All Dax Function Definition With Examples

Datasets:- https://www.kaggle.com/datasets/apoorvaappz/global-super-store-dataset

1. Aggregation Functions

(i) AVERAGE:- The AVERAGE function calculates the arithmetic mean (average) of a column that contains numeric values. Non-numeric values like text or blanks are ignored.

Syntax: AVERAGE(<Column>)

Example:-

- Avg_Sales= AVERAGE(Global_Superstore2[Sales])
- Avg_Discount = AVERAGE(Global_Superstore2[Discount])
- Avg_Progit_Margin = AVERAGE(Global_Superstore2[Profit])
- Avg_Qty_Sold = AVERAGE(Global_Superstore2[Quantity])
- Avg_Shipp_Cost = AVERAGE(Global_Superstore2[Shipping Cost])

(ii). AVERAGEA:- The AVERAGEA function calculates the arithmetic mean (average) of a column but considers non-numeric values differently:

Syntax: AVERAGEA(<Column>)

Example:-

- AverageShipModeRatings(A) = AVERAGEA(Global_Superstore2[Ship Mode])
- AverageDiscountWithBlanks(A) = AVERAGEA(Global_Superstore2[Discount])
- AverageOrderPriority(A) = AVERAGEA(Global_Superstore2[Order Priority])
- AverageQuantityWithNumeric(A) = AVERAGEA(Global_Superstore2[Quantity])

Goutam Kuiri

(iii) AVERAGEX:- The AVERAGEX function calculates the arithmetic mean of an expression evaluated for each row in a table.

Syntax: AVERAGEX(<Table>, <Expression>)
Example:-

- AverageProfitPerCategory=
  AVERAGEX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Category],"Total_Profit",SUM(Global_Superstore2[Profit])),SUM(Global_Superstore2[Profit]))

- Average(x)_Sales_Per_Region =
  AVERAGEX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"Total_Sales",SUM(Global_Superstore2[Sales])),SUM(Global_Superstore2[Sales]))

- AverageDiscountPerOrder(X) =
  AVERAGEX(Global_Superstore2,Global_Superstore2[Discount]*Global_Superstore2[Quantity])

- AverageShippingDays(X) =
  AVERAGEX(Global_Superstore2,DATEDIFF(Global_Superstore2[Order Date],Global_Superstore2[Ship Date],DAY))

- AverageTop10CustomerSales(X) =
  AVERAGEX(TOPN(10,Global_Superstore2,Global_Superstore2[Sales],DESC),Global_Superstore2[Sales])

(iv) Counts:- Count he number of values in a column (ignoring blank values).

Syntax: COUNT(<column>)

- TotalOrders = COUNT(Global_Superstore2[Order ID])
- NonBlankCustomerNames = COUNT(Global_Superstore2[Customer Name])
- ProductIDCount = COUNT(Global_Superstore2[Product ID])
- DiscountedSalesCount = COUNT(Global_Superstore2[Discount])
- OrdersWithShippingCosts = COUNT(Global_Superstore2[Shipping Cost])

(v) COUNTAX:- A DAX function that counts rows in a table where a given expression evaluates to a non-blank value.

Syntax: COUNTAX(<table>, <expression>)

- HighDiscountOrders(countax) =
  COUNTAX(Global_Superstore2,IF(Global_Superstore2[Discount]>0.1,1,BLANK()))

- LargeOrderCount(Countax) =
  COUNTAX(Global_Superstore2,IF(Global_Superstore2[Quantity]>10,1,BLANK()))
- OrdersWithProfit(Countax) =
  COUNTAX(Global_Superstore2,IF(Global_Superstore2[Profit]>0,1,BLANK()))
- ZeroProfitOrders(countax) =
  COUNTAX(Global_Superstore2,IF(Global_Superstore2[Profit]=0,1,BLANK()))

(vi) COUNTBLANK :- A DAX function that counts the number of blank or empty values in a specified column.

Syntax: COUNTBLANK(<column>)

- MissingCustomerName(CB) = COUNTBLANK(Global_Superstore2[Customer Name])
- MissingOrderPriorities(CB) = COUNTBLANK(Global_Superstore2[Order Priority])
- MissingProductIDs(CB) = COUNTBLANK(Global_Superstore2[Product ID])
- MissingSippingCost(CB) = COUNTBLANK(Global_Superstore2[Shipping Cost])
- NoDiscountOrders(CB) = COUNTBLANK(Global_Superstore2[Discount])

(vii) COUNTROWS :- A DAX function that counts the total number of rows in a table or a table expression.

Syntax: COUNTROWS(<table>)

- TotalRows(CR) = COUNTROWS(Global_Superstore2)
- ProductCategoriesCount(CR) = COUNTROWS(VALUES(Global_Superstore2[Category]))
- ShippingModesCount(CR) = COUNTROWS(VALUES(Global_Superstore2[Ship Mode]))
- SubcategoriesCount(CR) = COUNTROWS(VALUES(Global_Superstore2[Sub-Category]))
- UniqueRegion(CR) = COUNTROWS(VALUES(Global_Superstore2[Region]))

(viii) DISTINCTCOUNT:- A DAX function that counts the number of unique (distinct) values in a column.

Syntax: DISTINCTCOUNT(<column>)

- UniqueCustomerCount(DC) = DISTINCTCOUNT(Global_Superstore2[Customer Name])
- UniqueCategoriesCount(DC) = DISTINCTCOUNT(Global_Superstore2[Category])
- UniqueOrderIDCount(DC) = DISTINCTCOUNT(Global_Superstore2[Order ID])
- UniqueProductIDCount(DC) = DISTINCTCOUNT(Global_Superstore2[Product ID])
- UniqueShippingModesCount(DC) = DISTINCTCOUNT(Global_Superstore2[Ship Mode])

(ix) MAX:- A DAX function that returns the largest value in a column.

Syntax: MAX(<column>)

- MaxSales(M) = MAX(Global_Superstore2[Sales])
- MaxProfit(M) = MAX(Global_Superstore2[Profit])
- MaxDiscount(M) = MAX(Global_Superstore2[Discount])
- MaxShippingCost(M) = MAX(Global_Superstore2[Shipping Cost])
- MaxQuantityOrdered(M) = MAX(Global_Superstore2[Quantity])

(x) MAXX:- A DAX function that evaluates an expression for each row of a table and returns the largest value resulting from those calculations.

Syntax: MAXX(<table>, <expression>)

SUMMARIZE(<table>, <groupBy_columnName>, [<groupBy_columnName>], ..., [<name>, <expression>])

- MaxDiscountByProduct(MX) =
  MAXX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductDiscount",SUM(Global_Superstore2[Discount])),[ProductDiscount])

- MaxSalesPerCustomer(MX) =
  MAXX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Customer Name],"CustomerSales",SUM(Global_Superstore2[Sales])),[CustomerSales])

- MaxProfitByRegion(MX) =
  MAXX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"RegionProfit",SUM(Global_Superstore2[Profit])),[RegionProfit])

- MaxQuantityByProduct(MX) =
  MAXX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductQuantity",SUM(Global_Superstore2[Quantity])),[ProductQuantity])

- MaxShippingCostPerOrder(MX) =
  MAXX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Order ID],"OrderShippingCost",SUM(Global_Superstore2[Shipping Cost])),[OrderShippingCost])

(Xi) MIN:- Min returns the smallest numeric value in a column.

Syntax: MIN(<column>)

- MinSales(Min) = MIN(Global_Superstore2[Sales])
- MinShippingCost(Min) = MIN(Global_Superstore2[Shipping Cost])
- MinQuantity(Min) = MIN(Global_Superstore2[Quantity])
- MinProfit(Min) = MIN(Global_Superstore2[Profit])
- MinDiscount(Min) = MIN(Global_Superstore2[Discount])

(Xii) MINX:- MINX evaluates an expression for each row of a table and returns the smallest value.

Syntax: MINX(<table>, <expression>)

- MinSalesPerCustomer(MinX) =
  MINX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Customer Name],"CustomerSales",sum(Global_Superstore2[Sales])),[CustomerSales])

- MinShippingCostPerOrder(MinX) =
  MINX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Order ID],"OrderShippingCost",SUM(Global_Superstore2[Shipping Cost])),[OrderShippingCost])

- MinQuantityByProduct(MinX) =
  MINX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductQuantity",SUM(Global_Superstore2[Quantity])),[ProductQuantity])

- MinProfitByRegion(Minx) =
  MINX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"RegionProfit",SUM(Global_Superstore2[Profit])),[RegionProfit])

- MinDiscountByProduct(MinX) =
  MINX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductDiscount",SUM(Global_Superstore2[Discount])),[ProductDiscount])

(Xiii) SUM:- SUM returns the total (sum) of all numeric values in a column.

Syntax: SUM(<column>)

- TotalSales(S) = SUM(Global_Superstore2[Sales])
- TotalShippingCosts(S) = SUM(Global_Superstore2[Shipping Cost])
- TotalQuantity(S) = SUM(Global_Superstore2[Quantity])
- TotalProfit(S) = SUM(Global_Superstore2[Profit])
- TotalDiscount(S) = SUM(Global_Superstore2[Discount])

(Xiv)SUMX:- SUMX evaluates an expression for each row in a table and returns the sum of the resulting values.

Syntax: SUMX(<table>, <expression>)

- TotalSalesPerCustomer(SumX) =
  SUMX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Customer Name],"CustomerSales",SUM(Global_Superstore2[Sales])),[CustomerSales])

- TotalShippingCostPerOrder(SumX) =
  SUMX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Order ID],"OrderShippingCost",SUM(Global_Superstore2[Shipping Cost])),[OrderShippingCost])

- TotalQuantityByCategory(SumX) =
  SUMX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Category],"CategoryQuantity",SUM(Global_Superstore2[Quantity])),[CategoryQuantity])

- TotalProfitByRegion(SumX) =
  SUMX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"RegionProfit",SUM(Global_Superstore2[Profit])),[RegionProfit])

- TotalDiscountByProduct(SumX) =
  SUMX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductDiscount",SUM(Global_Superstore2[Discount])),[ProductDiscount])

(XV) PRODUCT :- PRODUCT calculates the product of all non-blank numeric values in a column.

Syntax: PRODUCT(<column>)

Use Case
Use **PRODUCT** when you need to calculate a multiplicative total of a single column's values, such as finding the compounded growth rate or multiplicative factors across multiple items.
Represents Cumulative Multiplication

Example: Compounded Growth Rate

Year	Growth Factor
2020	1.05
2021	1.10
2022	1.08

If you calculate the compounded growth rate, each year's growth factor multiplies with the next:

Cumulative Growth=1.05×1.10×1.08=1.2474

- ProductSales(Prod) = PRODUCT(Global_Superstore2[Sales])
- ProductOfQuantities(Prod) = PRODUCT(Global_Superstore2[Quantity])
- ProductOfDiscounts(Prod) = PRODUCT(Global_Superstore2[Discount])

Both **PRODUCT** and **PRODUCTX** can be used for **cumulative multiplication calculations**, where values are multiplied together across rows or columns.
This is particularly useful in scenarios like:

- **Compounded Growth Rate:** Multiplying growth factors over multiple periods.
- **Overall Efficiency:** Combining multiple efficiency factors in a system.
- **Weighted Adjustments:** Applying multiple adjustment factors cumulatively.

(Xvi) PRODUCTX :- **PRODUCTX** calculates the product of an expression evaluated for each row in a table.

Syntax: PRODUCTX(<table>, <expression>)

Row-Wise Example with PRODUCTX:
For row-wise cumulative calculations, where expressions differ for each row, use **PRODUCTX**. For instance, in a production scenario:

Machine	Efficiency Factor	Adjusted Efficiency (Expression: Factor×0.9)
A	0.95	0.95×0.9=0.855
B	0.90	0.90×0.9=0.81
C	0.85	0.85×0.9=0.765

To calculate the **cumulative efficiency**:
TotalEfficiency = PRODUCTX(Machines, Machines[EfficiencyFactor] * 0.9)
This multiplies the adjusted efficiency for all rows, giving the overall system efficiency.

- ProductOfProfitsByCustomer(ProdX) =
  PRODUCTX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Customer Name],"CustomerProfit",SUM(Global_Superstore2[Profit])),[CustomerProfit])

- CompoundedDiscountByProduct(ProdX) =
  PRODUCTX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Product ID],"ProductDiscount",SUM(Global_Superstore2[Discount])),[ProductDiscount])
-

- CumulativeShippingCostPerOrder(ProdX) =
  PRODUCTX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Order ID],"OrderShippingCost",SUM(Global_Superstore2[Shipping Cost])),[OrderShippingCost])

- ProductOfQuantitiesByCategory(ProdX) =
  PRODUCTX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Category],"CategoryQuantity",SUM(Global_Superstore2[Quantity])),[CategoryQuantity])

- ProductofSalesByRegion(ProdX) =
  PRODUCTX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"RegionSales",SUM(Global_Superstore2[Sales])),[RegionSales])

Why Use PRODUCT or PRODUCTX for Cumulative Calculations?

- Handles scenarios where cumulative multiplication is necessary across rows or columns.
- **PRODUCT** simplifies operations on a single column.
- **PRODUCTX** provides flexibility to include complex row-wise expressions.

These functions are essential for tasks like financial modeling, production analysis, or any cumulative scaling calculations.

(XVII) MEDIAN:- **MEDIAN** calculates the middle value of a numeric column. If the column has an even number of values, it returns the average of the two middle values.

Syntax: MEDIAN(<column>)

- MedianSales(Median) = MEDIAN(Global_Superstore2[Sales])
- MedianShippingCost(Median) = MEDIAN(Global_Superstore2[Shipping Cost])
- MedianQuantity(Median) = MEDIAN(Global_Superstore2[Quantity])
- MedianProfit(Median) = MEDIAN(Global_Superstore2[Profit])
- MedianDiscount(Median) = MEDIAN(Global_Superstore2[Discount])

(XVIII) MEDIANX:- **MEDIANX** evaluates a custom expression for each row in a table and returns the median of the calculated results.

Syntax: MEDIANX(<table>, <expression>)

- MedianSalesPerCustomer(MedX) = MEDIANX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Customer Name],"CustomerName",SUM(Global_Superstore2[Sales])),[CustomerName])

- MedianShippingCostPerOrder(MedX) = MEDIANX(SUMMARIZE(Global_Superstore2, Global_Superstore2[Order ID],"OrderShippingCost",SUM(Global_Superstore2[Shipping Cost])),[OrderShippingCost])

- MedianQuantityByCategory(MedX) = MEDIANX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Category],"CategoryQuantity",SUM(Global_Superstore2[Quantity])),[CategoryQuantity])

- MedianProfitByRegion(MedX) = MEDIANX(SUMMARIZE(Global_Superstore2,Global_Superstore2[Region],"RegionProfit",SUM(Global_Superstore2[Sales])),[RegionProfit])

⭐ What is Standard Deviation?

Standard deviation (SD) tells us how spread out the values in a dataset are from the **mean** (average). A **smaller SD** means the data points are closer to the mean, while a **larger SD** means the data points are more spread out.

Using Standard Deviation with a Range

To understand the data spread, we can use the rule of thumb: **Mean ± Standard Deviation**.

Example with Sample SD (129.10129.10129.10):

•Mean = 650

•Range: 650−129.10=520.90 ,650+129.1=779.10 **(520.90 to 779.10)**

•**Interpretation**: Most data points will likely fall between **520.90 and 779.10**

Example with Population SD (111.80):

•Mean = 650

•Range: 650−111.80=538.20 , 650+ 111.80 =761.80

•**Interpretation**: Most data points will likely fall between **538.20 and 761.80.**

1. STDEV.S

Definition

STDEV.S calculates the standard deviation for a sample of data. It assumes the data is a representative subset of a larger population.

Formula

The formula for sample standard deviation is:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Where:

•s: Sample standard deviation

•n: Number of observations

•Xi: Each individual value

•$\bar{x}$: Mean of the sample

Syntax: STDEV.S(<column>)

Example

SampleStdDev = STDEV.S(Sales[Revenue])

Steps to Calculate Standard Deviation

1. **Calculate the Mean ($\bar{x}$):**

   The mean is calculated as:

$$\bar{x} = \frac{\text{Sum of all values}}{\text{Number of values}}$$

$$\bar{x} = \frac{500 + 600 + 700 + 800}{4} = 650$$

Example

Consider the following dataset:

SalesID	Revenue
1	500
2	600
3	700
4	800

2.Find the Squared Differences from the Mean:

Step: Squared Differences from the Mean

For each value in the dataset, the squared difference from the mean is calculated as:

$$(x_i - \bar{x})^2$$

Where:

- x_i: Each individual value in the dataset
- $\bar{x}$: The mean (average) of the dataset

Dataset Values

Sales (x_i)	Difference from Mean ($x_i - \bar{x}$)	Squared Difference $(x_i - \bar{x})^2$
500	$500 - 650 = -150$	$(-150)^2 = 22500$
600	$600 - 650 = -50$	$(-50)^2 = 2500$
700	$700 - 650 = 50$	$(50)^2 = 2500$
800	$800 - 650 = 150$	$(150)^2 = 22500$

3. **Calculate the Variance:**

   The variance is the average of these squared differences, divided by $n - 1$:

$$\text{Variance} = \frac{\text{Sum of squared differences}}{n - 1}$$

$$\text{Variance} = \frac{22500 + 2500 + 2500 + 22500}{4 - 1} = \frac{50000}{3} = 16666.67$$

4. **Take the Square Root of the Variance:**

   The standard deviation is the square root of the variance:

$$s = \sqrt{\text{Variance}}$$

$$s = \sqrt{16666.67} \approx 129.10$$

Summary of Results

- **Mean**: $\bar{x} = 650$
- **Variance**: 16666.67
- **Standard Deviation**: ≈ 129.10

The sample standard deviation (s) of the Revenue column is approximately **129.10**.

