DAX in Power BI

Parwaaz Solutions

Contents

1. Introduction to DAX

Data Analysis Expressions (DAX) is a powerful formula language used primarily in Microsoft Power BI, Excel Power Pivot, and Analysis Services Tabular models. It is designed for data modeling and analysis, enabling users to create custom calculations and aggregations to work with data effectively. Here's an introduction to DAX:

1.2 Features of DAX

1. Purpose of DAX:

DAX is used to perform calculations, data modeling, and analysis on structured data. Its primary use cases include creating custom calculations, defining measures, aggregating data, and generating insights from your data.

2. Data Modeling:

DAX works in conjunction with data modeling. When you import and transform data in Power BI or other Microsoft tools, you create relationships between tables. DAX operates on these tables and relationships to provide a dynamic and interactive way to analyze data.

3. Calculations in DAX:

DAX formulas are used for creating calculations. These calculations can be as simple as basic arithmetic or as complex as statistical analysis. DAX supports a wide range of functions, operators, and expressions.

4. Measures:

In Power BI, one of the most common uses of DAX is to create measures. Measures are calculations that aggregate data for reporting purposes. They're used in visuals like charts, tables, and matrices, and they update dynamically as you interact with your reports.

5. Aggregation and Filtering:

DAX allows you to perform aggregations and filtering operations. You can calculate sums, averages, counts, and more. Filters applied to visuals and slicers can change the context in which these calculations are performed.

6. Time Intelligence:

DAX includes a set of functions specifically designed for working with date and time data. You can easily perform calculations related to time, such as year-to-date, month-to-date, or rolling averages.

7. Context in DAX:

DAX calculations operate within two main contexts: filter context and row context. Filter context is defined by filters applied to visuals, while row context focuses on individual rows in tables when performing calculations.

8. Variables and Error Handling:

DAX allows you to use variables to store intermediate results, making your code more readable and efficient. It also supports error handling to deal with potential issues in your calculations.

9. Optimization:

Optimizing DAX code is crucial for performance. You need to be mindful of formula complexity and the number of calculations required to render reports quickly.

In summary, DAX is a versatile language for data analysis and modeling, especially in the context of Power BI. It empowers users to create custom calculations, visualize data effectively, and gain valuable insights from their data. As you become more familiar with DAX, you can perform increasingly sophisticated data analysis tasks, making it a valuable tool for business intelligence and reporting.

1.3 Data Types in DAX

Model data type	DAX data type	Description
Whole number	64-bit integer	-2^{63} through $2^{63}-1$
Decimal number	64-bit real	Negative: -1.79×10^{308} through -2.23×10^{-308} - zero (0) - positive: 2.23×10^{-308} through 1.79×10^{308} - Limited to 17 decimal digits
Boolean	Boolean	TRUE or FALSE
Text	String	Unicode character string
Date	Date/time	Valid dates are all dates after March 1, 1900
Currency	Currency	-9.22×10^{14} through 9.22×10^{14} - limited to four decimal digits of fixed precision
N/A	BLANK	In some cases, it's the equivalent of a database (SQL) NULL

1.4 Operators in DAX

Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

Comparison operators

Operator	Description
=	Equal to
==	Strict equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Text concatenation operator

Use the ampersand (&) character to connect, or concatenate, two text values to produce one continuous text value. For example, consider the following calculated column definition:

DAXCopy

Model Color = 'Product'[Model] & "-" & 'Product'[Color]

Logical operators

Use logical operators to combine expressions that produce a single result. The following table lists all logical operators.

Operator	Description
&&	Creates an AND condition between two expressions where each has a Boolean result. If both expressions return TRUE, the combination of the expressions also returns TRUE; otherwise the combination returns FALSE.
|| (double pipe)	Creates an OR condition between two logical expressions. If either expression returns TRUE, the result is TRUE; only when both expressions are FALSE is the result FALSE.
IN	Creates a logical OR condition between each row that is being compared to a table. Note: The table constructor syntax uses braces.
NOT	Inverts the state of a Boolean expression (FALSE to TRUE, and vice versa).

