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

# LAB: LOGICAL FUNCTIONS

## The if() function

**Syntax**

IF(<logical\_test>, <value\_if\_true> , <value\_if\_false>)

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| logical\_test | Any value or expression that can be evaluated to TRUE or FALSE. |
| value\_if\_true | The value that's returned if the logical test is TRUE. |
| value\_if\_false | (Optional) The value that's returned if the logical test is FALSE. If omitted, BLANK is returned. |

## Get the Data

1. Download the “**OrderData**” Excel file fromdatasets 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.

## Create an if() Function

1. In the Power BI file that you are currently working on, create a new page and name it **Logical Functions**. Keep all your work in one PowerBI file.
2. The business goal is to create a new field that will list products as **inexpensive** if their price is below $35 and **expensive** if their price is equal or greater than $35.
3. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**if(OrderData[ProductPrice] < 35 , "Inexpensive" , "Expensive")**

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



1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **ProductPriceOneCondition** attribute to your Stacked Column Chart visual.
3. Add the **InvoiceAmount measure** to your Stacked Column chart visual. The default aggregate function in the Y-axis should be sum().
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 screenshot of a graph

Description automatically generated

1. The management clearly sees that the products with lower prices generate more revenue.

## Create Nested if() functions

**Syntax**

if(<logical\_test>, <value\_if\_true> ,

if(<logical\_test>, <value\_if\_true> , <value\_if\_false>))

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| logical\_test | Any value or expression that can be evaluated to TRUE or FALSE. |
| value\_if\_true | The value that's returned if the logical test is TRUE. |
| value\_if\_false | (Optional) The value that's returned if the logical test is FALSE. If omitted, BLANK is returned. |

1. The management was thrilled with the sales insight you have provided for inexpensive vs. expensive products sales and they want to expand on that concept. They are now asking to create three product categories as follows: Products will be labeled as **inexpensive** if their price is below $20, **moderate\_priced** if their price is between $20 and $35 including those priced at $20 or $35, and **expensive** if their price is greater than $35.
2. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

ProductPriceTwoConditions = if(OrderData[ProductPrice] < 20 ,  "inexpensive" ,  if(OrderData[ProductPrice] <= 35, "moderate\_priced" , "expensive" ))

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



1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **ProductPriceTwoConditions** attribute to your Stacked Column Chart visual.
3. Add the **InvoiceAmount measure** to your Stacked Column chart visual. The default aggregate function in the Y-axis should be sum().
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 (it does not have to be identical):

A graph of blue rectangular objects

Description automatically generated with medium confidence

1. Your dashboard should look like the following:

A screenshot of a graph

Description automatically generated

## Create conditions with the switch() function

**Syntax**

SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])

**Parameters**

| **Term** | **Definition** |
| --- | --- |
| expression | Any DAX expression that returns a single scalar value where the expression is to be evaluated multiple times (for each row/context). |
| value | A constant value to be matched with the results of **expression**. |
| result | Any scalar expression to be evaluated if the results of **expression** match the corresponding **value**. |
| else | Any scalar expression to be evaluated if the result of **expression** doesn't match any of the **value** arguments. |

**Return value**

If there’s a match with a **value**, a scalar value from the corresponding **result** is returned. If there isn’t a match with a **value**, a value from **else** is returned. If none of the **values** match and **else** isn’t specified, BLANK is returned.

1. The management was thrilled with the sales insight you have provided for three product categories and they now want to expand the analysis of inexpensive vs. expensive products sales with five product categories as follows: Products will be labeled as **inexpensive** if their price is below $20 including the $20 boundary, **relative\_inexpensive** if their price is between $20 and $25 including the $25 boundary, **moderate\_priced** if their price is between $25 and $30 including the $30 boundary, **relative\_expensive** if their price is between $30 and $40 including the $40 boundary, and **expensive** if their price is greater than $40.   
     