Example 1: Daily Sales in Two Stores

Store A and Store B are selling the same product. Here are their sales for 5 days:

- **Store A**: 50, 51, 49, 52, 48
- **Store B**: 20, 80, 30, 90, 10

1. **Mean Sales**:
   - Store A: $\frac{50+51+49+52+48}{5} = 50$
   - Store B: $\frac{20+80+30+90+10}{5} = 46$
2. **Standard Deviation**:
   - Store A: **Low SD** (around 2) → Sales are very consistent, close to the mean of 50.
   - Store B: **High SD** (around 35) → Sales fluctuate a lot from the mean of 46.

Interpretation:

- Store A has steady sales, suggesting stable demand.
- Store B's sales are highly inconsistent, possibly due to promotions or other factors.

Example 2: Student Test Scores

Two classes took the same test, and their scores are as follows:

- **Class A**: 80, 82, 81, 79, 83
- **Class B**: 50, 90, 55, 95, 60

1. **Mean Scores**:
   - Class A: $\frac{80+82+81+79+83}{5} = 81$
   - Class B: $\frac{50+90+55+95+60}{5} = 70$
2. **Standard Deviation**:
   - Class A: **Low SD** → Most students scored close to the mean of 81.
   - Class B: **High SD** → Scores vary widely, showing some students excelled while others struggled.

Interpretation:

- Class A students are performing uniformly.
- Class B has a mix of high achievers and struggling students.

Example 4: Investment Returns

You invest in two stocks:

- **Stock A**: Returns over 5 years are 5%, 6%, 4%, 7%, 5%.
- **Stock B**: Returns over 5 years are -10%, 20%, -5%, 25%, -15%.

1. **Mean Returns**:
   - Stock A: $\frac{5+6+4+7+5}{5} = 5.4\%$
   - Stock B: $\frac{-10+20-5+25-15}{5} = 3\%$
2. **Standard Deviation**:
   - Stock A: **Low SD** → Steady returns close to 5.4%.
   - Stock B: **High SD** → Highly volatile returns.

Interpretation:

- Stock A is a safer investment due to its consistent returns.
- Stock B is riskier but offers the potential for higher gains (or losses).

Key Takeaway

A **low standard deviation** indicates consistency, while a **high standard deviation** suggests variability. Use this to assess:

•**Stability in sales, scores, or performance** (e.g., business planning).
•**Risk in investments** (e.g., choosing steady vs. volatile stocks).
•**Consistency in processes** (e.g., manufacturing quality).

- STDDEVSales(STDEV.S) = STDEV.S(Global_Superstore2[Sales])
- STDDEVPROFIT(STDDEV.S) = STDEV.S(Global_Superstore2[Profit])
- STDDEVSHIPPINGCOST(STDDEV.S) = STDEV.S(Global_Superstore2[Shipping Cost])
- STDDEVDiscount(STDDEV.S) = STDEV.S(Global_Superstore2[Discount])

Meaning:
Calculates the variation or spread in sales amounts across transactions.
Business Implication:
A higher standard deviation indicates significant variability in sales values, suggesting that some transactions are significantly larger or smaller than others. A lower value implies more consistency.

2. STDEV.P

Definition

STDEV.P calculates the standard deviation for the entire population. It assumes the data represents the full set of values.

Formula

The formula for population standard deviation is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Where:

- σ: Population standard deviation
- n: Number of observations
- x_i: Each individual value
- μ: Mean of the population

Syntax: STDEV.P(<column>)

Steps:

1. Mean remains the same:

$$\mu = 650$$

2. Squared differences remain the same:

$$(500 - 650)^2 = 22500,\ (600 - 650)^2 = 2500,\ (700 - 650)^2 = 2500,\ (800 - 650)^2 = 22500$$

3. Calculate variance (divide by n):

$$\text{Variance} = \frac{22500 + 2500 + 2500 + 22500}{4} = 12500$$

4. Take the square root:

$$\sigma = \sqrt{12500} \approx 111.80$$

Result: `111.80`

Key Differences

Function	Assumes Data Is	Formula Divisor	Use Case
STDEV.S	A Sample	$n - 1$	When analyzing a sample of data.
STDEV.P	Entire Population	n	When analyzing the full dataset.

- Sales(Stddev.P) = STDEV.P(Global_Superstore2[Sales])
- Quantity(STDDEV.P) = STDEV.P(Global_Superstore2[Quantity])
- Discount(STDDEV.P) = STDEV.P(Global_Superstore2[Discount])

2. Date and Time Functions

(i) Date:- The DATE function in DAX creates a specific date using the year, month, and day provided as inputs.

Syntax: DATE(<year>, <month>, <day>)

- CustomDate = DATE(2023, 12, 31)

- OrderYearDate(M Date) =
  FORMAT(DATE(YEAR(Global_Superstore2[Order_Date].[Date]),1,1),"DD MMM YYYY")

- Order Date(M Date) =
  FORMAT(DATE(YEAR(Global_Superstore2[Order_Date].[Date]),MONTH(Global_Superstore2[Order_Date].[Date]),DAY(Global_Superstore2[Order_Date].[Date])),"DD MMM YYYY")

- DefaultDate(M Date) =
  IF(ISBLANK(Global_Superstore2[Order_Date].[Date]),DATE(2000,1,1),Global_Superstore2[Order_Date])

- FiscalYearStart(M Date) =
  FORMAT(DATE(YEAR(Global_Superstore2[Order_Date].[Date]),4,1),"DD MMM YYYY")

(ii) DATEDIFF:- The **DATEDIFF** function calculates the difference between two dates in specified units such as days, months, quarters, or years. The syntax is:

Syntax :- DATEDIFF(<start_date>, <end_date>, <unit>)

- DaysToShip(DateDiff) =
  DATEDIFF(Global_Superstore2[Order_Date].[Date],Global_Superstore2[Ship_Date].[Date],DAY)

- MonthsSinceOrder(DateDiff) =
  DATEDIFF(Global_Superstore2[Order_Date].[Date],TODAY(),MONTH)

(iii) DATEADD:- The **DATEADD** function shifts a date by a specified number of intervals (days, months, quarters, or years) forward or backward in time. The syntax is:

Syntax :- DATEADD(<dates>, <number_of_intervals>, <interval>)

- PreviousYearDate(DateADD) = DATEADD(Global_Superstore2[Order_Date].[Date],-1,YEAR)
- NextQuarterDate(DateADD) = DATEADD(Global_Superstore2[Order_Date],1,QUARTER)
- PreviousWeekDate = DATEADD(Global_Superstore2[Order_Date].[Date],-7,DAY)
- RollingSixMonth(DateADD) = DATEADD(Global_Superstore2[Order_Date].[Date],-6,MONTH)
- NextMonthSameMonth(DateAdd) = DATEADD(Global_Superstore2[Order_Date].[Date],12,MONTH)

(iv) DAY:- The **DAY** function extracts the day of the month (1–31) from a given date. The syntax is:

Syntex:- DAY(<date>)

- OrderDay(Day) = DAY(Global_Superstore2[Order_Date].[Date])
- IsFirstOrLastDay(Day) = IF(DAY(Global_Superstore2[Order_Date].[Date])=1||DAY(Global_Superstore2[Order_Date].[Date])=DAY(EOMONTH(Global_Superstore2[Order_Date].[Date],0)),"Yes","No")
- IsMidMonth(Day) = IF(DAY(Global_Superstore2[Order_Date].[Date])>=10 && DAY(Global_Superstore2[Order_Date].[Date])<=20, "Mid-Month","Other")
- DiscountFlag(Day) = IF(MONTH(Global_Superstore2[Order_Date].[Date])=8 && DAY(Global_Superstore2[Order_Date].[Date])=15,"Special Discount","No Discount") -- 15th August Independence Day Discount Offer
- Promotion_Flag(Day) = IF(DAY(Global_Superstore2[Order_Date].[Date])=15,"Special Promo","No Promo")

(v) ENDOFMONTH:- The **ENDOFMONTH** function returns the last date of the month for a specified date column. The syntax is:

Syntex:- ENDOFMONTH(<date_column>)

- LastSippingDate(EOM) = ENDOFMONTH(Global_Superstore2[Ship_Date].[Date])

- LastOrderDate(EOM) = ENDOFMONTH(Global_Superstore2[Order_Date].[Date])

- EOMonthSales(EOM) =

```
VAR LastDayMonth=ENDOFMONTH(Global_Superstore2[Order_Date].[Date])
RETURN
    CALCULATE(
        SUM(Global_Superstore2[Sales]),
            Global_Superstore2[Order_Date]=LastDayMonth
            )
```

- EOMProfitMargin(EOM) =

```
VAR LastOrderDate = ENDOFMONTH(Global_Superstore2[Order_Date].[Date])
RETURN
  ROUNDUP(
    DIVIDE(
      CALCULATE(
        SUM(Global_Superstore2[Profit]),
        FILTER(Global_Superstore2, Global_Superstore2[Order_Date] = LastOrderDate)
      ),
      CALCULATE(
        SUM(Global_Superstore2[Sales]),
        FILTER(Global_Superstore2, Global_Superstore2[Order_Date] = LastOrderDate)
      )
    ),
    2              -- Rounding up to two decimal places
  )
```

(vi) ENDOFYEAR:-The **ENDOFYEAR** function returns the last date of the year for a specified date column. The syntax is:

Syntex:- ENDOFYEAR(<date_column>[, <year_end_date>])

- **LastOrderDate(EOY)** = ENDOFYEAR(Global_Superstore2[Order_Date].[Date])

- **YearEndSales(EOY)** =

```
VAR LastDateOfYear = ENDOFYEAR(Global_Superstore2[Order_Date].[Date])
RETURN
   ROUND(CALCULATE(
      SUM(Global_Superstore2[Sales]),
      FILTER(
         Global_Superstore2,
         Global_Superstore2[Order_Date]= LastDateOfYear
           )
       ),2
      )
```

- **FiscalYearClosingDate(EOY)** = ENDOFYEAR(Global_Superstore2[Order_Date].[Date],3)

- **YearEndProfit(EOY)** =

```
var Lastorderdate=ENDOFYEAR(Global_Superstore2[Order_Date].[Date])
Return
    ROUND(CALCULATE(
      SUM(Global_Superstore2[Profit]),
      FILTER(Global_Superstore2,Global_Superstore2[Order_Date].[Date]=Lastorderdate
      )
    ),2
    )
```

📝 (vii) FORMAT

The FORMAT function in DAX is used to convert a value into a **string** with a specified format.

- **Syntax**

  FORMAT(<value>, <format_string>)

  <value>: The value to be formatted (can be a date, number, or currency).

  <format_string>: A string that defines how the value should be displayed.

```
➢ FormattedDate(Format) =
  FORMAT(Global_Superstore2[Order_Date].[Date],"MMM DD,YYYY")
➢ FormattedSales(Format) = FORMAT(Global_Superstore2[Sales],"#,##0")
➢ FormattedProfit(Format) =
  FORMAT(Global_Superstore2[Profit],"₹#,##0.00")
```

⚠️ **Important Notes**

1.FORMAT() **returns a text value**, so you **cannot use it in calculations**.

2.It is best used for **display purposes** in reports.

📌 (viii) QUARTER

The QUARTER function in DAX returns the **quarter number (1 to 4)** for a given date.

- Syntax

  QUARTER(<date>)

➢ <date>: A column or expression containing a date.

- OrderQuarter(Quarter) = "Q"&QUARTER(Global_Superstore2[Order_Date].[Date])

- FinancialQuarter =

```
SWITCH(QUARTER(Global_Superstore2[Order_Date].[Date]),
      1, 4,  -- Jan-Mar → Q4
      2, 1,  -- Apr-Jun → Q1
      3, 2,  -- Jul-Sep → Q2
     4, 3   -- Oct-Dec → Q3
    )
```

- QuarterYear = "Q" & QUARTER(Global_Superstore2[Order_Date].[Date]) & " - " & YEAR(Global_Superstore2[Order_Date].[Date])

```
QuarterName =
SWITCH(
    QUARTER(Global_Superstore2[Order_
Date].[Date]),
    1, "Q1 (Jan - Mar)",
    2, "Q2 (Apr - Jun)",
    3, "Q3 (Jul - Sep)",
    4, "Q4 (Oct - Dec)"
)
```

💡 **Key Takeaways**

✅ QUARTER(date) returns **1, 2, 3, or 4** based on the date.

✅ You can use SWITCH() to display **formatted quarter names**.

✅ Use CALCULATE() to **aggregate sales by quarter**.

✅ Adjust quarters for a **Fiscal Year starting in April**.

📌 (ix) STARTOFMONTH

The STARTOFMONTH function is used to **return the first date of the month** for a given date column or table. This is useful when working with **monthly aggregations, time intelligence calculations, or financial reports**.

Syntax:- STARTOFMONTH(<dates>)

- <dates> → A column containing date values or a table with a date column.
- Returns: The first date of the month for each row in the column.

```
FirstDayOfMonth = STARTOFMONTH(Global_Superstore2[Order_Date].[Date])

MonthlySales(Som) =
CALCULATE(SUM(Global_Superstore2[Sales]),FILTER(Global_Superstore2,Globa
l_Superstore2[Order_Date].[Date]=STARTOFMONTH(Global_Superstore2[Order_D
ate].[Date])))
```

📌 **Key Takeaways**

✅ STARTOFMONTH(date_column) **returns the first day of each month**.

✅ Useful for **time-based calculations**, such as **monthly comparisons & aggregations**.

✅ Can be used in **measures & calculated columns** to filter data.

📌 (x) STARTOFYEAR

The STARTOFYEAR function in DAX is used to **return the first date of the year** for a given date column. It is commonly used in **time intelligence calculations** like Year-to-Date (YTD) analysis, comparing current vs. previous year sales, and more.

Syntax:- STARTOFYEAR(<dates> [, <fiscal_year_end_month>])

🔹 **Parameters**

1.<dates> → A column containing date values.

2.<fiscal_year_end_month> (Optional) → A month number (1 to 12) indicating the **end of the fiscal year**.

- If omitted, the function assumes a **calendar year** (January to December).
- If provided, it adjusts to **fiscal year calculations**.

🔹 **Returns**

👉 The **first date of the year** (or fiscal year) for each row in the column.

```
FirstDayofYear = STARTOFYEAR(Global_Superstore2[Order_Date].[Date])
FirstDayofFiscalYear = STARTOFYEAR(Global_Superstore2[Order_Date].[Date],"3/31")
```

📌 (xi) TODAY()

The TODAY() function in **DAX** returns the **current system date** without the time component.

Syntax:- Today()

- **No parameters required.**
- Returns a **date** value with only the **current date (no time).**
- The value updates automatically every day when the report refreshes.

```
CurrentDate = TODAY()
DaysSinceOrder = DATEDIFF(Global_Superstore2[Order_Date].[Date], TODAY(), DAY)
CurrentYearSales = CALCULATE(SUM(Global_Superstore2[Sales]),
YEAR(Global_Superstore2[Order_Date]) = YEAR(TODAY()))
```

🚀 **Key Points**

✅ TODAY() returns the current **system date** (without time).

✅ Updates automatically when the **Power BI dataset refreshes**.

✅ Useful for **calculating age, filtering data, and comparisons**.

📌 (xii) YEAR()

The YEAR() function in **DAX** extracts the **year** from a given date value.

Syntax:- YEAR(<date>)

- <date> → A column or expression containing a date value.
- Returns an **integer** representing the **year** of the given date.

- OrderYear = YEAR(Global_Superstore2[Order_Date].[Date])
- CurrentYearSales(Year) =

```
    CALCULATE(
        SUM(Global_Superstore2[Sales]),
        FILTER(Global_Superstore2,
YEAR(Global_Superstore2[Order_Date]) =
YEAR(MAX(Global_Superstore2[Order_Date].[Date]))
)
)
```

- LastYearSales =

```
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    FILTER(Global_Superstore2,
YEAR(Global_Superstore2[Order_Date].[Date]) =
YEAR(MAX(Global_Superstore2[Order_Date].[Date])) - 1)
)
```

- FinancialYear =

```
IF(
   MONTH(Global_Superstore2[Order_Date].[Date]) >= 4,
   YEAR(Global_Superstore2[Order_Date].[Date]),
   YEAR(Global_Superstore2[Order_Date].[Date]) - 1
)
```

🚀 Key Points

✅ YEAR() extracts the **year** from a date value.
✅ Works well with **filters, calculations, and comparisons**.
✅ Combine with TODAY() for **dynamic time-based analysis**.

