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

Discovering DAX

Presented by: Rebekyah Brewer

Date: May 21, 2025

Session: #37063

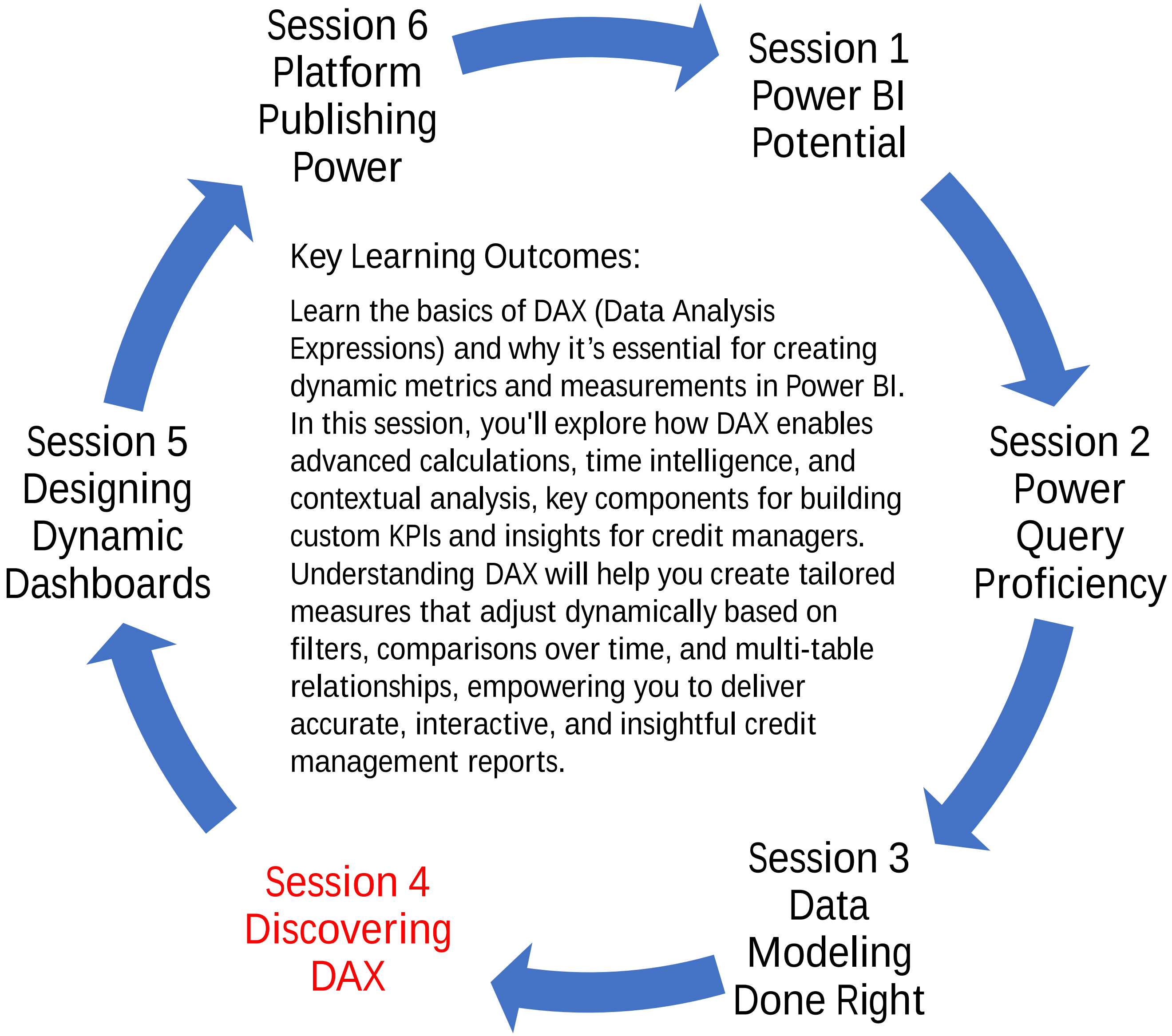
Session 4 Files Download:

Discovering DAX

Password: NACM2025

https://tinyurl.com/NACMDiscoveringDAX

Link Expiration: 6/18/2025



Session 1
Power BI
Potential

Session 2
Power
Query
Proficiency

Session 3
Data
Modeling
Done Right

Session 4
Discovering
DAX

Session 5
Designing
Dynamic
Dashboards

Session 6
Platform
Publishing
Power

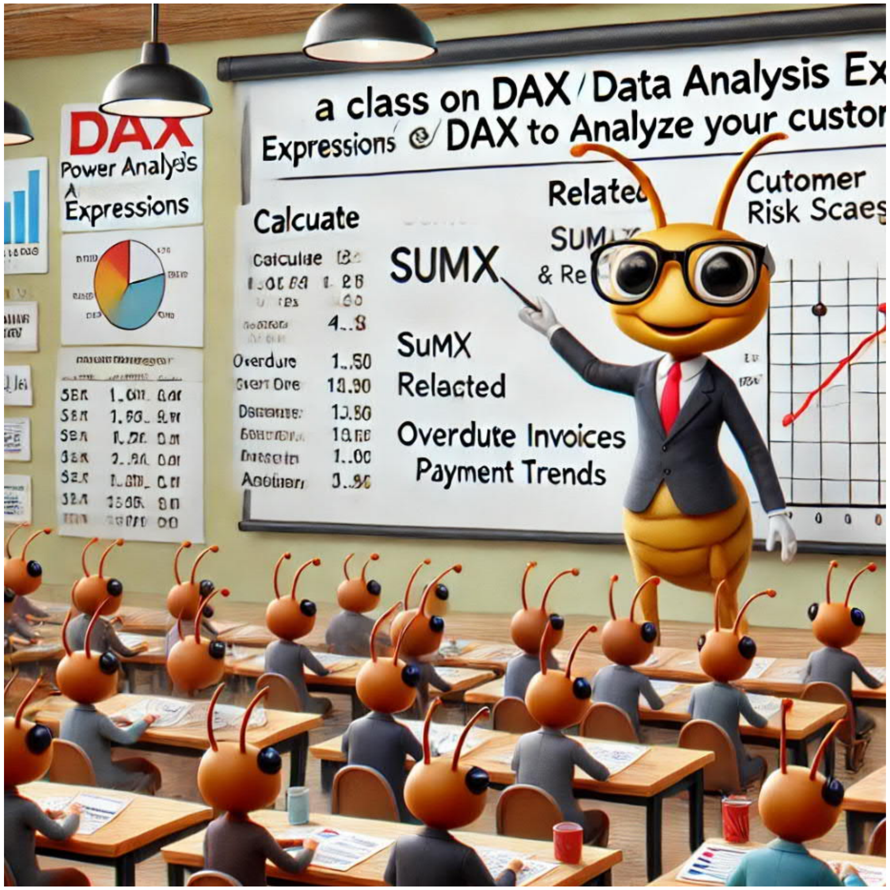
Key Learning Outcomes:

Learn the basics of DAX (Data Analysis Expressions) and why it’s essential for creating dynamic metrics and measurements in Power BI. In this session, you'll explore how DAX enables advanced calculations, time intelligence, and contextual analysis, key components for building custom KPIs and insights for credit managers. Understanding DAX will help you create tailored measures that adjust dynamically based on filters, comparisons over time, and multi-table relationships, empowering you to deliver accurate, interactive, and insightful credit management reports.

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



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.
- Internet Speed – If working with cloud data, a stable internet connection is necessary.

Prerequisites – Experience

Before diving into DAX, it's helpful when a beginner has good grasp of:

Excel Functions & Formulas 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.

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).

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.

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

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:

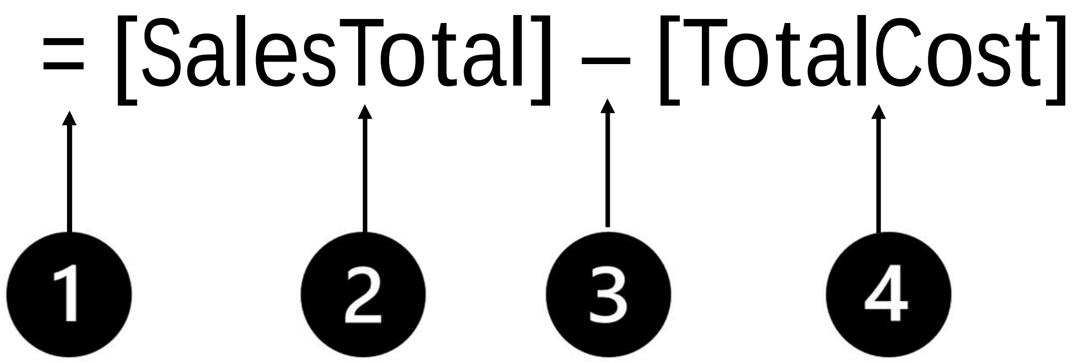
Aggregation and Filtering – Functions like SUM(), AVERAGE(), FILTER(), and CALCULATE() allow powerful data manipulation

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:



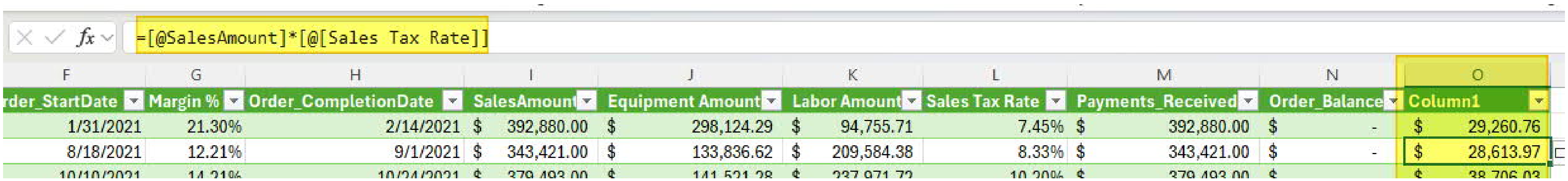
= [SalesTotal] – [TotalCost]

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 [ ]

Practice: Calculated Columns

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

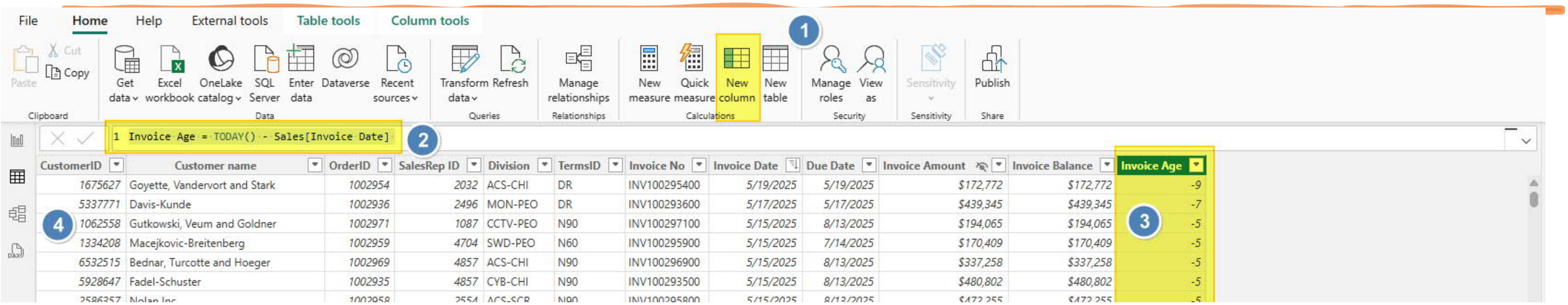
If you have ever added a new column to a ‘Table’ in Excel and enjoyed the auto calculations all the way down, Calculated Columns are very similar.



```
=[@SalesAmount]*[@[Sales Tax Rate]]
```

F	G	H	I	J	K	L	M	N	O
rder_StartDate	Margin %	Order_CompletionDate	SalesAmount	Equipment Amount	Labor Amount	Sales Tax Rate	Payments_Received	Order_Balance	Column1
1/31/2021	21.30%	2/14/2021	$ 392,880.00	$ 298,124.29	$ 94,755.71	7.45%	$ 392,880.00	$ -	$ 29,260.76
8/18/2021	12.21%	9/1/2021	$ 343,421.00	$ 133,836.62	$ 209,584.38	8.33%	$ 343,421.00	$ -	$ 28,613.97
10/10/2021	14.21%	10/24/2021	$ 379,493.00	$ 141,521.28	$ 237,971.72	10.20%	$ 379,493.00	$	$ 38,706.03

Power BI Calculated Column Example:



```
Invoice Age = TODAY() - Sales[Invoice Date]
```

CustomerID	Customer name	OrderID	SalesRep ID	Division	TermsID	Invoice No	Invoice Date	Due Date	Invoice Amount	Invoice Balance	Invoice Age
1675627	Goyette, Vandervort and Stark	1002954	2032	ACS-CHI	DR	INV100295400	5/19/2025	5/19/2025	$172,772	$172,772	-9
5337771	Davis-Kunde	1002936	2496	MON-PEO	DR	INV100293600	5/17/2025	5/17/2025	$439,345	$439,345	-7
1062558	Gutkowski, Veum and Goldner	1002971	1087	CCTV-PEO	N90	INV100297100	5/15/2025	8/13/2025	$194,065	$194,065	-5
1334208	Macejkovic-Breitenberg	1002959	4704	SWD-PEO	N60	INV100295900	5/15/2025	7/14/2025	$170,409	$170,409	-5
6532515	Bednar, Turcotte and Hoeger	1002969	4857	ACS-CHI	N90	INV100296900	5/15/2025	8/13/2025	$337,258	$337,258	-5
5928647	Fadel-Schuster	1002935	4857	CYB-CHI	N90	INV100293500	5/15/2025	8/13/2025	$480,802	$480,802	-5
2586357	Nolan Inc	1002958	2554	ACS-SCR	N90	INV100295800	5/15/2025	8/13/2025	$472,255	$472,255	-5

```
Invoice Age = TODAY() - Sales[Invoice Date]
```

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:

```
DBTAge  = TODAY() - Sales[Due Date]
```

```
(REL YRS) Years of Relationship wBusiness =  (Today( ) – Customer[Creation Date])/365
```

```
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(Customer[Credit Risk]) = "High Risk", "Alert", " ")
```

Best Practice: Calculated Columns

When to Use a Calculated Column

- ✓ Row-Level Calculations (e.g., Concatenating names, Classification)
- ✓ Sorting or Filtering needs, Slicer
- ✓ Required for Relationships between tables
- ✓ Data Model Constraint - Conditional Flags for later aggregation

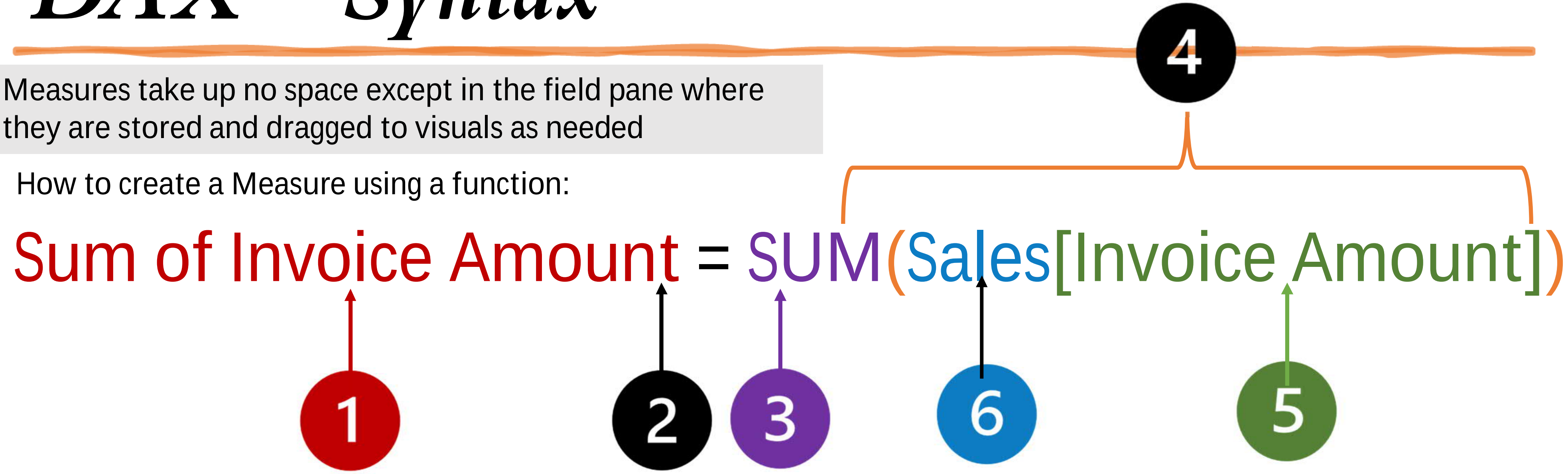
When NOT to Use a Calculated Column

- ❌ Aggregations – Use Measures instead
- ❌ Simple Transformations – Use Power Query
- ❌ Large Data Models – Reduces performance efficiency
- ❌ Anything that can be calculated dynamically with measures

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:



Sum of Invoice Amount = SUM(Sales[Invoice Amount])

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]

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)

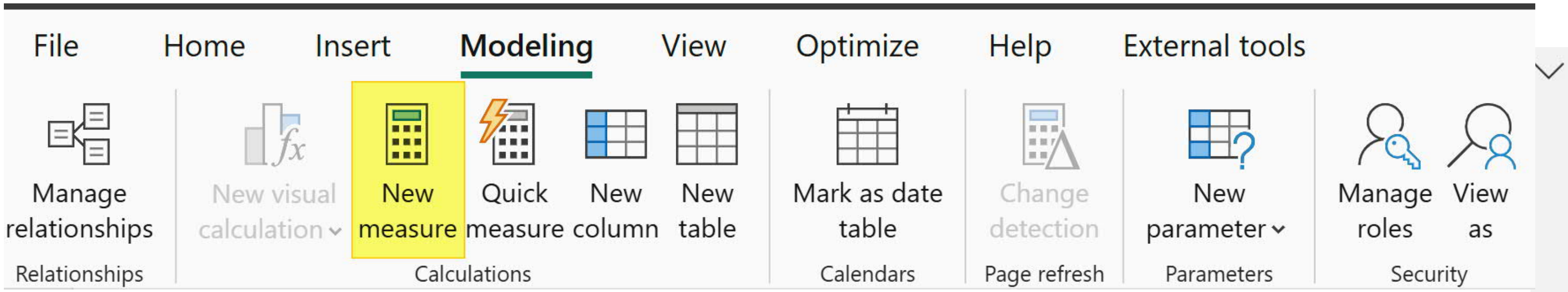
Basic Concepts: DAX Measures (DAX)

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.