   Now you are thinking that to create four nested if() functions will result in a very complicated DAX formula that will be very difficult to read and above all to maintain and update. It is time to use the switch() function.
2. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **ProductPriceFiveConditions**.
2. Add the DAX expression to the right of the equal sign as (use ALT + ENTER if you need to create multiple lines):

SWITCH (

TRUE,

OrderData[ProductPrice] <= 20 , " inexpensive",

OrderData[ProductPrice] <= 25 , "relative\_inexpensive",

OrderData[ProductPrice] <= 30 , "moderate\_priced",

OrderData[ProductPrice] <= 40 , "relative\_expensive ",

OrderData[ProductPrice] > 40 , "expensive "

)

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

A screenshot of a computer code

Description automatically generated

1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **ProductPriceFiveConditions** attribute to your Stacked Column Chart visual.
3. Add the **InvoiceAmount measure** to your Stacked Column chart visual. The default aggregate function in the Y-axis should be sum().
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 graph of blue rectangular bars with white text

Description automatically generated

1. Your dashboard should look like the following (it does not have to be identical):

A screenshot of a graph

Description automatically generated

## The switch() function with the OR and AND operators

**Logical operators**

Use logical operators (&&) and (||) to combine expressions to produce a single result.

| **Text operator** | **Meaning** | **Examples** |
| --- | --- | --- |
| && (double ampersand) | Creates an AND condition between two expressions that each have a Boolean result. If both expressions return TRUE, the combination of the expressions also returns TRUE; otherwise the combination returns FALSE. | ([Region] = "France") && ([BikeBuyer] = "yes")) |
| || (double pipe symbol) | Creates an OR condition between two logical expressions. If either expression returns TRUE, the result is TRUE; only when both expressions are FALSE is the result FALSE. | (([Region] = "France") || ([BikeBuyer] = "yes")) |
| IN | Creates a logical OR condition between each row being compared to a table. Note: the table constructor syntax uses curly braces. | 'Product'[Color] IN { "Red", "Blue", "Black" |

1. The management of the company is impressed by your work and the quality of analytics you have provided. Now, they need something more sophisticated with multiple criteria for their analyses in relation to product sales. Specifically, they need the sales figures for products in the following custom categories:  
   1. Products will be labeled as **inexpensive** if their price is below $20 including the $20 boundary, and their quality rating is “**very good**” and their IngredientType is “**Organic**” or “**GMO-free**”.
   2. Products will be labeled as **relative\_inexpensive**  if their price is between $20 and $25 including the $25 boundary, and their quality rating is “**very good**” or “**acceptable**”, and their IngredientType is “**Organic**” or “**GMO-free**” or “**all**-**natural**”.
   3. Products will be labeled as **moderate\_\_priced** if their price is between $25 and $30 including the $30 boundary, and if their quality rating is “**Excellent**“ or “**very good**” or “**acceptable**”, and their IngredientType is “**Organic**” or “**GMO-free**” or “**all**-**natural**”.
   4. Products will be labeled as **relative\_expensive** if their price is between $30 and $40 including the $40 boundary, and their quality rating is “**Excellent**“ or “**very good**” or “**acceptable**”.
   5. Products will be labeled as **expensive** if their price is above $40.
2. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **ProductPriceFiveConditions\_AND\_OR** .
2. Add the DAX expression to the right of the equal sign as (use ALT + ENTER if you need to create multiple lines) :

SWITCH (

TRUE,

OrderData[ProductPrice] <= 20 && ([QualityRating] = "very good") && ([IngredientType] = "Organic" || [IngredientType] = "GMO-Free")  , "inexpensive",

OrderData[ProductPrice] <= 25 && ([QualityRating] = "very good" || [QualityRating] = "acceptable") && ([IngredientType] = "Organic" || [IngredientType] = "GMO-Free" || [IngredientType] = " all-natural") , "relative\_inexpensive",

