EXPLORE AI ACADEMY

Calculated columns with DAX

DAX functions

Diving deeper into DAX functions

> There are roughly **250 different functions** within Power BI, but the most popular ones can be grouped into the following **broad categories**.

Aggregation	These functions are used for **summarising** or **aggregating** data in columns or tables into a **single value**. They correspond closely to spreadsheet functions.
Count	**Count functions** are used for returning the number of rows/values in a column or table. Columns are the only accepted input for count functions.
Logical	**Logical** functions return **values or sets** based on whether an **expression** is true or false (i.e. whether a **condition** has been met or not).
Information	These functions are useful when **examining data** quality or types. A cell/row of data is provided as input and the function returns whether the cell/row value **matches** the **expected** type.

Diving deeper into DAX functions

Text	These functions are similar to **string** functions found in spreadsheets, with the ability to be applied to **columns** and **tables.**
Temporal	Used when doing calculations based on **dates** and **time**. These functions use a **datetime** data type and can use values from a column as input.
Filter	**Filter** functions in Power BI allow us to **manipulate data context**, resulting in dynamic calculations.
Other	**Power BI documentation** contains detailed information on **mathematical, financial, statistical,** and many other categories of functions that could be of use.

Aggregation functions

> These functions perform **calculations** on a set of values to produce a **single, summarising value,** often used in measures.

Syntax

```
Measure = FUNCTION(Table_name[column_name])
Measure_with_expression = FUNCTIONX(Table_name, Expression)
```

Common aggregation functions

`SUM()` Aggregates the values in a column.

`SUMX()` Aggregates the values in an expression.

`MIN()/MAX()` Returns the min/max value in a column.

`MINX()/MAXX()` Returns the min/max value of an expression.

`AVERAGE()` Returns the arithmetic mean of a column.

`AVERAGEX()` Returns the arithmetic mean of an expression.

Power BI has extensive documentation explaining syntax for all the DAX functions.

X for expression

The **X** at the end of these aggregation functions refers to **expression** and it means that we can add any expression which will then be used as the **'input column'.**

Aggregation example:

> In this example, we are interested in the **number of males living in rural areas** in the different countries in our `Gender_parity_2022` table. By multiplying the **population size** with the **proportion** of **rural male population**, we can easily see this metric.

```
Total_pop = SUM(Gender_parity_2022[Population size])
```

```
Rural_male_pop = SUMX(Gender_parity_2022, Gender_parity_2022[Population size] * Gender_parity_2022[Rural population, male (% of total)])
```

Country	Sum of Population size	Sum of Rural population, male (% of total)	Rural_male_pop
Egypt	110990103	28.85	3,201,844,210.58
Ethiopia	123379924	40.38	4,981,485,318.49
Ghana	33475870	23.18	775,831,354.42
Kenya	54027487	36.85	1,990,681,006.19
Morocco	37457971	19.89	744,989,333.09
Nigeria	218541212	25.97	5,676,327,071.01
Rwanda	13776698	33.10	456,008,814.70
South Africa	59893885	16.55	991,028,146.42
Total	**651543150**	**224.75**	**18,818,195,254.91**

Count functions

> We use **count functions** to count certain elements in a table or column. These functions follow the same format as the common aggregation functions.

Syntax

```
Count_of_something = FUNCTION(Table_name[column_name])
Count_of_some_expression = FUNCTIONX(Table_name, Expression)
```

Common count functions

`COUNT()` Returns the number of non-blank values in a column. Supports strings, integers, or dates.

`COUNTX()` Returns the number of non-blank values as the result of an expression. Uses table as input.

`COUNTROWS()` Counts the number of rows in a table. Uses table as input.

`DISTINCTCOUNT()` Counts the number of distinct values in a column, including the BLANK value.