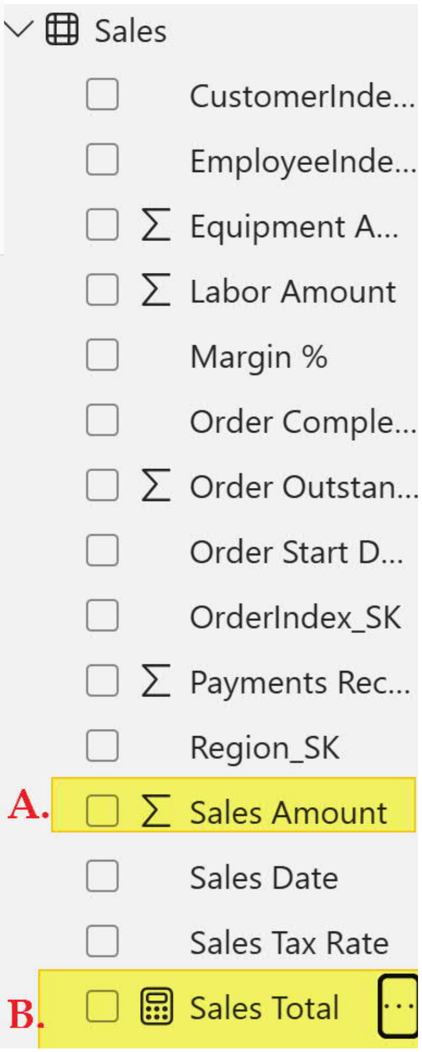


Created measures are shown in the Fields list beneath their assigned table with a little calculator icon beside them instead of the sum icon.

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	Σ Invoice Amount Implicit Measures	Invoice Amt 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

Best Practice: Take Time to Organize

Get in a habit while you connecting your relationships in your data model, setting the data types in your Power BI, setting your date table and sorting, to also create explicit measures for all your implicit measures and then hide your implicit measures along with your unnecessary sort keys ID’s .

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>)	Invoice Amt = SUM('Sales'[Invoice Amount])
AVERAGE	Returns the average (arithmetic mean) of a column.	AVERAGE(<column>)	Average Sale LTD = AVERAGE('Sales'[Invoice Amount])
MIN	Returns the smallest value in a column.	MIN(<column>)	Smallest Sale LTD = MIN('Sales'[Invoice Amount])
MAX	Returns the largest value in a column.	MAX(<column>)	Highest Sale LTD = MAX('Sales'[Invoice Amount])
COUNT	Counts the number of numeric values in a column.	COUNT(<column>)	Open AR Transactions = COUNTROWS(Sales)
COUNTA	Counts the number of non-empty values in a column.	COUNTA(<column>)	# Collection Notes = COUNTA(Collections[Collection Note])
COUNTROWS	Counts the number of rows in a table.	COUNTROWS(<table>)	Open AR Transactions = CALCULATE(COUNTROWS(Sales), ALL(Sales))
DISTINCTCOUNT	Counts the number of distinct values in a column.	DISTINCTCOUNT(<column>)	# Customers = DISTINCTCOUNT('Customer'[CustomerID]) # Invoices = DISTINCTCOUNT('Sales'[Invoice No])
SUMX	Returns the sum of an expression evaluated for each row in a table.	SUMX(<table>, <expression>)	Work Order Balance = SUMX('Work Orders', [WO Sale Amount] - [WO Cash TTD])
AVERAGEX	Returns the average of an expression evaluated for each row in a table.	AVERAGEX(<table>, <expression>)	AVERAGEX(Sales, Sales[Quantity] * Sales[Sales Amount])
MINX	Returns the smallest value of an expression evaluated for each row in a table.	MINX(<table>, <expression>)	First Sales Date = MINX('Work Orders', 'Work Orders'[SalesDate])
MAXX	Returns the largest value of an expression evaluated for each row in a table.	MAXX(<table>, <expression>)	Last Sales Date = MAXX('Work Orders', 'Work Orders'[SalesDate])

Context. Context. Context.

Understanding context is essential in DAX. There are two primary types: row context and filter context.

Row Context –

Row context refers to the current row being processed.

Example: A 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.

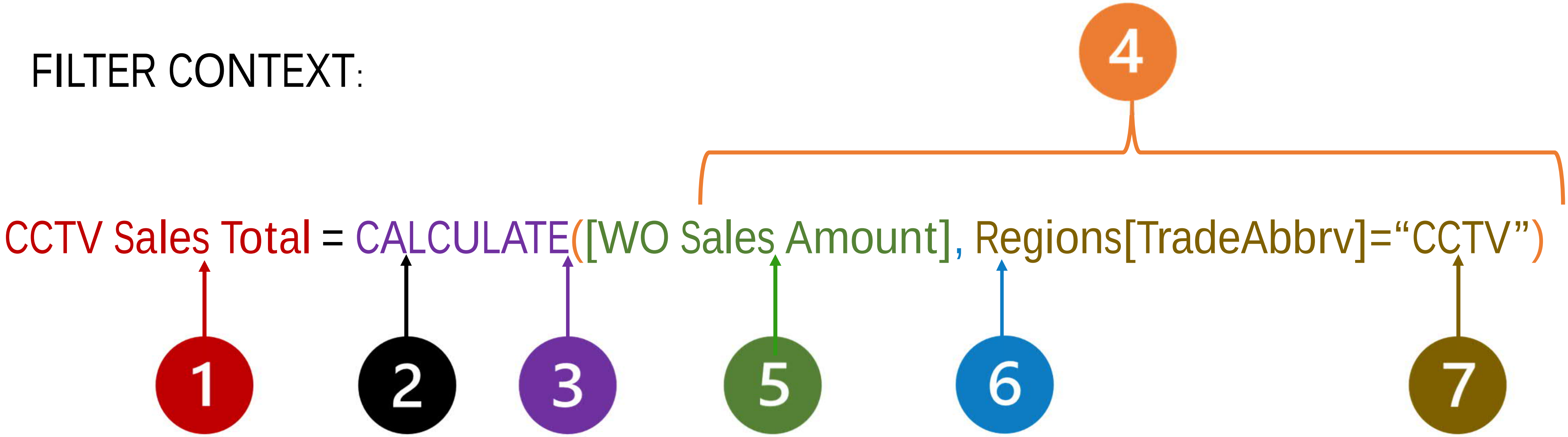
Row Context exists not just in Calculated Columns but in the SUMX, AVERAGEX, MINX and MAXX Functions.

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:



CCTV Sales Total = CALCULATE([WO Sales Amount], Regions[TradeAbbrv]="CCTV")

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 DimRegion with value "CCTV".