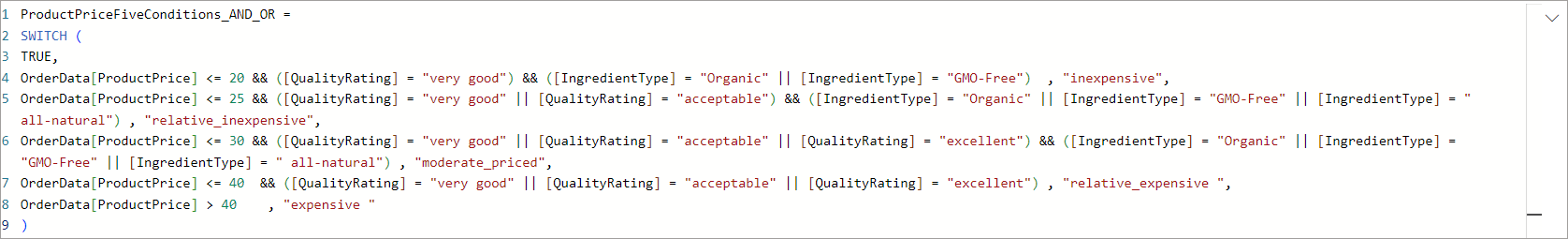
OrderData[ProductPrice] <= 30 && ([QualityRating] = "very good" || [QualityRating] = "acceptable" || [QualityRating] = "excellent") && ([IngredientType] = "Organic" || [IngredientType] = "GMO-Free" || [IngredientType] = " all-natural") , "moderate\_priced",

OrderData[ProductPrice] <= 40  && ([QualityRating] = "very good" || [QualityRating] = "acceptable" || [QualityRating] = "excellent") , "relative\_expensive ",

OrderData[ProductPrice] > 40    , "expensive "

)

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

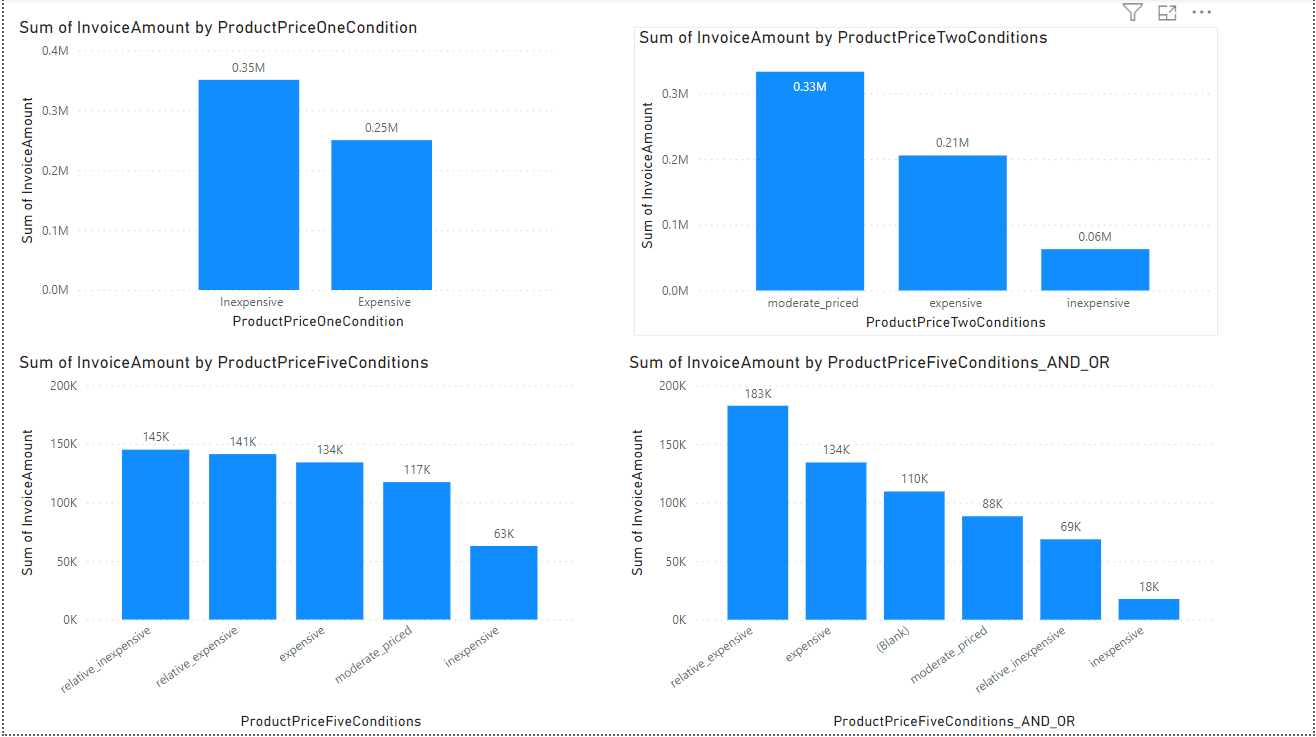


1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **ProductPriceFiveConditions\_AND\_OR** attribute to your Stacked Column Chart visual.
3. Add the **InvoiceAmount measure** to your Stacked Column chart visual. The default aggregate function in the Y-axis should be sum().
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 graph of blue rectangular objects

Description automatically generated

1. Your dashboard should look like the following:



# LAB: DATE FUNCTIONS

## The Year() Function

Returns the year of a date as a four-digit integer in the range 1900-9999.