2. Application of DAX in Power BI

2.1 Introduction

By using Data Analysis Expressions (DAX), you can add three types of calculations to your data model:

- Calculated tables
- Calculated columns
- Measures

Note

DAX can also be used to define row-level security (RLS) rules, which are expressions that enforce filters over model tables. However, rules aren't considered to be model calculations so they're out of scope for this module. For more information, see Row-level security (RLS) with Power BI.

2.2 Calculated tables

You can write a DAX formula to add a calculated table to your model. The formula can duplicate or transform existing model data, or create a series of data, to produce a new table. Calculated table data is always imported into your model, so it increases the model storage size and can prolong data refresh time.

Note

A calculated table can't connect to external data; you need to use Power Query to accomplish that task.

Calculated tables can be useful in various scenarios:

- Date tables
- Role-playing dimensions
- What-if analysis

Date tables

Date tables are required to apply special time filters known as time intelligence. DAX time intelligence functions only work correctly when a date table is set up. When your source data doesn't include a date table, you can create one as calculated tables by using the CALENDAR or CALENDARAUTO DAX functions.

Role-playing dimensions

When two model tables have multiple relationships, it could be because your model has a role-playing dimension. For example, if you have a table named Sales that includes two date columns, OrderDateKey and ShipDateKey, both columns are related to the Date column in the Date table.

What-if analysis

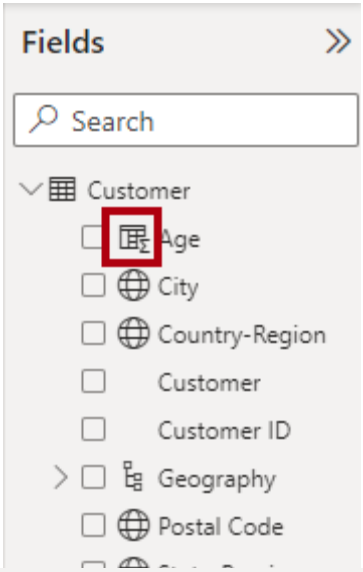
Power BI Desktop supports a feature called What-if parameters. When you create a what-if parameter, a calculated table is automatically added to your model.

What-if parameters allow report users to select or filter by values that are stored in the calculated table. Measure formulas can use selected value(s) in a meaningful way. For example, a what-if parameter could allow the report user to select a hypothetical currency exchange rate, and a measure could divide revenue values (in a local currency) by the selected rate.

2.3 Calculated columns

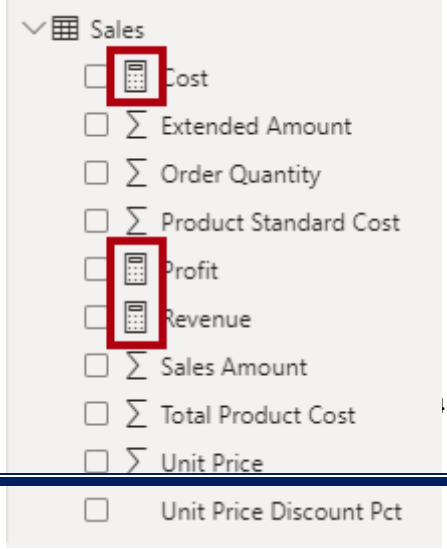
You can write a DAX formula to add a calculated column to any table in your model. The formula is evaluated for each table row and it returns a single value. When added to an Import storage mode table, the formula is evaluated when the data model is refreshed, and it increases the storage size of your model. When added to a DirectQuery storage mode table, the formula is evaluated by the underlying source database when the table is queried.

In the **Fields** pane, calculated columns are enhanced with a special icon. The following example shows a single calculated column in the **Customer** table called **Age**.



2.4 Measures

You can write a DAX formula to add a measure to any table in your model. The formula is concerned with achieving summarization over model data. Similar to a calculated column, the formula must return a single value. Unlike calculated columns, which are evaluated at data refresh time,



measures are evaluated at query time. Their results are never stored in the model.

In the **Fields** pane, measures are shown with the calculator icon. The following example shows three measures in the **Sales** table: **Cost**, **Profit**, and **Revenue**.

2.5 RLS using DAX

Row-Level Security () in Power BI allows you to control and restrict data access for different users or roles within your reports and dashboards. It's a security feature that uses DAX expressions to filter the data a user can see at the row level. Here's a brief overview of how to implement using DAX:

1. Create a Security Table:

- Create a table in your data model that contains at least two columns: one for role names and another for DAX filter expressions.

