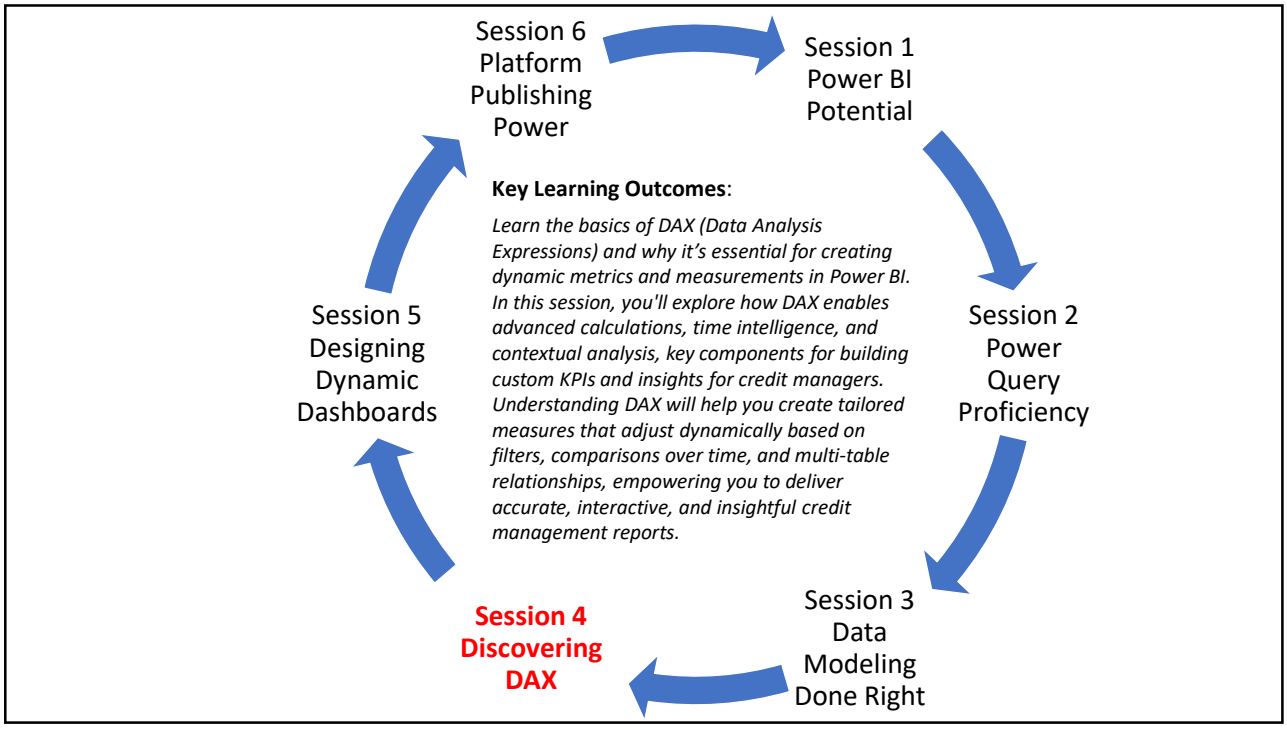
NACM'S 129TH CREDIT CONGRESS & EXPO Cleveland MAY 18-21, 2025

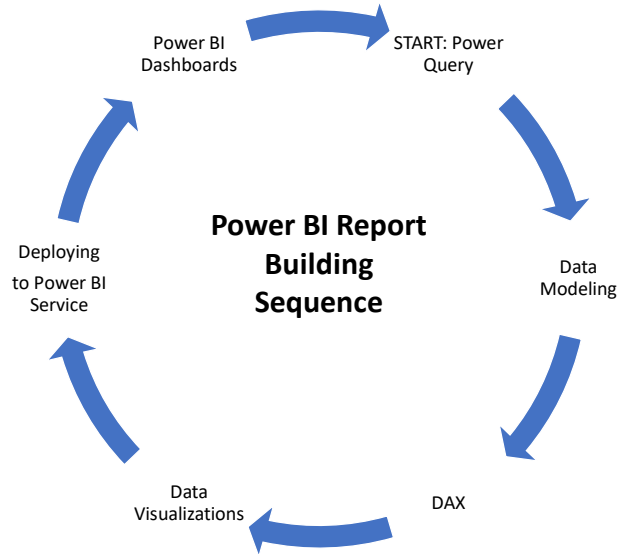
Discovering DAX

Presented by: Rebekyah Brewer

Date: May 21, 2025

Session: #37063





Prerequisites – Technical

Software Requirements

- **Power BI Desktop** (Free) – Power Query is built into Power BI for data transformation.
- **Excel (2016 and later, or Microsoft 365)** – Power Query is available in the "Get & Transform" section.
- **Windows OS (Windows 10 or later recommended)** – Power Query in Power BI is optimized for Windows.

Optional:

- **Power BI Service (Pro or Premium Per User License)** – If publishing reports online, you'll need a Power BI account

Prerequisites – Technical

Computer Capabilities & Performance Considerations

Power Query processes data transformations, and performance can be impacted by your system specs.

- **RAM** – 8GB minimum; 16GB+ recommended for handling large datasets.
- **Processor** – Intel i5/i7 or AMD Ryzen 5/7 or higher for better performance.
- **Storage (SSD Recommended)** – Faster SSD drives improve data processing speed compared to HDD.
- **Internet Speed** – If working with cloud data, a stable internet connection is necessary.

Prerequisites – Experience

Before diving into DAX, a beginner needs a good grasp of:

Basic Understanding of Power BI

- Power BI Desktop: Know how to import data, create visualizations, and use different report elements.
- Power Query - While DAX is for calculations, Power Query is for data transformation. A basic understanding of ETL (Extract, Transform, Load) in Power Query helps.
- Data Modeling Basics: Understand relationships between tables, star schema vs. snowflake schema, and cardinality.

Relational Databases & Tables

- Familiarity with concepts like tables, columns, rows, primary keys, and foreign keys
- Knowing how different tables relate to each other (one-to-many, many-to-one, many-to-many).

Excel Functions & Formulas (Optional but VERY Helpful)

- If you are comfortable with Excel formulas, especially SUMIFS, COUNTIFS, VLOOKUP, INDEX/MATCH, and ARRAY formulas, learning DAX will be easier.
- Understanding how Excel PivotTables work can also be helpful since DAX operates on columnar data similar to PivotTables.

Prerequisites – Experience

Logical Thinking & Problem Solving

- Since DAX is a functional language, writing formulas requires structured thinking.
- Debugging DAX errors requires patience and an analytical mindset.

Understanding Data Types & context

- **Data Types in Power BI:** Understand different data types like Text, Whole Number, Decimal, Boolean, and Date/Time.
- **Row Context vs. Filter Context:** One of the most fundamental DAX concepts.
- **Evaluation Context:** How filters and row context change based on calculations.