**Syntax**: YEAR(<datefield>)

1. Create a new Page in your Power BI report and name it **Date Functions**.
2. Add a table visual to this Power BI Report.
3. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **OrderYear**.
2. Add the DAX expression to the right of the equal sign as: **Year(OrderData[OrderDate])**.
3. The expression in the DAX editor should be as in the image below:

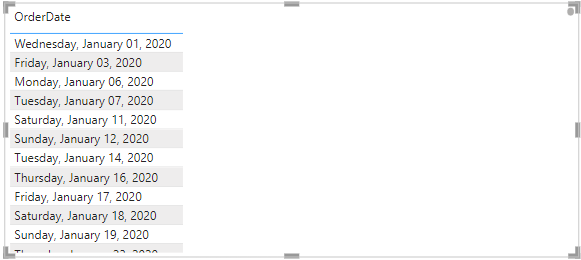


1. From the visualizations tab, add a **Table** visual to the report area.
2. Add the **OrderDate** field to your table visual. When you do that, Power BI by default displays the hierarchy values. Power BI creates data hierarchies by default when it finds date fields. We want to display only the date field.
3. Go to the **Visualizations tab**, click the down arrow right next to the OrderDate field, and select OrderDate.

A screenshot of a computer

Description automatically generated

1. Now, the visual changes and displays actual order dates from the dataset.



1. Add your new column **OrderYear** to the table visual. You notice that Power BI displays gibberish and summarizes incorrectly on a date field!
2. Go to the **Visualizations tab**, click the down arrow right next to the Sum of Order Year field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

Now, Power BI displays the correct OrderYear value in the table visual.

A screenshot of a computer

Description automatically generated

1. The **OrderDate** field in the table visual displays in text format. We would like to convert it to date format. In the **Data** pane, click on the OrderDate field and then in the Ribbon at the top of the screen, go to the **Formatting** group. For the Format property, click the drop-down menu, and select the **Short Date** format. Your table should look similar to the following:

A white board with blue dots

Description automatically generated with medium confidence

1. Now we would like to increase the font of the table visual to 12. Select the table visual, go to the **Visualizations Pane**, and click on the **Format Your Visual** tab. Click on the values option and change the font size to 12.

## The DateDiff() Function

Returns the difference between two dates using an interval such as days.

**Syntax**: DATEDIFF(date field 1, date field 2, date interval)

**Parameters**

|  |  |
| --- | --- |
| Date1 | A datetime field. |
| Date2 | A datetime field. |
| Interval | The interval to use when comparing dates. The value can be one of the following:  - SECOND - MINUTE - HOUR - DAY - WEEK - MONTH - QUARTER - YEAR |

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**DateDiff(OrderData[OrderDate], OrderData[ShippedDate], day)**

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



1. Add the **OrderCycleTime** field to your table visual, right next to the OrderYear. PowerBI by default displays a sum of values for this column. What we need is the actual difference in days between the OrderDate and ShippedDate so that we can track if our orders are processed on time.
2. Go to the **Visualizations tab**, click the down arrow right next to the Sum of the **OrderCycleTime** field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

Now, Power BI displays the correct **OrderCycleTime** values in the table visual.

A screenshot of a computer

Description automatically generated

## The Weekday() Function

Returns a number from 1 to 7 identifying the day of the week of a date. By default the day ranges from 1 (Sunday) to 7 (Saturday).