📌 (xiii) YEARFRAC

The YEARFRAC() function in **DAX** calculates the **fractional year** between two dates. It returns a **decimal number** representing the total number of days between the two dates as a fraction of a year.

Syntax:- YEARFRAC(<start_date>, <end_date> [, <basis>])

Arguments:

- **<start_date>** → The beginning date.
- **<end_date>** → The ending date.
- **<basis>** *(optional)* → Determines how the year is calculated.
  - If omitted, the default basis is 0 (360-day year).

🔢 Basis Options:

Basis	Description
0	30/360 (US) (default)
1	Actual/Actual
2	Actual/360
3	Actual/365
4	European 30/360

YearFraction = YEARFRAC(DATE(2023, 01, 01), DATE(2023, 07, 01))
Output: 0.5 (Half a year between January 1 and July 1)

YearDiff = YEARFRAC(DATE(2022, 05, 10), DATE(2024, 05, 10), 3)

- 🔹 Since the basis is 3 (Actual/365), it calculates the exact fraction of years.

Output: 2.0000 (Exactly 2 years)

```
LastOrdTenure =
ROUND(YEARFRAC(Global_Superstore2[Order_Date].[Date],TODAY(),3),2)
```

🚀 **Key Takeaways**

✅ YEARFRAC() returns the **fractional difference** in years.
✅ Useful for **calculating tenure, age, and time durations**.
✅ The **basis argument** allows flexibility in how the year length is computed.

📌 (xiv) SAMEPERIODLASTYEAR

The **SAMEPERIODLASTYEAR** function returns the same period (day, month, or year) from the previous year, based on a given date column. It is mainly used for year-over-year (YoY) comparisons.

Syntax:- SAMEPERIODLASTYEAR(<date_column>)

--> <date_column>: A column with date values.

--> Returns: A table containing dates from the same period in the previous year.

- Sales_LastYear(SPLY) =
  CALCULATE(SUM(Global_Superstore2[Sales]),SAMEPERIODLASTYEAR(Global_Superstore2[Order_Date].[Date]))

- YOY_Sales_Growth(SPLY) =
```
var
lastyearsales=CALCULATE(SUM(Global_Superstore2[Sales]),SAMEPERIODLASTYEAR(Global_Superstore2[Order_Date].[Date]))
VAR currentYearSales=SUM(Global_Superstore2[Sales])
RETURN
    IF(ISBLANK(lastyearsales),0,DIVIDE(currentYearSales-lastyearsales,lastyearsales,0))
```

- Order_LastYear(SPLY) = CALCULATE(COUNT(Global_Superstore2[Order ID]),SAMEPERIODLASTYEAR(Global_Superstore2[Order_Date].[Date]))

- Profit_LastYear(SPLY) =
  CALCULATE(SUM(Global_Superstore2[Sales]),SAMEPERIODLASTYEAR(Global_Superstore2[Order_Date].[Date]))

- Discount_LastYear(SPLY) =
  CALCULATE(SUM(Global_Superstore2[Discount]),SAMEPERIODLASTYEAR(Global_Superstore2[Order_Date].[Date]))

SAMEPERIODLASTYEAR helps compare sales, orders, profit, discounts, or any metric with the same period last year.

📌 (xv) TOTALYTD

The **TOTALYTD** function calculates the **Year-to-Date (YTD) total** of a given measure (e.g., Sales, Profit) based on a specified date column.

Syntax:- TOTALYTD(<expression>, <date_column>, [<filter>], [<year_end_date>])

- **<expression>**: The measure or column to aggregate (e.g., SUM(Sales))
- **<date_column>**: A column with date values
- **[<filter>]** *(optional)*: Conditions to filter data
- **[<year_end_date>]** *(optional)*: Custom fiscal year-end date (default: December 31)

- Sales_YTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESYTD(Global_Superstore2[Order_Date].[Date]))

- Orders_YTD = TOTALYTD(COUNT(Global_Superstore2[Order ID]),Global_Superstore2[Order_Date].[Date])

- Profit_YTD =TOTALYTD(SUM(Global_Superstore2[Profit]),Global_Superstore2[Order_Date].[Date])

- Discount_YTD =TOTALYTD(SUM(Global_Superstore2[Discount]), Global_Superstore2[Order_Date].[Date])

- Sales_YTD_FirstClass = TOTALYTD(CALCULATE(SUM(Global_Superstore2[Sales]),Global_Superstore2[Ship Mode]="First Class"),Global_Superstore2[Order_Date].[Date])

- **TOTALYTD** calculates Year-to-Date totals for **Sales, Profit, Orders, or any measure**.
- **Filters can be applied** (e.g., for specific product categories).

📌 (xvi) DATESBETWEEN

The **DATESBETWEEN** function returns a **date range** between a **start date** and an **end date**, helping to filter calculations within a specific timeframe.

Syntax:- DATESBETWEEN(<date_column>, <start_date>, <end_date>)

- **<date_column>**: The column containing date values.
- **<start_date>**: The beginning of the date range.
- **<end_date>**: The end of the date range.

- **Sales_Between_Dates(DateBetween) =**
  CALCULATE(SUM(Global_Superstore2[Sales]),DATESBETWEEN(Global_Superstore2[Order_Date].[Date],DATE(2012,01,01),DATE(2013,12,31)))

- **Sales_Last_30_Days(DateBetween) =**
  CALCULATE(SUM(Global_Superstore2[Sales]),DATESBETWEEN(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date])-30,MAX(Global_Superstore2[Order_Date])))

- **Sales_Last_Month(DateBetween) =**

```
VAR LastMonthStart=EOMONTH(MAX(Global_Superstore2[Order_Date].[Date]),-1)+1
VAR LastMonthEnd= EOMONTH(MAX(Global_Superstore2[Order_Date].[Date]),0)
RETURN
CALCULATE(SUM(Global_Superstore2[Sales]),DATESBETWEEN(Global_Superstore2[Order_Date].[Date],LastMonthStart,LastMonthEnd))
```

- **Sales_NatCarroll_Q4_2014(DateBetween) =**
  CALCULATE(SUM(Global_Superstore2[Sales]),DATESBETWEEN(Global_Superstore2[Order_Date].[Date],date(2014,10,1),date(2014,12,31)),Global_Superstore2[Customer Name]="Nat Carroll")

- **Orders_Q2_2014(DatesBetween) =**
  CALCULATE(SUM(Global_Superstore2[Sales]),DATESBETWEEN(Global_Superstore2[Order_Date].[Date],DATE(2014,4,1),DATE(2014,6,30)))

- **DATESBETWEEN** helps extract a **specific date range**.
- Can be used for **sales, profit, order count, customer filtering, and more**.

📌 (xvii) DATESINPERIOD

The **DATESINPERIOD** function returns a **date range** based on a specific start date and a relative time period (days, months, quarters, or years). It is useful for creating **rolling periods** like the last 7 days, last 3 months, or the last full year.

Syntax:- DATESINPERIOD(<date_column>, <start_date>, <number_of_intervals>, <interval_type>)

- **<date_column>**: The column containing date values.
- **<start_date>**: The base date from which the period is calculated.
- **<number_of_intervals>**: The number of intervals (positive for future, negative for past).
- **<interval_type>**: The type of period (DAY, MONTH, QUARTER, YEAR).

➢ Sales_Last_7_Days(DATESINPERIOD) =
CALCULATE(SUM(Global_Superstore2[Sales]),DATESINPERIOD(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date]),-7,DAY))

➢ Sales_Last_Quarter(DATESINPERIOD) =
CALCULATE(SUM(Global_Superstore2[Sales]),DATESINPERIOD(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date]),-1,QUARTER))

➢ Profit_Last_3_Months(DATESINPERIOD) =
CALCULATE(SUM(Global_Superstore2[Sales]),DATESINPERIOD(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date]),-3,MONTH))

➢ Orders_Last_Year(DATESINPERIOD) =
CALCULATE(SUM(Global_Superstore2[Sales]),DATESINPERIOD(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date]),-1,YEAR))

➢ Future_Sales_3_Months(DATESINPERIOD) =
CALCULATE(SUM(Global_Superstore2[Sales]),DATESINPERIOD(Global_Superstore2[Order_Date].[Date],MAX(Global_Superstore2[Order_Date]),3,MONTH))

- DATESINPERIOD helps create **rolling periods** dynamically.
- Can be used for **last 7 days, last 3 months, last quarter, last year, or future periods.**

📌 (xviii) DATESMTD

The **DATESMTD** function returns a set of dates from the **beginning of the current month** to the latest available date in the month. It is commonly used to calculate **Month-to-Date (MTD) metrics**, such as **MTD Sales, MTD Profit, and MTD Orders**.

Syntax:- DATESMTD(<date_column>)

❖ **<date_column>:** A column containing date values.

➢ Sales_MTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Profit_MTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Orders_MTD = CALCULATE(COUNT(Global_Superstore2[Order ID]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Sales_MTD_2012 = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]),YEAR(Global_Superstore2[Order_Date])=2012)

- **DATESMTD** is used for **Month-to-Date (MTD) calculations**.
- Useful for **MTD Sales, MTD Profit, MTD Orders, and Running Totals**.

- Creates a **running total of sales from the start of the month** to today.

🔹 **Example Output**

Order Date	Sales	Running Sales MTD
2024-02-01	$2,000	$2,000
2024-02-05	$3,000	$5,000
2024-02-12	$1,500	$6,500

📌 (xviii) DATESMTD

The **DATESMTD** function returns a set of dates from the **beginning of the current month** to the latest available date in the month. It is commonly used to calculate **Month-to-Date (MTD) metrics**, such as **MTD Sales, MTD Profit, and MTD Orders**.

Syntax:- DATESMTD(<date_column>)

❖ **<date_column>:** A column containing date values.

➢ Sales_MTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Profit_MTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Orders_MTD = CALCULATE(COUNT(Global_Superstore2[Order ID]),DATESMTD(Global_Superstore2[Order_Date].[Date]))

➢ Sales_MTD_2012 = CALCULATE(SUM(Global_Superstore2[Sales]),DATESMTD(Global_Superstore2[Order_Date].[Date]),YEAR(Global_Superstore2[Order_Date])=2012)

- **DATESMTD** is used for **Month-to-Date (MTD) calculations**.
- Useful for **MTD Sales, MTD Profit, MTD Orders, and Running Totals**.

📌 (xix) DATESQTD

The **DATESQTD** function returns a set of dates from the **beginning of the current quarter** to the latest available date in that quarter. It is commonly used to calculate **Quarter-to-Date (QTD) metrics**, such as **QTD Sales, QTD Profit, and QTD Orders**.

Syntax:- DATESQTD(<date_column>)

- **<date_column>:** A column containing date values

- Orders_QTD = CALCULATE(SUM(Global_Superstore2[Order ID]),DATESQTD(Global_Superstore2[Order_Date].[Date]))

- Profit_QTD = CALCULATE(SUM(Global_Superstore2[Profit]),DATESQTD(Global_Superstore2[Order_Date].[Date]))

- Sales_QTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESQTD(Global_Superstore2[Order_Date].[Date]))

- Sales_QTD_2013 = CALCULATE(SUM(Global_Superstore2[Sales]),DATESQTD(Global_Superstore2[Order_Date].[Date]),YEAR(Global_Superstore2[Order_Date])=2013)

- DATESQTD is used for **Quarter-to-Date (QTD)** calculations.
- Useful for **QTD Sales, QTD Profit, QTD Orders, and Running Totals.**

📌 (XX) DATESYTD

The **DATESYTD** function returns a set of dates from **the beginning of the year** to the latest available date within the same year. It is used to calculate **Year-to-Date (YTD) metrics**, such as **YTD Sales, YTD Profit, and YTD Orders**.

Syntax:- DATESYTD(<date_column>)

- **<date_column>:** A column containing date values.

- Orderss_YTD=CALCULATE(COUNT(Global_Superstore2[OrderID]),DATESYTD(Global_Superstore2[Order_Date].[Date]))

- Profits_YTD = CALCULATE(SUM(Global_Superstore2[Profit]),DATESYTD(Global_Superstore2[Order_Date].[Date]))

- Sales_YTD_2013 = CALCULATE(SUM(Global_Superstore2[Sales]),DATESYTD(Global_Superstore2[Order_Date]),YEAR(Global_Superstore2[Order_Date]) = 2013)

- Saless_YTD = CALCULATE(SUM(Global_Superstore2[Sales]),DATESYTD(Global_Superstore2[Order_Date].[Date]))

📌 (xxi) PREVIOUSMONTH

The **PREVIOUSMONTH** function returns a **table of dates from the previous month**, based on a given date column. It is used to **compare sales, profit, or other metrics from last month**.

- **Syntax: PREVIOUSMONTH(<date_column>)**
- **<date_column>:** A column containing date values.

➢ Orders_PreviousMOnth = CALCULATE(count(Global_Superstore2[Order ID]),PREVIOUSMONTH(Global_Superstore2[Order_Date].[Date]))

➢ Profit_PreviousMonth = CALCULATE(SUM(Global_Superstore2[Profit]),PREVIOUSMONTH(Global_Superstore2[Order_Date].[Date]))

➢ Sales_Growth_LastMonth =

```
    VAR Sales_Current = SUM(Global_Superstore2[Sales])
    VAR Sales_Previous =
        CALCULATE(
            SUM(Global_Superstore2[Sales]),
            PREVIOUSMONTH(Global_Superstore2[Order_Date])
        )
RETURN
I       F(
        NOT(ISBLANK(Sales_Previous)),
        DIVIDE(Sales_Current - Sales_Previous, Sales_Previous, 0)
        )
```

➢ Sales_PreviousMonth = CALCULATE(SUM(Global_Superstore2[Sales]),PREVIOUSMONTH(Global_Superstore2[Order_Date].[Date]))

- **PREVIOUSMONTH** helps in **last month comparisons**.
- Useful for **Sales, Profit, Orders, Growth, and Running Totals**.

📌 (xxii) NEXTMONTH

The **NEXTMONTH** function returns a **table of dates from the next month**, based on a given date column. It is used to **compare sales, profit, or other metrics with the upcoming month**.

- **Syntax: NEXTMONTH(<date_column>)**
- **<date_column>:** A column containing date values.

➢ Sales_NextMonth =
CALCULATE(SUM(Global_Superstore2[Sales]),NEXTMONTH(Global_Superstore2[Order_Date].[Date]))

➢ Order_NextMonth = CALCULATE(COUNT(Global_Superstore2[Order ID]),NEXTMONTH(Global_Superstore2[Order_Date].[Date]))

➢ Profit_NextMonth =
CALCULATE(SUM(Global_Superstore2[Profit]),NEXTMONTH(Global_Superstore2[Order_Date].[Date]))

➢ Sales_Growth_NextMonth =

```
VAR Sales_Current=SUM(Global_Superstore2[Sales])
VAR
Sales_NextMonth=CALCULATE(SUM(Global_Superstore2[Sales]),NEXTMONTH(Global_Superstore2[Order_Date].[Date]))
RETURN
IF(Sales_NextMonth=0,0,(Sales_NextMonth-Sales_Current)/Sales_Current)
```

- **NEXTMONTH** helps in **future month comparisons**.
- Useful for **Sales, Profit, Orders, Growth, and Running Totals**.

📌 (xxiii) PARALLELPERIOD

The **PARALLELPERIOD** function returns a table of dates shifted **forward or backward** by a specified number of **months, quarters, or years**. It is useful for **comparative time analysis**, such as comparing sales from the same period in previous months or years.

- **Syntax: PARALLELPERIOD(<date_column>, <number>, <interval>)**

- **<date_column>** – A column containing date values.
- **<number>** – Number of intervals (positive for future, negative for past).
- **<interval>** – "Month", "Quarter", or "Year".

- Sales_PreviousMonth(PARALLELPERIOD) =
  CALCULATE(SUM(Global_Superstore2[Sales]),PARALLELPERIOD(Global_Superstore2[Order_Date].[Date],-1,MONTH))

- Profit_PreviousMonth(ParallelPeriod) =
  CALCULATE(SUM(Global_Superstore2[Profit]),PARALLELPERIOD(Global_Superstore2[Order_Date].[Date],-1,MONTH))

- Orders_LastYearSameMonth(ParallelPeriod) =
  CALCULATE(COUNT(Global_Superstore2[Order ID]),PARALLELPERIOD(Global_Superstore2[Order_Date].[Date],-1,YEAR))

- Sales_Growth_LastYear(PrallelPeriod) =

```
VAR Sales_Crrent=SUM(Global_Superstore2[Sales])
VAR
Sales_LastYear=CALCULATE(SUM(Global_Superstore2[Sales]),PARALLELPERIOD(Global_Superstore2[Order_Date].[Date],-1,YEAR))
  RETURN
        IF(Sales_LastYear=0,0,(Sales_Crrent-Sales_LastYear)/Sales_LastYear)
```

- **PARALLELPERIOD** shifts the time period by **months, quarters, or years**.
- Used for **previous month/quarter/year comparisons**.
- Useful for **Sales, Profit, Orders, Growth, and Running Totals**.

📌 (XXiV) FIRSTDATE

The **FIRSTDATE** function returns the **earliest date** in a column or within a given filter context. It is useful for identifying the start date of a dataset, a specific product launch, or the first sale of a period.