Hands-On Practice in Power BI

- Practice common DAX functions like
  - Aggregation: SUM, AVERAGE, COUNT, DISTINCTCOUNT
  - Filter-based calculations: CALCULATE, FILTER, ALL, ALLEXCEPT
  - Time intelligence: TOTALYTD, SAMEPERIODLASTYEAR, DATESYTD
  - Table functions: SUMMARIZE, ADDCOLUMNS, SELECTCOLUMNS
- Practice with sample datasets or in your own daily exports
- Practice. Practice. Practice

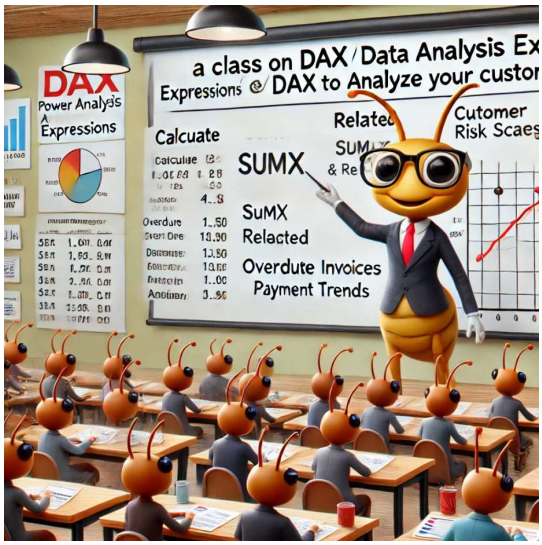
Who is DAX For?

	User Group	How Power BI Benefits Them
1	**Power BI Users**	Anyone building Power BI dashboards and needing custom calculations, dynamic aggregations, and time intelligence.
2	**Excel Data Analysts aka Data Wizards**	Those who want to move beyond SUMIFS and VLOOKUP to more efficient calculations in Excel.
3	**Financial Analysts and Accountants**	Useful for creating custom financial metrics, forecasts, and rolling average reports in Power BI & Excel or on top of others Power BI Reports.
4	**Self-Service BI User**	Business users who need to write custom formulas for KPIs and dynamic calculations.
6	**Credit Managers & Sales Teams**	Analyzing sales trends, year-over-year comparisons, and customer segmentation, AR Portfolio, Payment Trends.

Discovering DAX

Session Overview

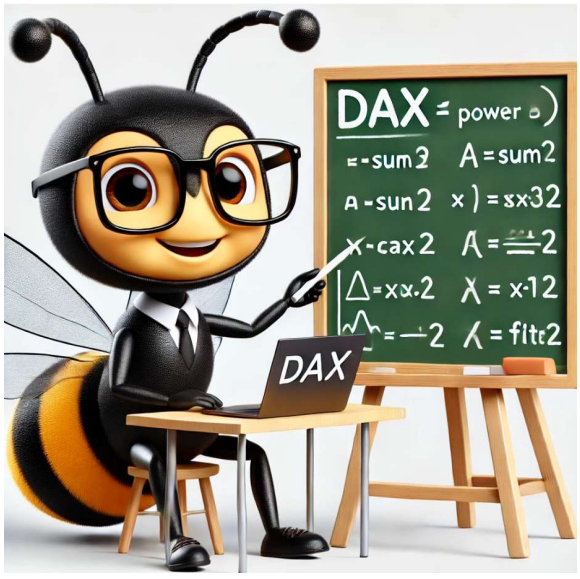
- Introduction & Prerequisites
- Understanding DAX
- Core Features & Functionalities
- Deep Dive into Measures
- Best Practices in DAX
- Wrap-Up, Q&A, Further Resources



Understanding DAX

(Data Analysis Expressions)

- **Definition of DAX**
  - What is DAX?
  - Purpose & Application
  - Basic Concepts
- **DAX Language Format**
  - Calculated Columns
  - Syntax & Expression Eval.



DAX – Definition

DAX (Data Analysis Expression) – DAX (Data Analysis Expressions) is the formula language used in Power BI, Excel Power Pivot, and Analysis Services.

It is designed for dimensional data modeling.

DAX allows users to create custom calculated columns, measures, and tables to enhance reports and dashboards.

DAX – Does: Purpose & Application:

DAX is the key behind dynamic calculations. It enhances every data model. It allows users to add their own analysis and calculations on top of a data model or data source.

**Functional Language – Unlike traditional procedural programming, DAX works like Excel formulas and is optimized for *columnar* data storage.

Context Awareness – DAX operates within row context (working on a single row at a time, like calculated columns) and filter context (evaluating measures based on filters applied in a report).

DAX – Purpose & Application:

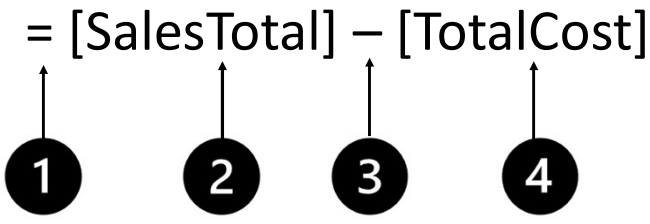
Aggregation and Filtering – Functions like SUM(), AVERAGE(), FILTER(), and CALCULATE() allow powerful data manipulation

Time Intelligence – DAX supports functions like TOTALYTD(), SAMEPERIODLASTYEAR(), and DATESBETWEEN() for time-based calculations.

Relationship Navigation – DAX can traverse table relationships, allowing complex multi-table calculations using functions like RELATED() and RELATEDTABLE().

DAX – Syntax

Calculating Margin:

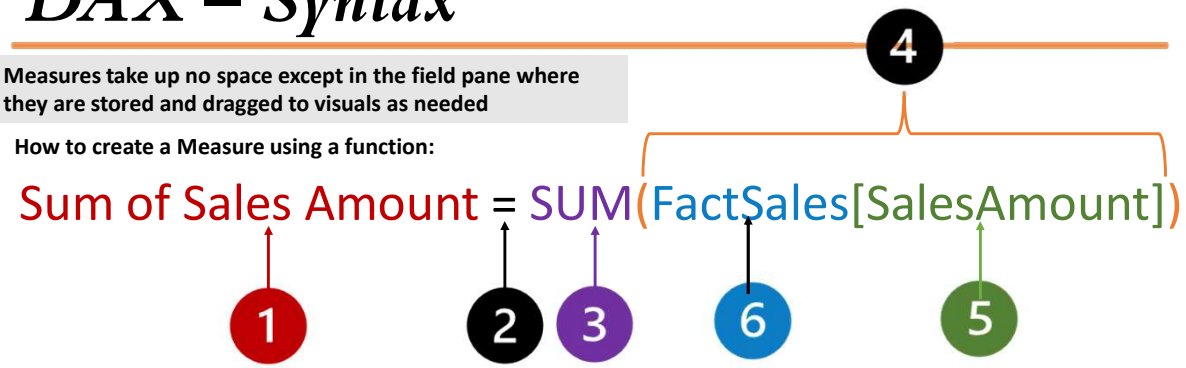


1. (=) Signs operator indicates beginning of formula, just like Excel.
2. First referenced column. Column references are always in brackets [ ]
3. (-) Subtract operator.
4. Referenced column [ ]