**Syntax**

WEEKDAY(<datefield>, <return\_type>)

**Parameters**

|  |  |
| --- | --- |
| datefield | A date value or field in **datetime** format. |
| return\_type | A number that determines the Return value:  Return type: **1**, week begins on Sunday (1) and ends on Saturday (7). numbered 1 through 7.  Return type: **2**, week begins on Monday (1) and ends on Sunday (7).  Return type: **3**, week begins on Monday (0) and ends on Sunday (6).numbered 1 through 7. |

**Return value**

An integer number from 1 to 7.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**weekday(OrderData[OrderDate], 1)**

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



1. Add the **WeekDay** field to your table visual right next to the **OrderCycleTime** column. PowerBI by default displays a sum of values for this column. What we need is the actual number of the week for every OrderDate in every order placed by the customer.
2. Go to the **Visualizations tab**, click the down arrow right next to the Sum of the **WeekDay** field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

Now, Power BI displays the correct **WeekDay** values in the table visual.

A screenshot of a computer

Description automatically generated

**The WeekNum() Function**

Returns the week number for a date value or field according to the **return\_type** value. The week number indicates where the week falls numerically within a year.

**Syntax**

WEEKNUM(<date>[, <return\_type>])

**Parameters**

|  |  |
| --- | --- |
| date | The date in **datetime** format. |
| return\_type | (Optional) A number that determines on which day the week begins. Default is 1. See Remarks. |
|  |  |

There are two *systems* used for this function:

* **System 1** - The week containing January 1 is the first week of the year and is numbered week 1.
* **System 2** - The week containing the first Thursday of the year is the first week of the year and is numbered as week 1. This system is the methodology specified in ISO 8601, which is commonly known as the European week numbering system.

**Return value**

An integer number.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**weeknum(OrderData[OrderDate])**

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



1. Add the **WeekOfTheYear** field to your table visual, right next to the WeekDay. PowerBI by default displays a sum of values for this column. What we need are the actual values that determine the week of the year.
2. Go to the **Visualizations tab**, click the down arrow right next to the Sum of the **WeekOfTheYear** field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

Now, Power BI displays the correct **WeekOfTheYear** values in the table visual.

A screenshot of a computer

Description automatically generated

# LAB: TEXT FUNCTIONS

## The Len() Function

Returns the number of characters in a text string.

**Syntax**: Len(text field)

**Return value**

A whole number indicating the number of characters in the text string.

1. Create a new Page in your Power BI report and name it **Text Functions**.
2. Add a table visual to this Power BI Report.
3. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **CustomerLastNameLength**.
2. Add the DAX expression to the right of the equal sign as: Len(OrderData[CustomerLastName]).
3. The expression in the DAX editor should be as in the image below:



1. Add the **CustomerLastNameLength** field to your table visual. Power BI by default applies a sum() function to this column because the results of the Len() function are numeric. This is not what we want.

A screenshot of a computer

Description automatically generated

1. Go to the **Visualizations tab**, click the down arrow right next to the Sum of **CustomerLastNameLength** field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

1. Now, add the **CustomerLastName** field to the table visual and use the **Columns** box in the **Build Visual** tab to place it to the left of the **CustomerLastNameLength** field. Your report should look similar to the following:

A screenshot of a computer

Description automatically generated

1. Now we would like to increase the font of the table visual to 12. Select the table visual, go to the **Visualizations Pane**, and click on the **Format Your Visual** tab. Click on the **values** option and change the font size to 12.

## The trim() Function

Removes all blank spaces from the beginning and end of the values in a text column except for single spaces between words. It will remove multiple blank spaces between words.

**Syntax**

TRIM(text field)

**Parameters**

**Text field** The field from which you want spaces removed.

**Return value**

The field column with spaces removed.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **ProductNameTrimmed**.
2. Add the DAX expression to the right of the equal sign as: **trim(OrderData[ProductName])**.
3. The expression in the DAX editor should be as in the image below:



1. Add the **ProductNameTrimmed** field to your table visual. Make sure the table visual is selected, otherwise Power BI will create a new visual. This is how your table visual should look like:

A screenshot of a computer

Description automatically generated

## The left() Function

Returns the specified number of characters from the start of a text string.

