

Power BI - DAX

Data Analysis Expressions

***Agenda***

**We are going to cover**

* Aggregation Functions
* Date & Time Functions
* Filter Functions
* Logical Functions
* Relationship Functions
* Multiple Functions Together
* Latest DAX Functions

**Aggregation Functions**

Aggregation functions calculate a (scalar) value such as count, sum, average, minimum, or maximum for all rows in a column or table as defined by the expression.

* **APPROXIMATEDISTINCTCOUNT**: Returns an estimated count of unique values in a column.
* **AVERAGE**: Returns the average (arithmetic mean) of all the numbers in a column.
* **AVERAGEA**: Returns the average (arithmetic mean) of the values in a column.
* **AVERAGEX**: Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.
* **COUNT**: Counts the number of rows in the specified column that contain non-blank values.
* **COUNTA**: Counts the number of rows in the specified column that contain non-blank values.
* **COUNTAX**: Counts non-blank results when evaluating the result of an expression over a table.
* **COUNTBLANK**: Counts the number of blank cells in a column.
* **COUNTROWS**: Counts the number of rows in the specified table, or in a table defined by an expression.
* **COUNTX**: Counts the number of rows that contain a number or an expression that evaluates to a number when evaluating an expression over a table.
* **DISTINCTCOUNT**: Counts the number of distinct values in a column.
* **DISTINCTCOUNTNOBLANK**: Counts the number of distinct values in a column, excluding blanks.
* **MAX**: Returns the largest numeric value in a column or between two scalar expressions.
* **MAXA**: Returns the largest value in a column, including logical values and numbers represented as text.
* **MAXX**: Evaluates an expression for each row of a table and returns the largest numeric value.
* **MIN**: Returns the smallest numeric value in a column or between two scalar expressions.
* **MINA**: Returns the smallest value in a column, including logical values and numbers represented as text.
* **MINX**: Returns the smallest numeric value that results from evaluating an expression for each row of a table.
* **PRODUCT**: Returns the product of the numbers in a column.
* **PRODUCTX**: Returns the product of an expression evaluated for each row in a table.
* **SUM**: Adds all the numbers in a column.
* **SUMX**: Returns the sum of an expression evaluated for each row in a table.

DAX Measures of most used Aggregate Functions:

1. **AVERAGE**:

Average Sales = AVERAGE(Orders[Sales])

1. **AVERAGEA**:

Average Profit = AVERAGEA(Orders[Profit])

1. **AVERAGEX**:

Average Profit per Order = AVERAGEX(Orders, Orders[Profit] / Orders[Quantity ordered new])

1. **COUNT**:

Count Product Names = COUNT(Orders[Product Name])

1. **COUNTA**:

Count Non-Blank Product Categories = COUNTA(Orders[Product Category])

1. **COUNTAX**:

Count Orders with Profit > 100 = COUNTAX(Orders, IF(Orders[Profit] > 100, 1, BLANK()))

1. **COUNTX**:

Count Orders with Quantity > 10 = COUNTX(Orders, IF(Orders[Quantity ordered new] > 10, 1, 0))

1. **COUNTBLANK**:

Count Blank Cities = COUNTBLANK(Orders[City])

1. **COUNTROWS**:

Count Rows in Returns = COUNTROWS(Returns)

1. **MAX**:

Max Sales = MAX(Orders[Sales])

1. **MAXX**:

Max Profit per Order = MAXX(Orders, Orders[Profit] / Orders[Quantity ordered new])

1. **MIN**:

Min Unit Price = MIN(Orders[Unit Price])

1. **MINX**:

Min Shipping Cost per Order = MINX(Orders, Orders[Shipping Cost] / Orders[Quantity ordered new])

1. **SUM**:

Total Saless = SUM(Orders[Sales])

1. **SUMX**:

Total Profit per Order = SUMX(Orders, Orders[Profit] / Orders[Quantity ordered new])

1. **PRODUCT**:

Total Product of Unit Prices = PRODUCT(Orders[Unit Price])

1. **PRODUCTX**:

Total Product of Profit per Order = PRODUCTX(Orders, Orders[Profit] / Orders[Quantity ordered new])

**Measures:**

1. Measure to calculate the total sales in the 'Orders' table.

Total Sales = SUM(Orders[Sales])

1. Write DAX measure to calculate the average discount in the 'Orders' table.

Average Discount = AVERAGE(Orders[Discount])

1. Write DAX measure to count the number of unique customers in the 'Orders' table.

Number of Customers = DISTINCTCOUNT(Orders[Customer Name])

1. Write DAX measure to calculate the total profit in the 'Orders' table.

Total Profit = SUM(Orders[Profit])

1. Write DAX measure to find the maximum shipping cost in the 'Orders' table.

Maximum Shipping Cost = MAX(Orders[Shipping Cost])

1. Measure to calculate the average sales per customer in the 'Orders' table.

Average Sales per Customer = DIVIDE([Total Sales], [Number of Customers], 0)

1. Measure to calculate the ratio of discounted sales to total sales.

Discounted Sales Ratio = DIVIDE([Total Sales], CALCULATE([Total Sales], Orders[Discount] > 0), 0)

1. Calculate the weighted average profit margin for each product sub-category based on the product's sales amount.

Weighted Avg Profit Margin =

SUMX(

SUMMARIZE(Orders, Orders[Product Sub-Category]),

VAR TotalSales = SUMX(FILTER(Orders, Orders[Product Sub-Category] = EARLIER(Orders[Product Sub-Category])), Orders[Sales])

VAR WeightedProfit = SUMX(FILTER(Orders, Orders[Product Sub-Category] = EARLIER(Orders[Product Sub-Category])), Orders[Profit])

RETURN DIVIDE(WeightedProfit, TotalSales) \* 100

)

1. Calculate the average sales per customer segment using the 'Customer Segment' column and the 'Sales' column in the 'Orders' table.

Avg Sales per Segment =

AVERAGEX(

SUMMARIZE(Orders, Orders[Customer Segment]),

[Total Sales]

)

1. Count the number of rows in the 'Users' table.

Count of Users = COUNTROWS(Users)

1. Find the smallest value in the 'State or Province' column of the 'Orders' table, including any logical values and numbers represented as text.

Minimum State = MINA(Orders[State or Province])

1. Evaluate an expression for each row of the 'Orders' table and return the smallest numeric value.

Min Sales per Order =

MINX(Orders, Orders[Sales])

**Calculated Columns:**

1. Crete calculated column to calculate the discounted price for each order.

Discounted Price = Orders[Unit Price] - (Orders[Unit Price] \* Orders[Discount])

1. Create calculated column to calculate the sales per quantity for each order.

Sales Per Quantity = Orders[Sales] / Orders[Quantity ordered new]

1. Create calculated column to calculate the total sales for each region.

Total Sales by Region = SUMX(FILTER(Orders, Orders[Region] = EARLIER(Orders[Region])), Orders[Sales])

**Date & Time Functions**

Date & Time functions are designed to assist in performing calculations based on dates and times. While many of the DAX functions are similar to their Excel counterparts, they use a datetime data type and can accept values from columns as arguments.