- Define the filtering criteria for each role in this table.

2. Create Roles:

- In Power BI Desktop, go to "Model" view and use the "Manage Roles" option to create the roles you want to implement.

- Link each role to the Security Table.

3. Define Role Filters:

- For each role, specify a DAX expression in the Security Table that defines the filtering condition.

- These expressions determine which data rows users in each role can access.

4. Assign Users to Roles:

- In the Power BI Service, assign users or groups to the roles you created.

- This step associates individual users or groups with specific roles.

5. Test :

- Publish your report to the Power BI Service.

- Verify that users can only access the data allowed by their role's DAX filter expression.

- Test the setup thoroughly to ensure data security.

using DAX is a powerful way to enforce data security and privacy in your Power BI reports by restricting access to specific data rows based on user roles and DAX filter expressions. It allows you to control and customize data visibility for different users, providing a tailored experience while maintaining data confidentiality.

3. Writing DAX Formulae

3.1 DAX for Calculated Tables

The syntax for creating a calculated table in DAX is as follows:

```
NewTableName = ADDCOLUMNS (
    BaseTableName,
    NewColumnName1, Expression1,
    NewColumnName2, Expression2,
    ...
)
```

Here's an example of creating a calculated table. Let's say you have a table called "Sales" with columns "Product," "Revenue," and "Quantity." You want to create a calculated table that summarizes the total revenue for each product category:

```
CategoryRevenue = ADDCOLUMNS (
    SUMMARIZE('Sales', 'Sales'[Product]),
    "Total Revenue", SUM('Sales'[Revenue])
)
```

In this example:

CategoryRevenue is the name of the calculated table we are creating.

SUMMARIZE('Sales', 'Sales'[Product]) generates a table with unique product categories from the "Sales" table using the SUMMARIZE function.

"Total Revenue" is the name of the new column we're adding to the calculated table.

SUM('Sales'[Revenue]) calculates the sum of revenue for each product category from the "Sales" table.

3.2 DAX for Calculated Columns

The syntax for creating a calculated column in DAX is as follows:

```
TableName[NewColumnName] = Expression
```

Let's break down the syntax and provide an example:

TableName: This is the name of the table where you want to add the calculated column.

NewColumnName: This is the name of the new column you want to create in the table.

Expression: This is the DAX formula that defines the calculation for the new column. The expression can reference other columns in the same table to perform calculations

Here's an example of creating a calculated column. Let's say you have a table called "Sales" with columns "Revenue" and "Quantity." You want to create a calculated column that calculates the "Total Sales" for each row by multiplying the revenue and quantity:

Sales[Total Sales] = Sales[Revenue] * Sales[Quantity]

In this example:

Sales is the name of the table where we are adding the calculated column.

Total Sales is the name of the calculated column we are creating.

Sales[Revenue] and Sales[Quantity] are references to the columns in the "Sales" table.

* is the multiplication operator in DAX.

3.3 DAX for Calculated Measures

The syntax for creating a calculated measure in DAX is as follows:

MeasureName = DAXFunction(Expression)

Let's break down the syntax and provide an example:

MeasureName: This is the name of the measure you want to create.

DAXFunction: This is a DAX function that you use to perform calculations on the data.

Expression: This is the DAX formula that defines the calculation performed by the measure. The expression can include DAX functions, column references, and operators.

Here's an example of creating a calculated measure. Let's say you have a table called "Sales" with columns "Revenue" and "Quantity." You want to create a measure that calculates the total revenue:

Total Revenue = SUM('Sales'[Revenue])

In this example:

Total Revenue is the name of the measure we are creating.

SUM() is a DAX aggregation function used to calculate the sum of the values in the specified column.

'Sales'[Revenue] is a reference to the "Revenue" column in the "Sales" table.