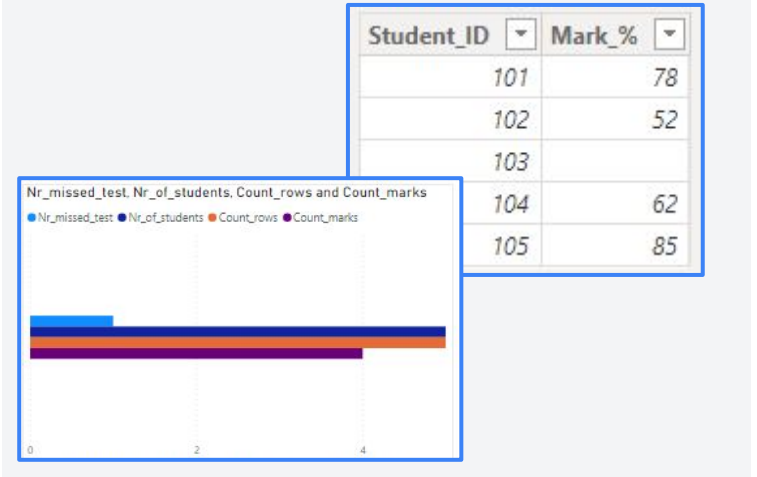
`COUNTBLANK()` Counts the number of BLANK cells in a column.

`COUNTA()` Counts the number of non-blank cells in a column. In contrast to **`COUNT()`**, it also supports Boolean values.

Count example

In the dataset below, we have a very small class with 5 students and a test mark per student. We want to know **how many** students didn't write the test (with a valid excuse).

```
Total_blanks = COUNTBLANK(Test_data[Mark_%])
Total_students = COUNT(Test_data[Student_ID])
Count_rows = COUNTROWS(Test_data)
Count_marks = COUNT(Test_data[Mark_%])
```



Why care about BLANKS?

Missing values might influence our dashboard's reliability. If a large proportion of data points are missing, the **summary** might present a **skewed view.**

Logical and conditional functions

> These functions enable us to make use of **conditional logic** when creating new columns.

Common logical functions

`AND()` Returns TRUE if both arguments are TRUE.

`OR()` Returns TRUE if one of the arguments is TRUE.

`NOT()` Returns TRUE if the condition is not met.

`IF()` Evaluates a condition, if TRUE, it returns one value, otherwise it returns the second value.

`IFERROR()` Evaluates an expression and returns an alternative pre-specified value if the expression results in an error.

`SWITCH()` Evaluates an expression (or column) against a list of values and returns one of the pre-specified results. By using Boolean values in the expression, we can also use this instead of nested `IF()` statements.

Important

Logical functions can only be used on **columns.**

IF()/SWITCH() example

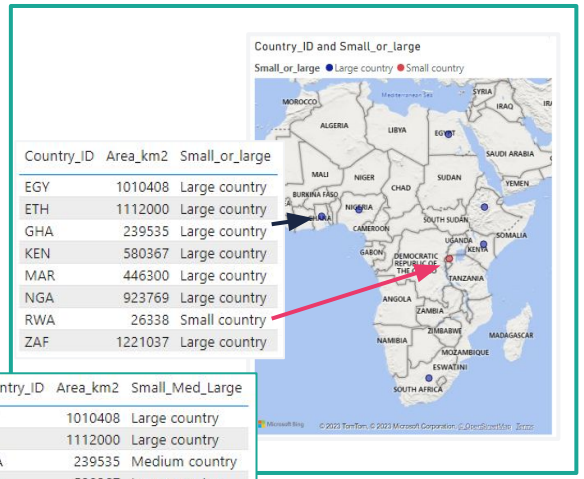
> Using our Demographic_info dataset, we're interested in creating a column that would **categorise countries as small or large,** based on the `Area_km2` column. The second code block shows a **nested** version of **`IF()`**, based on three categories for the country sizes.

```
Small_or_large = IF(Demographic_info[Area_km2] > 100000,
"Large country", "Small country")
```

```
Small_Med_Large = IF(Demographic_info[Area_km2] > 500000,
"Large country",  IF(Demographic_info[Area_km2] > 100000,
"Medium country", "Small country"))
```

This exact statement can also be written with a **`SWITCH()`** function:

```
Small_Med_Large = SWITCH(TRUE,
Demographic_info[Area_km2] > 500000, "Large country",
Demographic_info[Area_km2] > 100000, "Medium country",
Demographic_info[Area_km2] > 0, "Small country")
```



Country_ID	Area_km2	Small_or_large
EGY	1010408	Large country
ETH	1112000	Large country
GHA	239535	Large country
KEN	580367	Large country
MAR	446300	Large country
NGA	923769	Large country
RWA	26338	Small country
ZAF	1221037	Large country

Country_ID	Area_km2	Small_Med_Large
EGY	1010408	Large country
ETH	1112000	Large country
GHA	239535	Medium country
KEN	580367	Large country
MAR	446300	Medium country
NGA	923769	Large country
RWA	26338	Small country
ZAF	1221037	Large country

Information functions

> These functions operate on a cell/row level and evaluate whether the input argument is what is expected. If it is, the function returns **TRUE**, otherwise **FALSE**. They are often used to perform calculations wrapped within an **IF()** statement, to prevent errors in the results.