- **Syntax: FIRSTDATE(<date_column>)**
- **<date_column>** – A column containing date values.

- First_Order_Date(FirstDate) = FIRSTDATE(Global_Superstore2[Order_Date].[Date])

- Product_First_Sales(FirstDate) = CALCULATE(FIRSTDATE(Global_Superstore2[Order_Date].[Date]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Product ID]))

- Customer_FirstOrder(FirstDate) = CALCULATE(FIRSTDATE(Global_Superstore2[Order_Date].[Date]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Customer ID]))

- Categort_First_Sales(FirstDate) = CALCULATE(FIRSTDATE(Global_Superstore2[Order_Date].[Date]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Category]))

- **FIRSTDATE** finds the **earliest date in a dataset or filtered context**.
- Useful for **first orders, product launches, or tracking first purchases per customer**.
- Can be combined with ALLEXCEPT for **grouped first dates**.

📌 (XXV) LASTDATE

The **LASTDATE** function returns the **most recent date** in a column or within a given filter context. It is useful for identifying the latest order date, last purchase per customer, or the last recorded date for any analysis.

- **Syntax: LASTDATE(<date_column>)**
- **<date_column>** – A column containing date values.

- Last_Order_Date(LASTDATE) = LASTDATE(Global_Superstore2[Order_Date].[Date])

- Product_Last_Sale(LASTDATE) =

```
CALCULATE(
   LASTDATE(Global_Superstore2[Order_Date].[Date]),
   ALLEXCEPT(Global_Superstore2, Global_Superstore2[Product ID]))
```

- Customer_LastOrder(LastDate) =
  CALCULATE(LASTDATE(Global_Superstore2[Order_Date].[Date]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Customer ID]))

📌 (XXVi) LASTNONBLANK

The **LASTNONBLANK** function returns the **last (most recent) non-blank value** in a column, based on a given expression. This is useful when analyzing sales trends, last recorded data, or finding the last available entry before a missing value.

- **Syntax: LASTNONBLANK(<column>, <expression>)**

- **<column> –** A column containing date values (typically a date field).
- **<expression>** – An expression that evaluates whether a row is considered non-blank (e.g., **SUM(Sales), COUNT(Orders),** etc.).

- Last_Sales_Date(LASTNONBLANK) =
  LASTNONBLANK(Global_Superstore2[Order_Date].[Date],SUM(Global_Superstore2[Sales]))

- Customer_LastOrder(LASTNONBLANK) =
  CALCULATE(LASTNONBLANK(Global_Superstore2[Order_Date].[Date],SUM(Global_Superstore2[Sales])),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Customer ID]))

- Product_Last_Sales(LASTNONBLANK) =
  CALCULATE(LASTNONBLANK(Global_Superstore2[Order_Date].[Date],SUM(Global_Superstore2[Sales])),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Product ID]))

- **LASTNONBLANK** finds the **last non-empty date in a dataset** based on a condition.
- Useful for **tracking last sales dates, last purchases, and last data updates**.
- **Avoids blank values**, ensuring accuracy in sales reporting.

3. Filter Functions

📌 (i) ALL

The **ALL** function removes all filters from a table or specific columns. It is useful for **calculating totals, ignoring slicers, and removing context filters in calculations**.

- **Syntax:ALL(<table_or_column>)**
- **<table_or_column>** – The table or column to ignore filters on.

- Total_Sales_All =
  CALCULATE(SUM(Global_Superstore2[Sales]),ALL(Global_Superstore2))

- Total_Sales_All_Categories(All) =
  CALCULATE(SUM(Global_Superstore2[Sales]),ALL(Global_Superstore2[Category]))

- Total_Sales_All_Region(All) =
  CALCULATE(SUM(Global_Superstore2[Sales]),all(Global_Superstore2[Region]))

- Total_Orders_All_Customers(All) =
  CALCULATE(SUM(Global_Superstore2[Sales]),all(Global_Superstore2[Customer ID]))

- Sales_Percentage_Region(All) =
  DIVIDE(SUM(Global_Superstore2[Sales]),CALCULATE(SUM(Global_Superstore2[Sales]),ALL(Global_Superstore2[Region])))*100

- Average_Sales_Per_Order_All =
  ```
  CALCULATE(
      AVERAGE(Global_Superstore2[Sales]),
      ALL(Global_Superstore2))
  ```

🚀 **Summary**

- **ALL removes filters** from a table or column.
- Useful for **calculating totals, percentages, and ignoring slicers**.
- Works with **SUM, AVERAGE, COUNT, and other aggregate functions**.

📌 (II) ALLSELECTED

The **ALLSELECTED** function removes filters **except those applied in the visual or report-level filters**. It is useful when you want to **calculate totals while respecting filters applied in slicers**.

- **Syntax: ALLSELECTED(<table_or_column>)**
- **<table_or_column>** – The table or column where filters should be kept from the report.

➢ Total_Sales_Selected(AllSelected) =

```
CALCULATE(
   SUM(Global_Superstore2[Sales]),
   ALLSELECTED(Global_Superstore2)
)
```

➢ Total_Sales_Region_Selected(ALLSELECTED) =

```
CALCULATE(
   SUM(Global_Superstore2[Sales]),
   ALLSELECTED(Global_Superstore2[Region])
)
```

➢ Total_Orders_Selected_Customers(ALLSELECTED) =

```
CALCULATE(
   COUNT(Global_Superstore2[Order ID]),
   ALLSELECTED(Global_Superstore2[Customer ID])
)
```

➢ Sales_Percentage_Category_Selected(ALLSELECTED) =

```
DIVIDE(
   SUM(Global_Superstore2[Sales]),
   CALCULATE(SUM(Global_Superstore2[Sales]),
ALLSELECTED(Global_Superstore2[Category]))
) * 100
```

ALLSELECTED Ekdum Real-Life Example

Soch **ek Shopping Mall hai** jisme **4 brands ke stores hain**:

- **Nike**
- **Adidas**
- **Puma**
- **Reebok**

Ab maan lo **mall ke andar ek shopkeeper hai** jo **poore mall ka total sales dekhna chahta hai**, ek **customer hai jo sirf apni pasand ke brands dekhna chahta hai**, aur ek **manager hai jo ek specific filter lagake dekhta hai**.

Yeh 3 log alag tareeke se sales dekhna chahte hain:

🎯 Scenario 1: ALL() (Poora Mall ka Data Chahiye)

Mall ka **owner** chahta hai ki **chahe koi bhi brand select ho ya na ho, usko total sales dikhe**.

Matlab **poore mall ka total fix rahe**.

DAX Copy Edit

```
Total_Sales_ALL = CALCULATE(
    SUM(Sales[Amount]),
    ALL(Sales[Brand])
)
```

💡 **Koi bhi filter lagao, iska total fix rahega!**

➡️ **Nike select karo ya na karo, total sabka aayega (Nike + Adidas + Puma + Reebok).**

🎯 Scenario 2: ALLSELECTED() (Customer Sirf Apni Pasand Ke Brands Dekhna Chahta Hai)

Ab ek customer hai jo sirf "Nike" aur "Adidas" dekhna chahta hai.

Lekin usko Nike aur Adidas ka % contribution bhi dekhna hai sirf uske selected brands ke basis pe, na ki poore mall ke basis pe.

DAX Copy Edit

```
Sales_Percentage =
DIVIDE(
    SUM(Sales[Amount]),
    CALCULATE(SUM(Sales[Amount]), ALLSELECTED(Sales[Brand]))
)
```

💡 Yeh kya karega?

➡️ Agar customer "Nike" aur "Adidas" select karega, toh sirf in dono ka total lega, pura mall ka nahi!

Brand	Sales	% of Total
Nike	5000	41.67%
Adidas	7000	58.33%

- Sirf Nike aur Adidas ka % show hoga, pura mall ka nahi!
- ALLSELECTED() ka magic yahi hai ki yeh external filters ko maanega!

🎯 Scenario 3: ALLSELECTED() vs ALL() (Manager Ka Filter)

Ek **mall manager hai jo sirf 2023 ka data dekhna chahta hai.**

Woh chahta hai ki **jo bhi brands uske slicer me hain sirf unka total calculate ho**.

💡 Agar hum **ALL() use karenge**, toh **poore mall ka total aa jayega** chahe koi filter laga ho ya nahi.

💡 Agar hum **ALLSELECTED() use karenge**, toh sirf **jo filter select kiya hai uske andar ka total ayega**.

Year	Brand	Sales	% of Selected Total
2023	Nike	5000	**41.67%**
2023	Adidas	7000	**58.33%**

- **ALL() hota toh pura mall ka total leta** (Nike, Adidas, Puma, Reebok sabka).
- **ALLSELECTED() ne sirf "Nike" aur "Adidas" ka total liya** jo slicer me selected the.

💡 Final Conclusion (Ekdum Simple)

✅ **ALL() → Koi bhi filter ho, hata do! (Pura mall ka total le aayega)**

✅ **ALLSELECTED() → Sirf jo slicer me selected hai unka total le aayega!**

🎯 **ALL() kab use karein?**

- Jab aapko **poora dataset ka total** dikhana ho **chahe user kuch bhi select kare**.

🎯 **ALLSELECTED() kab use karein?**

- Jab aapko **sirf selected filters ka respect karna ho** aur **baaki internal filters hatane ho**.

🔥 ALL vs ALLSELECTED Summary Table

Function	Filters from Report?	Filters from DAX?	Use Case
ALL()	❌ Remove	❌ Remove	Pura dataset ka total dikhana
ALLSELECTED()	✅ Respect	❌ Remove	Sirf selected filters ka total dikhana

Difference Between ALL & ALLSELECTED

Function	Kaam Kya Karta Hai?	Kab Use Kare?
ALL(Global_Superstore2[Region])	Poore dataset ka filter hata deta hai, slicer ka koi effect nahi hota	Jab tumhe **poore dataset ka total** chahiye bina kisi filter ke
ALLSELECTED(Global_Superstore2[Region])	Sirf **slicer se selected values** ka total calculate karega	Jab tumhe **selected values ka total** chahiye

- **ALLSELECTED removes row-level filters but keeps slicer selections**.
- Useful for **calculating totals while respecting user selections in visuals**.
- Works with **SUM, AVERAGE, COUNT, and percentage calculations**.

📌 (III) ALLEXCEPT

The **ALLEXCEPT** function removes **all filters from a table except for the specified column(s)**. It is useful when you want to calculate totals while keeping certain filters applied.

(ALLEXCEPT ek DAX function hai jo **table ke sabhi filters ko hata deta hai, sirf un columns ke filter ko rakh ke jo hum specify karte hain**. Matlab, ye **ALL** function jaisa kaam karta hai, lekin ek ya zyada columns ke filter ko preserve karta hai.)

- **Syntax: ALLEXCEPT(<table>, <column1>, <column2>, ...)**
- **<table> –** The table from which all filters will be removed.(Jis table se filters hatane hain.)
- **<column1>, <column2>, ... –** The columns that will retain their filters.(Ye wo columns hain jinke filters **preserve** karne hain.)

ALLEXCEPT vs ALL

Function	Kya karta hai?
ALL(Table)	Pura filter hata deta hai (poori table ke liye).
ALLEXCEPT(Table, Column1)	Sirf diye gaye column ka filter rakhta hai, baaki sab hata deta hai.

Conclusion:

•**ALLEXCEPT** ka use tab hota hai jab tum **kuch specific filters rakhna chahte ho, par baaki sab hata dena chahte ho**.

•**ALL** ke comparison me **ALLEXCEPT zyada flexible hota hai**, kyunki tum **choose kar sakte ho ki kaunse columns ka filter hatana hai** aur kaunse rakhna hai.

- Total_Sales_Except_Category(AllExcept) =
  CALCULATE(SUM(Global_Superstore2[Sales]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Category]))

- Total_Sales_Except_Region_SubCategory(Allexcept) =
  CALCULATE(SUM(Global_Superstore2[Sales]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Region],Global_Superstore2[Sub-Category]))

- Total_Profit_Except_OrderDate(AllExcept) =

```
CALCULATE(
   SUM(Global_Superstore2[Profit]),
   ALLEXCEPT(Global_Superstore2, Global_Superstore2[Order_Date].[Date]))
```

- Total_Orders_Except_Segment(AllExcept) =
  CALCULATE(count(Global_Superstore2[Order ID]),ALLEXCEPT(Global_Superstore2,Global_Superstore2[Segment]))

- Average_Discount_Except_Product(AllExcept) =

```
CALCULATE(
   AVERAGE(Global_Superstore2[Discount]),
   ALLEXCEPT(Global_Superstore2, Global_Superstore2[Product Name])
)
```

•**ALLEXCEPT** removes all filters **except for the columns you specify**.

•Works well with **SUM, AVERAGE, COUNT, and percentage calculations**.

•**Ensures only selected columns retain filtering in reports**.

📌 (iV) REMOVEFILTERS

The **REMOVEFILTERS** function removes **all filters** from a table or a column. It is useful when you want to calculate values without being affected by filters applied in a report.

(REMOVEFILTERS() ek DAX function hai jo kisi table ya column se **saare filters hata deta hai**. Yeh mostly tab use hota hai jab hume **poore dataset ka aggregate result dikhana ho bina kisi filter ke.)**

- **Syntax: REMOVEFILTERS(<table | column>)**

<table> – Removes all filters from the specified table.
<column> – Removes all filters from the specified column.
(Yahan tum ek **table ya specific column** ka naam dete ho jiska filter hatana hai.)

➢ Total_Sales_Without_Filters(REMOVEFILTERS) =

```
CALCULATE(
   SUM(Global_Superstore2[Sales]),
   REMOVEFILTERS(Global_Superstore2))
```

➢ Total_Profit_Without_Category_Filter(REMOVEFILTERS) =

```
CALCULATE(
   SUM(Global_Superstore2[Profit]),
   REMOVEFILTERS(Global_Superstore2[Category]))
```

➢ Total_Orders_Without_Region_Filter(REMOVEFILTERS) =

```
CALCULATE(
   COUNT(Global_Superstore2[Order ID]),
   REMOVEFILTERS(Global_Superstore2[Region]))
```

➢ Average_Discount_Without_Segment(REMOVEFILTERS) =

```
CALCULATE(
   AVERAGE(Global_Superstore2[Discount]),
   REMOVEFILTERS(Global_Superstore2[Segment]))
```

- **REMOVEFILTERS** removes filters **from an entire table or specific columns**.
- **Ignores slicers, filters, or selections in a report**.
- **Works well with SUM, COUNT, AVERAGE calculations**.

🔹 Example 1: Pura Table Ka Filter Hatana

Scenario:

Maan lo ek dataset hai `Sales_Data`, jisme columns hain:

- `Region`
- `Product`
- `Sales`

Agar tum **poore table ka filter hata kar total sales dekhna chahte ho**, to yeh formula use kar sakte ho:

DAX Copy Edit

```
Total_Sales_Without_Filters =
CALCULATE( SUM(Sales_Data[Sales]), REMOVEFILTERS(Sales_Data) )
```

✅ Kya Hoga?

Chahe tum Power BI me **koi bhi filter lagao**, yeh formula **poore table ka total sales** hi dikhayega.

🔹 Example 2: Sirf Specific Column Ka Filter Hatana

Scenario:

Maan lo tum **Product ka filter hata kar** lekin **Region ka filter maintain rakhna chahte ho**:

DAX Copy Edit

```
Total_Sales_Without_Product_Filter =
CALCULATE( SUM(Sales_Data[Sales]), REMOVEFILTERS(Sales_Data[Product]) )
```

✅ Kya Hoga?

- **Region ka filter apply rahega**
- **Product ka filter hata diya jayega**
- Agar tum **kisi specific product pe filter lagate ho to bhi iska effect nahi hoga.**

🔹 Example 3: REMOVEFILTERS vs ALLEXCEPT

Case: Mujhe sirf Region ka filter rakhna hai, baaki sab hatana hai

DAX Copy

```
Total_Sales_Region_Wise =
CALCULATE( SUM(Sales_Data[Sales]), ALLEXCEPT(Sales_Data, Sales_Data[Region]) )
```

✅ Kya Hoga?

- Region ka filter apply rahega
- Baaki sab filters remove ho jayenge

🔹 REMOVEFILTERS कब Use करें?

✅ Jab **poore dataset ka aggregate** dekhna ho bina kisi filter ke.

✅ Jab **sirf ek ya kuch specific columns ke filters hata kar calculation karni ho.**

✅ Jab tum **comparison measures bana rahe ho, jaise ki Total vs Filtered Values.**

📌 (V) SELECTEDVALUE

The **SELECTEDVALUE** function returns the value of a column when **only one value is selected**. If multiple values are selected, it returns **the alternate result (optional) or BLANK() by default**.

(SELECTEDVALUE() ek DAX function hai jo **ek column se single selected value return karta hai**. Agar **ek se zyada values select hoti hain, toh yeh blank ya default value return karta hai**.)

- **Syntax: SELECTEDVALUE(<column>, [alternateResult])**

- **<column> –** The column from which to return a value. (Column ka naam jisme se selected value extract karni hai.)

- **[alternateResult] (Optional)** – A value to return if multiple values are selected. (Agar **multiple values selected hain ya koi value select nahi hai**, toh yeh default result return karega.)