DAX – Syntax

Measures take up no space except in the field pane where they are stored and dragged to visuals as needed

How to create a Measure using a function:

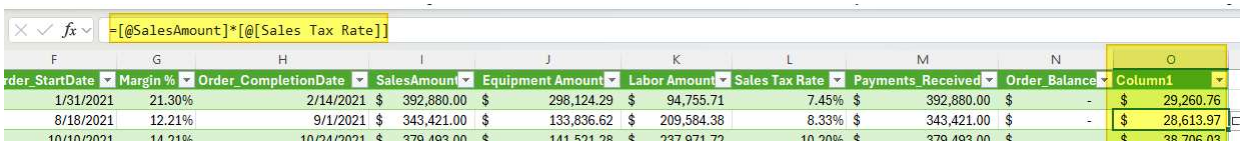


1. Name of Measure before (=)
2. (=) Signs operator indicates beginning of formula, just like Excel.
3. Function, SUM, AVERAGE, MIN, MAX, SUM adds up all of referenced columns
4. ( ) Parenthesis surround the argument just like they would in Excel.
5. Reference Column in brackets
6. Table name in which the column resides. If spaces are in column name, you must enclose with single quotation marks. ' ' as in **'Fact Sales'[SalesAmount]**

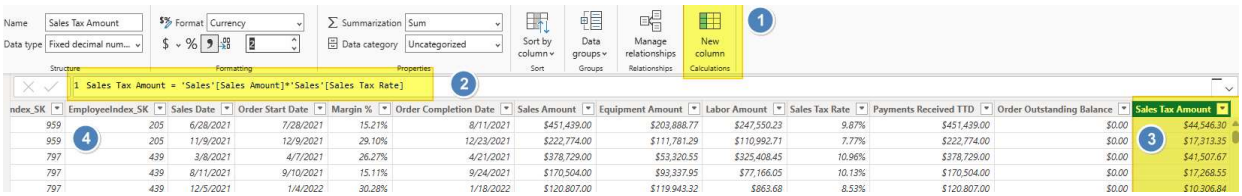
Practice: Calculated Columns

Columnar Calculations – Are used to create new columns in a table referred to as **Calculated Columns**

If you have ever added a new column to a 'Table' in Excel and enjoyed the auto calculations all the way down, Calculated Columns are very similar.



Power BI Calculated Column Example:



Sales Tax Amount = 'Sales'[Sales Amount]*'Sales'[Sales Tax Rate]

1. Name your column before the "=" symbol
2. Identify the table with Apostrophe Symbols '___'. IntelliSense will provide you a list of available tables to select from.
3. Identify and Select the Column you want to aggregate. Columns are identified between [___]
4. Type your operator, *, +, -, etc…
5. Identify the table and column to be operated on.

Calculated Column Examples:

Sales Tax Amount = 'Sales'[Sales Amount]*'Sales'[Sales Tax Rate]

Salesperson Name = RELATED(Salesperson[EmployeeName])

Salesperson Name & Location =
RELATED(Salesperson[EmployeeName]) & "-" & RELATED('Salesperson'[Location])

**Notice the table identifiers ' apostrophes are not always required to write a Calculated Column.*

Credit Risk Alert = IF(RELATED(Customers[Credit Risk Level]) = "High Risk", "Alert", " ")

Margin Size Bonus = IF([Margin %] >=.25, .02, 0)

Best Practice: Calculated Columns

When to Use a Calculated Column	When NOT to Use a Calculated Column
✔ Row-Level Calculations (e.g., Concatenating names, Classification)	❌ Aggregations – Use Measures instead
✔ Sorting or Filtering needs, Slicer	❌ Simple Transformations – Use Power Query
✔ Required for Relationships between tables	❌ Large Data Models – Reduces performance efficiency
✔ Data Model Constraint - Conditional Flags for later aggregation	❌ Anything that can be calculated dynamically with measures

Calculated Columns vs Measures

Feature	Calculated Column	Measure
Calculation Type	Computed row by row during data model refresh	Computed on the fly based on user interaction.
Storage	Stored in the model, consuming memory	Not stored, recalculated dynamically when needed
Evaluation Context	Works at the row level (row context)	Works at the aggregation level (filter context)
Performance Impact	Increases memory usage and file size	More efficient, as it's calculated only when needed
Use Case	Used when you need a new column field in your data table	Used for aggregations (SUM, AVERAGE, COUNT, etc.) in reports
Example	Sales[Profit] = Sales[Revenue] - Sales[Cost] (adds a new column to the table)	Total Sales = SUM(Sales[Revenue]) (computed dynamically)

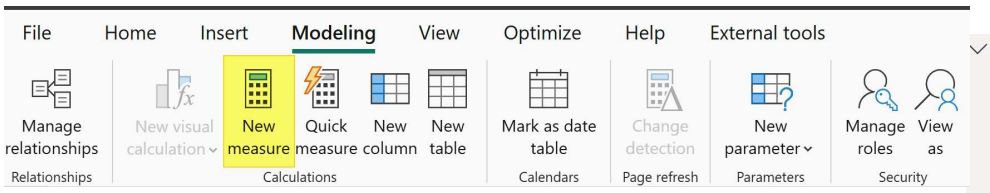
Basic Concepts: DAX Measures (DAX)

Measures perform calculations on data at the time of query, responding to user interactions such as filtering and slicing.

They are dynamic formulas that aggregate data more efficiently then calculated columns.

The value changes based on the interaction of the reports and context of the filters.

- **Calculated at Query Time** – Unlike calculated columns, which are computed when the data is loaded or refreshed, measures are evaluated dynamically when used in a report.
- **Aggregated Results** – Measures perform calculations across multiple rows rather than row by row.
- **Context-Aware** – Measures change based on the filter and row context applied in a report (e.g., filtering by region, date, or product category).
- **Stored in the Model –** Unlike Excel formulas, measures do not exist as part of the dataset but as metadata inside the data model.