Common information functions

`ISBLANK()` Checks whether a value/expression is BLANK.

`ISNUMBER()` Checks whether a value/expression is numeric.

`ISTEXT()` Checks whether a value is of type text.

`ISNONTEXT()` Checks whether a value is non-text.

`ISERROR()` Checks whether value/expression results in error.

`CONSTAINSSTRING()` Checks if the specified text is found in a string.

Case sensitive?

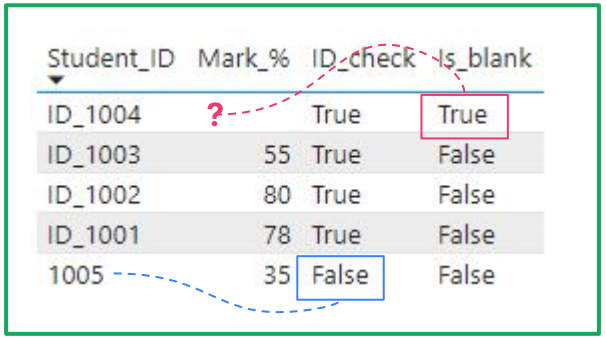
Certain functions are not case sensitive, such as **`CONTAINSSTRING()`**. If we are trying to find a case sensitive string, we'd use **`CONTAINSSTRINGEXACT()`** instead.

Example

> We need to determine whether all the **Student_IDs** in our data are in the **right format** (contains "ID_" as a prefix), and also whether the **marks** are **populated**. We can use two information functions to solve this issue.

```
ID_check = CONTAINSSTRING(Test_data2[Student_ID],"ID")

Is_blank = ISBLANK(Test_data2[Mark_%])
```



Power BI BLANK functions

BLANK cells in Power BI can display different behaviours depending on column types. It is good practice to check that **ISBLANK()** and **COUNTBLANK()** functions behave in the way assumed, especially if used for quality checks.

Text functions

> **Text columns** frequently need to be **manipulated** in some way in order to be used in **visualisations**. Power BI functions can help us create columns in a **specific format**, with the benefit that they correspond closely to string functions found in spreadsheets and SQL.

Common text functions

`SUBSTITUTE()` Replaces existing text with new text in a text string. Text to be replaced is explicitly stated.

`REPLACE()` Replaces part of a string with new text, based on the number of characters and position specified.

`SEARCH()` Returns the number of the character at which a specific string/character first occurs in text, from left to right. Can specify starting position for the search.

`FIXED()` Rounds a number to specified decimals, then converts to string.

`CONCATENATE()` Joins two text strings together into one.

`UPPER/LOWER()` Converts the case of a text string to all uppercase or all lowercase.

`LEFT()` Returns the specified number of characters from the start of a text string.

`RIGHT()` Returns the last character or characters in a text string, based on the number of characters specified.

String manipulation example

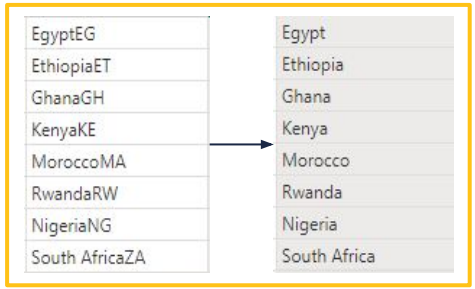
> Our `Country` column in our dataset accidentally includes the first two letters of the `Country_ID` column, as **something went wrong** during the data processing phase. We need to **clean the data** by making use of some **text manipulation functions.**

Column of interest

```
Country_new = REPLACE(Country_simple[Country],
LEN(Country_simple[Country]) - 1 ,2 ,"")
```

Getting the position of where the replacement text should go, in this case the length of the name, -1 position

Specifying that we want to replace 2 characters, with nothing ("")



EgyptEG	Egypt
EthiopiaET	Ethiopia
GhanaGH	Ghana
KenyaKE	Kenya
MoroccoMA	Morocco
RwandaRW	Rwanda
NigeriaNG	Nigeria
South AfricaZA	South Africa

Power BI documentation

Here we used **`LEN()`** (returning the length of a string) within our text manipulation formula. **Power BI documentation** contains a full list of useful functions not directly covered here.