- Selected_Category(SelectedValue) = SELECTEDVALUE(Global_Superstore2[Category], "All Categories")

- Selected_Region(SelectedValue) = SELECTEDVALUE(Global_Superstore2[Region],"Multiple Regions Selected")

- Selected_Segment(SELECTEDVALUE) = SELECTEDVALUE(Global_Superstore2[Segment], "Multiple Segments Selected")

- Selected_Ship_Mode(SELECTEDVALUE) = SELECTEDVALUE(Global_Superstore2[Ship Mode], "Select a Ship Mode")

- Selected_Year(SelectedValue) = SELECTEDVALUE(Global_Superstore2[OrderYear],"Multiple Years Selected")

1️⃣ Display the Selected Region Name

👉 Returns the name of the selected **Region** from the slicer.

```
Selected_Region =
SELECTEDVALUE(Global_Superstore2[Region], "Multiple Regions Selected")
```

✅ What it does?

- If **one** region (e.g., "West") is selected → it returns **"West"**.
- If **multiple** regions are selected → it returns **"Multiple Regions Selected"**.

🔹 **Example Output**

Selected Region(s)	Output
West	West
West, East	Multiple Regions Selected

2️⃣ : Measure Use Karna Card Visualization ke Liye

Maan lo tum ek **Card Visual** me selected Region dikhana chahte ho:

```
Selected_Region = SELECTEDVALUE(Sales_Data[Region], "All Regions")
```

✅ Kya Hoga?

- **Agar ek Region select hai** → Wahi Region dikhayega.
- **Agar multiple ya koi bhi select nahi hai** → "All Regions" dikhayega

3️⃣: Conditional Logic ke Saath SELECTEDVALUE

Agar tum **Sales Target ka Status** check karna chahte ho, toh aisa use kar sakte ho:

```
Sales_Status =
IF(
   SELECTEDVALUE(Sales_Data[Sales]) > 5000,
   "Above Target",
   "Below Target"
)
```

✅ Kya Hoga?

- **Agar Sales 5000 se zyada hai**, toh "Above Target" return karega.
- **Agar 5000 se kam hai ya multiple values select hain**, toh "Below Target" return karega.

🚀 **Summary**

•**SELECTEDVALUE** returns the selected value of a column when only **one** value is chosen.
•**If multiple values are selected, it returns an alternate result (or BLANK)**.
•**Very useful for slicers and user selections in reports**.

📌 **SELECTEDVALUE कब Use Karein?**

✅ **Slicer ya filter ke saath** → Jab hume pata ho ki **sirf ek hi value select hogi.**
✅ **Card Visual ke andar dynamic text dikhane ke liye.**
✅ **Conditional Logic ke andar.**
✅ **Default Value set karne ke liye, agar selection na ho ya multiple ho.**

🔥 **Final Thoughts:**

•**SELECTEDVALUE single value ko extract karne ke liye best hai**, lekin agar multiple values hain toh iska use **carefully karna chahiye.**
•**Agar tumhe multiple values manage karni hain, toh VALUES() ya CONCATENATEX() ka use kar sakte ho.**
•**Conditional logic aur dynamic text ke liye yeh ek useful function hai.**

📌 **(VI) VALUES**

The **VALUES** function returns a **single-column table** containing unique values from a column. It is commonly used to:
•Retrieve unique values for filtering.
•Work with slicers and dynamic measures.
•Handle relationships in calculated columns and measures.

(VALUES() ek DAX function hai jo **ek column ki unique values ka table return karta hai**. Yeh **filtered data ke basis par dynamic list deta hai aur slicer ya filter ke saath kaam karta hai**.)

🔹 **Syntax: VALUES(<column>)**
🔹 **<column> –** The column from which to return unique values. (Column ka naam jisme se unique values extract karni hai.)
→ Measure me use nahi hota hai direct use karna hai to calculated column me use hoga

VALUES() Function Kya Karta Hai?

VALUES() ek **table function hai**, jo **ek column ke unique values ka table return karta hai**.
✅ **Agar ek hi value hai** → Yeh **wohi value return karega**.
✅ **Agar multiple values hai** → Yeh **table return karega** (aur measure table return nahi kar sakta, isliye error dega!).

```
➢ Unique_Customer_Segments(VALUES)=VALUES(Global_Superstore2[Segment])

➢ Unique_Order_Years(VALUES) = VALUES(Global_Superstore2[OrderYear])

➢ Selected_Ship_Mode(VALUES) = IF(HASONEVALUE(Global_Superstore2[Ship
  Mode]),VALUES(Global_Superstore2[Ship Mode]),"Multiple Selected")

➢ Category_Count(VALUES)=COUNTROWS(VALUES(Global_Superstore2[Category]))

➢ SELECTEDVALUES_VALUES(VALUES) = VALUES(Global_Superstore2[Region])
```

1️⃣ Get Unique Regions

👉 Returns a table containing **unique** values from the Region column.

Unique_Regions = VALUES(Global_Superstore2[Region])

Unique Regions
East
West
Central
South

✅ What it does?

- Retrieves **unique region names** from the dataset.
- Can be used in **tables, slicers, or filters**.

🔹 **Example Output**

🚀 Summary

- **VALUES** returns a table of **unique values** from a column.
- **Used in filtering, slicers, and calculations**.
- **When combined with COUNTROWS, HASONEVALUE, or IF, it can create powerful measures.**

🚀 Key Takeaways:

1. **VALUES() ko measure me use mat karo** → Measure me CONCATENATEX() use karo.
2. **VALUES() calculated column me work karega** → Direct Region column use karo.
3. **Agar unique region ka slicer banana hai** → Global_Superstore2[Region] ko directly slicer me drag karo.

📌 (VII) FILTER

The **FILTER** function returns a table with rows that meet a specific condition. It is commonly used for:

- Creating custom **filtered tables**.
- Working with **CALCULATE** for conditional aggregations.
- Applying **row-level filters** in DAX measures.

(FILTER() ek **table function** hai jo ek **specified condition ke basis pe ek filtered table return karta hai**. Yeh function tab use hota hai jab tumhe kisi table ke rows ko dynamically filter karna ho **based on a condition**. ⚡)

🔹 **Syntax: FILTER(<table>, <condition>)**

- **<table>** – The table to filter.(Jisme se filter lagana hai.)
- **<condition>** – The condition to apply (returns TRUE for rows to keep). (Jo condition satisfy hone par rows select hongi.)

❑ Orders_2014(FILTER) =
CALCULATE(SUM(Global_Superstore2[Sales]),FILTER(Global_Superstore2,YEAR(Global_Superstore2[Order_Date])=2014))

❑ Last30Days_Sales(FILTER) =
CALCULATE(SUM(Global_Superstore2[Sales]),filter(Global_Superstore2,Global_Superstore2[Order_Date].[Date]>=MAX(Global_Superstore2[Order_Date].[Date])-30))

❑ High_Value_Sales(FILTER) =
CALCULATE(SUM(Global_Superstore2[Sales]),filter(Global_Superstore2,Global_Superstore2[Sales]>1000))

❑ East_Region_Sales(FILTER) =
CALCULATE(SUM(Global_Superstore2[Sales]),FILTER(Global_Superstore2,Global_Superstore2[Region]="East"))

❑ Count_High_Value_Orders(FILTER) = CALCULATE(COUNT(Global_Superstore2[Order ID]),FILTER(Global_Superstore2,Global_Superstore2[Sales]>1000))

Example 1: Filtered Sales for a Specific Region

Agar tumhe **sirf 'East' region ke sales ka total chahiye**, toh FILTER ka use aise hoga:

```
East_Region_Sales =
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    FILTER(Global_Superstore2, Global_Superstore2[Region] = "East")
)
```

🔎 **Explanation:**

✓ **FILTER**(Global_Superstore2, Global_Superstore2[Region] = "East") → Table se sirf ‘East’ region ke rows return karega.

✓ **CALCULATE**(**SUM**(Global_Superstore2[Sales]), ...) → East region ke total sales ka sum karega.

Example 2: Sales Above a Certain Value

Agar tumhe **sirf un orders ka sales total chahiye jisme Sales > 1000 hai**, toh FILTER ka use hoga:

```
High_Value_Sales =
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    FILTER(Global_Superstore2, Global_Superstore2[Sales] > 1000)
)
```

🔎 **Explanation:**

✓ **FILTER**(Global_Superstore2, Global_Superstore2[Sales] > 1000) → Sirf un rows ko select karega jisme sales 1000 se zyada hai.

✓ **CALCULATE**(**SUM**(Global_Superstore2[Sales]), ...) → Un filtered orders ka sum return karega.

🚀 **Summary**

- **FILTER returns a table** with rows meeting a condition.
- **Commonly used with CALCULATE for advanced aggregations**.
- **Powerful for filtering data dynamically in reports**.

✓ **FILTER** table return karta hai, isliye **direct measure me use nahi hota**.

✓ CALCULATE() ka use karo **FILTER ke saath** taaki measure properly kaam kare.

✓ Agar sum ya count chahiye, toh CALCULATE(SUM(...)) ya CALCULATE(COUNT(...)) ka use karo.

📌 (VIII) CALCULATE

The **CALCULATE** function **modifies the context** of a calculation by applying **filters**. It is one of the most powerful DAX functions because it allows:

- **Conditional aggregations** (e.g., Total Sales for a specific year).
- **Dynamic filtering** of tables.
- **Combining multiple conditions** in calculations

(**CALCULATE()** DAX function ka **sabse powerful function** hai. Yeh kisi bhi expression (SUM, AVERAGE, COUNT, etc.) ko ek **specified filter context** ke saath evaluate karne ke liye use hota hai.)

- **Syntax: CALCULATE(<expression>, <filter1>, <filter2>, ...)**

- **<expression>** – The aggregation or calculation to perform. (Jo calculation karni hai (SUM, COUNT, AVERAGE, etc.))
- **<filter1>, <filter2>, ...** – One or more conditions applied to modify the context. (Jitne filters chahiye, utne laga sakte ho)

❑ Total_Sales_2014(CALCULATE) =
CALCULATE(SUM(Global_Superstore2[Sales]),YEAR(Global_Superstore2[Order_Date])=2014)

❑ Total_Sales_High_Discount(CALCULATE) =
CALCULATE(SUM(Global_Superstore2[Sales]),Global_Superstore2[Discount]>0.10)

❑ Total_Profit_West(CALCULATE) =
CALCULATE(SUM(Global_Superstore2[Sales]),Global_Superstore2[Region]="West")

❑ Consumer_Orders_Count(CALCULATE)=CALCULATE(count(Global_Superstore2[Order ID]),Global_Superstore2[Segment]="Consumer")

❑ Avg_Discount_Technology(CALCULATE) =
CALCULATE(AVERAGE(Global_Superstore2[Discount]),Global_Superstore2[Category]="Technology")

🔑 Summary (Why is CALCULATE() Important?)

✅ CALCULATE() kisi bhi **measure ko filter context ke saath evaluate** karne ke liye use hota hai.
✅ Iska **sabse bada advantage hai ki tum multiple filters apply kar sakte ho**.
✅ Isko **SUM, COUNT, AVERAGE, etc. ke saath combine** karke powerful calculations bana sakte ho.

📌 (IX) CALCULATETABLE

The **CALCULATETABLE** function **returns a filtered table** instead of a single aggregated value. It allows you to:

- **Apply dynamic filters** to a table.
- **Modify the context of calculations**.
- **Use the filtered table inside other DAX functions**.

(CALCULATETABLE() function **poore table ka data modify** karta hai **filters apply karke.** Yeh **CALCULATE() ke jaisa hi hai**, bas yeh ek table return karta hai instead of a single value.)

👉 **Syntax: CALCULATETABLE(<table>, <filter1>, <filter2>, …)**

- **<table> –** The original table you want to filter. (Jis table ka filter apply karna hai)
- **<filter1>, <filter2>, ...** – One or more conditions applied to modify the table's context. (Jo conditions apply karni hai)

CALCULATETABLE() sirf New Table banane ke liye use hota hai, isko Measure me nahi likh sakte.

👇 **Sahi Tarika (New Table ke liye):**

```
West_Region_Table = CALCULATETABLE(Global_Superstore2,
   Global_Superstore2[Region] = "West")
```

✔️ **Yeh Power BI me "New Table" create karega**, sirf **West region ka data rakh kar**.
✔️ **Isko "Table Visual" me dalo, to sirf West region ka data dikhega**.

- Orders_2024
  =CALCULATETABLE(Global_Superstore2,YEAR(Global_Superstore2[Order_Date]) = 2014)

- High_Value_Orders = CALCULATETABLE(Global_Superstore2,Global_Superstore2[Sales] > 1000)

❑ **West_2024_Orders =**
CALCULATETABLE(Global_Superstore2,Global_Superstore2[Region] = "West",
YEAR(Global_Superstore2[Order_Date]) = 2013)

❑ **All_Regions_Orders =**
CALCULATETABLE(Global_Superstore2,REMOVEFILTERS(Global_Superstore2[Region]))

🔑 **Summary (Why is CALCULATETABLE() Important?)**
✅ **Poore table ke data ko filter karke modify kar sakta hai**.
✅ **Multiple filters apply kar sakta hai (Region, Year, Sales, etc.)**.
✅ **Ye ek new filtered table return karta hai, jo table visual me dikhaya ja sakta hai**.
✅ **Agar kisi specific filter ko remove karna ho, toh REMOVEFILTERS() ka use kar sakte ho**.

📌 **(X) TREATAS**

The **TREATAS** function applies **a table of values as a filter** to another unrelated table. It is useful when you need to:

- **Apply filters from one table to another** (without direct relationships).
- **Create virtual relationships** in DAX calculations.
- **Use aggregated values from one table to filter another**.

TREATAS() ek powerful function hai jo **ek table ke column(s) ko dusre table ke filter ke roop me treat** karta hai. Yeh **relationships bina create kiye tables ko connect** karne me kaam aata hai.

📜 **Syntax: TREATAS(<Table>, <Column1>, [<Column2>], ...)**

🔹 **<Table>** → A table containing values that will be treated as filters. (Yeh woh table hai jiska data filter ke roop me use hoga.)
🔹 **<Column1>, <Column2>, ...** → The columns where these filters will be applied. (Yeh woh column(s) hai jo filter apply karenge dusre table me.)

1️⃣ Filter Global_Superstore2 Sales Using a Separate Region Table

👉 We **use a table of selected regions** as a filter for Global_Superstore2.

```
Filtered_Sales =
CALCULATE(
   SUM(Global_Superstore2[Sales]),
   TREATAS(
      {"East", "West"},
      Global_Superstore2[Region]
   )
)
```

✅ **What it does?**

- **Filters Global_Superstore2 to only show East and West regions**.
- **Calculates total sales for these regions** without needing a direct relationship.

🔹 **Example Output:**

💰 **Total Sales in East & West → $ 24334894**

- Filtered_Orders = CALCULATE(COUNTROWS(Global_Superstore2),TREATAS(VALUES(YearTable[Year]),Global_Superstore2[Order Year]))
- Filtered_Profit = CALCULATE(SUM(Global_Superstore2[Profit]), TREATAS(VALUES(SegmentTable[Segment]), Global_Superstore2[Segment]))

🔥 **Key Points to Remember:**

1.TREATAS relationships ki tarah kaam karta hai, but bina relationship create kiye.
2.Multiple columns ka filter bhi apply kar sakta hai.
3.Yeh tab useful hai jab tables ke beech direct relationship na ho.

🛠️ 🔍 **Kab Use Karein?**

✔️ Jab tables ke beech **relationship nahi bana sakte**
✔️ Jab hume ek table ke data se doosre table ko **indirectly filter** karna ho
✔️ Jab multiple columns ke basis pe filtering karni ho

📌 (XI) USERELATIONSHIP

The **USERELATIONSHIP** function **activates an inactive relationship** between two tables for a specific calculation.

👉 Why use USERELATIONSHIP?

- When **multiple relationships** exist between two tables, Power BI **only uses the active one**.
- If you need to use an **inactive relationship** (e.g., for different date fields like Order Date & Ship Date), USERELATIONSHIP helps **override the default active relationship**.
- Used **inside CALCULATE** to change the relationship context for that calculation.

🔹 **Syntax: USERELATIONSHIP(<column1>, <column2>)**

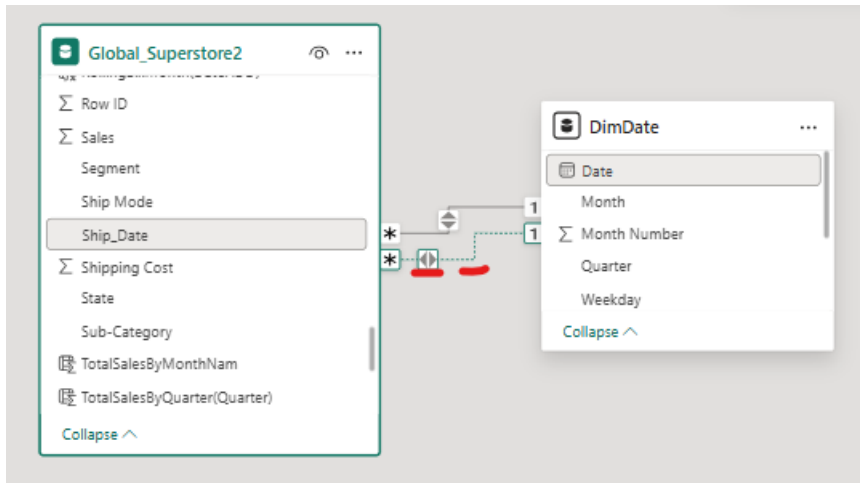
❖ **<column1> –** The column from Table A.
❖ **<column2> –** The column from Table B that forms an inactive relationship with **<column1>.**

USERELATIONSHIP() ek **DAX function** hai jo **inactive relationship ko active** karne ke liye use hota hai. Agar **Power BI ya SSAS me do tables ke beech multiple relationships** ho, toh Power BI **ek relationship ko active rakhta hai aur doosre ko inactive bana deta hai**.

