# POWER BI LAB - Data Analysis Expressions (DAX) I

# THEORY: INTRODUCTION TO DATA ANALYSIS EXPRESSIONS

## What is DAX?

DAX is a collection of functions that can be used in a formula, or expression, to calculate and return one or more values. DAX helps you create new information from data already in your model.

## Why is DAX so important?

It’s easy to create a new Power BI Desktop file and import some data into it. You can even create reports that show valuable insights without using any DAX formulas at all. But, what if you need to analyze growth percentage across product categories and for different date ranges? Or, you need to calculate year-over-year growth compared to market trends? DAX formulas provide this capability and many other important capabilities as well. Learning how to create effective DAX formulas will help you get the most out of your data. When you get the information you need, you can begin to solve real business problems that affect your bottom line.

**Familiarity with Excel Functions will help a lot**

You might already be familiar with creating formulas in Microsoft Excel, and that knowledge will be extremely helpful in understanding DAX. But even if you have no experience with Excel formulas, the concepts described here will help you get started creating DAX formulas and solving real-world BI problems right away.

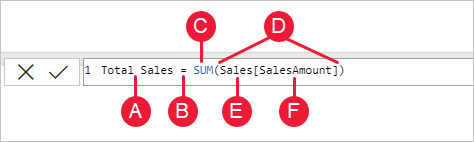
We’ll focus on understanding DAX formulas used in calculations, more specifically, in measures and calculated columns. You should already be familiar with using Power BI Desktop to import data and add fields to a report.

## Understanding DAX Functions

We'll frame our understanding of DAX around three fundamental concepts: *Syntax*, *Functions*, and *Context*. There are other important concepts in DAX, but understanding these three concepts will provide the best foundation on which to build your DAX skills.

## Syntax

Before you create your own formulas, let’s take a look at DAX formula syntax. Syntax includes the various elements that make up a formula, or more simply, how the formula is written. For example, here's a simple DAX formula for a measure:



This formula includes the following syntax elements:

**A.** The measure (field) name, **Total Sales**.

**B.** The equals sign operator (**=**), which indicates the beginning of the formula. When calculated, it will return a result.

**C.** The DAX function **SUM**, which adds up all of the numbers in the **Sales[SalesAmount]** column. You’ll learn more about functions later.

**D.** Parenthesis **()**, which surround an expression that contains one or more arguments. Most functions require at least one argument. An argument passes a value to a function.

**E.** The referenced table, **Sales**.

**F.** The referenced column, **[SalesAmount]**, in the Sales table. With this argument, the SUM function knows on which column to aggregate a SUM.

When trying to understand a DAX formula, it's often helpful to break down each of the elements into a language you think and speak every day. For example, you can read this formula as:

*For the measure named Total Sales, calculate (=) the SUM of values in the [SalesAmount ] column in the Sales table.*

When added to a report, this measure calculates and returns values by summing up sales amounts for each of the other fields we include, for example, Cell Phones in the USA.

Let’s go over a few more things about this formula. In particular, we introduced the SUM() function. Functions are pre-written formulas that make it easier to do complex calculations and manipulations with numbers, dates, time, text, and more.

## Functions vs formulas

Functions are predefined formulas that perform calculations by using specific values, called arguments, in a particular order or structure. Arguments can be other functions, another formula, expression, column references, numbers, text, logical values such as TRUE or FALSE, or constants.

DAX includes the following categories of functions: [Date and Time](https://learn.microsoft.com/en-us/dax/date-and-time-functions-dax), [Time Intelligence](https://learn.microsoft.com/en-us/dax/time-intelligence-functions-dax), [Information](https://learn.microsoft.com/en-us/dax/information-functions-dax), [Logical](https://learn.microsoft.com/en-us/dax/logical-functions-dax), [Mathematical](https://learn.microsoft.com/en-us/dax/math-and-trig-functions-dax), [Statistical](https://learn.microsoft.com/en-us/dax/statistical-functions-dax), [Text](https://learn.microsoft.com/en-us/dax/text-functions-dax), [Parent/Child](https://learn.microsoft.com/en-us/dax/parent-and-child-functions-dax), and [Other](https://learn.microsoft.com/en-us/dax/other-functions-dax) functions. If you’re familiar with functions in Excel formulas, many of the functions in DAX will appear similar to you; however, DAX functions are unique in the following ways:

* A DAX function always references a complete column or a table. If you want to use only particular values from a table or column, you can add filters to the formula.
* If you need to customize calculations on a row-by-row basis, DAX provides functions that let you use the current row value or a related value as a kind of argument to perform calculations based on the context. You'll learn more about context later.
* DAX includes many functions that return a table rather than a value. The table isn't displayed, but is used to provide input to other functions. For example, you can retrieve a table and then count the distinct values in it, or calculate dynamic sums across filtered tables or columns.
* DAX includes various time intelligence functions. These functions let you define or select date ranges, and perform dynamic calculations based on them. For example, you can compare sums across parallel periods.
* Excel has a popular function, VLOOKUP. DAX functions don’t take a cell or cell range as a reference like VLOOKUP does in Excel. DAX functions take a column or a table as a reference. Keep in mind, in Power BI Desktop you’re working with a relational data model. Looking up values in another table is easy, and in most cases you don’t need to create any formulas at all.

As you can see, functions in DAX can help you create powerful formulas. We only touched on the basics of functions. As your DAX skills grow, you'll create formulas by using many different functions. One of the best places to learn details about each of the DAX functions is in the [DAX Function Reference](https://learn.microsoft.com/en-us/dax/).

## Reference of DAX Functions

<https://learn.microsoft.com/en-us/dax/dax-function-reference>

You also see that the column name [SalesAmount] was preceded by the Sales table in which the column belongs. This name is known as a fully qualified column name in that it includes the column name preceded by the table name. Columns referenced in the same table don't require the table name be included in the formula, which can make long formulas that reference many columns shorter and easier to read. However, it's a good practice to include the table name in your measure formulas, even when in the same table.

**Note**

If a table name contains spaces, reserved keywords, or disallowed characters, you must enclose the table name in single quotation marks. You’ll also need to enclose table names in quotation marks if the name contains any characters outside the ANSI alphanumeric character range, regardless of whether your locale supports the character set or not.

It’s important your formulas have the correct syntax. In most cases, if the syntax isn't correct, a syntax error is returned. In other cases, the syntax might be correct, but the values returned might not be what you're expecting. The DAX editor in Power BI Desktop includes a suggestions feature, used to create syntactically correct formulas by helping you select the correct elements.

Let’s create an example formula. This task will help you further understand formula syntax and how the suggestions feature in the formula bar can help you.

## Functions QuickQuiz

1. What does a function always reference?
2. Can a formula contain more than one function?
3. What category of functions would you use to concatenate two text strings into one string?

Answers are provided at the end of this article.

## Context

Context is one of the most important DAX concepts to understand. There are two types of context in DAX: row context and filter context. We’ll first look at row context.

## Row context

Row context is most easily thought of as the current row. It applies whenever a formula has a function that applies filters to identify a single row in a table. The function will inherently apply a row context for each row of the table over which it's filtering. This type of row context most often applies to measures.

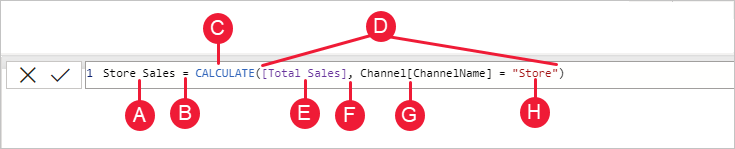
## Filter context

Filter context is a little more difficult to understand than row context. You can most easily think of filter context as: One or more filters applied in a calculation that determines a result or value.

Filter context doesn’t exist in place of row context; rather, it applies in addition to row context. For example, to further narrow down the values to include in a calculation, you can apply a filter context, which not only specifies the row context, but also specifies a particular value (filter) in that row context.

Filter context is easily seen in your reports. For example, when you add TotalCost to a visualization, and then add Year and Region, you're defining a filter context that selects a subset of data based on a given year and region.

Why is filter context so important to DAX? You've seen that filter context can be applied by adding fields to a visualization. Filter context can also be applied in a DAX formula by defining a filter with functions such as ALL, RELATED, FILTER, CALCULATE, by relationships, and by other measures and columns. For example, let’s look at the following formula in a measure named Store Sales:



To better understand this formula, we can break it down, much like with other formulas.

This formula includes the following syntax elements:

**A.** The measure name, **Store Sales**.

**B.** The equals sign operator (**=**), which indicates the beginning of the formula.

**C.** The **CALCULATE** function, which evaluates an expression, as an argument, in a context that is modified by the specified filters.

**D.** Parenthesis **()**, which surround an expression containing one or more arguments.

**E.** A measure **[Total Sales]** in the same table as an expression. The Total Sales measure has the formula: =SUM(Sales[SalesAmount]).

**F.** A comma (**,**), which separates the first expression argument from the filter argument.

**G.** The fully qualified referenced column, **Channel[ChannelName]**. This is our row context. Each row in this column specifies a channel, such as Store or Online.

**H.** The particular value, **Store**, as a filter. This is our filter context.

This formula ensures only sales values defined by the Total Sales measure are calculated only for rows in the Channel[ChannelName] column, with the value *Store* used as a filter.