* **CALENDAR**: Generates a table with a single column named "Date" containing a continuous sequence of dates.
* **CALENDARAUTO**: Produces a table with a single column named "Date" that holds an uninterrupted series of dates.
* **DATE**: Returns a specified date in datetime format.
* **DATEDIFF**: Calculates the count of interval boundaries between two dates.
* **DATEVALUE**: Converts a date represented in text form into a date in datetime format.
* **DAY**: Provides the day of the month as a number from 1 to 31.
* **EDATE**: Determines the date that falls a given number of months before or after a reference start date.
* **EOMONTH**: Fetches the last day of a month, before or after a specified number of months, in datetime format.
* **HOUR**: Returns the hour as a number ranging from 0 (12:00 A.M.) to 23 (11:00 P.M.).
* **MINUTE**: Offers the minute as a number from 0 to 59, considering a date and time value.
* **MONTH**: Delivers the month as a number from 1 (January) to 12 (December).
* **NETWORKDAYS**: Computes the count of complete workdays between two dates.
* **NOW**: Provides the current date and time in datetime format.
* **QUARTER**: Identifies the quarter as a number from 1 to 4.
* **SECOND**: Retrieves the seconds of a time value as a number from 0 to 59.
* **TIME**: Transforms hours, minutes, and seconds represented as numbers into a time in datetime format.
* **TIMEVALUE**: Converts a time in text format into a time in datetime format.
* **TODAY**: Returns the current date.
* **UTCNOW**: Supplies the current UTC date and time.
* **UTCTODAY**: Offers the current UTC date.
* **WEEKDAY**: Determines a number from 1 to 7, signifying the day of the week for a given date.
* **WEEKNUM**: Calculates the week number for a given date and year based on the specified return type.
* **YEAR**: Retrieves the year of a date as a four-digit integer within the range of 1900 to 9999.
* **YEARFRAC**: Computes the fraction of the year represented by the number of full days between two dates.

DAX calculated columns of most used Date & Time Functions:

1. **CALENDER:**

DateTable = CALENDAR(DATE(2020, 1, 1), DATE(2023, 12, 31))

1. **NOW**:

CurrentDateTime = NOW()

1. **TODAY**:

CurrentDate = TODAY()

**Measures:**

1. Create a measure to calculate the average number of network days (workdays) between order dates.

Avg Network Days Between Orders = AVERAGEX('Orders', NETWORKDAYS('Orders'[Order Date], 'Orders'[Ship Date]))

1. Create a measure to calculate the average time it takes to ship orders in hours.

Avg Time to Ship Orders = AVERAGEX('Orders', DATEDIFF('Orders'[Order Date], 'Orders'[Ship Date], HOUR))

**Calculated Columns:**

1. Create a calculated column that calculates the number of whole workdays between the 'Order Date' and 'Ship Date' in the 'Orders' table.

Workdays Difference = DATEDIFF(Orders[Order Date], Orders[Ship Date], DAY)

1. Create a calculated column that extracts the day of the week for the 'Order Date' in the 'Orders' table.

Day of Week = WEEKDAY(Orders[Order Date])

1. Create a calculated column that extracts the quarter for the 'Order Date' in the 'Orders' table.

Order Quarter = QUARTER(Orders[Order Date])

1. Create a calculated column that returns the current date for each row in the 'Orders' table.

Today Date = TODAY()

**Filter Functions**

DAX's filter and value functions allow you to work with data context and relationships, making it easier to perform complex calculations and aggregations on large datasets. By leveraging tables, relationships, and measures, you can create dynamic calculations that respond to the filters and slicers applied to your data.

* **ALL**: Returns all the rows in a table or all the values in a column, ignoring any filters that might have been applied.
* **ALLCROSSFILTERED**: Clears all filters applied to a table.
* **ALLEXCEPT**: Removes all context filters in a table except filters that have been applied to the specified columns.
* **ALLNOBLANKROW**: From the parent table of a relationship, returns all rows but the blank row or all distinct values of a column but the blank row, disregarding any context filters.
* **ALLSELECTED**: Removes context filters from columns and rows in the current query while retaining other context filters or explicit filters.
* **CALCULATE**: Evaluates an expression in a modified filter context.
* **CALCULATETABLE**: Evaluates a table expression in a modified filter context.
* **EARLIER**: Returns the current value of the specified column in an outer evaluation pass of the mentioned column.
* **EARLIEST**: Returns the current value of the specified column in an outer evaluation pass of the specified column.
* **FILTER**: Returns a table that represents a subset of another table or expression.
* **INDEX**: Returns a row at an absolute position within the specified partition, sorted by the specified order or on the specified axis.
* **KEEPFILTERS**: Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function.
* **LOOKUPVALUE**: Returns the value for the row that meets specified search conditions. The function can apply one or more search conditions.
* **MATCHBY**: In window functions, defines the columns used to determine how to match data and identify the current row.
* **OFFSET**: Returns a single row positioned before or after the current row within the same table, by a given offset.
* **ORDERBY**: Defines the columns that determine the sort order within each of a WINDOW function's partitions.
* **PARTITIONBY**: Defines the columns used to partition a WINDOW function's relation parameter.
* **RANK**: Returns the ranking of a row within the given interval.
* **REMOVEFILTERS**: Clears filters from the specified tables or columns.
* **ROWNUMBER**: Returns the unique ranking of a row within the given interval.
* **SELECTEDVALUE**: Returns the value when the context for columnName has been filtered down to one distinct value only. Otherwise, it returns alternateResult.
* **WINDOW**: Returns multiple rows positioned within the given interval.