DAX Filters for Measures – Context Override

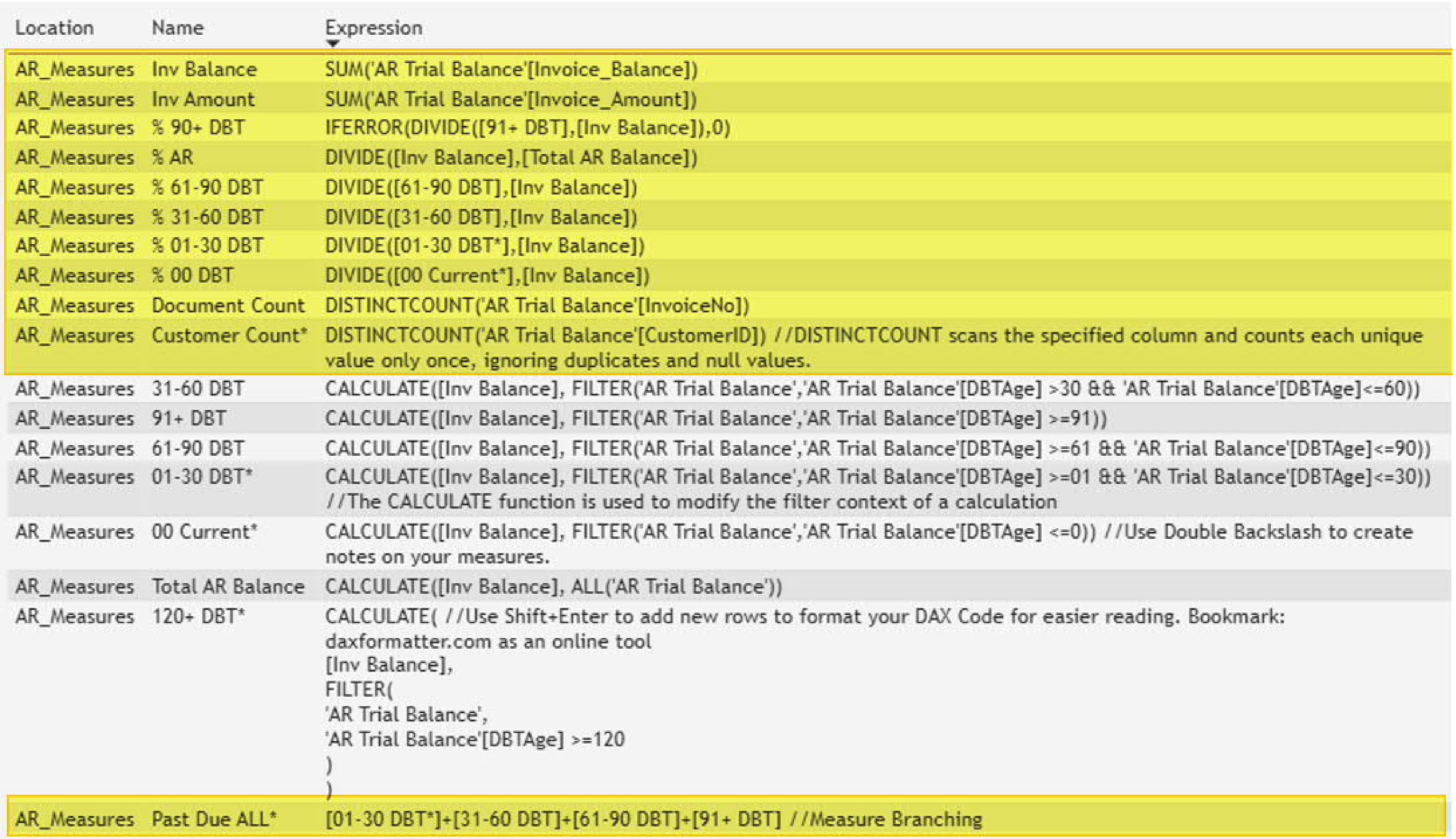
Function	Description	Syntax	Example

FILTER	Returns a filtered table based on a condition.	FILTER(<table>, <condition>)
		FILTER(Sales, Sales[Amount] > 1000)
			High Risk Balances = CALCULATE([Invoice Balance], FILTER('Customer', Customer[Credit Risk] = "High Risk"))

ALL	Removes all filters from a table or column.	ALL(<table_or_column>)	Total AR Balance = CALCULATE([Invoice Balance], ALL('Sales'))

ALLEXCEPT	Removes all filters except on specified columns.	ALLEXCEPT(<table>, <column1>, <column2>, ...)
		ALLEXCEPT(Sales, Sales[Region])
			Total AR Balance Division AllExcept = CALCULATE([Invoice Balance], ALLEXCEPT('Sales',Sales[Division])

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])



Location	Name	Expression
AR_Measures	Inv Balance	SUM('AR Trial Balance'[Invoice_Balance])
AR_Measures	Inv Amount	SUM('AR Trial Balance'[Invoice_Amount])
AR_Measures	% 90+ DBT	IFERROR(DIVIDE([91+ DBT],[Inv Balance]),0)
AR_Measures	% AR	DIVIDE([Inv Balance],[Total AR Balance])
AR_Measures	% 61-90 DBT	DIVIDE([61-90 DBT],[Inv Balance])
AR_Measures	% 31-60 DBT	DIVIDE([31-60 DBT],[Inv Balance])
AR_Measures	% 01-30 DBT	DIVIDE([01-30 DBT*],[Inv Balance])
AR_Measures	% 00 DBT	DIVIDE([00 Current*],[Inv Balance])
AR_Measures	Document Count	DISTINCTCOUNT('AR Trial Balance'[InvoiceNo])
AR_Measures	Customer Count*	DISTINCTCOUNT('AR Trial Balance'[CustomerID]) //DISTINCTCOUNT scans the specified column and counts each unique value only once, ignoring duplicates and null values.
AR_Measures	31-60 DBT	CALCULATE([Inv Balance], FILTER('AR Trial Balance','AR Trial Balance'[DBTAge] >30 && 'AR Trial Balance'[DBTAge]<=60))
AR_Measures	91+ DBT	CALCULATE([Inv Balance], FILTER('AR Trial Balance','AR Trial Balance'[DBTAge] >=91))
AR_Measures	61-90 DBT	CALCULATE([Inv Balance], FILTER('AR Trial Balance','AR Trial Balance'[DBTAge] >=61 && 'AR Trial Balance'[DBTAge]<=90))
AR_Measures	01-30 DBT*	CALCULATE([Inv Balance], FILTER('AR Trial Balance','AR Trial Balance'[DBTAge] >=01 && 'AR Trial Balance'[DBTAge]<=30)) //The CALCULATE function is used to modify the filter context of a calculation
AR_Measures	00 Current*	CALCULATE([Inv Balance], FILTER('AR Trial Balance','AR Trial Balance'[DBTAge] <=0)) //Use Double Backslash to create notes on your measures.
AR_Measures	Total AR Balance	CALCULATE([Inv Balance], ALL('AR Trial Balance'))

AR_Measures	120+ DBT*	CALCULATE(//Use Shift+Enter to add new rows to format your DAX Code for easier reading. Bookmark: daxformatter.com as an online tool
		[Inv Balance],
		FILTER(
		'AR Trial Balance',
		'AR Trial Balance'[DBTAge] >=120
		)
		)

AR_Measures	Past Due ALL*	[01-30 DBT*]+[31-60 DBT]+[61-90 DBT]+[91+ DBT] //Measure Branching
	Credit Remaining	[Credit Limit Amt – [Invoice Balance]

DAX Logical Conditional Measures

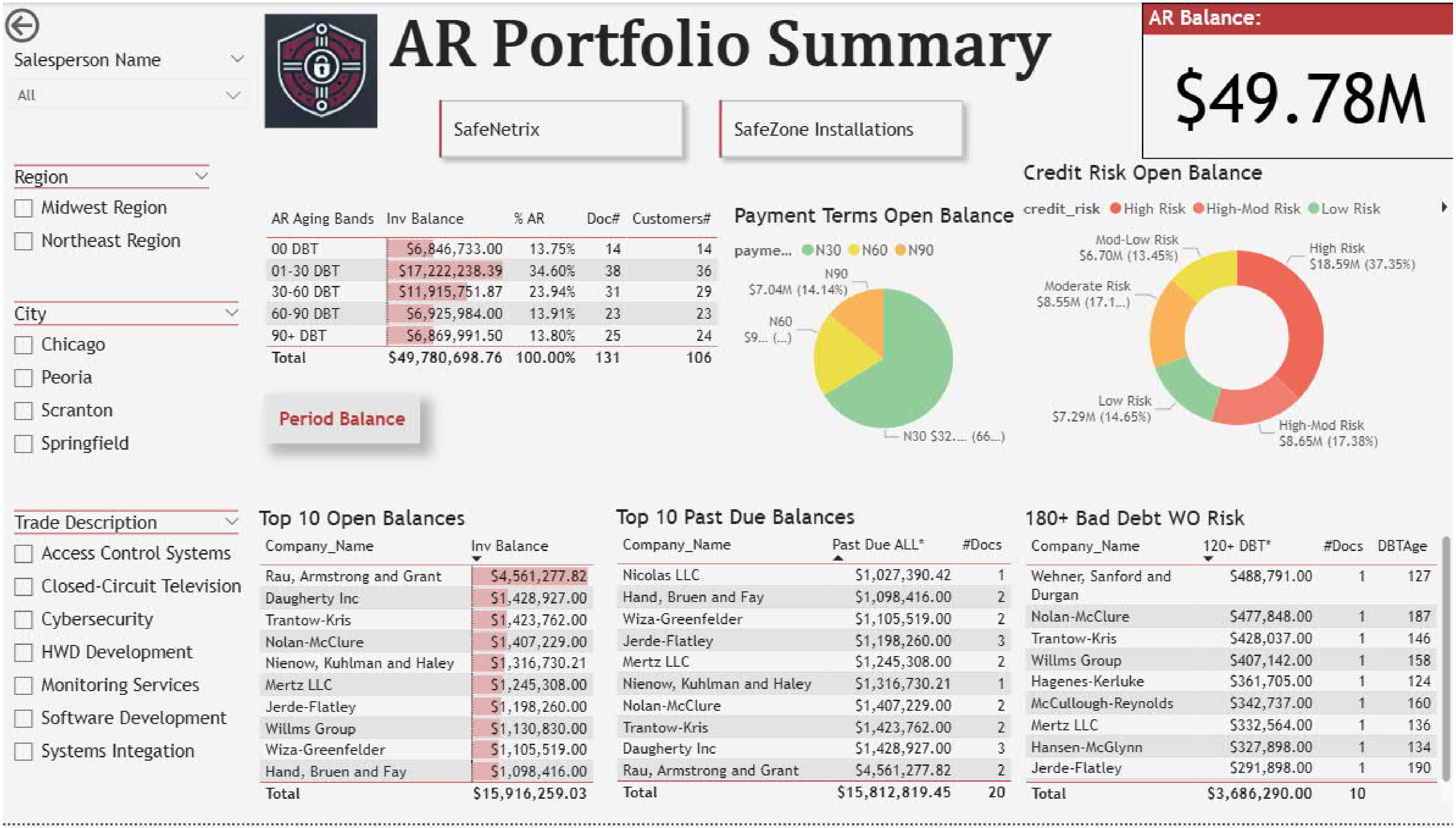
Function	Description	Syntax	Example

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")
			Over Credit Limit Check = IF([Credit Remaining] <0, "Review", "")

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])
			Collection Note Check = ISBLANK('AR Measures'[# Collection Notes])

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()



AR Portfolio Summary

Salesperson Name

All

SafeNetrix

SafeZone Installations

AR Balance:

$49.78M

Region

- □ Midwest Region
- □ Northeast Region

City

- □ Chicago
- □ Peoria
- □ Scranton
- □ Springfield

Trade Description

- □ Access Control Systems
- □ Closed-Circuit Television
- □ Cybersecurity
- □ HWD Development
- □ Monitoring Services
- □ Software Development
- □ Systems Integation