As you can imagine, being able to define filter context within a formula has immense and powerful capabilities. The ability to reference only a particular value in a related table is just one such example. Don’t worry if you don't completely understand context right away. As you create your own formulas, you'll better understand context and why it’s so important in DAX.

## Context QuickQuiz

1. What are the two types of context?
2. What is filter context?
3. What is row context?

Answers are provided at the end of this article.

Summary

Now that you have a basic understanding of the most important concepts in DAX, you can begin creating DAX formulas for measures on your own. DAX can indeed be a little tricky to learn, but there are many resources available to you. After reading through this article and experimenting with a few of your own formulas, you can learn more about other DAX concepts and formulas that can help you solve your own business problems. There are many DAX resources available to you; most important is the [Data Analysis Expressions (DAX) Reference](https://learn.microsoft.com/en-us/dax/).

Because DAX has been around for several years in other Microsoft BI tools such as Power Pivot and Analysis Services Tabular models, there are many great sources information out there. You can find more information in books, whitepapers, and blogs from both Microsoft and leading BI professionals. The [DAX Resource Center Wiki on TechNet](https://social.technet.microsoft.com/wiki/contents/articles/dax-resource-center.aspx) is also a great place to start.

# THEORY: CALCULATED COLUMNS in POWER BI

Sometimes the data you’re analyzing does not contain a field or fields that you need to obtain your desired results. In this case, we can use concatenated fields or calculated fields. For example, let’s say your data has a **City** and a **State** field, but you want a single **Location** field that has both, like "Miami, FL". We obtain the location field by concatenating the city and State fields.

With calculated columns, you can add new data to a table already in your model. But instead of querying and loading values into your new column from a data source, you create a Data Analysis Expressions (DAX) formula that defines the column's values. In Power BI Desktop, calculated columns are created by using the new column feature in **Report** view, **Data** view, or **Model** view.

Calculated columns are created in **Report** view, **Data** view, or **Model** view and are based on data you've already loaded into the model. Calculated columns you create appear in the **Fields** list just like any other field, but they'll have a special icon showing its values are the result of a formula. You can name your columns whatever you want and add them to a report visualization just like other fields.

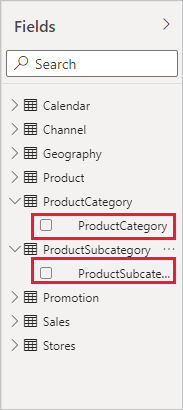
# LAB 1: CONCATENATED COLUMNS in DAX

## The Data

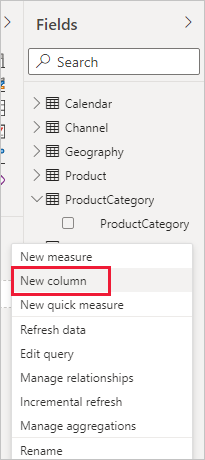
This lab uses the Contoso Sales Data which is a Power BI Desktop file. Download the file from the week 5 folder.

## Concatenated columns with fields from related tables

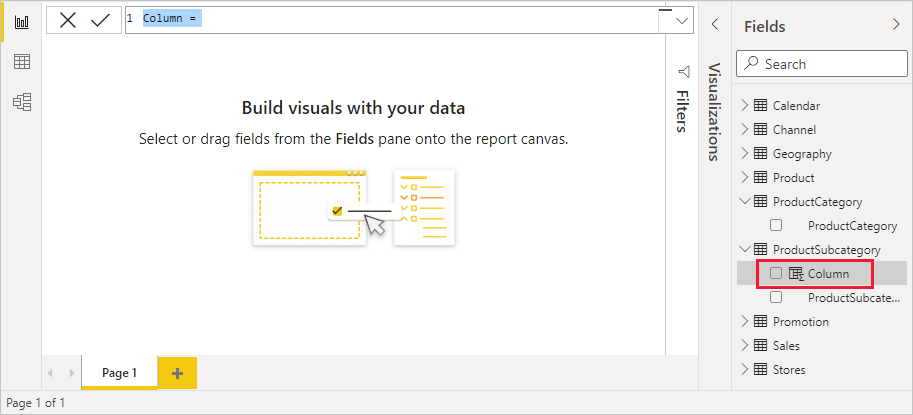
In your Sales Report, you want to display product categories and subcategories as single values, like "Cell phones – Accessories", "Cell phones – Smartphones & PDAs", and so on. There's no field in the **Fields** list that gives you that data, but there's a **ProductCategory** field and a **ProductSubcategory** field, each in its own table. You can create a calculated column that combines values from these two columns. DAX formulas can use the full power of the model you already have, including relationships between different tables that already exist.



1. Create a new page in your report and name it **Concatenated Fields**.
2. To create your new column in the **ProductSubcategory** table, right-click or select the ellipsis **...** next to **ProductSubcategory** in the **Fields** pane and choose **New column** from the menu.

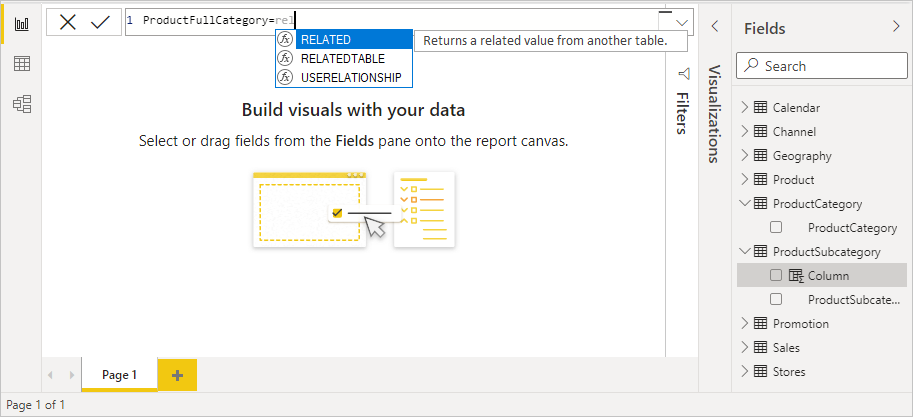


When you choose **New column**, the **Formula bar** appears along the top of the Report canvas, ready for you to name your column and enter a DAX formula.

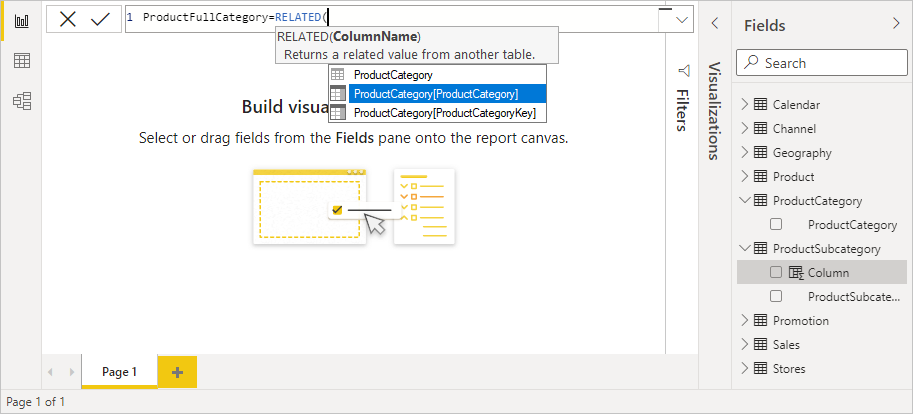


1. By default, a new calculated column is named **Column**. If you don’t rename it, new columns will be named **Column 2**, **Column 3**, and so on. You want your column to be more identifiable, so while the **Column** name is already highlighted in the formula bar, rename it by typing **ProductFullCategory**, and then type an equals (**=**) sign.
2. You want the values in your new column to start with the name in the **ProductCategory** field which is in a different but related table. You can use the [RELATED](https://learn.microsoft.com/en-us/dax/related-function-dax) function to help you get it.

After the equals sign, type **re** or **rel**. A dropdown suggestion list shows all of the DAX functions beginning with the letter R. Selecting each function shows a description of its effect. As you type, the suggestion list scales closer to the function you need. Select **RELATED**, and then press **Enter**.



An opening parenthesis appears, along with another suggestion list of the related columns you can pass to the RELATED function, with descriptions and details of expected parameters.



1. You want the **ProductCategory** column from the **ProductCategory** table. Select **ProductCategory[ProductCategory]**, press **Enter**, and then type a closing parenthesis.

**Tip**

Syntax errors are most often caused by a missing or misplaced closing parenthesis, although sometimes Power BI Desktop will add it for you.