DAX Measures of most used Filter Functions:

1. **ALL**:

Total Sales (All Categories) = CALCULATE(SUM(Orders[Sales]), ALL(Orders[Product Category]))

1. **ALLEXCCEPT:**

Total Sales (All Except Furniture) = CALCULATE(SUM(Orders[Sales]), ALLEXCEPT(Orders, Orders[Product Category]), NOT(Orders[Product Category] = "Furniture"))

1. **CALCULATE:**

Total Sales (Express Air) = CALCULATE(SUM(Orders[Sales]), Orders[Ship Mode] = "Express Air")

1. **LOOKUPVALUE:**

Customer Name = LOOKUPVALUE(Orders[Customer Name], Orders[Order ID], 88522)

1. **SELECTEDVALUE:**

Selected Category Sales = CALCULATE(SUM(Orders[Sales]), Orders[Product Category] = SELECTEDVALUE(Orders[Product Category]))

**Measures:**

1. Calculate the total sales for high priority orders in the 'East' region.

Total Sales (High Priority, East) =

CALCULATE(

SUM(Orders[Sales]),

FILTER(ALL(Orders), Orders[Order Priority] = "High"),

FILTER(ALL(Users), Users[Region] = "East")

)

This measure calculates the sum of sales for high-priority orders but only in the 'East' region using the FILTER and ALL functions.

1. Calculate the maximum quantity ordered by a customer in the 'West' region.

Max Quantity Ordered (West) =

CALCULATE(

MAXX(

SUMMARIZE(Orders, Orders[Customer ID], "Max Quantity", MAX(Orders[Quantity ordered new])),

[Max Quantity]

),

FILTER(ALL(Users), Users[Region] = "West")

)

This measure finds the maximum quantity ordered per customer but only for the 'West' region using the MAXX and SUMMARIZE functions.

1. Calculate the average discount in the 'East' region.

Average Discount (Central) =

CALCULATE(

    AVERAGE(Orders[Discount]),

    FILTER(ALL(Users), Users[Region] = "Central")

)

This measure calculates the average discount but only for the 'East' region using the FILTER and ALL functions.

**Calculated Columns:**

1. Calculate the ratio of shipping cost to unit price for each order.

Shipping Cost to Unit Price Ratio =

DIVIDE(Orders[Shipping Cost], Orders[Unit Price])

This calculated column calculates the ratio of shipping cost to unit price, providing insights into the cost-effectiveness of shipping.

1. Create a calculated column to categorize discounts into 'Low', 'Medium', and 'High'.

Discount Category =

IF(Orders[Discount] <= 0.03, "Low",

IF(Orders[Discount] <= 0.06, "Medium", "High")

)

This calculated column assigns discount categories based on the discount value.

1. Calculate the number of days it took to ship the order.

Days to Ship =

IF(ISBLANK(Orders[Ship Date]),

BLANK(),

DATEDIFF(Orders[Order Date], Orders[Ship Date], DAY)

)

This calculated column calculates the number of days between the order date and ship date.

1. Create a calculated column to categorize orders into 'Fast', 'Normal', and 'Delayed' based on days to ship.

Days to Ship Category =

IF([Days to Ship] <= 1, "Fast",

IF([Days to Ship] <= 3, "Normal", "Delayed")

)

This calculated column assigns categories based on the number of days it took to ship an order.

**Logical Functions**

Logical functions in DAX are powerful tools for making decisions, performing logical operations, and evaluating conditions within your data.