**Syntax**

LEFT(<text field>, <num\_chars>)

**Parameters**

|  |  |
| --- | --- |
| text field | The text string containing the characters you want to extract, or a reference to a column that contains text. |
| num\_chars | (optional) The number of characters you want LEFT to extract; if omitted, 1. |

**Return value**

A text string.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **CustomerEmail\_LastNamePart**
2. Add the DAX expression to the right of the equal sign as: **left(OrderData[CustomerLastName], 5)**
3. The expression in the DAX editor should be as in the image below:



1. Add the **CustomerEmail\_LastNamePart** field to your table visual. Make sure the table visual is selected, otherwise Power BI will create a new visual. This is how your table visual should look like:

A screenshot of a menu

Description automatically generated

## Concatenate functions to create an email address

The goal is to create an email address for our customers.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



1. Edit the name of the new column to the left of the equal sign and name it **CustomerEmail**
2. Add the DAX expression to the right of the equal sign as: **CustomerEmail = left(OrderData[CustomerFirstName], 1) & left(OrderData[CustomerLastName], 5) & "@" & "newhaven.edu"**
3. The expression in the DAX editor should be as in the image below:



1. Add the **CustomerEmail** field to your table visual. Make sure the table visual is selected, otherwise Power BI will create a new visual. This is how your table visual should look like:

A screenshot of a computer

Description automatically generated

# LAB: AGGREGATE FUNCTIONS

1. Create a **new page** in Power BI desktop and name it “**Aggregate Functions**”.

## Average Product Cost by Supplier

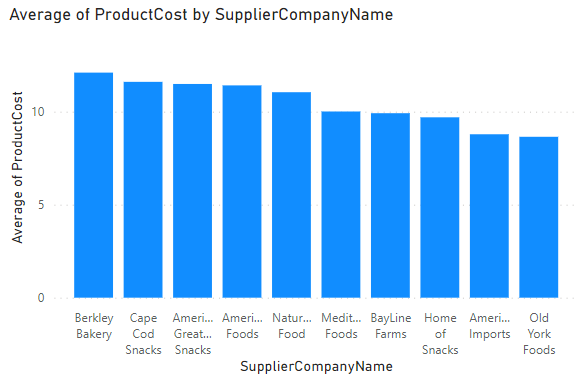
The business goal in this case is to calculate the average cost of raw materials by supplier. We could have calculated the total material cost by supplier but that will not give us a good indication of how expensive a supplier is since we might have ordered from a particular supplier more times.

1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **SupplierCompanyName** attribute to your to your Stacked Column Chart visual. Your chart should look like:

A screenshot of a computer

Description automatically generated

1. Add the **ProductCost** measure to your Stacked Column chart visual. Change the aggregate function in the Y-axis from sum() to average()Your chart should look like the following:



1. Notice that the supplier “Berkley Bakery” has the highest average product cost. We might want to advise the company to look at purchasing more from other suppliers or ask the most expensive companies to provide better prices or discounts.

## Average Product Cost Per Unit by Supplier

To be certain of our inventory recommendations to the company we will explore the average product cost per unit. This is because although the average product cost per supplier is a good indicator, we might have bought more units by a specific supplier or set of suppliers and that will skew our results. The per unit cost will verify or alter our conclusions.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**ProductCostPerUnit = OrderData[ProductCost] / OrderData[ProductQuantityInOrder]**

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



1. From the visualizations tab, add a **Stacked Column chart** visual to the report area.
2. Add the **SupplierCompanyName** attribute to your Stacked Column Chart visual. Your chart should look like:

A screenshot of a computer

Description automatically generated

1. Add the **ProductCostPerUnit** measure to your Stacked Column chart visual. Change the aggregate function in the Y-axis from sum() to average(). Your chart should look like the following:

A graph of blue bars

Description automatically generated

1. Your dashboard should now look like the one in the image below:

A comparison of blue bars

Description automatically generated

## Immediate Inventory Needs

The next item the management wants to see on the Inventory Dashboard is a list of products for which we need to order new inventory immediately. To achieve this goal, we need to create a new DAX expression. Practically, we need to add the fields UnitsInStock and UnitsOnOrder and subtract this number from the ReorderLevel.