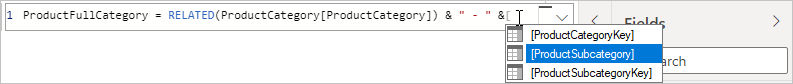
1. You want dashes and spaces to separate the **ProductCategories** and **ProductSubcategories** in the new values, so after the closing parenthesis of the first expression, type a space, ampersand (**&**), double-quote (**"**), space, dash (**-**), another space, another double-quote, and another ampersand. Your formula should now look like this:

ProductFullCategory = RELATED(ProductCategory[ProductCategory]) & " - " &

**Tip**

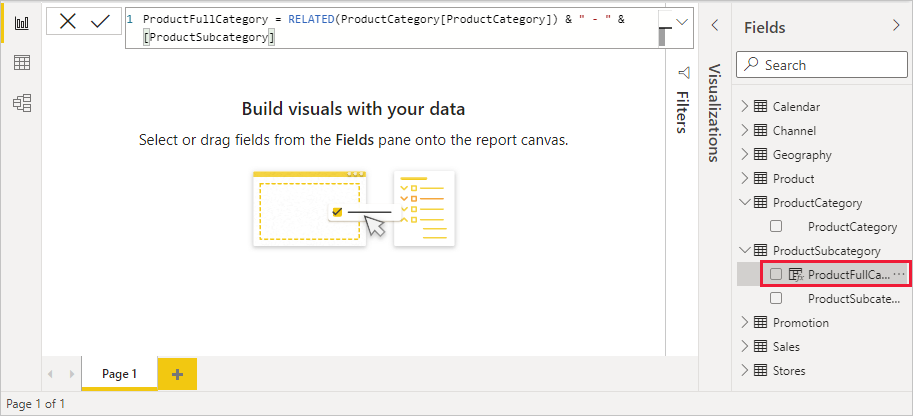
If you need more room, select the down chevron on the right side of the formula bar to expand the formula editor. In the editor, press **Alt + Enter** to move down a line, and **Tab** to move things over.

1. Enter an opening bracket (**[**), and then select the **[ProductSubcategory]** column to finish the formula.



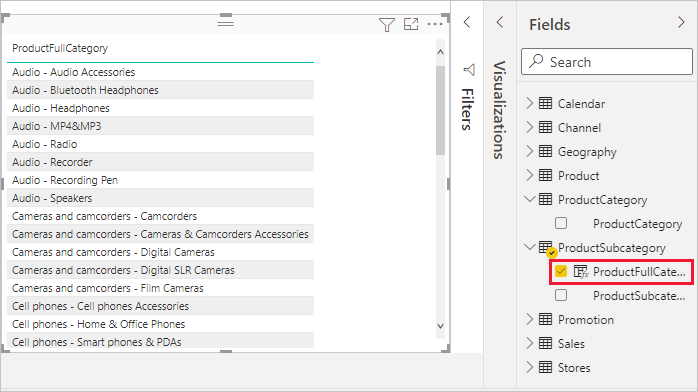
You didn’t need to use another RELATED function to call the **ProductSubcategory** table in the second expression, because you're creating the calculated column in this table. You can enter **[ProductSubcategory]** with the table name prefix (fully qualified) or without (non-qualified).

1. Complete the formula by pressing **Enter** or selecting the checkmark to the left of the formula bar. The formula validates, and the **ProductFullCategory** column name appears in the **ProductSubcategory** table in the **Fields** pane.

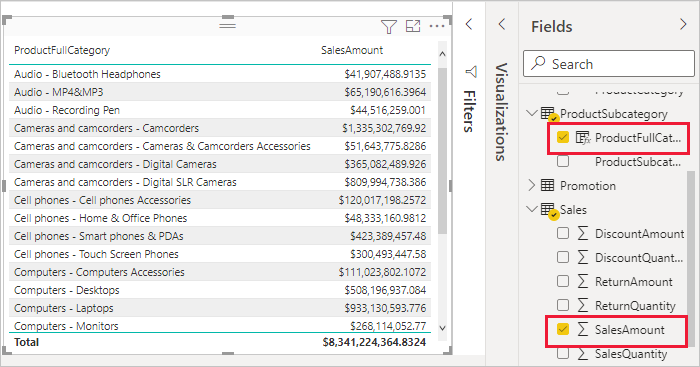


Now you can use your new **ProductFullCategory** column to look at **SalesAmount** by **ProductFullCategory**.

1. Select or drag the **ProductFullCategory** column from the **ProductSubcategory** table onto the Report canvas to create a table showing all of the **ProductFullCategory** names.



1. Select or drag the **SalesAmount** field from the **Sales** table into the table to show the **SalesAmount** for each **ProductFullCategory**.



# LAB 2: CONDITIONAL FUNCTIONS in DAX

## The Data

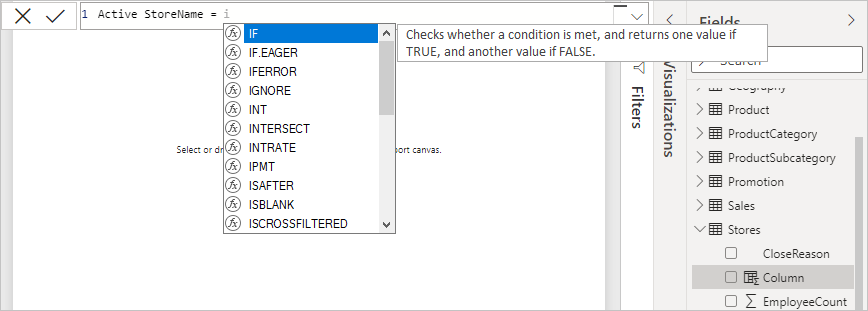
This lab uses the Contoso Sales Data which is a Power BI Desktop file. Download the file from the Week 5 folder.

## **The if() function**

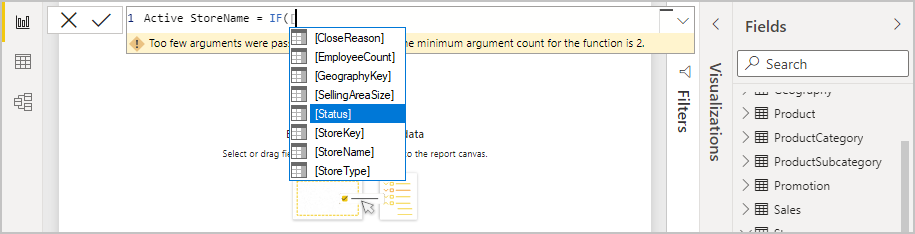
The Contoso Sales Sample contains sales data for both active and inactive (closed) stores. You want to ensure that active store sales are clearly separated from inactive store sales in your report by creating an **Active StoreName** field. In the new **Active StoreName** calculated column, each active store will appear with the store's full name, while the sales for inactive stores will be grouped together in one line item called **Inactive**.

Fortunately, the **Stores** table has a column named **Status**, with values of "On" for active stores and "Off" for inactive stores, which we can use to create values for our new **Active StoreName** column. Your DAX formula will use the logical [IF](https://learn.microsoft.com/en-us/dax/if-function-dax) function to test each store's **Status** and return a particular value depending on the result. If a store's **Status** is "On", the formula will return the store's name. If it’s "Off", the formula will assign an **Active StoreName** of "Inactive".

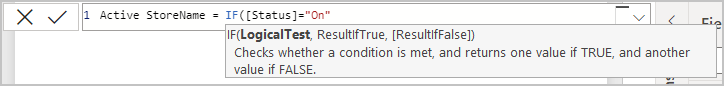
1. Create a new page in your report and name it **Conditional Function**.
2. Create a new calculated column in the **Stores** table and name it **Active StoreName** in the formula bar. To do so right click the table name or click the ellipsis ... to the right of the table name and choose New column from the menu.
3. After the **=** sign, begin typing **IF**. The suggestion list will show what you can add. Select **IF**.



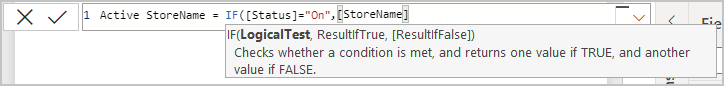
1. The first argument for **IF** is a logical test of whether a store's **Status** is "On". Type an opening bracket **[**, which lists columns from the **Stores** table, and select **[Status]**.



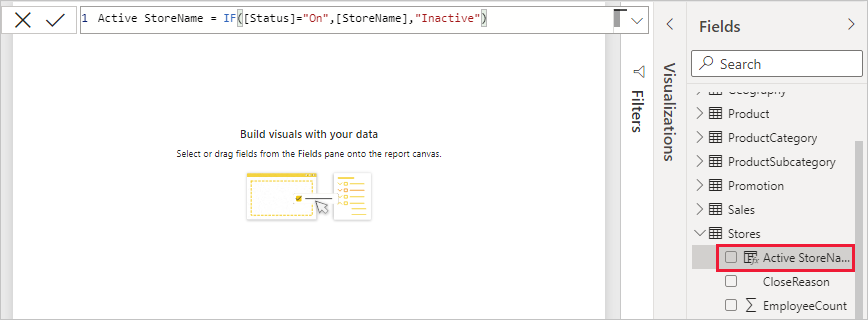
1. Right after **[Status]**, type **="On"**, and then type a comma (**,**) to end the argument. The tooltip suggests that you now need to add a value to return when the result is TRUE.



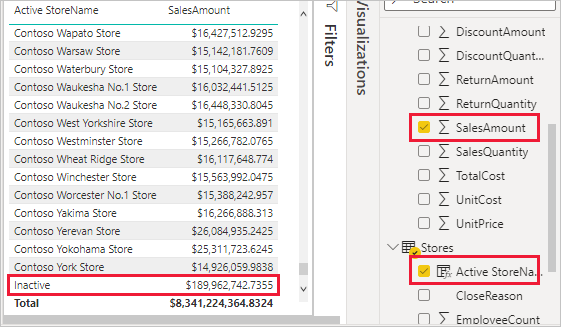
1. If the store's status is "On", you want to show the store’s name. Type an opening bracket (**[**) and select the **[StoreName]** column, and then type another comma. The tooltip now indicates that you need to add a value to return when the result is FALSE.