🔹 **USERELATIONSHIP() ka use tab hota hai jab hume inactive relationship activate karna ho kisi specific calculation ke liye.**

```
DimDate = ADDCOLUMNS(
    CALENDAR(DATE(2011,1,1), DATE(2014,12,31)),
    "Year", YEAR([Date]),
    "Month", FORMAT([Date], "MMMM"),
    "Month Number", MONTH([Date]),
    "Quarter", "Q" & FORMAT([Date], "Q"),
    "Weekday", FORMAT([Date], "dddd")
)
```

DimDate = CALENDARAUTO()



1️⃣ Model Relationships Check Karo

1.Power BI ke "Model" View me jao.
2.DimDate[Date] aur Global_Superstore2[Order_Date] ka **"Active" One-to-Many relationship check karo.**
3.DimDate[Date] aur Global_Superstore2[Ship_Date] ka **"Inactive" relationship hona chahiye (dashed line dikhni chahiye).**
4. 📌 **Agar Ship_Date ka relationship nahi bana hai, toh manually banao but mark it as inactive.**

USERELATIONSHIP ka **seedha funda** ye hai ki jab kisi table me **multiple relationships** hote hain, to ye **temporarily ek specific relationship activate** karne ke liye use hota hai.

🔥 **Maal lo tumhare paas Global_Superstore2 table hai jisme do date columns hain:**

1.Order_Date

2.Ship_Date

Aur **DimDate table ke Date column** ke saath **dono relationships bana rakhe ho**:

- **Order_Date → Date (Active)**
- **Ship_Date → Date (Inactive)**

💡 **By default, Power BI sirf Active Relationship (Order_Date → Date) use karega.**

Agar tum Ship_Date ke basis pe calculation karna chahte ho, to **USERELATIONSHIP use karna padega!**

```
Total_Sales_By_ShipDate =
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    USERELATIONSHIP(Global_Superstore2[Ship_Date], DimDate[Date]),
    REMOVEFILTERS(Global_Superstore2[Order_Date])
)
```

Order_Date	Ship_Date	Total_Sales_By_ShipDate	Sum of Sales
03/01/2011	03/01/2011	311.99	311.99
01/01/2011	05/01/2011	110.99	110.99
01/01/2011	06/01/2011	722.52	408.30
02/01/2011	06/01/2011	722.52	314.22
03/01/2011	07/01/2011	2,878.67	2,520.42
05/01/2011	07/01/2011	2,878.67	296.67
07/01/2011	07/01/2011	2,878.67	61.58
01/01/2011	08/01/2011	1,794.69	289.28
03/01/2011	08/01/2011	1,794.69	268.20
04/01/2011	08/01/2011	1,794.69	319.65
07/01/2011	08/01/2011	1,794.69	917.57
03/01/2011	09/01/2011	7,292.85	1,402.94
04/01/2011	09/01/2011	7,292.85	2,229.16
05/01/2011	09/01/2011	7,292.85	3,316.86
06/01/2011	09/01/2011	7,292.85	324.45
07/01/2011	09/01/2011	7,292.85	19.44
Total		**12,642,563.29**	**12,642,563.29**

```
Check_Active_Relationship =
IF(
   ISCROSSFILTERED(DimDate[Date]),
   "Active",
   "Not Active"
)
```

Cross-filter direction "Both" karo.

👆 **Iska matlab hai:**

*"Sales ka sum nikalo lekin Ship_Date aur DimDate[Date] wale **Inactive Relationship ko temporarily Activate** karke!"*

🚀 **TL;DR (Ek Line Me Samajho)**

👉 **USERELATIONSHIP tab use hota hai jab ek table me multiple relationships ho aur tum kisi specific (inactive) relationship ko temporarily activate karna chaho!**

🚀 **Summary**

- **USERELATIONSHIP lets you activate an inactive relationship in a calculation.**
- **Used inside CALCULATE to override the default relationship.**
- **Useful when dealing with multiple date fields (Order Date, Ship Date, Return Date, etc.).**

📌 (XII) CROSSFILTER

The **CROSSFILTER** function **modifies the filter direction** between two related tables.

👉 **Why use CROSSFILTER?**

- Power BI **normally applies filters in one direction** (e.g., from DateTable to Global_Superstore2).
- CROSSFILTER lets you **change the filter direction** dynamically:
  - **None → Removes filtering** between tables.
  - **Both → Filters work in both directions**.
  - **One → Resets to default filtering direction**.

CROSSFILTER function **table relationships ka filter direction control** karne ke liye use hota hai. Agar tumhe kisi relationship ka filter flow **change** ya **disable** karna ho, to ye kaam karta hai.

🚀 **Kab Use Karte Hain?**

1.Default Relationship Ka Filter Direction Change Karne Ke Liye
2.Bi-Directional Filtering Enable/Disable Karne Ke Liye
3.Table Relationships Ko Control Karne Ke Liye

- Syntax: CROSSFILTER(<Column1>, <Column2>, <Filter_Direction>)

- **<Column1>** → The column from Table 1.(Pehli table ka column)
- **<Column2>** → The column from Table 2 (which is related to Column1).(Dusri table ka column (jo Column1 se related hai))
- **<Filter_Direction>** →(Kaunsa filter direction apply karna hai)
  - "None" → **Removes filter between tables**
  - "Both" → **Applies filters in both directions**
  - "One" → **Keeps default direction (table to fact table)**

Mode	Kya Karta Hai?
`NONE`	Relationship ka filter **completely hata deta hai**
`BOTH`	Filter ko **bi-directional** bana deta hai (dono tables ek dusre ko filter kar sakti hain)
`ONEWAY`	Default direction maintain karta hai (Dim Table → Fact Table)

✅ **Example 1: Relationship ka Filter Disable Karna (NONE)**

```
CALCULATE(SUM(Global_Superstore2[Sales]),CROSSFILTER(DimDate[Date],
Global_Superstore2[Order_Date], NONE))
```

👆 **Iska Matlab:**
*"Total Sales ka sum nikalo, **lekin** DimDate[Date] aur Order_Date ke beech jo relationship hai, usko **ignore** kar do (disable kar do)!"*
👉 **Result:** DimDate ka filter kaam nahi karega, aur sales **poore dataset ka sum** return karega.

✅ **Example 2: Bi-Directional Filtering Enable Karna (BOTH)**

```
CALCULATE(SUM(Global_Superstore2[Sales]),CROSSFILTER(DimDate[Date],
Global_Superstore2[Order_Date], BOTH))
```

👆 **Iska Matlab:**
*"Total Sales ka sum nikalo, **aur** DimDate aur Order_Date ke beech jo relationship hai usko **bi-directional filtering** me convert kar do!"*
👉 **Result:** Ab DimDate bhi Global_Superstore2 ko filter karega **aur** vice versa.

✅ **Example 3: Default Filter Direction Maintain Karna (ONEWAY)**

```
CALCULATE( SUM(Global_Superstore2[Sales]), CROSSFILTER(DimDate[Date],
Global_Superstore2[Order_Date], ONEWAY) )
```

👆 **Iska Matlab:**
*"Total Sales ka sum nikalo, **aur** filter direction ko **default (one-way) rakhne do!**"*
👉 **Result:** Filter sirf DimDate se Global_Superstore2 ki taraf jayega, par ulta nahi.

🚀 **Summary**

- **CROSSFILTER modifies the filter direction between two related tables.**
- **"None"** → Removes filtering.
- **"Both"** → Enables bidirectional filtering.
- **"One"** → Resets to default filtering.
- **Used inside CALCULATE to override relationships dynamically.**

👉 **CROSSFILTER tables ke relationships ka filter control karta hai!**

- **NONE** → Filter **hata do (disable relationship)**
- **BOTH** → Filter **dono taraf apply karo (bi-directional)**
- **ONEWAY** → Default direction **maintain karo** (Dim → Fact)

📌(XIII) KEEPFILTERS

👉 The **KEEPFILTERS function** in DAX is used inside CALCULATE to **preserve existing filters when applying new ones**.

👉 Normally, CALCULATE **replaces existing filters** with the new ones applied.

👉 **KEEPFILTERS ensures that both old and new filters work together** instead of overwriting each other.

KEEPFILTERS **existing filters ko preserve karta hai**, jab tum CALCULATE ya CALCULATETABLE use kar rahe hote ho. Yeh function **new filters apply karne ke saath-saath** purane filters ko bhi maintain karta hai, taaki **multiple filters ek saath kaam karein!**

- **Syntax: KEEPFILTERS(<Expression>)**

<Expression> → The filtering condition you want to apply without removing previous filters. (Jo bhi calculation tum CALCULATE ya CALCULATETABLE me kar rahe ho)

```
Total_Sales_Overwritten =
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    Global_Superstore2[Sub-Category]
    = "Chairs")

Total_Sales_KeepFilters =
CALCULATE(
    SUM(Global_Superstore2[Sales]),
    KEEPFILTERS(Global_Superstore2[Sub-Category]
    = "Chairs"))
```

Sub-Category	Total_Sales_KeepFilters	Total_Sales_Overwritten
Accessories		1,501,681.76
Appliances		1,501,681.76
Art		1,501,681.76
Binders		1,501,681.76
Bookcases		1,501,681.76
Chairs	1,501,681.76	1,501,681.76
Copiers		1,501,681.76
Envelopes		1,501,681.76
Fasteners		1,501,681.76
Furnishings		1,501,681.76
Total	**1,501,681.76**	**1,501,681.76**

- Total_Profit_KeepFilters =
  CALCULATE(SUM(Global_Superstore2[Profit]),KEEPFILTERS(Global_Superstore2[Region]="West"),KEEPFILTERS(Global_Superstore2[Segment]="Consumer"))

- Sales_2014_Keepfilter =
  CALCULATE(SUM(Global_Superstore2[Sales]),KEEPFILTERS(YEAR(Global_Superstore2[Order_Date])=2014))

- Sales_Furniture_Technology(KeepFilter) =
  CALCULATE(SUM(Global_Superstore2[Sales]),KEEPFILTERS(Global_Superstore2[Category] IN {"Furniture","Technology"}))

🚀 **Summary**

- **KEEPFILTERS preserves existing filters** when using CALCULATE.
- **Without KEEPFILTERS**, new filters **replace old ones**.
- **Useful in dashboards to avoid removing user-selected filters**.
- **Best for multi-filter scenarios where different filters should remain active together.**

📌 (XIV) ISFILTERED

👉 The **ISFILTERED function** checks if a column **has an active direct filter applied** in the current context.
👉 It **returns TRUE** if the column is filtered, otherwise, it **returns FALSE**.
👉 This is useful when creating **dynamic calculations** that change based on whether a filter is applied.

ISFILTERED(ColumnName) is a DAX function that **checks whether a column is being filtered or not** in the current context.

- **Returns TRUE** → If the column is filtered.
- **Returns FALSE** → If the column is **not filtered** (i.e., showing all values).

🔹 **Syntax: ISFILTERED(<ColumnName>)**

ColumnName → The column for which you want to check if a filter is applied.

- Category_Filter_Status(ISFILTERED) =
  IF(ISFILTERED(Global_Superstore2[Category]),"Filtered","Not Filtered")

- Date_Filter_Status(ISFILTERED) =
  IF(ISFILTERED(Global_Superstore2[Order_Date]),"Filtered","No filter")

- Dynamic_Sales_Calculation(ISFILTERED) = IF(ISFILTERED(Global_Superstore2[Sub-Category]),AVERAGE(Global_Superstore2[Sales]),SUM(Global_Superstore2[Sales]))

- Is_Category_Filtered(ISFILTERED) = ISFILTERED(Global_Superstore2[Category])

- Sales_Only_If_Region_Filtered(ISFILTERED) =
  IF(ISFILTERED(Global_Superstore2[Region]),SUM(Global_Superstore2[Sales]),0)

🚀 **Summary**

- **ISFILTERED(ColumnName) checks if a column has an active filter.**
- **Useful for dynamic calculations in dashboards & reports.**
- **Combines well with IF to create conditional logic.**
- **Works for both categorical (text) and date columns**

📌 (XV) ISBLANK

ISBLANK(Value) is a **DAX function** that checks whether a value is **blank (empty/null)** or not.

- **Returns TRUE** → If the value is **blank (BLANK())**.
- **Returns FALSE** → If the value is **not blank** (i.e., it contains data).

🔹 **Syntax: ISBLANK(<expression>)**

expression → The value or calculation you want to check for blank (BLANK()).

- Is_Profit_Blank(ISBLANK) = ISBLANK(Global_Superstore2[Profit])

- Discount_Status(ISBLANK)=IF(ISBLANK(Global_Superstore2[Discount]),0
  ,Global_Superstore2[Discount])

- Sales_Handling_Blank = IF(ISBLANK(Global_Superstore2[Sales]),0,
  Global_Superstore2[Sales])

❑ Total_Profit_No_Blank(ISBLANK)
=IF(ISBLANK(Global_Superstore2[Profit]),0,Global_Superstore2[Profit])

❑ Total_Profit_No_Blank(ISBLANK) =
IF(ISBLANK(Global_Superstore2[Profit]),0,Global_Superstore2[Profit])

🚀 **Summary**

- **ISBLANK(expression) checks if a value is blank (BLANK()).**
- **Useful for handling missing data and avoiding errors.**
- **Combines well with IF to create conditional logic.**
- **Works with numbers, text, dates, and measures.**

4. Information Functions

📌 (I) ISERROR

👉 The **ISERROR function** checks whether an expression or calculation results in an **error** in DAX.
👉 It **returns TRUE** if the expression results in an error and **FALSE** otherwise.
👉 Useful for handling errors in calculations and preventing report failures.

🔹 **Syntax: ISERROR(<expression>)**

expression → The calculation or value you want to check for errors.

❑ Is_Error_Division(ISERROR) =
ISERROR(Global_Superstore2[Sales]/Global_Superstore2[Quantity])

❑ Sales_Per_Quantity(ISERROR) =
IF(ISERROR(Global_Superstore2[Sales]/Global_Superstore2[Quantity]),0,Global_Superstore2[Sales]/Global_Superstore2[Quantity])

❑ Total_Profit_NO_Error(ISERROR) =
IF(ISERROR(SUM(Global_Superstore2[Profit])),0,SUM(Global_Superstore2[Profit]))

❑ Is_Error_Profit_Per_Order(ISERROR)=ISERROR(SUM(Global_Superstore2[Profit]) /
COUNT(Global_Superstore2[Order ID]))

❑ Discount_Percentage(ISERROR)
=IF(ISERROR(Global_Superstore2[Discount]/Global_Superstore2[Sales]),
"Error",FORMAT(Global_Superstore2[Discount]/Global_Superstore2[Sales], "0.00%"))

🚀 **Summary of ISERROR**

✅ The ISERROR function in DAX is used to **detect errors** in calculations and return TRUE if an error occurs, otherwise FALSE.

✅ It helps prevent report failures by allowing error handling and replacement with alternative values.

📌(II) ISNONTEXT

✅ **Definition:**

The ISNONTEXT function in DAX checks whether a value is **not** a text (string). If the value is **not text**, it returns TRUE; otherwise, it returns FALSE.

- **Syntax: ISNONTEXT(<value>)**
- **<value>** → The value you want to check.

- **How It Works:**

•Returns TRUE if the value is **not text** (e.g., numbers, dates, Boolean, blank, errors).
•Returns FALSE if the value **is** text.

❑ Non_Text() = ISNONTEXT(Global_Superstore2[Sales])

❑ Customer_Name_Not_Text(ISNONTEXT) = ISNONTEXT(Global_Superstore2[Customer Name])

❑ Handling_Non_Text_Value = IF(ISNONTEXT(Global_Superstore2[Discount]),"Numeric Value","Text Value")

- **Why Use ISNONTEXT?**

✓ Helps validate and clean data in Power BI reports.
✓ Ensures correct data types before performing calculations.
✓ Useful for conditional formatting or handling errors.

📌(III) ISNUMBER

✅ **Definition:**

The ISNUMBER function in DAX checks whether a given value is a **number**.
•If the value **is a number**, it returns TRUE.
•If the value **is not a number** (text, date, blank, Boolean, or error), it returns FALSE.

- **Syntax: ISNUMBER(<value>)**
- **<value>** → The value you want to check.