* **AND**: Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE.
* **BITAND**: Returns a bitwise 'AND' of two numbers.
* **BITLSHIFT**: Returns a number shifted left by the specified number of bits.
* **BITOR**: Returns a bitwise 'OR' of two numbers.
* **BITRSHIFT**: Returns a number shifted right by the specified number of bits.
* **BITXOR**: Returns a bitwise 'XOR' of two numbers.
* **COALESCE**: Returns the first expression that does not evaluate to BLANK.
* **FALSE**: Returns the logical value FALSE.
* **IF**: Checks a condition and returns one value when TRUE, otherwise it returns a second value.
* **IF.EAGER**: Similar to IF but uses eager execution, always executing the branch expressions regardless of the condition expression.
* **IFERROR**: Evaluates an expression and returns a specified value if the expression returns an error.
* **NOT**: Changes FALSE to TRUE, or TRUE to FALSE.
* **OR**: Checks whether one of the arguments is TRUE to return TRUE.
* **SWITCH**: Evaluates an expression against a list of values and returns one of multiple possible result expressions.
* **TRUE**: Returns the logical value TRUE.

DAX Measures of most used Filter Functions:

1. **AND**:

HighProfitOrders = CALCULATE(

    SUM(Orders[Sales]),

    FILTER(Orders, Orders[Order Priority] = "High" && Orders[Profit] > 1000)

)

1. **OR**:

ExpressOrTruckOrders = CALCULATE(

SUM(Orders[Sales]),

FILTER(Orders, Orders[Ship Mode] = "Express Air" || Orders[Ship Mode] = "Delivery Truck")

)

1. **NOT**:

NotLowPriorityOrders = CALCULATE(

COUNTROWS(Orders),

FILTER(Orders, NOT(Orders[Order Priority] = "Low"))

)

1. **IF**:

TotalProfit = SUMX(

Orders,

IF(Orders[Profit] > 1000, Orders[Profit], 0)

)

1. **TRUE**:

IsHighPriority = TRUE()

1. **FALSE**:

IsLowPriority = FALSE()

**Measures:**

1. Write a measure to count Orders with High Priority and Profit Greater Than 100.

High Priority High Profit Orders =

COUNTROWS(

FILTER(

Orders,

AND(

Orders[Order Priority] = "High",

Orders[Profit] > 100

)

)

)

1. Write a measure to calculate the Total Sales for Orders with Profit Less Than 0 or a Discount Greater Than 0.1.

Total Sales Low Profit High Discount =

SUMX(

FILTER(

Orders,

OR(

Orders[Profit] < 0,

Orders[Discount] > 0.1

)

),

Orders[Sales]

)

1. How does the "Sales Category" calculated column categorize orders based on their total sales, and what are the criteria for each category?

Sales Category =

SWITCH(

TRUE(),

SUM(Orders[Sales]) <= 100, "Low Sales",

SUM(Orders[Sales]) <= 500, "Medium Sales",

SUM(Orders[Sales]) > 500, "High Sales",

"Other"

)

**Calculated Columns:**

1. Categorize orders into three groups: "Low Sales," "Medium Sales," and "High Sales" based on the "Sales" column

Sales Category =

SWITCH(

TRUE(),

Orders[Sales] <= 100, "Low Sales",

Orders[Sales] <= 500, "Medium Sales",

Orders[Sales] > 500, "High Sales",

"Other"

)

1. Determines whether a product is popular based on the quantity ordered.

Product Popularity =

IF(Orders[Quantity ordered new] > 50, "Popular", "Not Popular")

**Relationship Functions**

Relationship functions are designed to handle and leverage the relationships that exist between tables in your data model.

* **CROSSFILTER**: It allows you to define the cross-filtering direction to be used in calculations for a relationship between two columns.
* **RELATED**: This function retrieves a related value from another table based on the established relationships.
* **RELATEDTABLE**: You can use this function to evaluate a table expression within a context that has been modified by specific filters, considering the relationships in place.
* **USERELATIONSHIP**: This function enables you to specify a particular relationship to be used in a specific calculation, allowing you to choose which relationship between columnName1 and columnName2 is applied.