AR Aging Bands	Inv Balance	% AR	Doc#	Customers#
00 DBT	$6,846,733.00	13.75%	14	14
01-30 DBT	$17,222,238.39	34.60%	38	36
30-60 DBT	$11,915,751.87	23.94%	31	29
60-90 DBT	$6,925,984.00	13.91%	23	23
90+ DBT	$6,869,991.50	13.80%	25	24
Total	$49,780,698.76	100.00%	131	106

Period Balance

Payment Terms Open Balance

Credit Risk Open Balance

Top 10 Open Balances

Company_Name	Inv Balance
Rau, Armstrong and Grant	$4,561,277.82
Daugherty Inc	$1,428,927.00
Trantow-Kris	$1,423,762.00
Nolan-McClure	$1,407,229.00
Nienow, Kuhlman and Haley	$1,316,730.21
Mertz LLC	$1,245,308.00
Jerde-Flatley	$1,198,260.00
Willms Group	$1,130,830.00
Wiza-Greenfelder	$1,105,519.00
Hand, Bruen and Fay	$1,098,416.00
Total	$15,916,259.03

Top 10 Past Due Balances

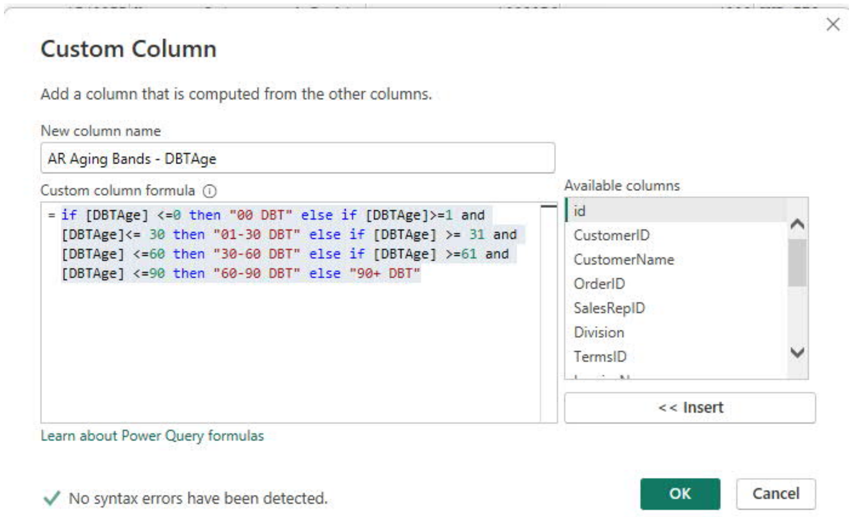
Company_Name	Past Due ALL*	#Docs
Nicolas LLC	$1,027,390.42	1
Hand, Bruen and Fay	$1,098,416.00	2
Wiza-Greenfelder	$1,105,519.00	2
Jerde-Flatley	$1,198,260.00	3
Mertz LLC	$1,245,308.00	2
Nienow, Kuhlman and Haley	$1,316,730.21	1
Nolan-McClure	$1,407,229.00	2
Trantow-Kris	$1,423,762.00	2
Daugherty Inc	$1,428,927.00	3
Rau, Armstrong and Grant	$4,561,277.82	2
Total	$15,812,819.45	20

180+ Bad Debt WO Risk

Company_Name	120+ DBT*	#Docs	DBTAge
Wehner, Sanford and Durgan	$488,791.00	1	127
Nolan-McClure	$477,848.00	1	187
Trantow-Kris	$428,037.00	1	146
Willms Group	$407,142.00	1	158
Hagenes-Kerluke	$361,705.00	1	124
McCullough-Reynolds	$342,737.00	1	160
Mertz LLC	$332,564.00	1	136
Hansen-McGlynn	$327,898.00	1	134
Jerde-Flatley	$291,898.00	1	190
Total	$3,686,290.00	10	

CALCULATED COLUMN FOR AR AGING BANDS ALLOWS FILTERING

Always created calculated column either in Power Query Custom Column or in Power BI Calculated Column for anything that will be sliced or filtered. Instead of using a measure.



Custom Column

Add a column that is computed from the other columns.

New column name

AR Aging Bands - DBTAge

Custom column formula

```
= if [DBTAge] <=0 then "00 DBT" else if [DBTAge]>=1 and
  [DBTAge]<= 30 then "01-30 DBT" else if [DBTAge] >= 31 and
  [DBTAge] <=60 then "30-60 DBT" else if [DBTAge] >=61 and
  [DBTAge] <=90 then "60-90 DBT" else "90+ DBT"
```

✓ No syntax errors have been detected.

Power Query M Code: AR Aging Bands - DBTAge

```
if [DBTAge] <=0 then "00 DBT" else if [DBTAge]>=1 and
[DBTAge]<= 30 then "01-30 DBT" else if [DBTAge] >= 31 and
[DBTAge] <=60 then "30-60 DBT" else if [DBTAge] >=61 and
[DBTAge] <=90 then "60-90 DBT" else "90+ DBT"
```

(IF Method) AR Aging Bands - DBTAge

```
AR Aging Bands - DBTAge =
  IF(
    [DBTAge] <= 0, "00 DBT",
    IF(
      [DBTAge] >= 1 && [DBTAge] <= 30, "01-30 DBT",
      IF(
        [DBTAge] >= 31 && [DBTAge] <= 60, "30-60 DBT",
        IF(
          [DBTAge] >= 61 && [DBTAge] <= 90, "60-90 DBT",
          "90+ DBT"
        )
      )
    )
  )
```

(Switch Method) AR Aging Bands - DBTAge

```
AR Aging Bands - DBTAge =
    SWITCH(
        TRUE(),
        [DBTAge] <= 0, "00 DBT",
        [DBTAge] >= 1 && [DBTAge] <= 30, "01-30 DBT",
        [DBTAge] >= 31 && [DBTAge] <= 60, "30-60 DBT",
        [DBTAge] >= 61 && [DBTAge] <= 90, "60-90 DBT",
        "90+ DBT"
    )
```

AR Aging Bands Measures Methods for DAX

DAX Method – IF AR Aging Bands

```
AR Aging Bands Measure (IF Method) =
    IF(
        MAX(Sales[DBTAge]) <= 0, "00 DBT",
        IF(
            MAX(Sales[DBTAge]) >= 1 && MAX(Sales[DBTAge]) <= 30, "01-30 DBT",
            IF(
                MAX(Sales[DBTAge]) >= 31 && MAX(Sales[DBTAge]) <= 60, "30-60 DBT",
                IF(
                    MAX(Sales[DBTAge]) >= 61 && MAX(Sales[DBTAge]) <= 90, "60-90 DBT",
                    "90+ DBT"
                )
            )
        )
    )
```

DAX Method – Switch AR Aging Bands

```
AR Aging Bands Measure Switch method =

    SWITCH(
        TRUE(),
        MAX('Sales'[DBTAge]) <= 0, "00 DBT",
        MAX('Sales'[DBTAge]) >= 1 && MAX('Sales'[DBTAge]) <= 30, "01-30 DBT",
        MAX('Sales'[DBTAge]) >= 31 && MAX('Sales'[DBTAge]) <= 60, "30-60 DBT",
        MAX('Sales'[DBTAge]) >= 61 && MAX('Sales'[DBTAge]) <= 90, "60-90 DBT",
        "90+ DBT"
    )
```

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]), Date[Date]) Sales YTD = TOTALYTD([WO Sale Amount], 'Dates'[Date])
TOTALQTD	Calculates quarter-to-date total for a measure.	TOTALQTD(<expression>, <dates_column>[, <filter>])	TOTALQTD(SUM(Sales[Amount]), ‘Date’[Date])
TOTALMTD	Calculates month-to-date total for a measure.	TOTALMTD(<expression>, <dates_column>[, <filter>])	TOTALMTD(SUM(Sales[Amount]), ‘Date’[[Date])
PREVIOUSYEAR	Returns the measure value for the previous year.	PREVIOUSYEAR(<dates_column>)	PREVIOUSYEAR(‘Date’[[Date])
PREVIOUSQUARTER	Returns the measure value for the previous quarter.	PREVIOUSQUARTER(<dates_column>)	PREVIOUSQUARTER(‘Date’[Date])
PREVIOUSMONTH	Returns the measure value for the previous month.	PREVIOUSMONTH(<dates_column>)	PREVIOUSMONTH(‘Date’[[Date])
PREVIOUSDAY	Returns the measure value for the previous day.	PREVIOUSDAY(<dates_column>)	PREVIOUSDAY(‘Date’[Date])
SAMEPERIODLASTYEAR	Returns the measure value for the same period in the previous year.	SAMEPERIODLASTYEAR(<dates_column>)	SAMEPERIODLASTYEAR(‘Date’[[Date])
DATEADD	Shifts dates forward or backward by a given number of intervals.	DATEADD(<dates_column>, <number_of_intervals>, <interval_type>)	DATEADD(‘Date’[Date], -1, YEAR)
DATEDIFF	Calculates the difference between two dates based on a specified time interval (e.g., day, month, year	DATEDIFF(Start_Date, End_Date, Interval)	Age Doc = MAXX('Sales', DATEDIFF('Sales'[Invoice Date],TODAY(),DAY))
PARALLELPERIOD	Returns a parallel period, shifting by a given number of intervals.	PARALLELPERIOD(<dates_column>, <number_of_intervals>, <interval_type>)	PARALLELPERIOD(‘Date’[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])