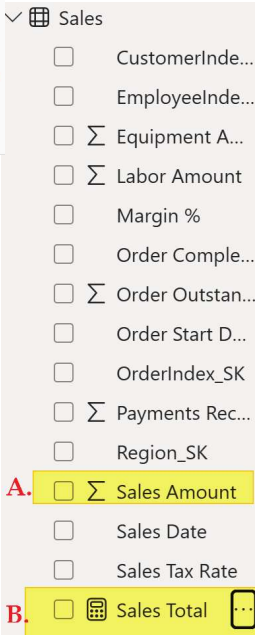


Created measures are shown in the Fields list beneath their assigned table with a little calculator icon beside them instead of the sum icon.

You can name them whatever you like.

*They are **Report Level** – custom metrics created in a report on top of the dataset, added by users or by data modelers.*

A. **Implicit Measure** – auto generated, based on fields you drag and drop.
B. **Explicit Measure** – are user-defined calculations created by DAX.
C. **Quick Measures –** Pre-built calculations in Power BI for common aggregations.
D. **Visual Measures –** Context Specific calculations applied directly within a visual, not stored in a column or a field.



Implicit & Explicit Measures

Feature	Implicit Measures	Explicit Measures
Definition:	Automatically created when dragging a numeric field into a visual	User-defined calculations written using DAX
Created By:	Power BI (Auto-generated)	Report Developer (Manually using DAX)
DAX Requirement:	No DAX needed	Requires DAX formula
Customization:	Limited (only basic aggregations)	Fully customizable with complex logic
Reusability:	Cannot be reused in other measures	Can be reused in multiple measures and calculations
Performance:	Generally optimized for quick visual calculations	Can be optimized using best DAX practices
Complexity:	Suitable for simple aggregations (SUM, AVERAGE, COUNT)	Suitable for complex calculations (Year-over-Year, Ratios, etc.)
Best Use Case:	Quick, ad-hoc analysis	Enterprise-level reporting, consistency, and scalability

Common DAX Functions

Aggregation: SUM(), AVERAGE(), MIN(), MAX()

Logical: IF(), SWITCH(), AND(), OR()

Filter and Context Modification: CALCULATE(), FILTER(), ALL(), REMOVEFILTERS()

Date & Time Intelligence: DATEADD(), TOTALYTD(), EOMONTH()

Text Functions: CONCATENATE(), SEARCH(), LEFT(), RIGHT()

Table Manipulation: SUMMARIZE(), ADDCOLUMNS(), UNION(), CROSSJOIN(), Relationship Navigation USERELATIONSHIP()

DAX Fundamental Aggregation Measures

Function	Description	Syntax	Example
SUM	Returns the sum of a column.	SUM(<column>)	SUM(Sales[Amount])
AVERAGE	Returns the average (arithmetic mean) of a column.	AVERAGE(<column>)	AVERAGE(Sales[Amount])
MIN	Returns the smallest value in a column.	MIN(<column>)	MIN(Sales[Amount])
MAX	Returns the largest value in a column.	MAX(<column>)	MAX(Sales[Amount])
COUNT	Counts the number of numeric values in a column.	COUNT(<column>)	COUNT(Sales[Amount])
COUNTA	Counts the number of non-empty values in a column.	COUNTA(<column>)	COUNTA(Sales[CustomerName])
COUNTROWS	Counts the number of rows in a table.	COUNTROWS(<table>)	COUNTROWS(Sales)
DISTINCTCOUNT	Counts the number of distinct values in a column.	DISTINCTCOUNT(<column>)	DISTINCTCOUNT(Sales[CustomerID])
SUMX	Returns the sum of an expression evaluated for each row in a table.	SUMX(<table>, <expression>)	SUMX(Sales, Sales[Quantity] * Sales[Price])
AVERAGEX	Returns the average of an expression evaluated for each row in a table.	AVERAGEX(<table>, <expression>)	AVERAGEX(Sales, Sales[Quantity] * Sales[Price])
MINX	Returns the smallest value of an expression evaluated for each row in a table.	MINX(<table>, <expression>)	MINX(Orders, Orders[OrderTotal])
MAXX	Returns the largest value of an expression evaluated for each row in a table.	MAXX(<table>, <expression>)	MAXX(Orders, Orders[OrderTotal])

Context. Context. Context.

Understanding context is essential in DAX. There are two primary types: **row context** and **filter context**. Let's start by examining row context.

Row Context –
Row context refers to the current row being processed.

Example: Our calculated column for Margin with the formula [SalesAmount] - [TotalCost].

This formula computes a value for *each row* by subtracting the TotalCost from the SalesAmount in the same row. DAX understands which values to use because it applies the calculation within the context of each row.

In a specific row where SalesAmount is $101.08 and TotalCost is $51.54, the Margin value is calculated as $49.54 by subtracting TotalCost from SalesAmount.

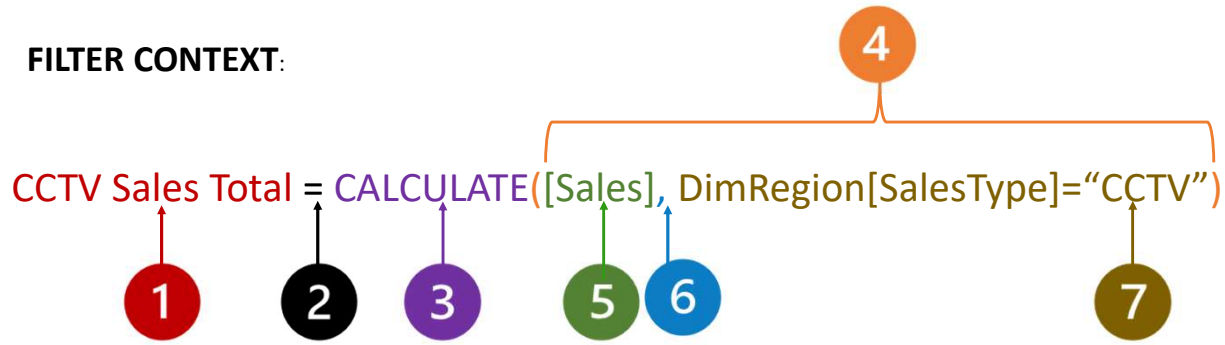
Row Context exists not just in Calculated Columns but in the SUMX, AVERAGEX, MINX and MAXX Functions.

Context. Context. Context.

Filter Context –
Filter context is crucial in DAX because it determines which data is used in calculations.
Pivot Tables are all about filter context.

- Visuals apply a filter context automatically.
- Slicers provide a filter context.
- Explicit filter functions in DAX like CALCULATE, ALL, RELATED, FILTER allow you to include additional filters to your measures and even override existing filter context as needed

FILTER CONTEXT:



1. Measure Name

2. = Beginning formula

3. CALCULATE Function evaluates an *expression,* as an argument, in a context that is modified by special filters.

4. Parenthesis () surround argument(s).

5. A measure [Sales] in the same table as expression. The sales measure has the same formula: =SUM(FactSales[SamesAmount])

6. A comma (,) separates each filter.

7. Referenced column with = "CCTV" as filter

Ensures that only sales values, defined by the filter are calculated only for rows in the DimChannel with value "CCTV".