Temporal functions

> By using temporal functions in Power BI, we can do **time-based analysis** on columns. It is useful for productivity measures, comparisons over different periods, and determining person-level indicators such as age.

Common date functions

`DATE()` Returns a specified date in datetime format.

`HOUR()` Returns the hour as a number from 0 to 23, based on 24-hour time notation (00:00 - 23:59).

`WEEKDAY()` Returns the day of the week as a number (1-7).

`NOW()` Returns the current date and time in a datetime format.

`EOMONTH()` Returns the date in datetime format of the last day of the month, before or after a specified number of months.

`DATEDIFF()` Counts the number of intervals between two dates. Intervals can be seconds up to years.

Date and time example

> We have a small dataset (`Grants`), but the day, month, and year have been captured in **separate columns**. Convert these columns to a **single date field**, showing the **actual day** of the week, and then calculate **how many months** have gone by between the end and start date.

```
Start_date = DATE(Grants[Start_year], Grants[Start_month], Grants[Start_day])
Weekday_nr = WEEKDAY(Grants[Start_date], 1)
Weekday_format = FORMAT(Grants[Weekday_nr], "DDD")
Diff_months = DATEDIFF(Grants[Start_date], Grants[End Date], MONTH)
```

Start_day	Start_month	Start_year	Start_date	Weekday_nr	Weekday_format	End Date	Diff_months
1	6	2022	6/1/2022 12:00:00 AM	4	Wed	Saturday, December 31, 2022	6
1	7	2023	7/1/2023 12:00:00 AM	7	Sat	Friday, December 01, 2023	5
3	8	2022	8/3/2022 12:00:00 AM	4	Wed	Thursday, December 01, 2022	4
4	6	2023	6/4/2023 12:00:00 AM	1	Sun	Friday, December 01, 2023	6
7	7	2020	7/7/2020 12:00:00 AM	3	Tue	Friday, January 01, 2021	6

Filter and relationship functions

> **Filter functions** in DAX enable us to manipulate the data context to create dynamic calculations, whereas **relationship functions** help to manage and utilise relationships between tables.

Common filter functions

`ALL()` Returns all rows in a table, ignoring any filters that may have been applied.

`CALCULATE()` Evaluates an expression in a modified filter context.

`CALCULATETABLE()` Evaluates a table expression in a modified filter context.

`FILTER()` Returns a table that is a subset of another table or expression.

`LOOKUPVALUE()` Returns a row based on criteria specified in a search condition. There can be multiple search conditions.

`USERELATIONSHIP()` The function specifies a relationship to be used in a specific calculation.

`RELATED()` Returns a related value from another table.

Filter example

> We are interested in creating a table (E_Africa) that contains a **subset** of the data in our original table. We also want to add the Country name, a column from another table. The linking columns have already been identified and relationships are set up.

```
E_Africa = FILTER(Demographic_info,
Demographic_info[Region] == "Eastern Africa")
```

Country_ID	Region
ETH	Eastern Africa
KEN	Eastern Africa
RWA	Eastern Africa
EGY	Northern Africa
MAR	Northern Africa
ZAF	Southern Africa
GHA	Western Africa
NGA	Western Africa

→

Country_ID	Region
ETH	Eastern Africa
KEN	Eastern Africa
RWA	Eastern Africa

```
Country_name =
RELATED(Country_simple[Country_name])
```

Country_ID	Region
ETH	Eastern Africa
KEN	Eastern Africa
RWA	Eastern Africa

→

Country_ID	Region	Country_name
ETH	Eastern Africa	Ethiopia
KEN	Eastern Africa	Kenya
RWA	Eastern Africa	Rwanda