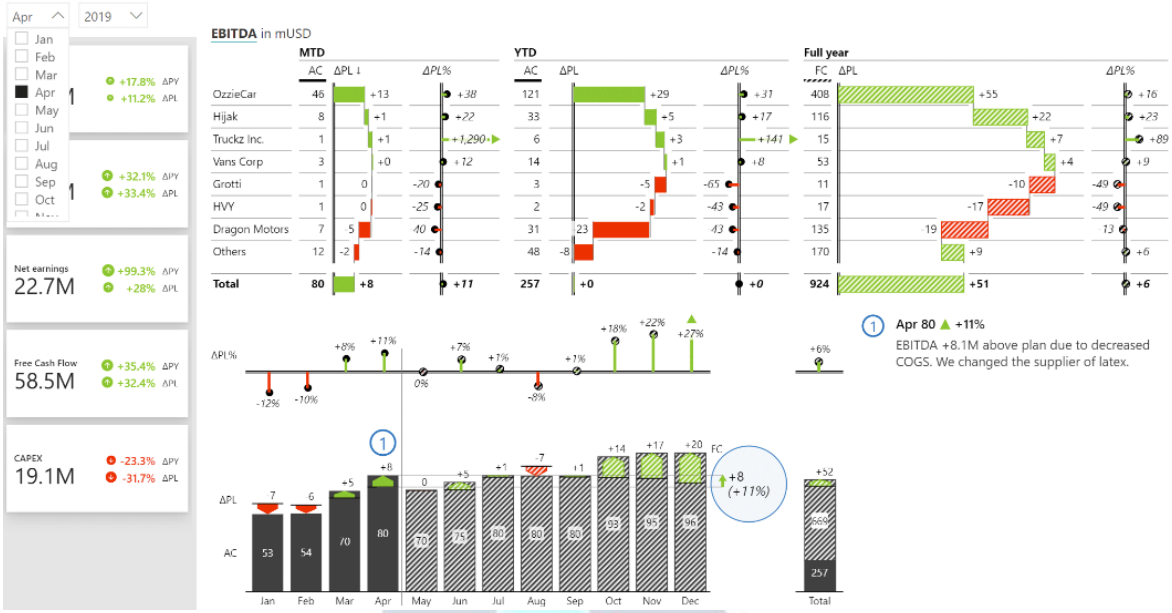
After creating this measure, you can use it in visualizations and reports to display the total revenue across your dataset.

Time intelligence functions in Data Analysis Expressions (DAX) are a set of functions that allow you to perform calculations and analysis on date and time-related data in a dynamic and context-aware manner. These functions are particularly useful for creating time-based reports, calculating year-to-date, month-to-date, or other time-based metrics, and comparing data over different periods. Time intelligence functions help you work with dates, times, and periods in your DAX formulas. Here are some commonly used time intelligence functions in DAX:

4. Time Intelligent Functions in DAX

Why T.I. functions are so important:

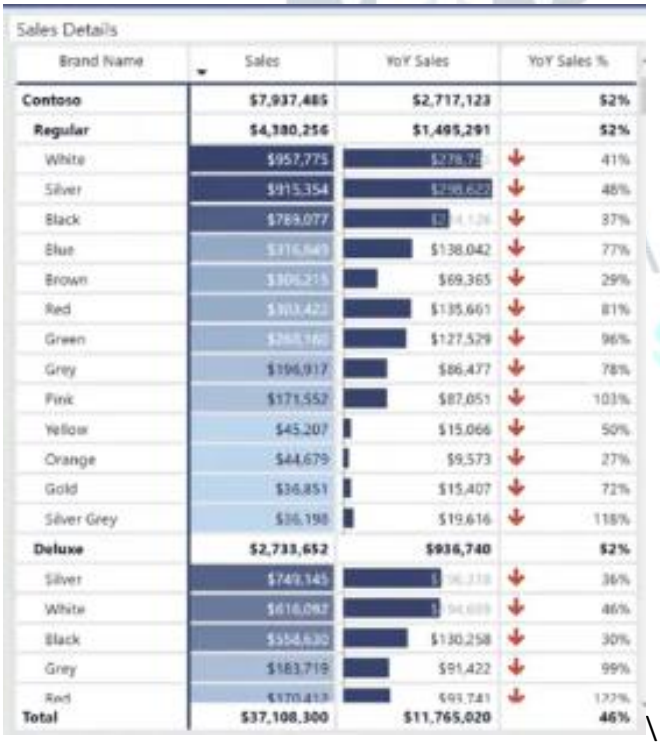
- T.I. functions are essential part of any professional analytical report or dashboard.



- Plays important role in many, industry specific KPIs.