DAX Filters for Measures – Context Override

Function	Description	Syntax	Example
FILTER	Returns a filtered table based on a condition.	FILTER(<table>, <condition>)	FILTER(Sales, Sales[Amount] > 1000)
ALL	Removes all filters from a table or column.	ALL(<table_or_column>)	ALL(Sales)
ALLEXCEPT	Removes all filters except on specified columns.	ALLEXCEPT(<table>, <column1>, <column2>, ...)	ALLEXCEPT(Sales, Sales[Region])
ALLSELECTED	Removes filters applied by visual interactions but retains others.	ALLSELECTED(<table_or_column>)	ALLSELECTED(Sales[Category])
REMOVEFILTERS	Removes all filters from the specified columns or tables.	REMOVEFILTERS(<table_or_column>)	REMOVEFILTERS(Sales[Product])
KEEPFILTERS	Applies existing filters before executing a calculation.	KEEPFILTERS(<expression>)	KEEPFILTERS(FILTER(Sales, Sales[Amount] > 1000))
CALCULATE	Evaluates an expression in a modified filter context.	CALCULATE(<expression>, <filter1>, <filter2>, ...)	CALCULATE(SUM(Sales[Amount]), Sales[Region] = "West")
CALCULATETABLE	Returns a table with a modified filter context.	CALCULATETABLE(<table>, <filter1>, <filter2>, ...)	CALCULATETABLE(Sales, Sales[Category] = "Electronics")
VALUES	Returns a single-column table of unique values.	VALUES(<column>)	VALUES(Sales[Product])
DISTINCT	Returns a table of distinct values from a column.	DISTINCT(<column>)	DISTINCT(Sales[CustomerID])

DAX Logical Conditional Measures

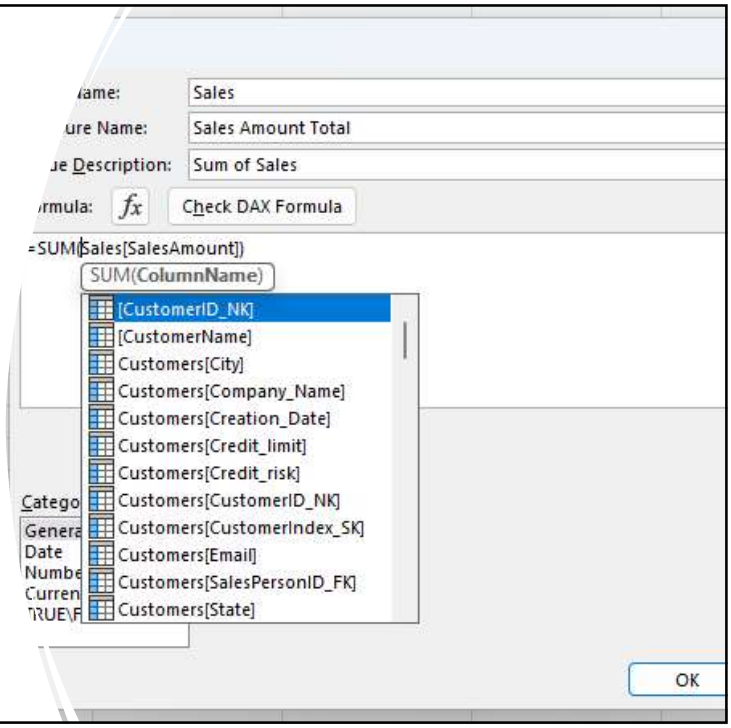
Function	Description	Syntax	Example
IF	Returns one value if a condition is TRUE and another if FALSE.	IF(<condition>, <true_value>, <false_value>)	IF(Sales[Amount] > 1000, "High", "Low")
SWITCH	Evaluates an expression against multiple conditions and returns a corresponding value.	SWITCH(<expression>, <value1>, <result1>, ..., <else_result>)	SWITCH(Sales[Category], "A", "Type 1", "B", "Type 2", "Other")
AND	Returns TRUE if all conditions are TRUE.	AND(<condition1>, <condition2>)	AND(Sales[Amount] > 1000, Sales[Discount] < 10)
OR	Returns TRUE if at least one condition is TRUE.	OR(<condition1>, <condition2>)	OR(Sales[Region] = "West", Sales[Region] = "East")
NOT	Returns the opposite of a Boolean expression.	NOT(<condition>)	NOT(Sales[Approved])
IFERROR	Returns a specified value if the expression results in an error.	IFERROR(<expression>, <alternate_value>)	IFERROR(Sales[Amount] / Sales[Quantity], 0)
ISBLANK	Checks if a value is blank (empty).	ISBLANK(<value>)	ISBLANK(Sales[CustomerID])
ISERROR	Checks if an expression results in an error.	ISERROR(<expression>)	ISERROR(Sales[Amount] / Sales[Quantity])
TRUE	Returns the Boolean value TRUE.	TRUE()	TRUE()
FALSE	Returns the Boolean value FALSE.	FALSE()	FALSE()

DAX Time Intelligence Measures

Function	Description	Syntax	Example
TOTALYTD	Calculates year-to-date total for a measure.	TOTALYTD(<expression>, <dates_column>[, <filter>])	TOTALYTD(SUM(Sales[Amount]), Sales[Date])
TOTALQTD	Calculates quarter-to-date total for a measure.	TOTALQTD(<expression>, <dates_column>[, <filter>])	TOTALQTD(SUM(Sales[Amount]), Sales[Date])
TOTALMTD	Calculates month-to-date total for a measure.	TOTALMTD(<expression>, <dates_column>[, <filter>])	TOTALMTD(SUM(Sales[Amount]), Sales[Date])
PREVIOUSYEAR	Returns the measure value for the previous year.	PREVIOUSYEAR(<dates_column>)	PREVIOUSYEAR(Sales[Date])
PREVIOUSQUARTER	Returns the measure value for the previous quarter.	PREVIOUSQUARTER(<dates_column>)	PREVIOUSQUARTER(Sales[Date])
PREVIOUSMONTH	Returns the measure value for the previous month.	PREVIOUSMONTH(<dates_column>)	PREVIOUSMONTH(Sales[Date])
PREVIOUSDAY	Returns the measure value for the previous day.	PREVIOUSDAY(<dates_column>)	PREVIOUSDAY(Sales[Date])
SAMEPERIODLASTYEAR	Returns the measure value for the same period in the previous year.	SAMEPERIODLASTYEAR(<dates_column>)	SAMEPERIODLASTYEAR(Sales[Date])
DATEADD	Shifts dates forward or backward by a given number of intervals.	DATEADD(<dates_column>, <number_of_intervals>, <interval_type>)	DATEADD(Sales[Date], -1, YEAR)
PARALLELPERIOD	Returns a parallel period, shifting by a given number of intervals.	PARALLELPERIOD(<dates_column>, <number_of_intervals>, <interval_type>)	PARALLELPERIOD(Sales[Date], -1, YEAR)
FIRSTDATE	Returns the first date in the column or table.	FIRSTDATE(<dates_column>)	FIRSTDATE(Sales[Date])
LASTDATE	Returns the last date in the column or table.	LASTDATE(<dates_column>)	LASTDATE(Sales[Date])