DAX Measures of most used Relationship functions:

1. **RELATED**:

Returned Order Status =

CALCULATE(

VALUES(Returns[Status]),

FILTER(Orders, NOT(ISBLANK(RELATED(Returns[Order ID])))

)

**Measures:**

1. Calculate the average sales per user in the East region, using the RELATEDTABLE function to evaluate a table expression.

Average Sales Per User in East Region with RELATEDTABLE =

AVERAGEX(

FILTER(

Users,

Users[Region] = "East"

),

SUMX(

RELATEDTABLE(Orders),

Orders[Sales]

)

)

1. Calculate the percentage of sales that received a discount, taking into account the cross-filtering direction using the CROSSFILTER function.

Discounted Sales Percentage with CROSSFILTER =

DIVIDE(

CALCULATE(

SUM(Orders[Sales]),

FILTER(

Orders,

Orders[Discount] > 0

),

CROSSFILTER(Orders[Order ID], Returns[Order ID], Both)

),

SUM(Orders[Sales]),

0

)

**Calculated Columns:**

1. Create column that calculates the profit margin for each order, taking into account the cross-filtering direction using the CROSSFILTER function.

Profit Margin with CROSSFILTER =

DIVIDE(

CALCULATE(

SUM(Orders[Profit]),

CROSSFILTER(Orders[Order ID], Returns[Order ID], Both)

),

SUM(Orders[Sales]),

0

)

1. Create a calculated column that identifies the return status for each order, using the RELATEDTABLE function to evaluate a table expression.

Return Status with RELATEDTABLE =

IF(

COUNTROWS(

FILTER(

RELATEDTABLE(Returns),

Returns[Order ID] = Orders[Order ID]

)

) > 0,

"Returned",

"Not Returned"

)

**Multiple Functions Together:**

**Measures:**

1. What is the "Customer Churn Rate," and how does it help analyze customer retention in the dataset?

Customer Churn Rate =

VAR CurrentMonth = MAX(Orders[Order Date])

VAR ThreeMonthsAgo = EDATE(CurrentMonth, -3)

RETURN

COUNTROWS(

    EXCEPT(

        ALL(Orders[Customer Name]),

        CALCULATETABLE(VALUES(Orders[Customer Name]), Orders[Order Date] >= ThreeMonthsAgo && Orders[Order Date] <= CurrentMonth)

    )

) / COUNTROWS(ALL(Orders[Customer Name]))

**Calculated Columns:**

1. Create calculated column that calculates the return rate for each order in the 'Orders' table by matching it with the 'Returns' table. It uses the RELATED and COUNTROWS functions.

Return Rate =

VAR OrderID = 'Orders'[Order ID]

RETURN

IF(COUNTROWS(FILTER('Returns', 'Returns'[Order ID] = OrderID)) > 0, 1, 0)

**Latest DAX Functions:**

**Measures:**

1. Create a measure that calculates the number of whole workdays between the Order Date and Ship Date for all orders.

Network Days Between Order Date and Ship Date (NETWORKDAYS) =

SUMX(

FILTER(

Orders,

NOT(ISBLANK(Orders[Order Date])) && NOT(ISBLANK(Orders[Ship Date]))

),

NETWORKDAYS(Orders[Order Date], Orders[Ship Date])

)

1. Create calculated column ranks orders by profit in descending order.

Rank by Profit (RANK) =

RANKX(

ALL(Orders),

CALCULATE(SUM(Orders[Profit])),

,

DESC

)

**Calculated Columns:**

1. Create calculated column that returns the total sales for a rolling window of the last three orders based on the Order Date.

Windowed Sales (WINDOW) =

SUMX(

TOPN(3, FILTER(Orders, Orders[Order Date] <= EARLIER(Orders[Order Date])), Orders[Order Date], ASC),

Orders[Sales]

)

This educational case study material is purely fictional and does not represent any actual companies or data. Any resemblance to real entities is coincidental, and it is intended solely for educational purposes.