1. Right click the OrderData dataset and click “**New Column**”. At the top of the visual area the DAX editor opens as in the image below:



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

**(OrderData[UnitsInStock] + OrderData[UnitsOnOrder]) - OrderData[ReorderLevel]**

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



1. From the visualizations tab, add a **Table** visual to the report area.
2. Add the **ProductName** attribute to your table visual.
3. Add the **InventoryReplenishmentNeeds** measure to your table visual. Power BI by default applies a sum() function to this column because the results of the DAX function are numeric. This is not what we want.
4. Go to the **Visualizations tab**, click the down arrow right next to the Sum of **InventoryReplenishmentNeeds** field, and select **Don’t summarize**.

A screenshot of a computer

Description automatically generated

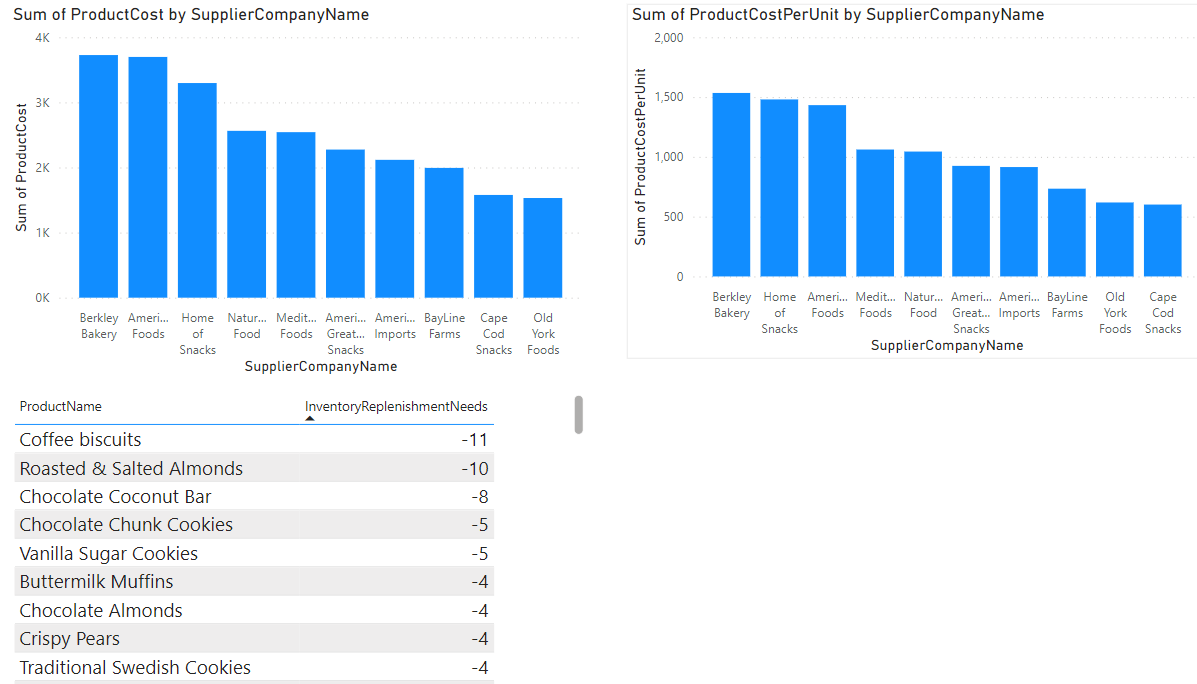
1. In the visualizations pane, click on the **Format Your Visual** tab and change the **Values** font to 14.
2. Now go back to your table visual and click on the heading of the **InventoryReplenishmentNeeds** field to sort its values so that those products with negative values show first. If a product has a negative InventoryReplenishmentNeeds value, it means that the current inventory is well below the Reorder Level. If it has a value close to zero, it means we need to pay attention to it because soon we will need to order more inventory.

1. Your table visual should look like the following:

A screenshot of a menu

Description automatically generated

1. Your dashboard should now look like the one in the image below:



**Product Cost by Container Type**

Next, the management wants to see a pie chart that shows the product cost by container type.

1. From the visualizations tab, add a **Pie Chart** visual to the report area.
2. Add the **ProductContainerType** attribute to your to your Pie Chart visual.
3. Add the **ProductCost** measure to your Pie chart visual. Your chart should look like the following:

A pie chart with numbers and a graph

Description automatically generated

1. Your dashboard should now look like the one in the image below:

A screenshot of a graph

Description automatically generated