Benefits of Learning DAX in Excel:

1. Excel is familiar territory for all of us
2. Increased opportunities for internal use leads to wider application
3. Increased usage cases leads to increased experience
4. Small quick wins will encourage motivation to learn more.
5. Logical transition to Power PI through PowerPivot



DAX Fundamental Table Functions

Function	Description	Syntax	Example
ADDCOLUMNS	Adds calculated columns to a table.	ADDCOLUMNS(<table>, <column_name>, <expression>[, <column_name>, <expression>, ...])	ADDCOLUMNS(Sales, "Profit", Sales[Revenue] - Sales[Cost])
SUMMARIZE	Returns a summary table with aggregated values.	SUMMARIZE(<table>, <group_by_column>[, <group_by_column>, ...][, <name>, <expression>, ...])	SUMMARIZE(Sales, Sales[Date], Sales[Region], "Total Sales", SUM(Sales[Amount]))
SUMMARIZECOLUMNS	Returns a summary table with filters applied.	SUMMARIZECOLUMNS(<group_by_column>[, <group_by_column>, ...][, <name>, <expression>, ...])	SUMMARIZECOLUMNS(Date[Year], Region[RegionName], "Total Sales", SUM(Sales[Amount]))
SELECTCOLUMNS	Returns a table with selected columns.	SELECTCOLUMNS(<table>, <name>, <column>[, <name>, <column>, ...])	SELECTCOLUMNS(OrderDetails, "Order ID", OrderDetails[OrderID], "Product", OrderDetails[Product])
FILTER	Returns a filtered table based on a condition.	FILTER(<table>, <condition>)	FILTER(Sales, Sales[Amount] > 1000)
ALL	Removes all filters from a table or column.	ALL(<table_or_column>)	ALL(Customer)
DISTINCT	Returns a table of distinct values from a column.	DISTINCT(<column>)	DISTINCT(Salesperson[SalespersonID])
VALUES	Returns a single-column table of unique values.	VALUES(<column>)	VALUES(Region[RegionName])
CROSSJOIN	Returns a Cartesian product of two tables.	CROSSJOIN(<table1>, <table2>)	CROSSJOIN(Salesperson, Region)
UNION	Returns the union of two tables, removing duplicates.	UNION(<table1>, <table2>[, <table3>, ...])	UNION(OrderDetails_2023, OrderDetails_2024)
INTERSECT	Returns the intersection of two tables.	INTERSECT(<table1>, <table2>)	INTERSECT(Customer_US, Customer_Canada)
EXCEPT	Returns the difference between two tables (values in first but not in second).	EXCEPT(<table1>, <table2>)	EXCEPT(Sales, Sales_Excluded)

Date Table – DAX Method

```
DateTable =
ADDCOLUMNS (
   CALENDAR (DATE(2015,1,1), DATE(2030,12,31)),
   "Year", YEAR([Date]),
   "Month", FORMAT([Date], "MMMM"),
   "Month Number", MONTH([Date]),
   "Quarter", "Q" & FORMAT([Date], "Q"),
   "Day of Week", FORMAT([Date], "dddd"),
   "Day of Year", FORMAT([Date], "DDD"),
   "Week Number", WEEKNUM([Date])
)
```