  Return on Investment (Finance), Customer Churn rate (Banking), Absenteeism Rate (HR), Patient Wait Time (Healthcare).

- T.I. measures open up all the possibilities to enrich the visuals.



Sales Details

Brand Name	Sales	YoY Sales	YoY Sales %
Contoso	**$7,937,485**	**$2,717,123**	**52%**
Regular	**$4,380,256**	**$1,495,291**	**52%**
White	$957,775	$278,75	41%
Silver	$915,354	$298,622	48%
Black	$789,077	$	37%
Blue	$316,940	$138,042	77%
Brown	$306,215	$69,365	29%
Red	$303,422	$135,661	81%
Green	$268,160	$127,529	96%
Grey	$196,917	$86,477	78%
Pink	$171,552	$87,051	103%
Yellow	$45,207	$15,066	50%
Orange	$44,679	$9,573	27%
Gold	$36,851	$15,407	72%
Silver Grey	$36,198	$19,616	118%
Deluxe	**$2,733,652**	**$936,740**	**52%**
Silver	$749,145	$	36%
White	$616,082	$	46%
Black	$358,630	$130,258	30%
Grey	$183,719	$91,422	99%
Red	$170,413	$93,741	122%
Total	**$37,108,300**	**$11,765,020**	**46%**

What is Time Intelligent Functions in DAX

- Example:

Month	Sales	YTD	YTD Pre	Variation
Jan-2023	$ 2,000.00	$ 2,000.00	$ 1,500.00	25.00%
Feb-2023	$ 2,500.00	$ 4,500.00	$ 4,000.00	11.11%
Mar-2023	$ 2,000.00	$ 6,500.00	$ 6,300.00	3.08%
Apr-2023	$ 3,000.00	$ 9,500.00	$ 9,300.00	2.11%
May-2023	$ 4,000.00	$ 13,500.00	$ 15,000.00	-11.11%
Jun-2023	$ 3,500.00	$ 17,000.00	$ 18,000.00	-5.88%

Time intelligence functions in Data Analysis Expressions (DAX) are a set of functions that allow you to perform calculations and analysis on date and time-related data in a dynamic and context-aware manner. These functions are particularly useful for creating time-based reports, calculating year-to-date, month-to-date, or other time-based metrics, and comparing data over different periods. Here are some commonly used time intelligence functions in DAX:

1. TOTALYTD: Calculates a year-to-date total for a given expression. This function is often used to calculate cumulative totals from the beginning of the year up to the current date.

Example:

Total Sales YTD = TOTALYTD(SUM(Sales[SalesAmount]), 'Date'[Date])

2. TOTALQTD: Calculates a quarter-to-date total for a given expression. This is useful for calculating cumulative totals from the start of the quarter to the current date.

Example:

Total Sales QTD = TOTALQTD(SUM(Sales[SalesAmount]), 'Date'[Date])

3. TOTALMTD: Calculates a month-to-date total for a given expression. It is used to calculate cumulative totals from the beginning of the month to the current date.

Example:

Total Sales MTD = TOTALMTD(SUM(Sales[SalesAmount]), 'Date'[Date])

4. DATESYTD: Returns a table that includes all dates from the beginning of the year to the last date in the filter context. This can be useful for creating custom time-based calculations.

Example:

Dates YTD = DATESYTD('Date'[Date])

5. SAMEPERIODLASTYEAR: Returns a table of dates in the same period (e.g., same month or same quarter) in the previous year. This is useful for year-over-year comparisons.

Example:

Sales Last Year = CALCULATE(SUM(Sales[SalesAmount]), SAMEPERIODLASTYEAR('Date'[Date]))

6. DATESBETWEEN: Returns a table of dates that fall within a specified date range. It is handy for creating custom date calculations and comparisons.

Example:

Sales Between Dates = CALCULATE(SUM(Sales[SalesAmount]), DATESBETWEEN('Date'[Date], [Start Date], [End Date]))

7. PREVIOUSMONTH: Returns a table of dates for the previous month, which can be used for month-over-month comparisons.

Example:

Sales Previous Month = CALCULATE(SUM(Sales[SalesAmount]), PREVIOUSMONTH('Date'[Date]))

These are just a few examples of time intelligence functions in DAX. They provide powerful capabilities for analyzing data over time, creating dynamic reports, and comparing performance across different time periods. When working with these functions, be sure to have a date table with a relationship to your data table to enable time-based calculations and filtering.