Advanced DAX – Rolling Average | Moving Calculations

A Rolling Average (or moving average) calculates the average of a specific measure over a defined time window. It is commonly used to smooth data and identify trends

Explanation:

- **DATESINPERIOD()** – Defines the rolling window of dates based on the current date context.
- **LASTDATE()** – Specifies the end date of the rolling window.
- **-3, MONTH** – Looks back three months from the last date in the current filter context.
- **AVERAGEX()** – Iterates over each date in the defined period and calculates the average of the specified measure ([Total Sales]).

```
Rolling 3-Month Average Sales =
    CALCULATE(
        AVERAGEX(
            DATESINPERIOD(
                ‘Calendar'[Date],
                LASTDATE('Date'[Date]),
                -3,
                MONTH
            ),
            [Total Sales]
        )
    )
```

```
Rolling 7-Day Average Sales =
    CALCULATE(
        AVERAGEX(
            DATESINPERIOD(
                'Date'[Date],
                LASTDATE('Date'[Date]),
                -7,
                DAY
            ),
            [Total Sales]
        )
    )
```

Advanced DAX – Cumulative Sales | Running Total | Rolling Sum

It calculates the sum of values from the start of a period up to the current point in time, adding the previous day's value to the current day's value consecutively.

Explanation:

- **CALCULATE()**: Modifies the filter context to include all dates up to the current date.
- **ALL('Date')**: Removes existing filters to ensure we are calculating across the entire date range.
- **FILTER()**: Applies the condition to include all dates up to the current date (MAX('Date'[Date]))

```
Cumulative Sales =
   CALCULATE(
      SUM(Sales[Invoice Amount]),
      FILTER(
         ALL('Date'),
         ‘Date'[Date] <= MAX(‘Date'[Date])
      )
   )
```

```
Daily Cumulative Sales =
   CALCULATE(
      SUM(Sales[SalesAmount]),
      FILTER(
         ALL(‘Date'),
         ‘Date'[Date] <= MAX(‘Date'[Date])
      )
   )
```

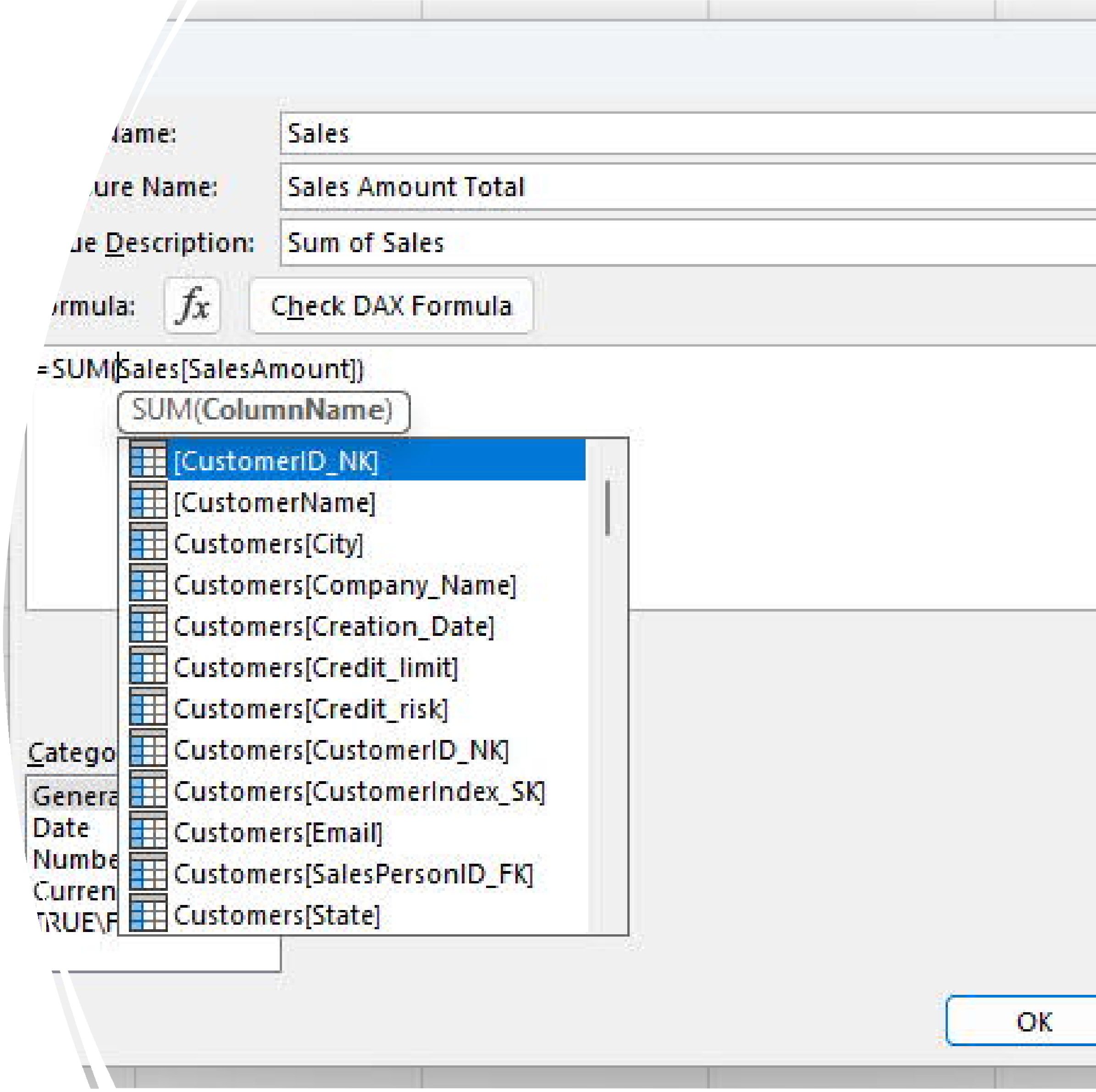
```
Monthly Cumulative Sales =
   CALCULATE(
      SUM(Sales[Sales Amount]),
      FILTER(
         ALL(‘Date'),
         'Date'[Date] <= MAX(‘Date'[Date]) &&
         MONTH(‘Date'[Date]) = MONTH(MAX(‘Date'[Date])) &&
         YEAR(‘Date'[Date]) = YEAR(MAX(‘Date'[Date]))
      )
   )
```

DAX Fundamental Table Functions

Function	Description	Syntax	Example
ADDCOLUMNS	Adds calculated columns to a table.	ADDCOLUMNS(<table>, <column_name>, <expression>[, <column_name>, <expression>, ...])	ADDCOLUMNS('Work Orders', "Profit", 'Work Orders'[SalesAmount]- 'Work Orders'[Equipment Amount])
SUMMARIZE	Returns a summary table with aggregated values.	SUMMARIZE(<table>, <group_by_column>[, <group_by_column>, ...][, <name>, <expression>, ...])	Summarize Table Division Sales by Period = SUMMARIZE('Work Orders', 'Dates'[YYYY-MMM], 'Work Orders'[Division_NK], "Total Sales", SUM('Work Orders'[SalesAmount]))
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>, ...])	Region by Year Total Sales = SUMMARIZECOLUMNS('Dates'[Fiscal Year], Regions[Region], "Total Sales", [WO Sale Amount])
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)

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



ame: Sales

ure Name: Sales Amount Total

ue Description: Sum of Sales

rmula:

=SUM(Sales[SalesAmount])