AR Portfolio Analysis Calculations

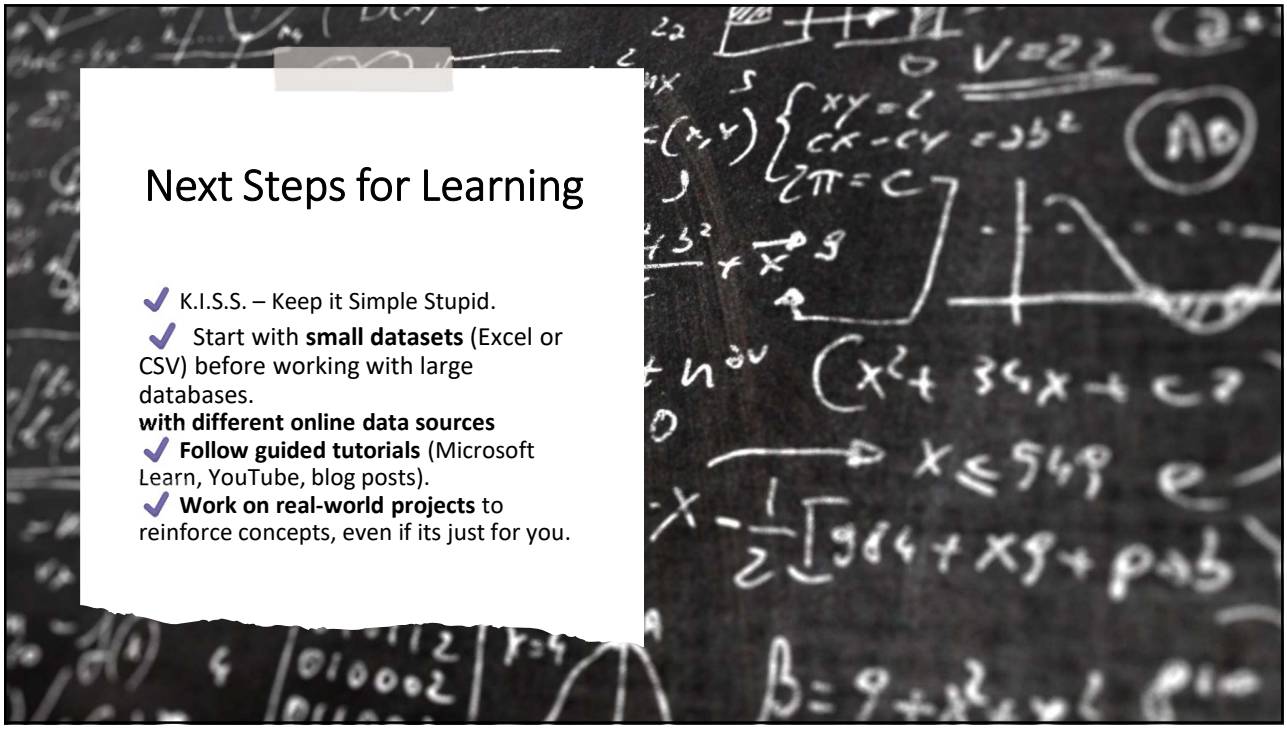
Measure Name	Description	DAX Formula
Total AR	Total outstanding Accounts Receivable (A/R) balance.	Total AR = SUM(Invoices[Outstanding Balance])
Current AR	Balance of invoices that are not overdue.	Current AR = CALCULATE([Total AR], Invoices[Due Date] >= TODAY())
Overdue AR	Total amount of past-due invoices.	Past Due AR = CALCULATE([Total AR], Invoices[Due Date] < TODAY())
AR 0-30 Days	Total A/R balance for invoices due within 0-30 days.	AR 0-30 Days = CALCULATE([Total AR], DATEDIFF(Invoices[Due Date], TODAY(), DAY) <= 30)
AR 31-60 Days	Total A/R balance for invoices due within 31-60 days.	AR 31-60 Days = CALCULATE([Total AR], DATEDIFF(Invoices[Due Date], TODAY(), DAY) > 30, DATEDIFF(Invoices[Due Date], TODAY(), DAY) <= 60)
AR 61-90 Days	Total A/R balance for invoices due within 61-90 days.	AR 61-90 Days = CALCULATE([Total AR], DATEDIFF(Invoices[Due Date], TODAY(), DAY) > 60, DATEDIFF(Invoices[Due Date], TODAY(), DAY) <= 90)
AR Over 90 Days	Total A/R balance for invoices due over 90 days.	AR Over 90 Days = CALCULATE([Total AR], DATEDIFF(Invoices[Due Date], TODAY(), DAY) > 90)
AR Aging Category	Categorization of AR into aging buckets (e.g., 0-30, 31-60, 61-90, 90+ days).	SWITCH(TRUE(), AR[DaysPastDue] <= 30, '0-30 Days', AR[DaysPastDue] <= 60, '31-60 Days', AR[DaysPastDue] <= 90, '61-90 Days', '90+ Days')
Overdue AR %	Percentage of total A/R that is overdue.	Overdue AR % = DIVIDE([Past Due AR], [Total AR], 0)
DSO	Days Sales Outstanding - average number of days to collect payment.	DSO = DIVIDE([Total AR] * 365, [Total Sales])
Top 10 Customers by AR	**Table** List of top 10 customers with the highest outstanding AR.	TOPN(10, AR, AR[OutstandingAmount])
Credit Utilization	Credit utilization ratio - how much of total credit limit is used.	Credit Utilization = DIVIDE(SUM(Invoices[Outstanding Balance]), SUM(Customers[Credit Limit]), 0)
Top 5 Customers	Ranks customers by outstanding A/R balance.	Top 5 Customers = RANKX(ALL(Customers), [Total AR], , DESC)
Expected Cash Inflow	Estimated cash inflow from outstanding invoices, adjusting for default risk.	Expected Cash Inflow = SUMX(Invoices, Invoices[Outstanding Balance] * (1 - Invoices[Estimated Default Rate]))

Sales Analysis Calculations

Measure Name	DAX Formula	Description
Total Sales	SUM('FactSales'[Order Amount])	Total revenue from sales
Total Equipment Sales	SUM('FactSales'[Equipment Portion])	Revenue from equipment sales
Total Labor Sales	SUM('FactSales'[Labor Portion])	Revenue from labor services
Total Sales Tax	SUMX('FactSales', 'FactSales'[Order Amount] * 'FactSales'[Sales Tax Rate])	Total sales tax collected
Equipment Contribution %	DIVIDE([Equipment Sales], [Total Sales], 0)	Percentage of total sales from equipment
Labor Contribution %	DIVIDE([Labor Sales], [Total Sales], 0)	Percentage of total sales from labor
Total Cost	SUM('FactSales'[Cost Portion])	Total cost of sales
Gross Profit	[Total Sales] - [Total Cost]	Total sales revenue minus total cost
Gross Profit Margin %	DIVIDE([Gross Profit], [Total Sales], 0)	Percentage of sales revenue that is profit
Sales YoY Growth %	VAR LastYearSales = CALCULATE([Total Sales], SAMEPERIODLASTYEAR('FactSales'[Sales Date])) RETURN DIVIDE([Total Sales] - LastYearSales, LastYearSales, 0)	Year-over-Year sales growth percentage
Sales MoM Growth %	VAR LastMonthSales = CALCULATE([Total Sales], PREVIOUSMONTH('FactSales'[Sales Date])) RETURN DIVIDE([Total Sales] - LastMonthSales, LastMonthSales, 0)	Month-over-Month sales growth percentage
Sales by Region	SUMX('FactSales', 'FactSales'[Order Amount])	Total sales categorized by region
Sales by Salesperson	SUMX('FactSales', 'FactSales'[Order Amount])	Total sales per salesperson
Average Order Value (AOV)	DIVIDE([Total Sales], DISTINCTCOUNT('FactSales'[Customer Name]), 0)	Average value of customer orders
Customer Sales % Contribution	DIVIDE([Total Sales], CALCULATE([Total Sales], ALL('FactSales'[Customer Name])), 0)	Percentage contribution of each customer to total sales
High Risk Customers	IF([Total Sales] > 100000, 'High Risk', 'Normal')	Flags customers with high sales as potentially high risk
Days Sales Outstanding (DSO)	DIVIDE([Total Receivables], [Total Sales]) * 30	Average number of days to collect payment
Late Payment %	DIVIDE(SUMX('FactSales', IF('FactSales'[Days Past Due] > 30, 'FactSales'[Order Amount], 0)), [Total Sales], 0)	Percentage of sales with late payments
Bad Debt %	DIVIDE(SUM('FactSales'[Write-offs]), [Total Sales], 0)	Percentage of total sales written off as bad debt
Top 10 Customers by Sales	RANKX(ALL('FactSales'[Customer Name]), [Total Sales], , DESC, DENSE)	Ranks customers based on sales volume
Sales by Trade Category	SUMX('FactSales', 'FactSales'[Order Amount])	Total sales categorized by trade category