1. You want the value to be "Inactive", so type **"Inactive"**, and then complete the formula by pressing **Enter** or selecting the checkmark in the formula bar. The formula validates, and the new column's name appears in the **Stores** table in the **Fields** pane.



1. You can use your new **Active StoreName** column in visualizations just like any other field. To show **SalesAmounts** by **Active StoreName**, select the **Active StoreName** field or drag it onto the Report canvas, and then choose the **SalesAmount** field from the Sales table or drag it into the report area. In this table, active stores appear individually by name, but inactive stores are grouped together at the end as **Inactive**.



# LAB 3: AUTOMATIC MEASURES in DAX

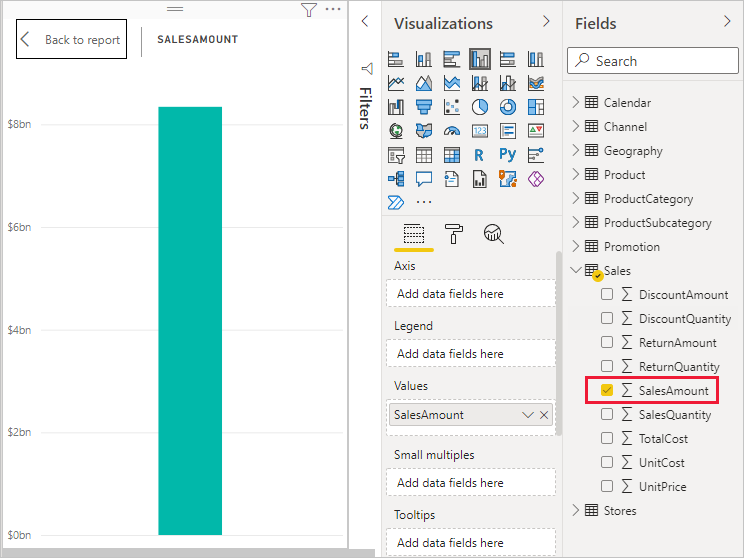
**The Data**

This lab uses the Contoso Sales Data which is a Power BI Desktop file. Download the file from the Week 5 folder.

Measures help you by performing calculations on your data as you interact with your reports. When Power BI Desktop creates a measure, it's most often created for you automatically. Follow these steps to see how Power BI Desktop creates an automatic measure:

1. In Power BI Desktop, select **File** > **Open**, browse to the *ContosoSales.pbix* file, and then choose **Open**.
2. Create a new page and name it **Measures**.
3. In the **Fields** pane, expand the **Sales** table. Then, either select the check box next to the **SalesAmount** field or drag **SalesAmount** onto the report canvas.

A new column chart visualization appears, showing the sum total of all values in the **SalesAmount** column of the **Sales** table.



Any field (column) in the **Fields** pane with a sigma icon Sigma icon is numeric, and its values can be aggregated. Rather than display a table with many values (2,000,000 rows for **SalesAmount**), Power BI Desktop automatically creates and calculates a measure to aggregate the data if it detects a numeric datatype. Sum is the default aggregation for a numeric datatype, but you can easily apply different aggregations like average or count. Understanding aggregations is fundamental to understanding measures, because every measure performs some type of aggregation.

To change the chart aggregation, follow these steps:

1. Select the **SalesAmount** visualization in the report canvas.
2. In the **Visualizations** pane, in the **Y-Axis** textbox , click the down arrow to the right of **SalesAmount**.
3. From the menu that appears, select **Average**.

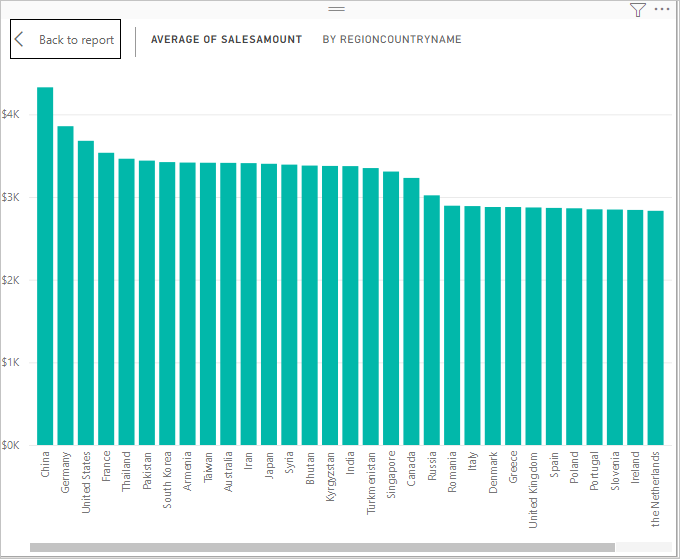
The visualization changes to an average of all sales values in the **SalesAmount** field.

A screenshot of a computer

Description automatically generated

Depending on the result you want, you can change the type of aggregation. However, not all types of aggregation apply to every numeric datatype. For example, for the **SalesAmount** field, Sum and Average are useful, and Minimum and Maximum have their place as well. However, Count doesn't make sense for the **SalesAmount** field, because while its values are numeric, they’re really currency.

Values calculated from measures change in response to your interactions with your report. For example, if you drag the **RegionCountryName** field from the **Geography** table onto your existing **SalesAmount** chart, it changes to show the average sales amounts for each country/region.



When the result of a measure changes because of an interaction with your report, you've affected your measure’s *context*. Every time you interact with your report visualizations, you're changing the context in which a measure calculates and displays its results.

# LAB 4: CUSTOM MEASURES in DAX

**The Data**

This lab uses the Contoso Sales Data which is a Power BI Desktop file. Download the file from the Week 5 folder.

## Create a Custom Measure using DAX

In some cases, you might want to create your own measures to perform more complex, unique calculations. With Power BI Desktop, you can create your own measures with the Data Analysis Expressions (DAX) formula language.

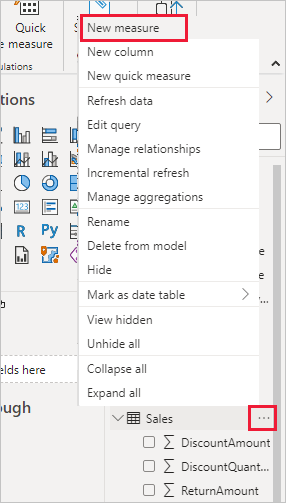
DAX formulas use many of the same functions, operators, and syntax as Excel formulas. However, DAX functions are designed to work with relational data and perform more dynamic calculations as you interact with your reports. There are over 200 DAX functions that do everything from simple aggregations like sum and average to more complex statistical and filtering functions.

When you create your own measure, it's called a ***model* measure**, and it's added to the **Fields** list for the table you select. Some advantages of model measures are that you can name them whatever you want, making them more identifiable. You can use them as arguments in other DAX expressions, and you can make them perform complex calculations quickly.

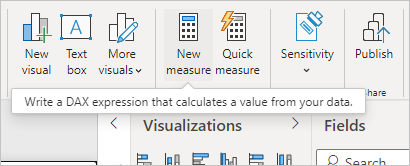
Let us visualize your net sales by subtracting discounts and customer returns from the total sales amounts. For the context that exists in your visualization, you need a measure that subtracts the sum of DiscountAmount and ReturnAmount from the sum of SalesAmount. There's no field for Net Sales in the **Fields** list, but you have the building blocks to create your own measure to calculate net sales.

1. Create a new page in your report and name it **Custom Measure**.
2. In the **Fields** pane, right-click the **Sales** table, or hover over the table and select **More options** (**...**). From the menu that appears, choose **New measure**.

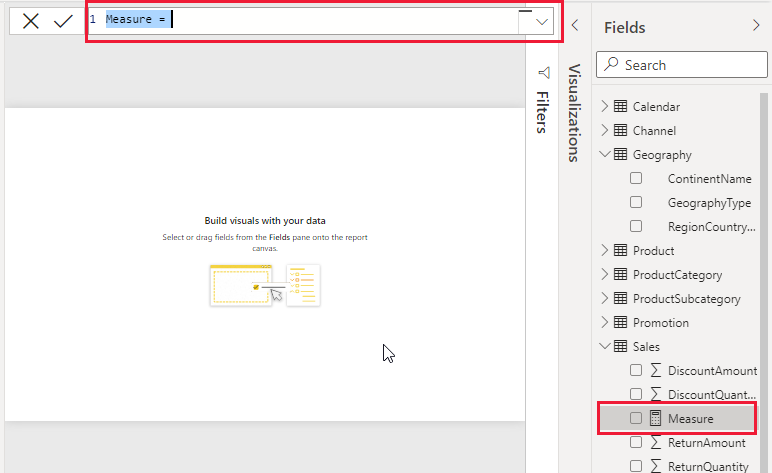
This action saves your new measure in the **Sales** table, where it's easy to find.