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

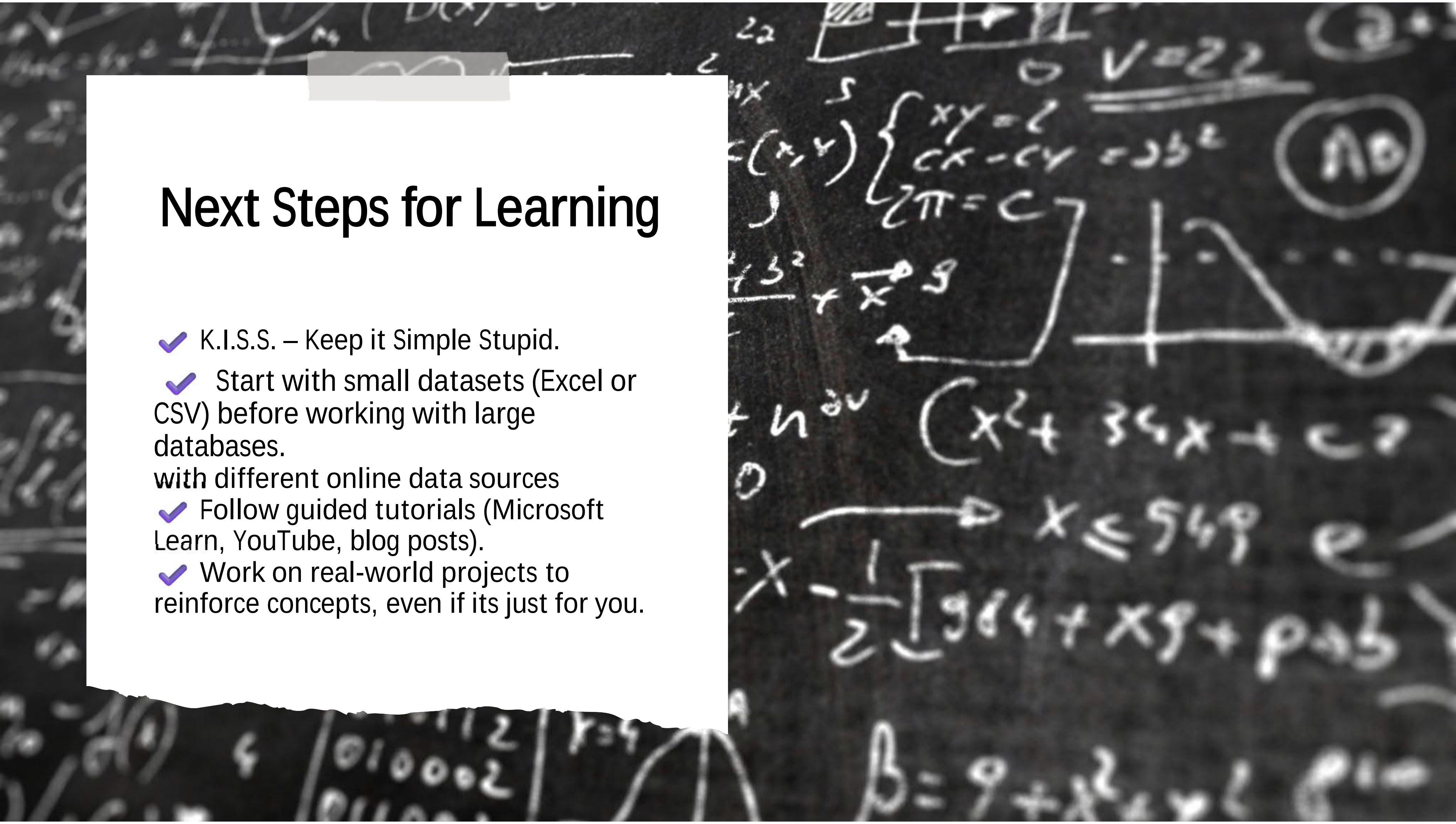
Total AR	Total outstanding Accounts Receivable (A/R) balance.	Total AR Balance = CALCULATE([Open Balance], ALL('Sales'))
00 DBT AR	Current AR. Balance of invoices that are not overdue.	00 DBT = CALCULATE([Open Balance], FILTER('Sales’, 'Sales'[DBTAge] <= 0))
01-30 DBT AR	Total A/R balance for invoices due within 0-30 days.	01-30 DBT = CALCULATE([Open Balance], FILTER('Sales','Sales'[DBTAge] >= 1 && 'Sales'[DBTAge] <= 30))
31-60 DBT AR	Total A/R balance for invoices due within 31-60 days.	31-60 DBT = CALCULATE([Open Balance], FILTER('Sales','Sales'[DBTAge] >= 31 && 'Sales'[DBTAge] <= 60))
61-90 DBT AR	Total A/R balance for invoices due within 61-90 days.	61-90 DBT (DateDiff) = CALCULATE([Open Balance], DATEDIFF(Sales[Due Date], TODAY(), DAY)>=61, DATEDIFF('Sales'[Due Date],TODAY(),DAY)<=90)
90+ DBT AR	Total A/R balance for invoices due over 90 days.	90+ DBT = CALCULATE([Open Balance], FILTER('Sales', 'Sales'[DBTAge] >= 91))
Past Due AR	Total amount of past-due invoices.	Past Due AR = CALCULATE([Open Balance], Sales[Due Date] < TODAY())
60+ DBT AR	Total A/R balance for invoices due over 60 days.	60+ DBT = [61-90 DBT] + [90+ DBT]
% Past Due AR	Percentage of total A/R that is overdue.	% Past Due AR= DIVIDE([Past Due AR], [Total AR Balance], 0)
% 90+ DBT AR	Percentage AR over 90 Days Beyond Terms	% 90+ DBT = DIVIDE([90+ DBT],[Total AR Balance])
% 60+ DBT AR	Percentage AR over 60 Days Beyond Terms	% 60+ DBT = DIVIDE([60+ DBT],[Total AR Balance])
% 00 DBT	Percentage AR that is Current	% 00 DBT = DIVIDE([00 DBT],[Total AR Balance])
% 01-30 DBT	Percentage AR 01-30 DBT	% 01-30 DBT = DIVIDE([01-30 DBT],[Total AR Balance])
% 31-60 DBT	Percentage AR 31-60 DBT	% 31-60 DBT = DIVIDE([31-60 DBT],[Total AR Balance])
% 61-90 DBT	Percentage AR 61-90 DBT	% 61-90 DBT = DIVIDE([61-90 DBT],[Total AR Balance])
% AR	Percentage Open Balane to Total AR	% AR = DIVIDE([Open Balance], [Total AR Balance])

AR Portfolio Analysis Calculations Cont’d

# Invoices	Distinct Count # of Invoice Documents on AR to track Transaction activity and volume	# Invoices = DISTINCTCOUNT('Sales'[Invoice No])
# Customers	Distinct Count of # Customers on AR with Balances	# Customers = DISTINCTCOUNT('Sales'[CustomerID])
		Average Days Outstanding = AVERAGEX('Sales','Sales'[DBTAge])
High Risk Balances	Calculates amount of outstanding AR that is categorized to High-Risk customers.	High Risk Balances = CALCULATE([Open Balance], FILTER('Customer', Customer[Credit Risk] = "High Risk"))
% High Risk Balance	Calculates percentage of outstanding AR that is categorized to High-Risk customers.	% High Risk Balance = DIVIDE([High Risk Balances],[Total AR Balance])
AR Aging Category	Categorization of AR into aging buckets (e.g., 00-30, 31-60, 61-90, 90+ days).	AR Aging Bands Measure= SWITCH(TRUE(), MAX('Sales'[DBTAge]) <= 0, "00 DBT", MAX('Sales'[DBTAge]) >= 1 && MAX('Sales'[DBTAge]) <= 30, "01-30 DBT", MAX('Sales'[DBTAge]) >= 31 && MAX('Sales'[DBTAge]) <= 60, "30-60 DBT", MAX('Sales'[DBTAge]) >= 61 && MAX('Sales'[DBTAge]) <= 90, "60-90 DBT", "90+ DBT“)
Amount Over Credit Limit	Calculates amount of open AR balance is over credit limit	Amount Over Credit Limit = SUMX(FILTER('Sales', 'Sales'[Invoice Balance]> RELATED(Customer[Credit Limit])), 'Sales'[Invoice Balance] - RELATED(Customer[Credit Limit]))
Message Over Credit Limit	Displays a text value “Review” if balance is over credit limit.	Msg Over Credit Limit Check = IF([Credit Remaining] <0, "Review", "")
Credit Utilization	Credit utilization ratio - how much of total credit limit is used.	Credit Utilization = DIVIDE([Open Balance],[Credit Limit Amt],0)
AR Concentration %	Percentage of AR Concentrated.	AR Concentration % = DIVIDE([Open Balance], [Total AR Balance])

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



Next Steps for Learning

✓ K.I.S.S. – Keep it Simple Stupid.

✓ Start with small datasets (Excel or CSV) before working with large databases.

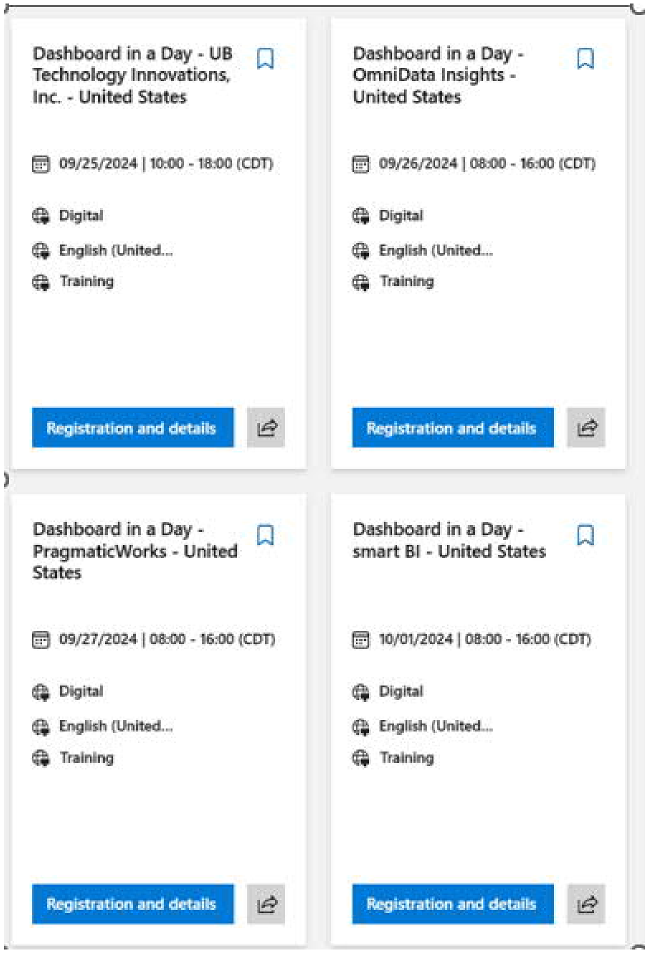
with different online data sources

✓ Follow guided tutorials (Microsoft Learn, YouTube, blog posts).