❑ Sales_Is_Number = ISNUMBER(Global_Superstore2[Sales])

❑ OrderDate_Is_Number = ISNUMBER(Global_Superstore2[Order_Date])

❑ CustomerName_Is_Number = ISNUMBER(Global_Superstore2[Customer Name])

❑ Discount_Type = IF(ISNUMBER(Global_Superstore2[Discount]), "Numeric", "Non-Numeric")

❑ Profit_Cleaned = IF(ISNUMBER(Global_Superstore2[Profit]), Global_Superstore2[Profit], 0)

🔹 **Why Use ISNUMBER?**

✓ Helps **correct data types** before performing calculations.
✓ Prevents errors when dealing with **mixed data formats**.
✓ Useful for **data validation** and **conditional formatting** in Power BI.
By using **ISNUMBER**, you can **clean and validate** your data before applying DAX calculations! 🚀

📌 (IV) ISTEXT

✅ **Definition:**

The ISTEXT function in DAX checks whether a given value is **text** (string).
•If the value **is text**, it returns TRUE.
•If the value **is not text** (number, date, Boolean, blank, or error), it returns FALSE.

🔹 **Syntax: ISTEXT(<value>)**
🔹 **<value>** → The value or column to check.

❑ CustomerName_Is_Text = ISTEXT(Global_Superstore2[Customer Name])

❑ OrderID_Is_Text = ISTEXT(Global_Superstore2[Order ID])

❑ Sales_Is_Text = ISTEXT(Global_Superstore2[Sales])

❑ Category_Type = IF(ISTEXT(Global_Superstore2[Category]), "Text", "Non-Text")

❑ Profit_Cleaned = IF(ISTEXT(Global_Superstore2[Profit]), "Unknown", Global_Superstore2[Profit])

🔹 **Why Use ISTEXT?**

✓ Helps **validate data types** before calculations.
✓ Prevents **errors in numeric calculations** by identifying text values.
✓ Useful for **data cleaning** and **conditional formatting** in Power BI.
By using **ISTEXT**, you can ensure your data is formatted correctly for analysis! 🚀

📌 (V) ISLOGICAL

✅ **Definition:**

The ISLOGICAL function in DAX checks whether a given value is a **Boolean (TRUE or FALSE)**.

- If the value **is Boolean**, it returns TRUE.
- If the value **is not Boolean** (number, text, date, blank, or error), it returns FALSE.

- **Syntax: ISLOGICAL(<value>)**
- **<value>** → The value or column to check.

❑ Discount_Is_Logical = ISLOGICAL(Global_Superstore2[Discount])

❑ High_Sales = IF(Global_Superstore2[Sales] > 500, TRUE(), FALSE())

❑ High_Sales_Is_Logical = ISLOGICAL([High_Sales])

❑ Order_Priority_Is_Logical = IF(ISLOGICAL(Global_Superstore2[Order Priority]), "Boolean", "Non-Boolean")

❑ Filtered_Logical_Values = FILTER(Global_Superstore2, ISLOGICAL(Global_Superstore2[Ship Mode]))

❑ Customer_Segment_Logical = IF(ISLOGICAL(Global_Superstore2[Customer Segment]), Global_Superstore2[Customer Segment], FALSE())

5. Logical Functions

(I) IFERROR

The IFERROR function in DAX is used to handle errors in calculations.

- If the given expression **returns an error**, IFERROR replaces it with a specified value.
- If there is **no error**, it simply returns the result of the expression.

- **Syntax: IFERROR(<value>, <alternate_result>)**
- **<value>** → The expression to evaluate.
- **<alternate_result>** → The value to return if an error occurs.

❑ **Sales_Per_Quantity =** IFERROR(Global_Superstore2[Sales] / Global_Superstore2[Quantity], 0)

❑ **Profit_Per_Order** = IFERROR(Global_Superstore2[Profit] /COUNT(Global_Superstore2[Order ID]), "Error in Calculation")

❑ **Valid_Discount** = **IFERROR**(**VALUE**(Global_Superstore2[Discount]), "Invalid Discount")
❑ **Average_Profit_Per_Sale** = **IFERROR**(AVERAGE(Global_Superstore2[Profit]), 0)
❑ **Converted_Profit** = **IFERROR**(**CONVERT**(Global_Superstore2[Profit], **INTEGER**), 0)

🔹 **Why Use IFERROR?**
✓ Prevents **calculation failures** in Power BI reports.
✓ Ensures **smooth data visualization** without unexpected errors.
✓ Helps **handle missing, invalid, or incorrect data** in datasets.
Using **IFERROR**, you can **fix potential issues in your calculations** and **ensure error-free reports**! 🚀

📌 (II) SWITCH

The SWITCH function in DAX is a conditional function that evaluates an expression and returns a corresponding result based on predefined conditions.
•It works like a **more efficient version of nested IF statements**.
•It is useful for categorizing data and creating **custom grouping logic**.

SWITCH() function ka kaam hai **multiple conditions check karna aur uske hisaab se output dena** – bilkul jaise **IF..ELSE LADDER** ki tarah!

🔹 **Syntax: SWITCH(<expression>, <value1>, <result1>, <value2>, <result2>, ..., [<else_result>])**

•**<expression>** → The value to compare. (Yeh evaluate hoga (e.g., column value ya measure)).
•**<value1>, <value2>, ...** → The possible values of the expression. (Yeh values compare ki jayengi.)
•**<result1>, <result2>, ... →** The output corresponding to each value. (Inke corresponding output return honge.)
•**[<else_result>]** ***(optional)*** → The default value if no match is found. (Agar koi match nahi mila, to yeh default value return karega.)

❑ Order_Priority_Caregory(SWITCH) = SWITCH(Global_Superstore2[Order Priority],
"Critical", " 🚨 High Priority",
"High", " ⚠️ Medium Priority",
"Medium", " 🔹 Low Priority",
"Low", " ✅ Normal Priority",
"Unknown")

Profit_Category(SWITCH) =

```
SWITCH(
    TRUE(),
    Global_Superstore2[Profit] > 1000, " 🟢 High Profit",
    Global_Superstore2[Profit] > 500, " 🟡 Medium Profit",
    Global_Superstore2[Profit] > 0, " 🟠 Low Profit",
    " 🔴 Loss"
)
```

Shipping_Type(SWITCH) =

```
SWITCH(
    Global_Superstore2[Ship Mode],
    "Same Day", " 🚀 Express Shipping",
    "First Class", " ✈️ Fast Shipping",
    "Second Class", " 🚢 Standard Shipping",
    "Standard Class", " 📦 Economy Shipping",
    "Unknown"
)
```

Discount_Label(SWITCH) =

```
SWITCH(
    TRUE(),
    Global_Superstore2[Discount] >= 0.5, " 🔥 Huge Discount",
    Global_Superstore2[Discount] >= 0.3, " ⚡ Big Discount",
    Global_Superstore2[Discount] > 0, " 🛒 Small Discount",
    "No Discount"
)
```

Customer_Segment(SWITCH) =

```
SWITCH(
    Global_Superstore2[Segment],
    "Consumer", " 🏠 Individual Customer",
    "Corporate", " 🏢 Business Client",
    "Home Office", " 💻 Small Business",
    "Unknown"
)
```

🔹 **Why Use SWITCH?**

✓ **Easier to read** than multiple IF statements.
✓ **Faster performance** in Power BI reports.
✓ **Great for categorization** and **grouping** data dynamically.
By using **SWITCH**, you can create **cleaner** and **more efficient** logic for your **Power BI dashboards**! 🚀

📌(III) ISINSCOPE

The ISINSCOPE function in DAX checks whether a specific column is currently being filtered within a hierarchical structure (e.g., when using row-level grouping in Power BI tables, matrices, or visuals).

- It is useful for identifying if a field is **inside a specific hierarchy level**.
- Mainly used in **dynamic calculations** and **conditional formatting** based on drill-down levels.

(**ISINSCOPE()** ka kaam hai **check karna ki koi column ek particular hierarchy level pe filter ho raha hai ya nahi**. Yeh **mainly hierarchies (jaise Category -> Sub-Category)** ke andar filter detect karne ke liye use hota hai.)

🔹 **Syntax: ISINSCOPE(<columnName>)**

- ❑ **<columnName>** → The column you want to check if it's currently in scope (filtered or grouped).
- ❑ Returns **TRUE** if the column is in the current scope. (Yeh TRUE return karta hai jab ek specific column ka filter apply ho)
- ❑ Returns **FALSE** if the column is not in scope. (FALSE return karega agar us column ka filter apply nahi hai)
- ❑ **Mostly tables ya matrix visualizations me use hota hai**

- ❑ Is_Category_In_Scope = **ISINSCOPE**(Global_Superstore2[Category])

- ❑ Is_SubCategory_In_Scope = **ISINSCOPE**(Global_Superstore2[Sub-Category])

- ❑ Sales_Display = **IF**(**ISINSCOPE**(Global_Superstore2[Category]),**SUM**(Global_Superstore2[Sales]), **AVERAGE**(Global_Superstore2[Sales])

- ❑ Format_Display = **IF**(**ISINSCOPE**(Global_Superstore2[Segment]), " 🔹 Detailed View"," 📊 Summary View")

🔹 **Why Use ISINSCOPE?**

✓ Helps in **dynamic measures and conditional formatting**.
✓ Useful for **drill-down levels** in Power BI tables and visuals.
✓ Helps in **custom ranking and dynamic aggregation calculations**.

By using ISINSCOPE, you can **customize your Power BI reports** dynamically, making them **more interactive and user-friendly**! 🚀

📌 (IV) COALESCE

The COALESCE function in DAX **returns the first non-blank value** from a list of expressions.

- If all values are **blank**, it returns BLANK().
- It is useful for handling **missing values** in reports by providing a fallback option.

COALESCE() ek **Power BI DAX function** hai jo **pehli non-blank value** return karta hai. Agar sab blank hai, to ek **default value** return hoti hai.

- **Syntax: COALESCE(<Expression1>, <Expression2>, ..., <ExpressionN>)**

- **Expression1, Expression2, ...** → A list of values or calculations.
- The function returns **the first non-blank value** it finds from left to right.
- If **all expressions are blank**, it returns BLANK().

❑ Sales_Corrected(COALESCE) = COALESCE(SUM(Global_Superstore2[Sales]), 0)

❑ Order_Quantity_Corrected(COALESCE) =COALESCE(SUM(Global_Superstore2[Quantity]), 1)

❑ Discount_Profit(COALESCE)
=COALESCE(SUM(Global_Superstore2[Discount]),SUM(Global_Superstore2[Profit]), 0)

✅ If Discount is blank, it **returns Profit**; if both are blank, it **returns 0**.

- **Why Use COALESCE?**

✓ Ensures **no blank values in reports**.
✓ Selects the **best available data** from multiple columns.
✓ Prevents errors when using **SUM, AVERAGE, or other aggregations**.
By using COALESCE, your Power BI dashboards become **more reliable and user-friendly!**

📌 (V) VAR

The VAR function in DAX allows you to define **variables** inside a DAX expression.

- It helps **simplify complex calculations** by storing intermediate values.
- **Improves performance** by reducing redundant calculations.
- **Enhances readability** of DAX formulas.

(VAR ka use DAX me **temporary variable** store karne ke liye hota hai, jisse calculations optimize ho jaye aur readability improve ho.)

- **Syntax:** VAR **<VariableName> = <Expression>**
  RETURN **<FinalExpression>**

•**VAR <VariableName>** → Defines a variable with a calculated value. (Yahan pe ek temporary variable store hota hai)
•**RETURN <FinalExpression>** → Uses the variable in the final calculation. (Final output generate karne ke liye hota hai)
•You can **declare multiple variables** inside one formula.

```
Profit_Margin(VAR) = VAR TotalSales = SUM(Global_Superstore2[Sales])
VAR TotalProfit = SUM(Global_Superstore2[Profit])
RETURN TotalProfit / TotalSales

Order_Category(VAR) = VAR OrderSales = SUM(Global_Superstore2[Sales])
RETURN IF(OrderSales > 1000, "High-Value Order", "Regular Order")

Discount_Percentage(VAR) =
VAR TotalDiscount = SUM(Global_Superstore2[Discount])
VAR TotalSales = SUM(Global_Superstore2[Sales])
RETURN IF(TotalSales > 0, TotalDiscount / TotalSales, 0)

Avg_Sales_Per_Order(VAR) =
VAR TotalSales = SUM(Global_Superstore2[Sales])
VAR OrderCount = DISTINCTCOUNT(Global_Superstore2[Order ID])
RETURN TotalSales / OrderCount

Adjusted_Region_Profit(VAR) =
VAR RegionSales = SUM(Global_Superstore2[Sales])
VAR RegionProfit = SUM(Global_Superstore2[Profit])
RETURN IF(RegionSales > 5000, RegionProfit * 1.1, RegionProfit)
```

- **Why Use VAR in DAX?**

✓ **Improves performance** by avoiding repetitive calculations.
✓ **Makes DAX easier to read** by breaking down complex formulas.
✓ **Enhances flexibility** for conditional logic and dynamic calculations.
Using VAR makes your DAX formulas **cleaner, faster, and easier to debug!**

6. Text Functions

📌(I) CONCATENATE

The CONCATENATE function in DAX is used to **combine two text values** into a single string.

- It **only accepts two arguments at a time** (unlike CONCATENATEX).
- If more than two values need to be combined, multiple CONCATENATE functions must be **nested**.

- **Syntax: Measure_Name = CONCATENATE(<Text1>, <Text2>)**

•**Text1** → The first text value.
•**Text2** → The second text value.
📌 **Important Notes:**
•Both arguments must be **text**; if they are numbers, they need to be converted using FORMAT().
•If **either argument is blank**, the function still returns a valid result without errors.

❑ Customer_City(CONCATENATE) = CONCATENATE(Global_Superstore2[Customer Name], " - " & Global_Superstore2[City])

❑ Product_Detail(CONCATENATE) = CONCATENATE(Global_Superstore2[Category], " - " & Global_Superstore2[Product Name])

❑ Order_Info(CONCATENATE) = CONCATENATE(Global_Superstore2[Order ID], " | " & FORMAT(Global_Superstore2[Order_Date], "DD-MMM-YYYY"))

❑ Sales_Profit_Summary(CONCATENATE) = CONCATENATE("Sales: $" & FORMAT(SUM(Global_Superstore2[Sales]), "0.00"), " | Profit: $" & FORMAT(SUM(Global_Superstore2[Profit]), "0.00"))

❑ Unique_Order_Key(CONCATENATE) = CONCATENATE(Global_Superstore2[Order ID], "-" & Global_Superstore2[Customer ID])

🚀 **Why Use CONCATENATE in DAX?**
✓ **Joins text fields dynamically** for better data presentation.
✓ **Improves readability** of reports with combined text values.
✓ **Useful for creating unique identifiers** in datasets.
📌 **For more flexibility with multiple values, use CONCATENATEX() instead!** 🚀

📌 (II) CONCATENATEX

The CONCATENATEX function in DAX **joins text values** from a table or column, separated by a **specified delimiter**. Unlike CONCATENATE, which works with only two values, CONCATENATEX can **concatenate multiple values** dynamically.

- Works well when **combining multiple rows into a single string**.
- Allows **custom sorting** of values before concatenation.
- Can **apply calculations** before concatenation (e.g., converting numbers to text).

(Ye **table ke multiple rows ka data ek string me combine** karne ke kaam aata hai, aur **custom delimiter** bhi add kar sakte hain!**)**

- **Syntax: CONCATENATEX(<Table>, <Expression>, [Delimiter], [OrderBy_Column], [Order])**

•**Table** → The table or filtered dataset containing values to concatenate.
•**Expression** → The column or calculated expression to concatenate.
•**Delimiter** (optional) → A separator (e.g., ", " or " | "). Default is "" (no separator).
•**OrderBy_Column** (optional) → The column to sort values before concatenation.
•**Order** (optional) → Sort direction: ASC (ascending) or DESC (descending).

📌 **Key Difference from CONCATENATE**
✓ CONCATENATE works on **two** values, while CONCATENATEX works on **multiple rows dynamically**.

Products_Per_Order(CONCATENATEX) = CONCATENATEX(VALUES(Global_Superstore2[Product Name]), Global_Superstore2[Product Name], ", ")

```
1  Products_Per_Order = CONCATENATEX(VALUES(Global_Superstore2[Product Name]), Global_Superstore2[Product Name], ", ")
2
```

Key(CONCATENATE)	
268-CS-12460	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
165-TR-21325	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
025-JC-15385	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
795-RB-19360	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
37-CM-12190	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
697-LC-17140	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
112-JB-15925	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
333-KB-16315	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
98-RF-19840	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
112-GT-14755	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
997-ES-14020	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
361-FH-14350	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
320-JK-15205	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
351-NC-18340	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
147-MO-17950	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
99-TB-21055	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
338-GH-14425	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
697-LC-17140	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
389-MC-18100	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
373-MC-17845	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
053-LT-17110	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
353-DC-12850	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,
307-TC-21295	Plantronics CS510 - Over-the-Head monaural Wireless Headset System, Novimex Executive Leather Armchair, Black, Nokia Smart Phone, with Caller ID, Motorola Smart Phone,

Customer_Purchases = **CONCATENATEX**(**FILTER**(Global_Superstore2, Global_Superstore2[Customer Name] = "John Doe"), Global_Superstore2[Product Name], " | ", Global_Superstore2[Product Name], **ASC**)

Orders_Per_Customer = **CONCATENATEX**(**VALUES**(Global_Superstore2[Order ID]), Global_Superstore2[Order ID], ", ", Global_Superstore2[Order Date], **ASC**)

Sales_Summary = **CONCATENATEX**(Global_Superstore2, "Order " & Global_Superstore2[Order ID] & " - $" & FORMAT(Global_Superstore2[Sales], "0.00"), " | ")

Categories_By_Region = **CONCATENATEX**(**VALUES**(Global_Superstore2[Category]), Global_Superstore2[Category], ", ")

🚀 **Why Use CONCATENATEX in DAX?**
✓ **Handles multiple rows dynamically** instead of just two values.
✓ **Supports custom sorting** before concatenation.
✓ **Works great in aggregating reports**, creating summaries, and improving data presentation.
📌 **For handling more than two values dynamically, CONCATENATEX is the best choice!** 🚀

📌 (III) REPLACE

The REPLACE function in DAX is used to **replace part of a text string** with a new text value. It is useful for modifying text data dynamically, such as correcting typos, masking sensitive information, or standardizing formats.