5. Some Other Important Functions in DAX

1. Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single value.

SUM : Adds up all the numbers in a column.

Total Sales = SUM(Sales[Amount])

AVERAGE : Calculates the average of the numbers in a column.

Average Sales = AVERAGE(Sales[Amount])

MIN : Finds the minimum value in a column.

Minimum Sales = MIN(Sales[Amount])

MAX : Finds the maximum value in a column.

Maximum Sales = MAX(Sales[Amount])

COUNT : Counts the number of values in a column.

Sales Count = COUNT(Sales[Amount])

2. Logical Functions

Logical functions perform logical tests and return information about values in expressions.

IF : Checks a condition and returns one value if true and another value if false.

Sales Category = IF(Sales[Amount] > 1000, "High", "Low")

SWITCH : Evaluates an expression against a list of values and returns one of several possible results.

Sales Rating = SWITCH(

```
TRUE(),
    Sales[Amount] > 5000, "Excellent",
    Sales[Amount] > 2000, "Good",
    "Average"
 )
```

AND : Checks if all arguments are TRUE.

High Sales and Discount = IF(AND(Sales[Amount] > 1000, Sales[Discount] > 0.1), "Yes", "No")

OR : Checks if any argument is TRUE.

High Sales or Discount = IF(OR(Sales[Amount] > 1000, Sales[Discount] > 0.1), "Yes", "No")

3. Date and Time Functions

Date and time functions create calculations based on dates and times.

TODAY : Returns the current date.

Current Date = TODAY()

YEAR : Extracts the year from a date.

Year of Sale = YEAR(Sales[OrderDate])

MONTH : Extracts the month from a date.

Month of Sale = MONTH(Sales[OrderDate])

DAY : Extracts the day from a date.

Day of Sale = DAY(Sales[OrderDate])

DATE : Creates a date from individual year, month, and day values.

Order Date = DATE(2024, 7, 16)

4. Text Functions

Text functions manipulate text strings.

CONCATENATE : Joins two text strings into one.

Full Name = CONCATENATE(Customers[FirstName], Customers[LastName])

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

First Letter = LEFT(Customers[FirstName], 1)

RIGHT : Returns the specified number of characters from the end of a text string.

Last Letter = RIGHT(Customers[LastName], 1)

LEN : Returns the number of characters in a text string.

Name Length = LEN(Customers[FirstName])

UPPER : Converts a text string to uppercase.

Uppercase Name = UPPER(Customers[FirstName])

LOWER : Converts a text string to lowercase.

Lowercase Name = LOWER(Customers[FirstName])

5. Filter Functions

Filter functions return specific data from a table.

FILTER : Returns a table that represents a subset of another table or expression.

Filtered Sales = FILTER(Sales, Sales[Amount] > 1000)

ALL : Removes filters from the specified columns or tables.

All Sales = CALCULATE(SUM(Sales[Amount]), ALL(Sales))

RELATED : Returns a related value from another table.

Customer City = RELATED(Customers[City])

RELATEDTABLE : Returns a table that contains all related rows from another table.

Related Orders = RELATEDTABLE(Orders)

6. Mathematical Functions

Mathematical functions perform arithmetic operations.

ROUND : Rounds a number to a specified number of digits.

Rounded Sales = ROUND(Sales[Amount], 2)

CEILING : Rounds a number up to the nearest integer or specified multiple.

Ceiling Sales = CEILING(Sales[Amount], 10)

FLOOR : Rounds a number down to the nearest integer or specified multiple.

Floor Sales = FLOOR(Sales[Amount], 10)

ABS : Returns the absolute value of a number.

Absolute Sales = ABS(Sales[Amount])

MOD : Returns the remainder after a number is divided by a divisor.

Sales Mod 100 = MOD(Sales[Amount], 100)

8. Statistical Functions

Statistical functions perform statistical operations.

MEDIAN : Returns the median of the numbers in a column.

Median Sales = MEDIAN(Sales[Amount])

VAR : Returns the variance of the numbers in a column.

Variance Sales = VAR(Sales[Amount])

STDEV : Returns the standard deviation of the numbers in a column.

Std Dev Sales = STDEV(Sales[Amount])