✓ Work on real-world projects to reinforce concepts, even if its just for you.

Attend a Free 1 Day Event Workshop:

Microsoft Dashboard in a Day



Dashboard in a Day - UB Technology Innovations, Inc. - United States

09/25/2024 | 10:00 - 18:00 (CDT)

Digital

English (United...

Training

Registration and details

Dashboard in a Day - OmniData Insights - United States

09/26/2024 | 08:00 - 16:00 (CDT)

Digital

English (United...

Training

Registration and details

Dashboard in a Day - PragmaticWorks - United States

09/27/2024 | 08:00 - 16:00 (CDT)

Digital

English (United...

Training

Registration and details

Dashboard in a Day - smart BI - United States

10/01/2024 | 08:00 - 16:00 (CDT)

Digital

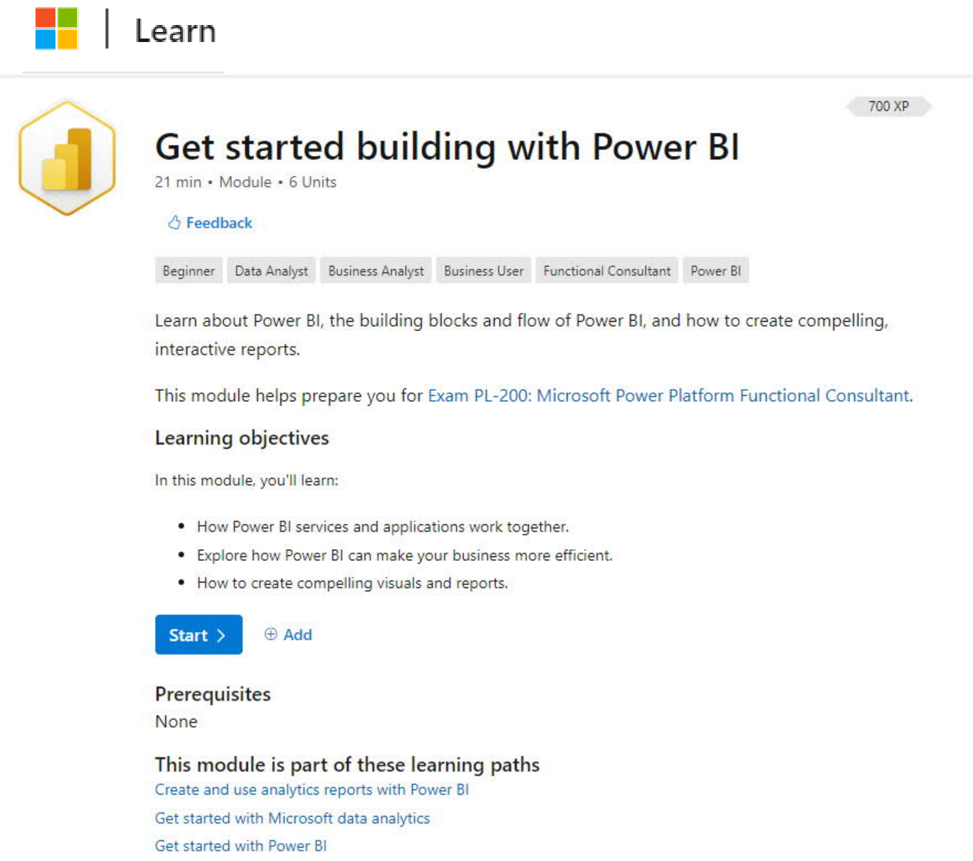
English (United...

Training

Registration and details

- 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

Pragmatic Works DAX Cheat Sheet for Beginners



Learn

700 XP

Get started building with Power BI

21 min • Module • 6 Units

Feedback

Beginner | Data Analyst | Business Analyst | Business User | Functional Consultant | Power BI

Learn about Power BI, the building blocks and flow of Power BI, and how to create compelling, interactive reports.

This module helps prepare you for Exam PL-200: Microsoft Power Platform Functional Consultant.

Learning objectives

In this module, you'll learn:

- How Power BI services and applications work together.
- Explore how Power BI can make your business more efficient.
- How to create compelling visuals and reports.

Start > ⊕ Add

Prerequisites

None

This module is part of these learning paths

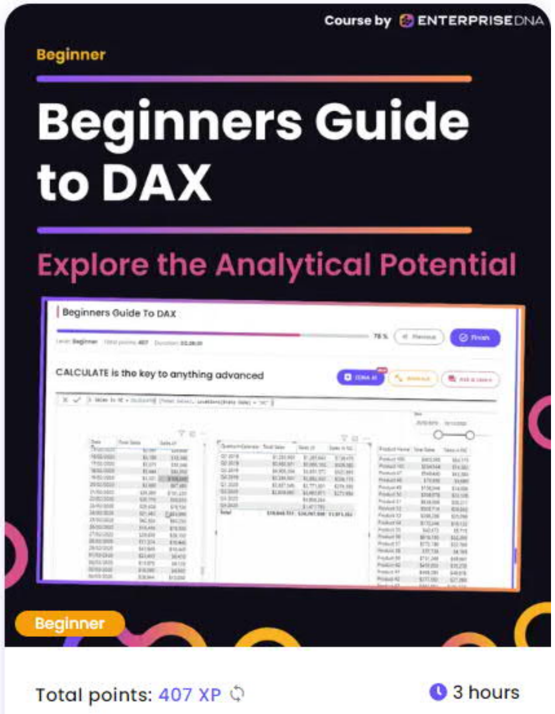
Create and use analytics reports with Power BI

Get started with Microsoft data analytics

Get started with Power BI

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

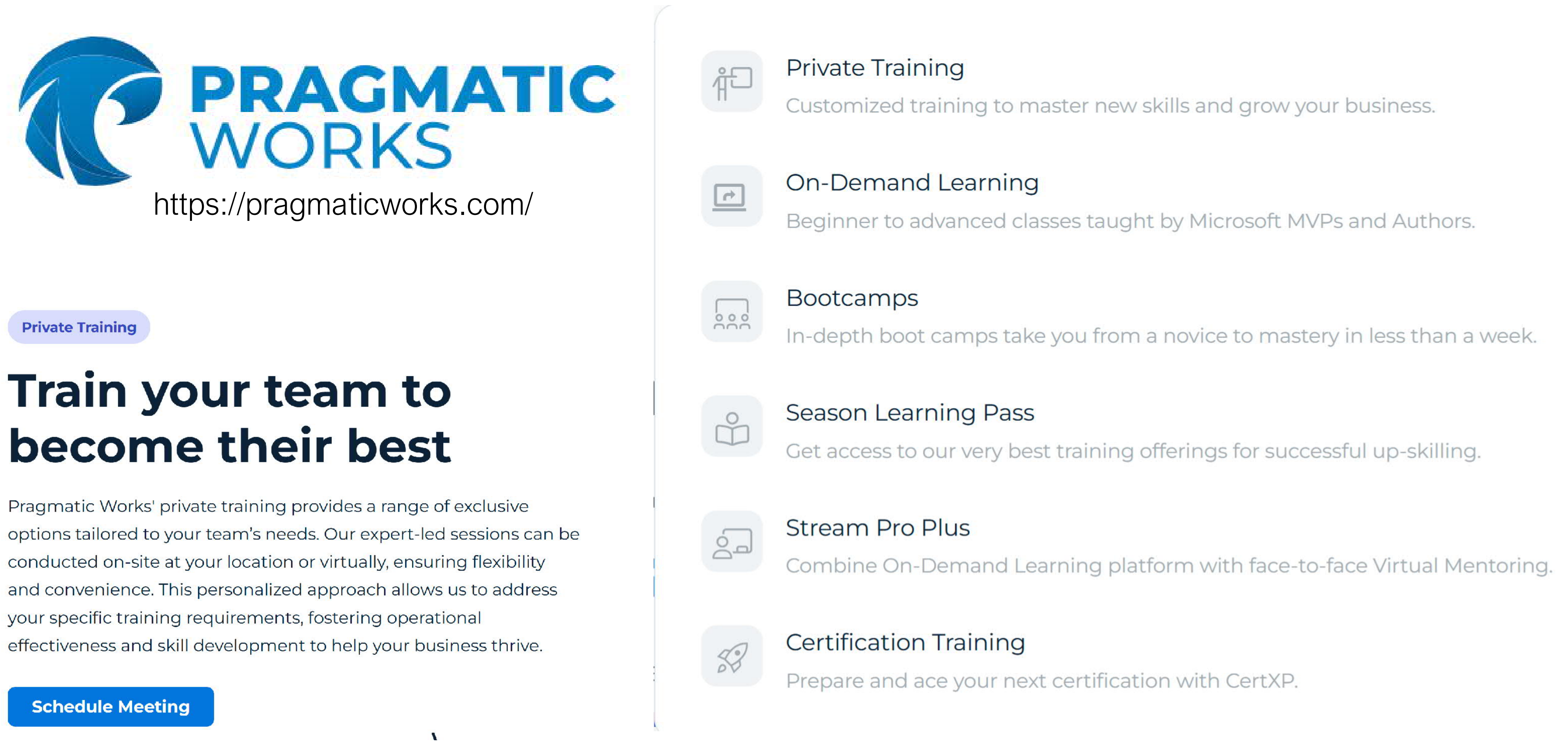
Click to start: ENTERPRISE DNA

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?