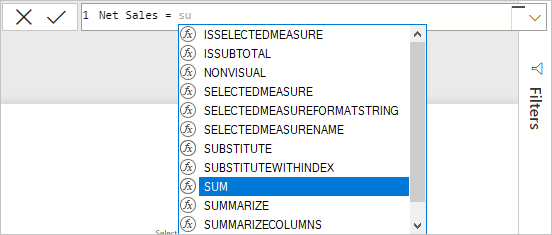
You can also create a new measure by selecting **New Measure** in the **Calculations** group on the **Home** tab of the Power BI Desktop ribbon.



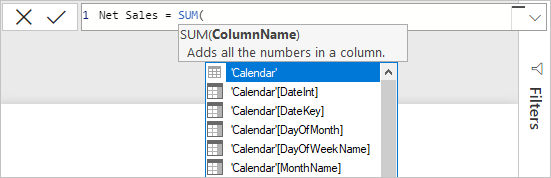
The formula bar appears along the top of the report canvas, where you can rename your measure and enter a DAX formula.



1. By default, each new measure is named *Measure*. If you don’t rename it, new measures are named *Measure 2*, *Measure 3*, and so on. Because we want this measure to be more identifiable, highlight *Measure* in the formula bar, and then change it to *Net Sales*.
2. Begin entering your formula. After the equals sign, start to type *Sum*. As you type, a drop-down suggestion list appears, showing all the DAX functions, beginning with the letters you type. Scroll down, if necessary, to select **SUM** from the list, and then press **Enter**.

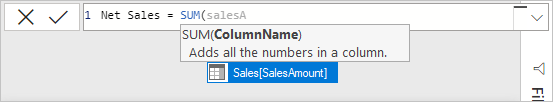


An opening parenthesis appears, along with a drop-down suggestion list of the available columns you can pass to the SUM function.



1. Expressions always appear between opening and closing parentheses. For this example, your expression contains a single argument to pass to the SUM function: the **SalesAmount** column. Begin typing *SalesAmount* until **Sales(SalesAmount)** is the only value left in the list.

The column name preceded by the table name is called the fully qualified name of the column. Fully qualified column names make your formulas easier to read.



1. Select **Sales[SalesAmount]** from the list, and then enter a closing parenthesis.

**Tip**

Syntax errors are most often caused by a missing or misplaced closing parenthesis.

1. Subtract the other two columns inside the formula:

a. After the closing parenthesis for the first expression, type a space, a minus operator (-), and then another space.

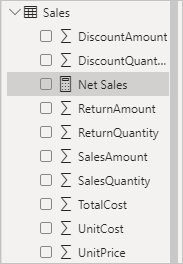
b. Enter another SUM function, and start typing *DiscountAmount* until you can choose the **Sales[DiscountAmount]** column as the argument. Add a closing parenthesis.

c. Type a space, a minus operator, a space, another SUM function with **Sales[ReturnAmount]** as the argument, and then a closing parenthesis.

Screenshot of the complete formula.

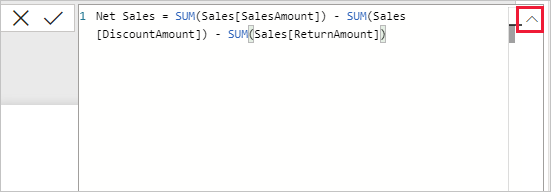
1. Press **Enter** or select **Commit** (checkmark icon) in the formula bar to complete and validate the formula.

The validated **Net Sales** measure is now ready to use in the **Sales** table in the **Fields** pane.

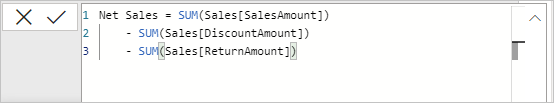


1. If you run out of room for entering a formula or want it on separate lines, select the down arrow on the right side of the formula bar to provide more space.

The down arrow turns into an up arrow and a large box appears.



1. Separate parts of your formula by pressing **Alt** + **Enter** for separate lines, or pressing **Tab** to add tab spacing.

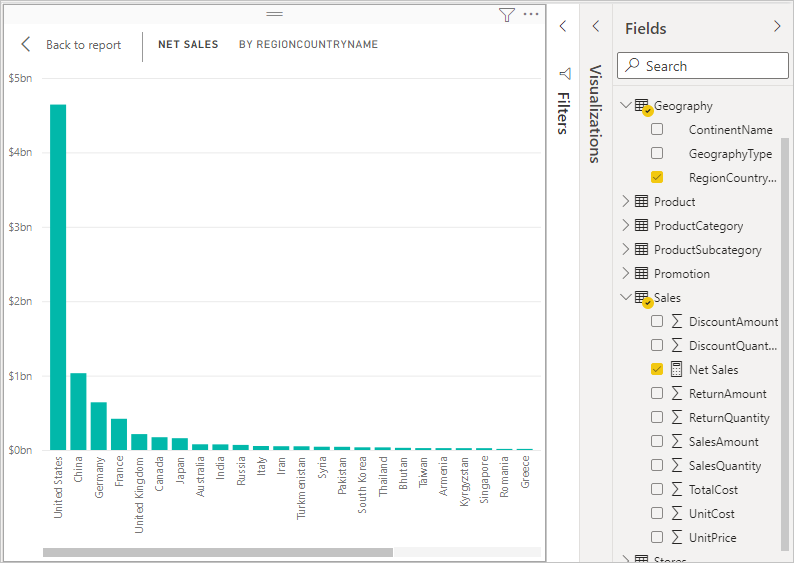


1. Your new measure is ready and you can see if as a field in the Sales table!

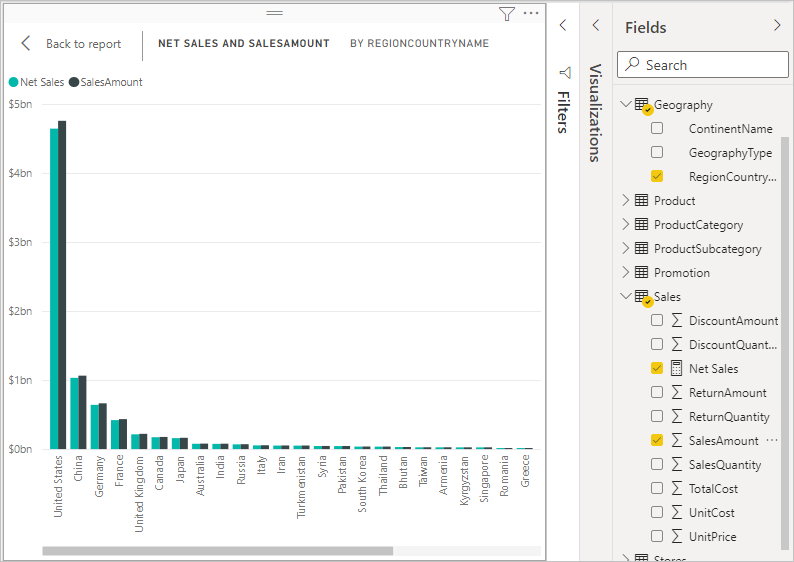
## Use Custom Measures in Reports

To look at net sales by country/region:

1. Select the **Net Sales** measure from the **Sales** table, or drag it onto the report canvas.
2. Select the **RegionCountryName** field from the **Geography** table, or drag it onto the **Net Sales** chart.



1. To see the difference between net sales and total sales by country/region, select the **SalesAmount** field from the Sales table or drag it onto the chart.

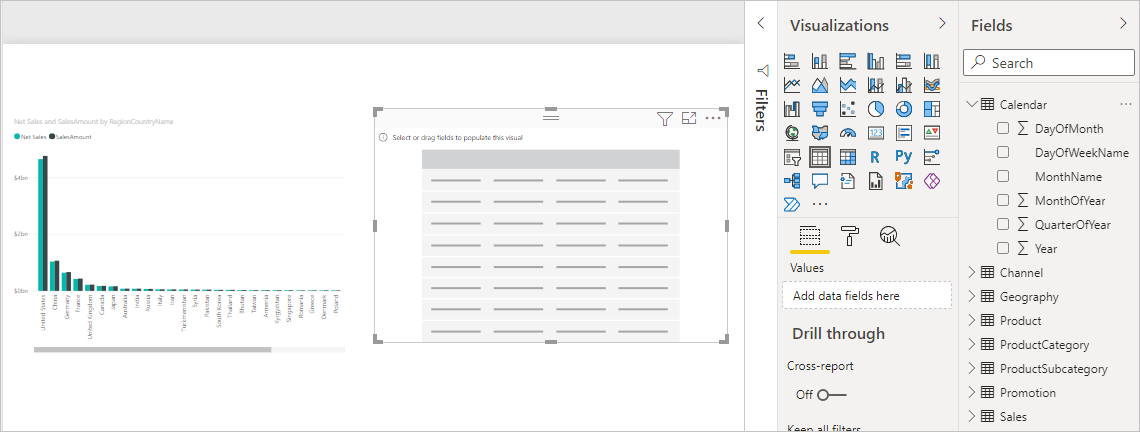


The chart now uses two measures: **SalesAmount**, which Power BI summed automatically, and the **Net Sales** measure, which you manually created. Each measure was calculated in the context of another field, **RegionCountryName**.