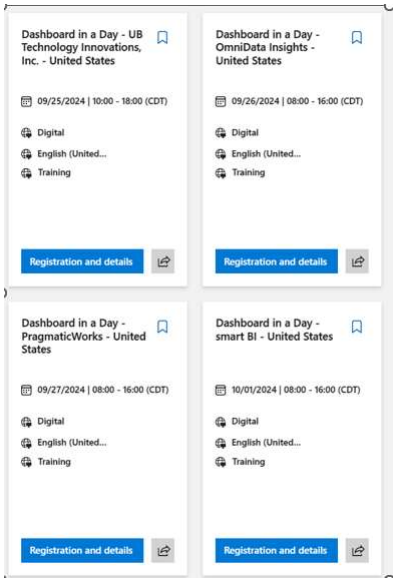
Top 10 Best Practices for DAX

Best Practice:	Why It's Important:	Example:
1. Use Measures Instead of Calculated Columns	Saves memory and improves performance	TotalSales = SUM(Sales[Amount])
2. Use Variables (VAR)	Avoids redundant calculations, improves readability	VAR Revenue = SUM(Sales[Revenue]) RETURN Revenue
3. Understand Context Transition	Ensures expected behavior when switching contexts	CALCULATE(SUM(Sales[Amount])) converts row to filter context
4. Keep Filters Explicit in CALCULATE()	Prevents unexpected filtering issues	CALCULATE(SUM(Sales[Amount]), SAMEPERIODLASTYEAR(Date[Date]))
5. Avoid FILTER() for Simple Conditions	Improves performance by reducing iterations	CALCULATE(SUM(Sales[Amount]), Sales[Amount] > 100)
6. Reduce Dependencies on Entire Tables	Limits computational overhead	CALCULATE(SUM(Sales[Amount]), Customer[Category] = "Premium")7
7. Use DIVIDE() Instead of /	Prevents division by zero errors	DIVIDE(SUM(Sales[Profit]), SUM(Sales[Revenue]), 0)
8. Organize Your Measures	Create Measure Tables & Folders	
9. Avoid Overuse of ALL()	Prevents unexpected filter removal	CALCULATE(SUM(Sales[Amount]), ALL(Sales)) (use with caution)
10. Use Clear Naming Conventions	Improves maintainability and collaboration	TotalSalesAmount, CustomerCount



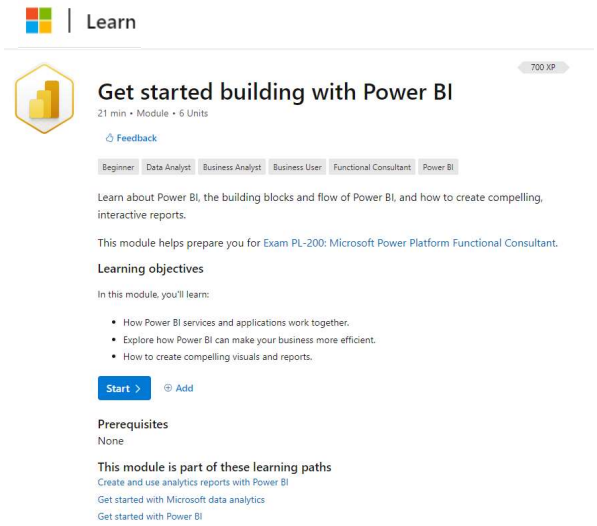
Attend a Free 1 Day Event Workshop:

Microsoft Dashboard in a Day



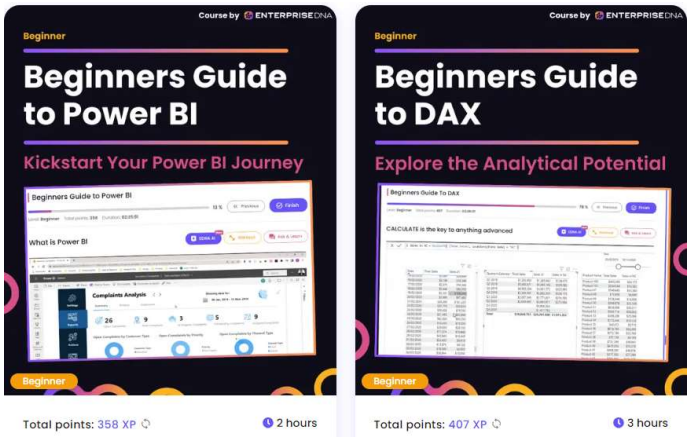
Pragmatic Works DAX Cheat Sheet for Beginners

- Hands-On, Practical Learning Experience
- Rapid Skill Acquisition
- Guided Instruction from Experts
- Structured Learning Agenda
- Real-World Application of Skills
- Access to Workshop Materials & Resources
- Networking Opportunities
- Personalized Feedback & Support
- Boosts Confidence with Power BI
- Preparation for Advanced Learning
- Cost-Effective Training Option
- Immediate Insight into Power BI's Capabilities
- Exposure to Power BI Service Features



Click to start: Microsoft Learn

- Free Access to High-Quality Content
- Structured Learning Paths
- Hands-on Labs and Interactive Exercises
- Official and Up-to-Date Content
- Integration with Certifications
- Gamified Learning Experiences (Points, Badges)
- Self-Paced Learning
- Community and Q&A Integration
- Comprehensive Coverage of Power BI Features
- Scenario-Based Learning Modules



FREE COURSE - Ultimate Beginners Guide To Power BI - http://portal.enterprisedna.co/p/ultimate-beginners-guide-to-power-bi
FREE COURSE - Ultimate Beginners Guide To DAX - http://portal.enterprisedna.co/p/ultimate-beginners-guide-to-dax
FREE - 60 Page DAX Reference Guide Download - https://enterprisedna.co/dax-formula-reference-guide-download

Click to start: ENTERPRISE DNA

Sam McKay, CFA

- Some Free Courses else Paid Subscription
- Expert-Led Training & Courses
- Focus on Real-World Scenarios & Problem-Solving
  - Finance Focused
- Comprehensive Course Catalog
- Access to Learning Summits & Workshops
- Extensive Resource Library
  - Power BI .pbix file downloads
- Customized Learning Paths
- Innovative Data Challenges & Projects
- Supportive Community Forum
- Access to Power BI Showcases
- Focus on Advanced Analytics & AI Integration
- On-Demand, Self-Paced Learning
- Gamified Learning Experience (Points & Badges)
- Certification Programs
- Emphasis on Visualization & Design

PRAGMATIC WORKS

https://pragmaticworks.com/

Private Training

Train your team to become their best

Pragmatic Works' private training provides a range of exclusive options tailored to your team's needs. Our expert-led sessions can be conducted on-site at your location or virtually, ensuring flexibility and convenience. This personalized approach allows us to address your specific training requirements, fostering operational effectiveness and skill development to help your business thrive.

Schedule Meeting

Private Training
Customized training to master new skills and grow your business.

On-Demand Learning
Beginner to advanced classes taught by Microsoft MVPs and Authors.

Bootcamps
In-depth boot camps take you from a novice to mastery in less than a week.

Season Learning Pass
Get access to our very best training offerings for successful up-skilling.

Stream Pro Plus
Combine On-Demand Learning platform with face-to-face Virtual Mentoring.

Certification Training
Prepare and ace your next certification with CertXP.

Q&A and Closing

Questions?