name the calculation: **Predict Median Sales.**
    - Enter the following formula:

MODEL_QUANTILE(0.5, SUM([Sales]),ATTR(DATETRUNC('month', [Order Date])))

Remember: The MODEL_QUANTILE function takes a given quantile and predicts values based on the predictors you input.

Let's break this down:

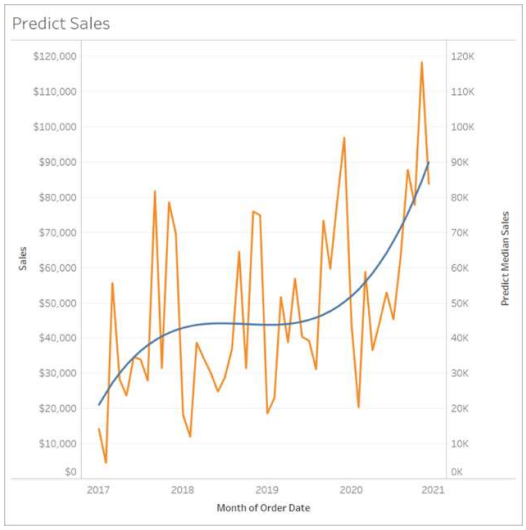
- In this case, the quantile = 0.5, which predicts the median.
- We want to predict sales, so the target expression is SUM([Sales]).
- We want to base the prediction on past performance, so we include date as a predictor, which is the last argument in the calculation.

3. When finished, click **OK**.

The prediction calculation is now added as a calculated field in the Data pane.

Step 3: Add the Prediction Calculation to the View

1. Drag the prediction calculation to the Rows shelf, to the right of SUM(Sales).
2. Right-click (Control-click on Mac) the measure and select **Dual Axis**.
3. To align the two axes in a dual axes chart to use the same scale, right-click (Control-click on Mac) the secondary axis, in this case **Predict Median Sales**, and select **Synchronize Axis**. This aligns the scale of the two axes.




That's all there is to it. To find out how you can extend a date axis and predict the future, see Predictive Modeling Functions in Time Series Visualizations.

Rules for Prediction Calculations

- You can't mix aggregate and non-aggregate arguments. If the target expression is an aggregate, so must the predictor.
- The functions are best used to predict values for individual records, on vizzes where each mark represents a discrete entity, such as a person, a product, a sale, etc.
- The functions are best used to predict values for aggregated target expressions using SUM and COUNT.
- The functions aren't recommended to predict values for aggregated target expressions using AVG, MEDIAN, MIN, or MAX.
- The functions should use predictors that are at the same level of detail or higher than the viz.

# Additional Functions

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

## 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 current International Components for Unicode (ICU), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the Regular Expressions(Link opens in a new window) page in the online ICU User Guide.

**Example**

REGEXP_REPLACE('abc 123', '\s', '-') = 'abc-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 current International Components for Unicode (ICU), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the Regular Expressions(Link opens in a new window) page in the online ICU User Guide.

**Example**

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

## 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 current International Components for Unicode (ICU), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the Regular Expressions(Link opens in a new window) 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 current International Components for Unicode (ICU), an open source project of mature C/C++ and Java libraries for Unicode support, software internationalization, and software globalization. See the Regular Expressions(Link opens in a new window) page in the online ICU User Guide.

**Example**

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

Hadoop Hive Specific Functions

**Note**: Only the PARSE_URL and PARSE_URL_QUERY functions are available for Cloudera Impala data sources.

**GET_JSON_OBJECT(JSON string, JSON path)**

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

## PARSE_URL(string, url_part)

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)

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'

## XPATH_BOOLEAN(XML string, XPath expression string)

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)

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)

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)

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)

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)

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)

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'

**Google BigQuery Specific Functions**

## DOMAIN(string_url)

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

**Example**

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

## GROUP_CONCAT(expression)

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"

## HOST(string_url)

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

**Example**

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

## LOG2(number)

Returns the logarithm base 2 of a number.

**Example**

LOG2(16) = '4.00'

## LTRIM_THIS(string, string)

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

**Example**

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

## RTRIM_THIS(string, string)

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

**Example**

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

## TIMESTAMP_TO_USEC(expression)

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

**Example**

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

## USEC_TO_TIMESTAMP(expression)

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

**Example**

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

## TLD(string_url)

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'

# FORMAT() Function Workarounds in Tableau

*Applies to: Tableau Cloud, Tableau Desktop, Tableau Public, Tableau Server*

Tableau has no FORMAT() function for formatting fields, but it does provide a variety of means to change the structure and appearance of fields in a workbook:

- For information on formatting geographic fields, see Assign Geographic Roles .
- For information on formatting date or number fields, see Set the default number format. And for information creating custom date formats, see Custom Date Formats.
- For information on symbols and conventions that you can use to specify field formats, see Literal expression syntax(Link opens in a new window).
- For information on formatting numbers and null values, see Format Numbers and Null Values.

Tableau also provides a range of string functions that you can use to customize the appearance of string fields in a view. See String Functions.

# Appendix

Tableau Functions (Alphabetical)- https://help.tableau.com/current/pro/desktop/en-us/functions_all_alphabetical.htm

Tableau Functions (by Category)- https://help.tableau.com/current/pro/desktop/en-us/functions_all_categories.htm

For Chart Types and its Importance- https://career.guru99.com/top-10-tableau-interview-questions/

Practical Question- https://www.simplilearn.com/tableau-interview-questions-and-answers-article#tableau_interview_questions_for_experienced

Relationships Vs Joins- https://help.tableau.com/current/pro/desktop/en-us/joining_tables.htm

https://help.tableau.com/current/pro/desktop/en-us/datasource_relationships_learnmorepage.htm