🔹 **Syntax: REPLACE(<Old_Text>, <Start_Position>, <Num_Characters>, <New_Text>)**

- **Old_Text** → The original text string.
- **Start_Position** → The position (starting from 1) where replacement begins.
- **Num_Characters** → The number of characters to replace.
- **New_Text** → The text that will replace the removed portion.

📌 **Key Notes:**
✓ The function is **position-based** (not case-sensitive).
✓ If Start_Position is greater than the length of Old_Text, it returns the original text.
✓ If Num_Characters is **0**, it inserts New_Text at Start_Position without deleting anything.

❑
```
Masked_Customer_Name(REPLACE) = REPLACE(Global_Superstore2[Customer Name], 2, 4, "****")
```

❑
```
Fixed_Product_Code = REPLACE(Global_Superstore2[Product ID], 1, 1, "P")
```

❑ Formatted_Order_ID = REPLACE(Global_Superstore2[Order ID], 3, 0, "-")

❑ Updated_Order_Year = REPLACE(Global_Superstore2[Order Date], 7, 4, "2024")

❑ Updated_Product_Name = REPLACE(Global_Superstore2[Product Name], 1, 7, "NewModel")

🚀 **Why Use REPLACE in DAX?**
✓ **Useful for modifying text dynamically** (e.g., formatting, masking, fixing errors).
✓ **Flexible for inserting, replacing, and standardizing text data**.
✓ **Works well in data cleaning and transformation tasks** in Power BI reports.
📌 **Use REPLACE when you need to modify parts of a string based on position!** 🚀

📌 (IV) SEARCH

The SEARCH function in DAX **returns the position** of a specified substring within a text string. It is useful for locating specific words, characters, or patterns in text fields, helping in **text analysis, filtering, and extraction** tasks.

- **Syntax: SEARCH(<Find_Text>, <Within_Text>, [Start_Position], [Not_Found_Value])**

•**Find_Text** → The substring you want to search for.
•**Within_Text** → The text string in which to search.
•**Start_Position** *(optional)* → The character position where the search should start (default is 1).
•**Not_Found_Value** *(optional)* → The value to return if the substring is not found (default is an error).

📌 **Key Notes:**
✓ SEARCH is **case-insensitive** (e.g., "Laptop" and "laptop" are treated the same).
✓ If Find_Text is not found, it returns an **error** unless you specify Not_Found_Value.
✓ If Start_Position is greater than the text length, it returns an error.

❑ Laptop_Position = **SEARCH**("Laptop", Global_Superstore2[Product Name], 1, -1)

- **Example Output:** "Laptop Bag" → 1
- **Example Output:** "Dell Inspiron Laptop" → 15
- **If "Laptop" is not found, returns -1 instead of an error.**

❑ First_Space_Position = **SEARCH**(" ", Global_Superstore2[Customer Name], 1, -1)

❑ Contains_CA = **IF**(**SEARCH**("CA", Global_Superstore2[Order ID], 1, 0) > 0, "Yes", "No")

🚀 **Why Use SEARCH in DAX?**
✓ **Helps extract, locate, or categorize data** based on text patterns.
✓ **Useful for conditional logic**, such as checking if a substring exists.
✓ **Essential for text cleaning, formatting, and transformations** in Power BI.
📌 **Use SEARCH when you need to find a substring's position dynamically!** 🚀

📌 (V) TEXT

The TEXT function in DAX **converts a value into a formatted text string** using a specified format. It is useful for customizing number, date, and time formats, making reports more readable and visually appealing.

🔹 **Syntax: TEXT(<Value>, <Format>)**

- **Value** → The number, date, or time value to format.
- **Format** → The format pattern (e.g., "DD/MM/YYYY", "0.00", "#,###", etc.).

📌 **Key Notes:**
✓ The TEXT function **does not affect the actual value**, only its display.
✓ It is useful for **formatting dates, numbers, and currency values** in reports.
✓ The Format parameter must be **in double quotes (" ")** and follow Excel-style formatting.

❑ Formatted_Order_Date = **TEXT**(Global_Superstore2[Order Date], "DD/MM/YYYY")

❑ Formatted_Sales = **TEXT**(Global_Superstore2[Sales], "#,###")

❑ Discount_Percentage = **TEXT**(Global_Superstore2[Discount], "0.00%")

❑ Formatted_Profit = **TEXT**(Global_Superstore2[Profit], "₹#,##0.00")

❑ Month_Year = **TEXT**(Global_Superstore2[Order Date], "MMM YYYY")

🚀 **Why Use TEXT in DAX?**
✓ **Enhances report readability** by displaying values in user-friendly formats.
✓ **Useful for custom date and number formatting** in Power BI dashboards.
✓ **Helps in text-based grouping and comparisons** (e.g., "Jan 2024" vs "Feb 2024").
📌 **Use TEXT to control how your data appears while keeping its original value unchanged!** 🚀

📌 (VI) VALUE

The VALUE function in DAX **converts a text string that represents a number into an actual numeric value**. This is useful when numbers are stored as text and need to be used in calculations.

- **Syntax: VALUE(<Text>)**

•**Text** → The text string that represents a number.

📌 **Key Notes:**

✓ If the text can be converted to a number, VALUE returns the numeric equivalent.
✓ If the text cannot be converted, it **returns an error**.
✓ Useful for working with **imported data where numbers are stored as text**.

❑ Order_ID_Numeric = VALUE(Global_Superstore2[Order ID])

❑ Sales_Numeric = VALUE(Global_Superstore2[Sales])

❑ Discount_Numeric = VALUE(Global_Superstore2[Discount]) / 100

❑ Profit_Numeric = VALUE(SUBSTITUTE(Global_Superstore2[Profit], "₹", ""))

- **Example Input:** "₹5000" → **Output:** 5000

❑ Shipping_Cost_Numeric = VALUE(Global_Superstore2[Shipping Cost])

🚀 **Why Use VALUE in DAX?**

✓ **Essential for calculations** when numbers are stored as text.
✓ **Prevents errors** when performing aggregations (SUM, AVERAGE, etc.).
✓ **Works well with imported datasets where numbers might be in text format.**
📌 **Use VALUE to convert text-based numbers into real numeric values for calculations!**

📌 (VII) SUBSTITUTE

The SUBSTITUTE function in DAX **replaces a specific part of a text string with another text string**. It is useful for cleaning or transforming text data.

- **Syntax: SUBSTITUTE(<Text>, <Old_Text>, <New_Text>, [Instance_Num])**

•**Text** → The original text string.
•**Old_Text** → The text that needs to be replaced.
•**New_Text** → The new text that will replace the old text.
•**Instance_Num** *(Optional)* → If provided, replaces only that specific occurrence. Otherwise, replaces all occurrences.

📌 **Key Notes:**

✓ If Instance_Num is **not provided**, all occurrences of Old_Text are replaced.
✓ If Old_Text is not found, the function **returns the original text**.
✓ It is **case-sensitive**.

❖ Sales_Cleaned = **SUBSTITUTE**(Global_Superstore2[Sales], "₹", "")

❖ Region_Standardized = **SUBSTITUTE**(Global_Superstore2[Region], "South-East", "Southeast")

❖ Product_Name_Cleaned = **SUBSTITUTE**(Global_Superstore2[Product Name], "_", " ")

❖ Formatted_Order_Date = **SUBSTITUTE**(Global_Superstore2[Order Date], "/", "-")

❖ Customer_Name_Cleaned = **SUBSTITUTE**(Global_Superstore2[Customer Name], " ", " ")

🚀 **Why Use SUBSTITUTE in DAX?**

✓ **Great for cleaning and transforming text data.**
✓ **Helps standardize inconsistent entries in datasets.**
✓ **Can be used in calculated columns or measures to format data dynamically.**
📌 **Use SUBSTITUTE to clean and refine text data for better accuracy in analysis!** 🚀

📌(VIII) FIND

The FIND function in DAX **searches for a substring within a text string and returns the starting position of the substring**. If the substring is not found, it returns an error.

🔹 **Syntax: FIND(<Find_Text>, <Within_Text>, [<Start_Position>], [<NotFoundValue>])**

•**Find_Text** → The substring you want to search for.
•**Within_Text** → The text string in which you are searching.
•**Start_Position** *(Optional)* → The position to start searching from (**default = 1**).
•**NotFoundValue** *(Optional)* → The value to return if the substring is **not found** (**default = error**).

📌 **Key Notes:**

✓ **Case-sensitive** (e.g., "Apple" ≠ "apple").
✓ Returns the **position (index)** of the first occurrence of Find_Text.
✓ If Find_Text is not found, it **throws an error** unless NotFoundValue is specified.
✓ Unlike Excel, **DAX does not support wildcard characters** (*, ?).

- ❑ Laptop_Position = **FIND**("Laptop", Global_Superstore2[Product Name], 1, -1)
- ❑ Office_Found = **FIND**("Office", Global_Superstore2[Category], 1, 0)
- ❑ Express_Found = **FIND**("Express", Global_Superstore2[Ship Mode], 1, -1)
- ❑ Hyphen_Position = **FIND**("-", Global_Superstore2[Order ID])

🚀 **Why Use FIND in DAX?**
✓ **Helps extract and analyze text-based data.**
✓ **Useful in data cleaning and validation (emails, product names, IDs).**
✓ **Enhances conditional logic in measures and calculated columns.**
📌 **FIND is essential for working with text-based filtering and data extraction in Power BI!**

7. Parent-Child Functions

📌(I) LOOKUPVALUE

The LOOKUPVALUE function **returns a single value from a column based on specified search conditions**. It works like a VLOOKUP in Excel but is more flexible and powerful.

🔹 **Syntax: LOOKUPVALUE(<Result_ColumnName>, <Search_ColumnName>, <Search_Value>[, <Alternate_Result>])**

- **Result_ColumnName** → The column from which you want to retrieve the value. **(Jo value return karni hai)**
- **Search_ColumnName** → The column where the function searches for a match. **(Jisme search karna hai)**
- **Search_Value** → The value to search for in Search_ColumnName. **(Jo value match karni hai)**
- **Alternate_Result** *(Optional)* → The value returned if no match is found (**default = error**). **(Agar match na mile to kya return kare)**

📌 **Key Notes:**
✓ Returns a **single value** (if multiple matches exist, it returns an error).
✓ Can use multiple search conditions (Search_ColumnName2, Search_Value2, etc.).
✓ Works well for **retrieving related data without relationships**.

LOOKUPVALUE tabhi sahi kaam karega jab search ki gayi value (Customer Name ya koi bhi column) UNIQUE ho.
Agar **multiple rows match hoti hain, to LOOKUPVALUE error de sakta hai** ya **BLANK return kar sakta hai.**

LOOKUPVALUE() ka Simple Example

Scenario:

Maan lo tumhare paas ek **Customers** table hai, jisme Customer Name aur Region diya gaya hai. Tumhe Orders table me har Customer Name ka corresponding Region lana hai.

✅ **Customers Table:**

Customer Name	Region
Aakash	North
Priya	South
Rohan	West

✅ **Orders Table:**

Order ID	Customer Name	Sales	Region (Required)
101	Aakash	500	???
102	Priya	700	???

Region_Lookup =
LOOKUPVALUE(Customers[Region], Customers[Customer Name], Orders[Customer Name])

Row ID Sales_Lookup =
LOOKUPVALUE(Global_Superstore2[Sales], Global_Superstore2[Row ID],15810)

Output (Orders Table after Lookup)

Order ID	Customer Name	Sales	Region
101	Aakash	500	North
102	Priya	700	South

🎯 **Kya Ho Raha Hai?**

- LOOKUPVALUE Customers table se Region ko Orders table ke Customer Name ke basis pe fetch kar raha hai.
- Agar match milta hai, to respective Region return karega. Agar match nahi milega, to BLANK() return hoga.

🚀 **Why Use LOOKUPVALUE in DAX?**
✓ **Works like a flexible VLOOKUP** for retrieving values from a table.
✓ **Avoids relationships when quick lookups are needed.**
✓ **Supports multiple conditions** for precise data retrieval.
📌 **Best for searching & retrieving values dynamically in Power BI reports!** 🚀

9. Rank Functions

📌 (I) RANKX

The RANKX function in DAX is used to **return the ranking of a value in a column based on a specified expression**. It assigns a rank to each row based on the order defined in the expression.

(RANKX() ek **ranking function** hai jo kisi **table ke ek column ke values ko compare karke unka rank assign** karta hai, **highest ya lowest value ke basis pe**.)

- **Syntax: RANKX(<Table>, <Expression>, [<Value>], [<Order>], [<Ties>])**

•**Table** → The table over which the ranking is applied.
•**Expression** → The calculation or column used for ranking (e.g., Sales).
•**Value** *(Optional)* → The specific value to rank (rarely used).
•**Order** *(Optional, Default = Descending)* →
•DESC (**Descending** → Highest value gets **Rank 1**).
•ASC (**Ascending** → Lowest value gets **Rank 1**).
•**Ties** *(Optional, Default = Skip)* →
•Dense → No gaps in ranking for ties.
•Skip → Skips ranks for ties (default).

📌 **Key Notes:**
✓ Ranks are assigned dynamically based on the filtered context.
✓ **Can be used with ALL, ALLEXCEPT, or ALLSELECTED** to control ranking scope.
✓ Works best with numerical data like **Sales, Profit, or Quantity Sold**.

```
Category_Rank_Segment =
RANKX(ALLEXCEPT(Global_Superstore2,Global_Superstore2[Segment]),CALCULATE(SUM(Global_Superstore2[Sales])), , DESC, Skip)
```

```
Customer_Profit_Rank = RANKX(ALLSELECTED(Global_Superstore2[Customer ID]),CALCULATE(SUM(Global_Superstore2[Profit])),,DESC,Dense)

Product_Sales_Rank = RANKX(ALL(Global_Superstore2[Product Name]),CALCULATE(SUM(Global_Superstore2[Sales])),,DESC,Dense)

Region_Sales_Rank = RANKX(ALL(Global_Superstore2[Region]),CALCULATE(SUM(Global_Superstore2[Sales])),,DESC,Skip)

Sales_Rank_By_Category =RANKX(ALLSELECTED(Global_Superstore2[Category]),CALCULATE(SUM(Global_Superstore2[Sales])),,DESC, DENSE)
```

Category	Sales_Rank_By_Category	Sum of Sales
Technology	1	4,744,557.50
Furniture	2	4,110,874.19
Office Supplies	3	3,787,131.61
Total	**1**	**12,642,563.29**

🎯 Sample Data Table (`Global_Superstore2`)

Product Name	Sales	Rank (Standard)	Rank (Dense)
Laptop	5000	1	1
Printer	4500	2	2
Mouse	4000	3	3
Keyboard	4000	3	3 (*Tie - Same Rank*)
Monitor	3500	5	4
Speakers	3000	6	5

SKIP

1 4SKIP

🚀 Why Use RANKX in DAX?

✓ **Easily ranks values dynamically** based on measure calculations.
✓ **Flexible ranking criteria** (Sales, Profit, Orders, etc.).
✓ **Can rank within specific groups** (Regions, Segments, Categories).
✓ **Useful for leaderboard reports, sales performance analysis, and trend tracking!**

📌 (II) RANK.EQ

The RANK.EQ function in DAX **returns the rank of a number within a column**, assigning the same rank to duplicate values and skipping the next rank. It works similarly to the RANKX function but is primarily used in **Excel**, while RANKX is preferred in DAX for Power BI.

(RANK.EQ() ek ranking function hai jo kisi column ya measure ki ranking calculate karta hai **ties ke saath**. Matlab agar **do values same hoti hain, to unko same rank assign hoti hai**, aur next rank **skip ho jata hai**.)

🔹 **Syntax: RANK.EQ(<value>, <column_name>, <order>)**

•**Number** → The value whose rank you want to determine. (Jo value rank karni hai (usually SUM, AVERAGE, etc.))
•**Column** → The column containing the values to rank against. (Jisme ranking karni hai)
Order *(Optional, Default = Descending)* → (
•ASC ya DESC (**Ascending ya Descending order**))
•0 (**Descending**) → Highest value gets **Rank 1**.
•1 (**Ascending**) → Lowest value gets **Rank 1**.