## Use Slicers with Custom Measures

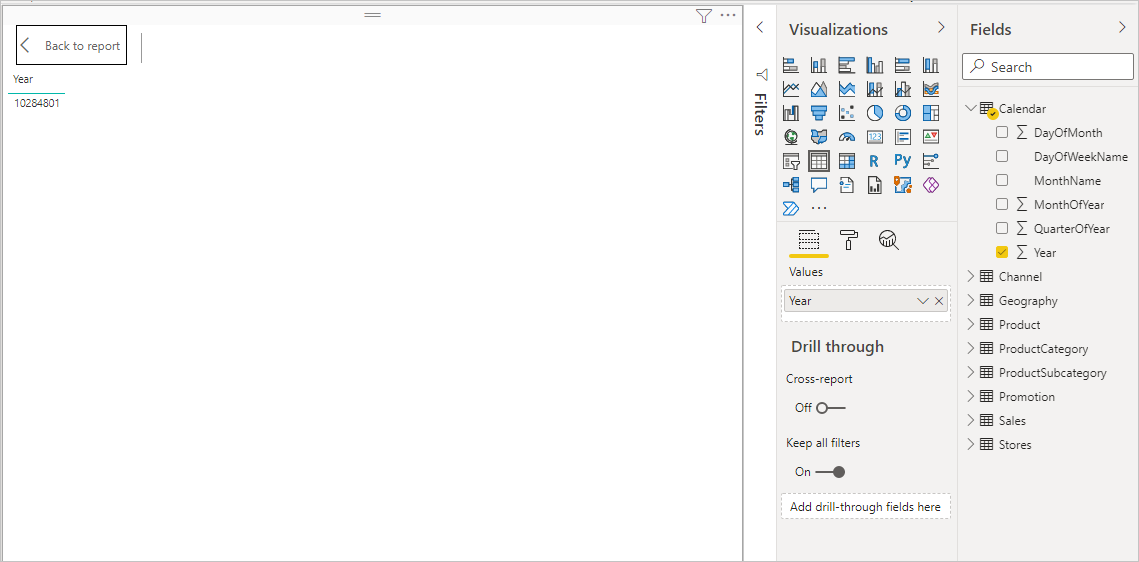
Add a slicer to further filter net sales and sales amounts by calendar year:

1. Select a blank area next to the chart. In the **Visualizations** pane, select the **Table** visualization. This action creates a blank table visualization on the report canvas.

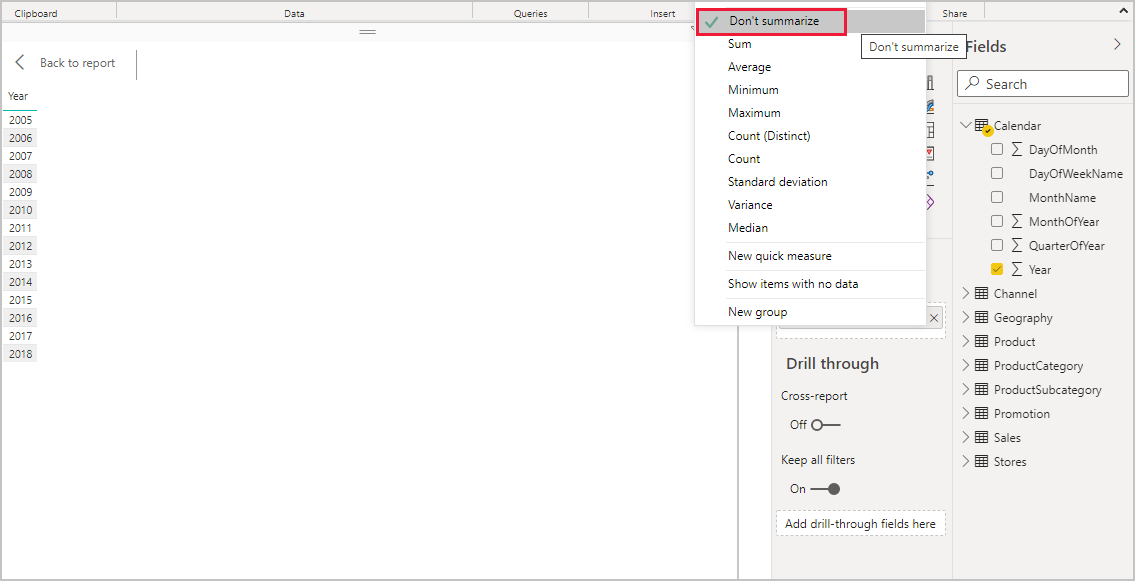


1. Drag the **Year** field from the **Calendar** table onto the new blank table visualization.

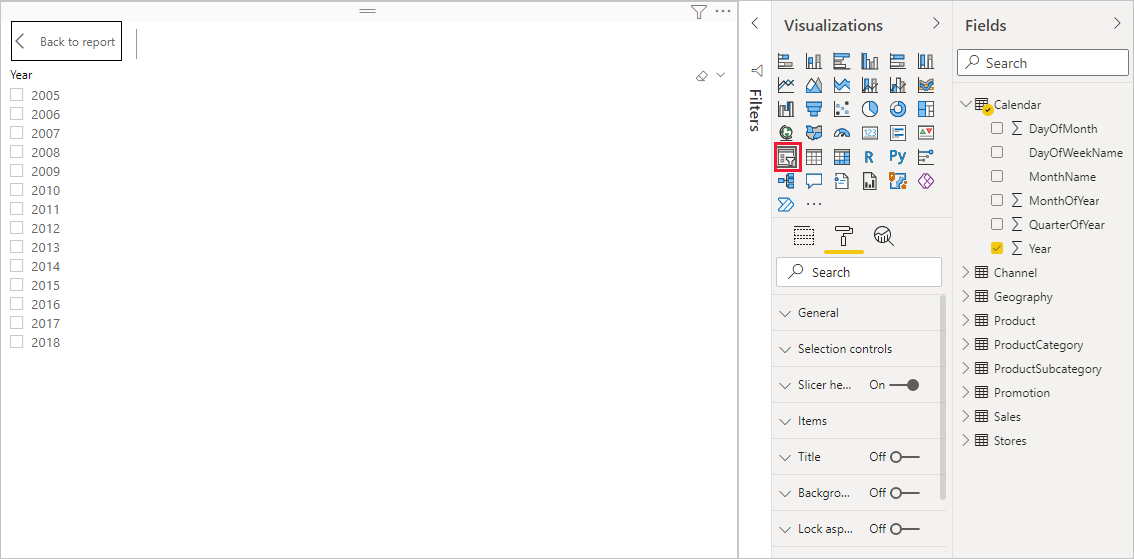
Because **Year** is a numeric field, Power BI Desktop sums up its values. This summation doesn’t work well as an aggregation; we'll address that in the next step.



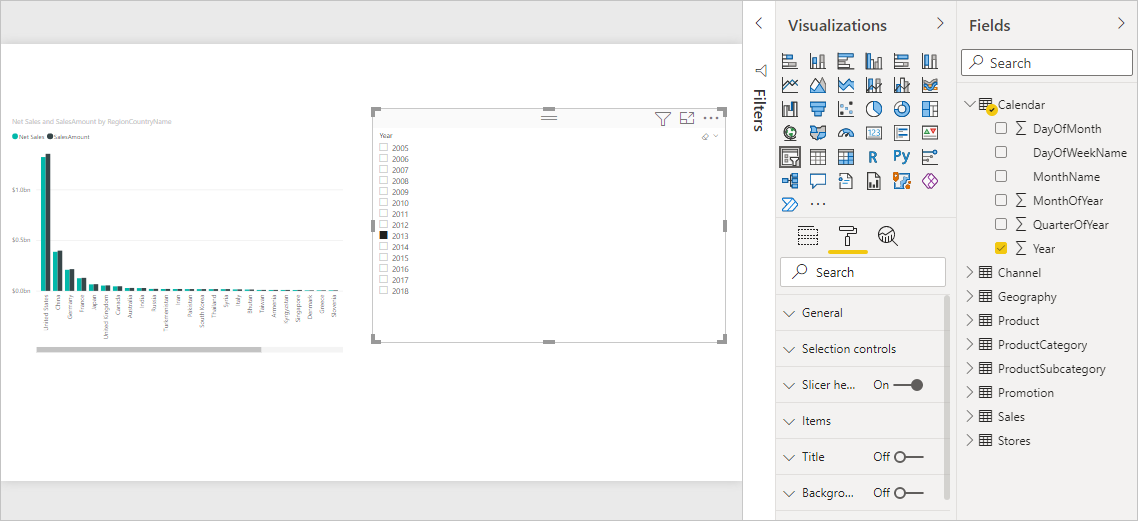
1. in the **Visualizations** pane, in the Build Visual tab, select the down arrow next to **Year**, and then choose **Don't summarize** from the list. The table now lists individual years.



1. Select the **Slicer** icon in the **Visualizations** pane to convert the table to a slicer. If the visualization displays a slider instead of a list, go to Format/Slicer **Settings/ Options/ Style** and select **Vertical List**.



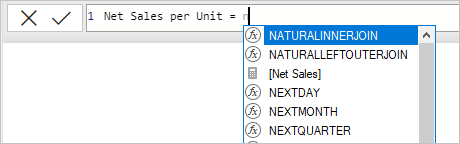
1. Select any value in the **Year** slicer to filter the **Net Sales and Sales Amount by RegionCountryName** chart accordingly. The **Net Sales** and **SalesAmount** measures recalculate and display results in the context of the selected **Year** field.



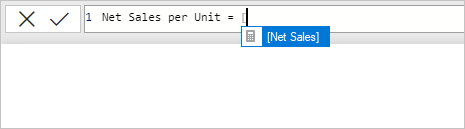
## Combine Custom Measures with Other Measures

Suppose you want to find out which products have the highest net sales amount per unit sold. You'll need a measure that divides net sales by the quantity of units sold. Create a new measure that divides the result of your **Net Sales** measure by the sum of **Sales[SalesQuantity]**.

1. In the **Fields** pane, create a new measure named **Net Sales per Unit** in the **Sales** table.
2. In the formula bar, begin typing *Net Sales*. The suggestion list shows what you can add. Select **[Net Sales]**.



1. You can also reference measures by just typing an opening bracket (**[**). The suggestion list shows only measures to add to your formula.



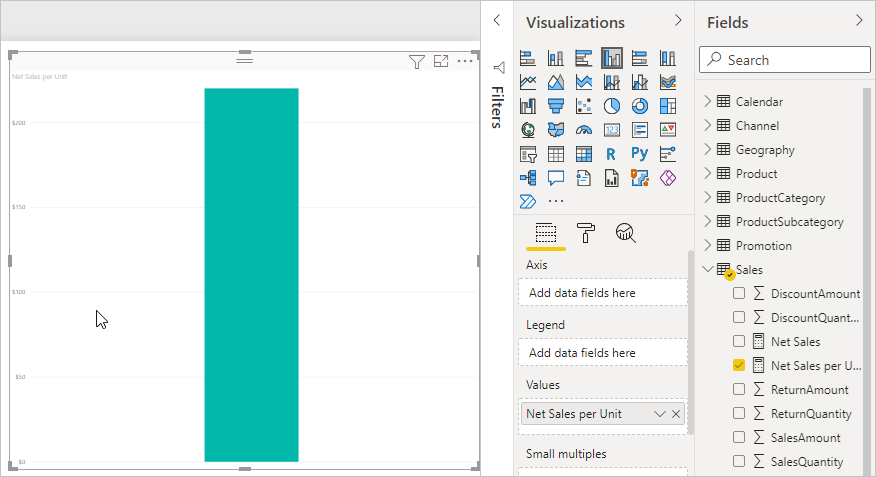
1. Enter a space, a divide operator (/), another space, a SUM function, and then type *Quantity*. The suggestion list shows all the columns with *Quantity* in the name. Select **Sales[SalesQuantity]**, type the closing parenthesis, and press **ENTER** or choose **Commit** (checkmark icon) to validate your formula.

The resulting formula should appear as:

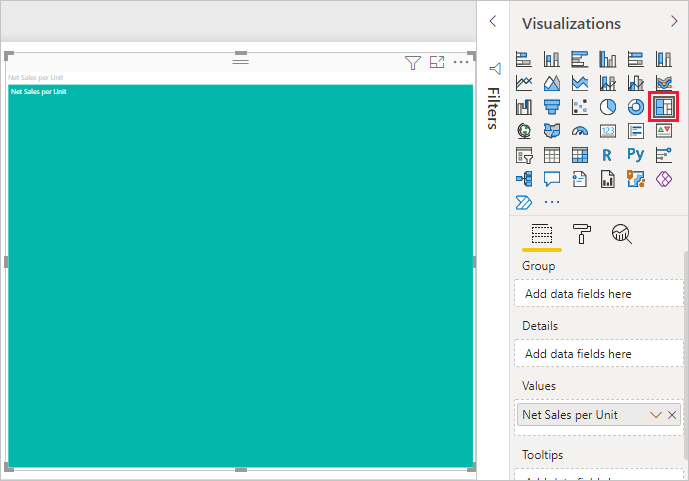
Net Sales per Unit = [Net Sales] / SUM(Sales[SalesQuantity])

1. Select the **Net Sales per Unit** measure from the **Sales** table, or drag it onto a blank area in the report canvas.

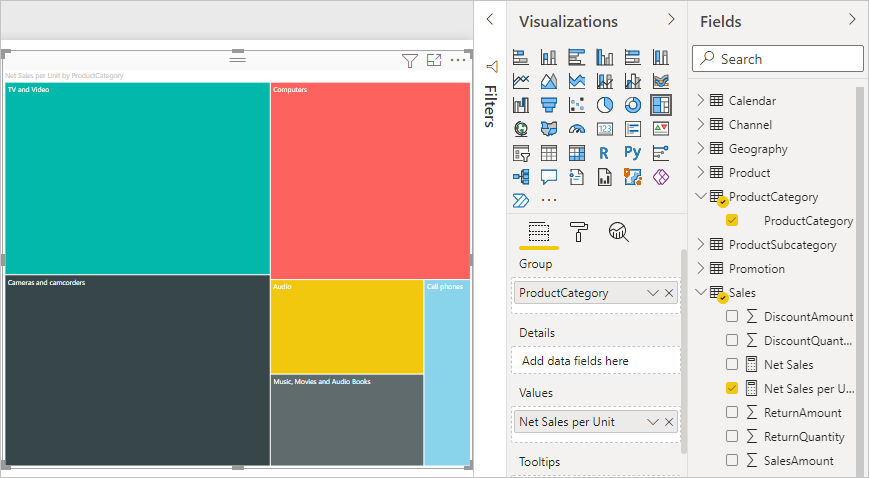
The chart shows the net sales amount per unit over all products sold. This chart isn't informative; we'll address it in the next step.



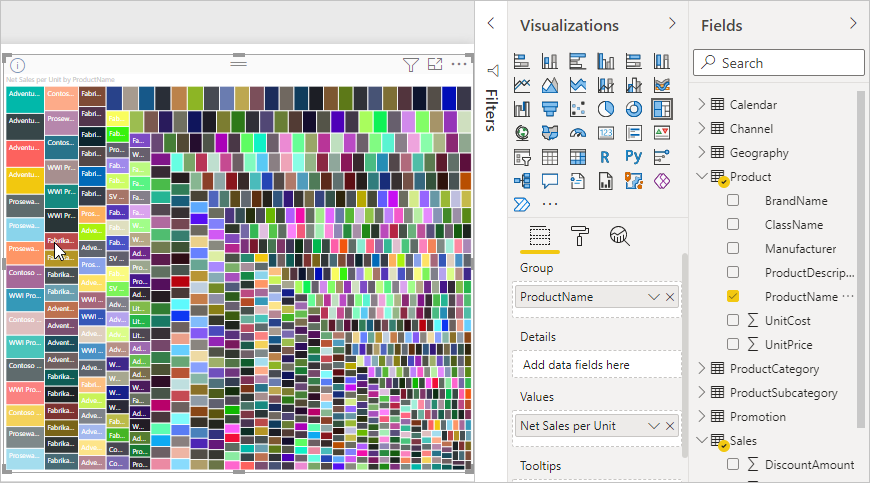
1. For a different look, change the chart visualization type to **Treemap**.



1. Select the **Product Category** field, or drag it onto the treemap or the **Group** field of the **Visualizations** pane. Now you have some good info!



1. Try removing the **ProductCategory** field, and dragging the **ProductName** field onto the chart instead.



Ok, now we're just playing, but you have to admit that's cool! Experiment with other ways to filter and format the visualization.

Save the Power BI file as **DataAnalysisExpressions\_I**.

**LAB 5: STATISTICAL FUNCTIONS**

**Download the Data**

1. Download the “**OrderData**” Excel file from datasets folder.
2. Open Power BI Desktop.
3. On the Ribbon at the top of the screen, click on "**Get Data**", select **Excel Workbook**, navigate to where you saved the "**OrderData**" Excel file, and select **Open**.
4. Click on the “**OrderData**” dataset and then click “**Load**”

A screenshot of a computer

Description automatically generated

1. Expand the OrderData dataset in the “**Data**” tab of PowerBI. As you can see, there are multiple fields , some of them are descriptors and some of them are measures.

**The Arithmetic Mean or Average Value – Central Tendency**

**LAB Theory**

It is a measure of central tendency and it returns the mean of the numbers in a column. It is the sum of values divided by the number of values. For example, we have ten values: 9, 8, 6, 11, 5, 7, 9, 4, 3, 8. The sum is 70. We divide the sum by the number of values which is 10 and the mean would be 7.

The mean is extremely sensitive to extreme values, that is to outliers, which significantly affect its value. It is highly recommended to use the mean together with other measures of central tendency such as the median to make conclusions. If you see big differences between the mean and the median, you need to investigate what is causing that difference. If the extreme values in your numbers are a rarity that happens in extreme cases, then you might decide to eliminate those values.

**Syntax**

average(<column>)

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| column | The column that contains the numbers for which you want the average. |

**Return value**

Returns a decimal number that represents the arithmetic mean of the numbers in the column.

**Remarks**

This function takes the specified column as an argument and finds the average of the values in that column. If you want to find the average of an expression that evaluates to a set of numbers, use the AVERAGEX function instead.

**Nonnumeric values in the column are handled as follows:**

If the column contains text, no aggregation can be performed, and the functions returns blanks.

If the column contains logical values or empty cells, those values are ignored.

* Cells with the value zero are included.
* When you average cells, you must keep in mind the difference between an empty cell and a cell that contains the value 0 (zero). When a cell contains 0, it is added to the sum of numbers and the row is counted among the number of rows used as the divisor. However, when a cell contains a blank, the row is not counted.
* Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Excel also returns a zero if no rows are found that meet the conditions.
* The AVERAGEX function enables you to evaluate expressions for each row of a table, and then take the resulting set of values and calculate its arithmetic mean. Therefore, the function takes a table as its first argument, and an expression as the second argument.
* In all other respects, AVERAGEX follows the same rules as AVERAGE. You cannot include non-numeric or null cells. Both the table and expression arguments are required.
* These functions are not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**LAB Steps**

1. Create a new page and name it **Central Tendency**.
2. Your task is to calculate the mean sales amount by product. Use the **ProductName** attribute and the **InvoiceAmount** measure on which to calculate the mean.
3. Right click the **OrderData** dataset and click “**New Measure**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new measure to the left of the equal sign and name it **ProductSalesMean**.
2. Add the DAX expression to the right of the equal sign as:

average(OrderData[InvoiceAmount])

1. From the visualizations tab, add a **Stacked Bar chart** visual to the report area.
2. Add the **Productname** attribute to your Stacked Bar Chart visual.
3. Add the **ProductSalesMean** measure to your Stacked Bar chart visual.
4. For the **Stacked bar Chart** visual, click on the “**more options**" three dots,  and sort the ProductName field ascending. Your sorting should look like the following:

A screenshot of a computer

Description automatically generated

1. For the **Stacked bar Chart** visual, go to the **Visualizations** pane, in the **Format Visual** tab, and turn **Data Labels** to **On**. Your chart should look like the following:

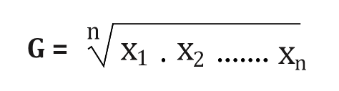
A bar chart with text and numbers

Description automatically generated

**The Geometric Mean – Central Tendency**

**LAB Theory**

Returns the geometric mean of the numbers in a column. It is a measure of central tendency that calculates averages. To calculate the geometric mean, we multiply the numbers in the dataset and then we obtain the root depending on the number of values: a square root (for two numbers), a cube root (for three numbers) etc.



We might prefer to use the geometric mean when the values in our dataset are not independent observations or when we have large fluctuations in values. For example, in finance we use the geometric mean for time series analysis of returns. It practically calculates the average rate of change of a measure such as the rate of return, growth rate, and interest rates. It is not influenced that much by outliers.

**Syntax**

GEOMEAN(<column>)

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| column | The column that contains the numbers for which the geometric mean is to be computed. |

**Return value**

A decimal number.

**Remarks**

* Only the numbers in the column are counted. Blanks, logical values, and text are ignored.
* GEOMEAN( Table[Column] ) is equivalent to GEOMEANX( Table, Table[Column] )
* This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**LAB Steps**

1. Your task is to calculate the geometric mean of the sales amount by product. Use the **ProductName** attribute and the **InvoiceAmount** measure on which to calculate the geometric mean.
2. Right click the **OrderData** dataset and click “**New Measure**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new measure to the left of the equal sign and name it **ProductSalesGeometricMean**.
2. Add the DAX expression to the right of the equal sign as:

geomean(OrderData[InvoiceAmount])

1. From the visualizations tab, add a **Stacked Bar chart** visual to the report area.
2. Add the **Productname** attribute to your Stacked Bar Chart visual.
3. Add the **ProductSalesGeometricMean** measure to your Stacked Bar chart visual.
4. For the **Stacked bar Chart** visual, click on the “**more options**" three dots,  and sort the ProductName field ascending. Your sorting should look like the following:

A screenshot of a computer

Description automatically generated

1. For the **Stacked bar Chart** visual, go to the **Visualizations** pane, in the **Format Visual** tab, and turn **Data Labels** to **On**. Your chart should look like the following:

A bar chart with text and numbers

Description automatically generated

**The median() function – Central Tendency**

**LAB Theory**

Returns the median of the numbers in a column. It is a measure of central tendency. The median is not affected by outliers unlike the mean which is affected by the presence of extreme values and outliers.

**Syntax**

MEDIAN(<column>)

**Parameters**

| Term | Definition |
| --- | --- |
| column | The column that contains the numbers for which the median is to be computed. |

**Return value**

A decimal number.

**Remarks**

* Only the numbers in the column are counted. Blanks are ignored. Logical values, dates, and text are not supported.
* MEDIAN( Table[Column] ) is equivalent to MEDIANX( Table, Table[Column] ).
* This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**LAB Steps**

1. Your task is to calculate the median sales amount by product. Use the **ProductName** attribute and the **InvoiceAmount** measure on which to calculate the median.
2. Right click the **OrderData** dataset and click “**New Measure**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new measure to the left of the equal sign and name it **ProductSalesMedian**.
2. Add the DAX expression to the right of the equal sign as:

median(OrderData[InvoiceAmount])

1. The expression in the DAX editor should be as in the image below:



1. From the visualizations tab, add a **Stacked Bar chart** visual to the report area.
2. Add the **Productname** attribute to your Stacked Bar Chart visual.
3. Add the **ProductSalesMedian** measure to your Stacked Bar chart visual.
4. For the **Stacked Column** visual, go to the **Visualizations** pane, in the **Format Visual** tab, and turn **Data Labels** to **On**. Your chart should look like the following:

A bar graph with numbers and text

Description automatically generated

**Dashboard**

Your dashboard should look like the following:

A screenshot of a graph

Description automatically generated

**The stdev.p() function – Looking for variation**

**LAB Theory**

Returns the standard deviation of the entire population. It shows how dispersed the data is with respect to the mean.

A diagram of a curve

Description automatically generated

**Syntax**

STDEV.P(<ColumnName>)

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

**Return value**

A number representing the standard deviation of the entire population.

**Remarks**

* STDEV.P assumes that the column refers to the entire population. If your data represents a sample of the population, then compute the standard deviation by using STDEV.S.
* STDEV.P uses the following formula:

√[∑(x - x̃)2/n]

where x̃ is the average value of x for the entire population and n is the population size.

* Blank rows are filtered out from *columnName* and not considered in the calculations.
* An error is returned if *columnName* contains less than 2 non-blank rows
* This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**LAB Steps**

1. Your task is to calculate the standard deviation of the sales amount by product. Use the **ProductName** attribute and the **InvoiceAmount** measure for your analysis.
2. Right click the **OrderData** dataset and click “**New Measure**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new measure to the left of the equal sign and name it **ProductSalesStDev**.
2. Add the DAX expression to the right of the equal sign as:

stdev.p(OrderData[InvoiceAmount])

1. From the visualizations tab, add a **Stacked Bar chart** visual to the report area.
2. Add the **Productname** attribute to your Stacked Bar Chart visual.
3. Add the **ProductSalesStDev** measure to your Stacked Bar chart visual.
4. For the **Stacked bar Chart** visual, click on the “**more options**" three dots,  and sort the ProductName field ascending. Your sorting should look like the following:

A screenshot of a computer

Description automatically generated

1. For the **Stacked bar Chart** visual, go to the **Visualizations** pane, in the **Format Visual** tab, and turn **Data Labels** to **On**. Your chart should look like the following:

A bar chart with text

Description automatically generated

**The var.p() function – Looking for variation**

**LAB Theory**

Returns the variance of the entire population. It provides a value of how much the values in the data set are away from the mean.

**Syntax**

VAR.P(<columnName>)

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

**Return value**

A number with the variance of the entire population.

**Remarks**

* VAR.P assumes that the column refers the entire population. If your data represents a sample of the population, then compute the variance by using VAR.S.
* VAR.P uses the following formula:

∑(x - x̃)2/n

where x̃ is the average value of x for the entire population

and n is the population size

* Blank rows are filtered out from *columnName* and not considered in the calculations.
* An error is returned if *columnName* contains less than 2 non-blank rows
* This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**LAB Steps**

1. Your task is to calculate the variance of the sales amount by product. Use the **ProductName** attribute and the **InvoiceAmount** measure for your analysis.
2. Right click the **OrderData** dataset and click “**New Measure**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new measure to the left of the equal sign and name it **ProductSalesVar**.
2. Add the DAX expression to the right of the equal sign as:

var.p(OrderData[InvoiceAmount])

1. From the visualizations tab, add a **Stacked Bar chart** visual to the report area.
2. Add the **ProductName** attribute to your Stacked Bar Chart visual.
3. Add the **ProductSalesVar** measure to your Stacked Bar chart visual.
4. For the **Stacked bar Chart** visual, click on the “**more options**" three dots,  and sort the ProductName field ascending. Your sorting should look like the following:

A screenshot of a computer

Description automatically generated

1. For the **Stacked bar Chart** visual, go to the **Visualizations** pane, in the **Format Visual** tab, and turn **Data Labels** to **On**. Your chart should look like the following:

A bar chart with text and numbers

Description automatically generated

**Dashboard**

Your dashboard should look like the following:

A screenshot of a graph

Description automatically generated