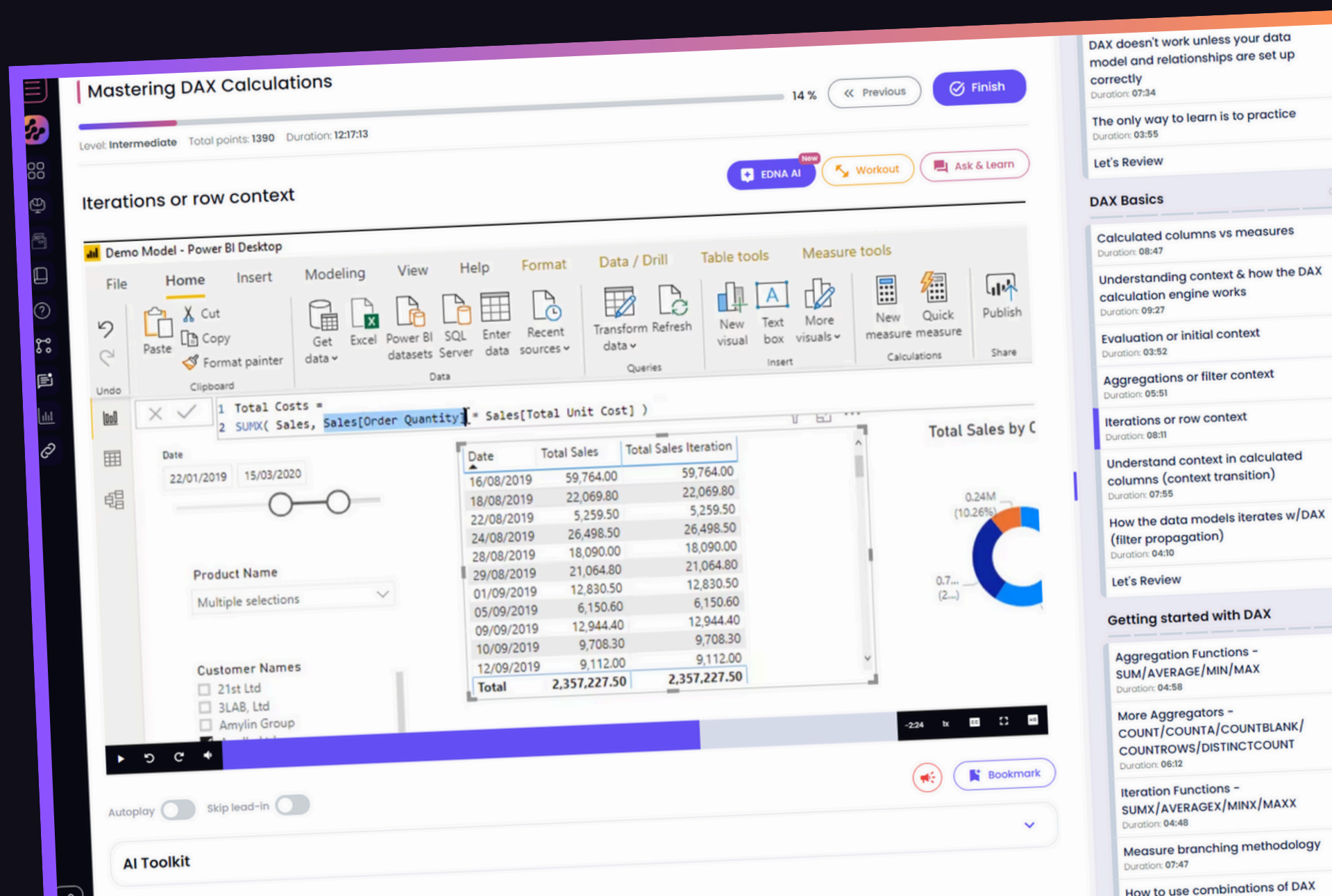


BROUGHT TO YOU BY  ENTERPRISEDNA

# Mastering DAX



## for Power BI – a detailed guide



**Mastering DAX Calculations** 14% [Previous](#) [Finish](#)

Level: Intermediate Total points: 1390 Duration: 12:17:13

[EDNA AI](#) [Workout](#) [Ask & Learn](#)

### Iterations or row context

Demo Model - Power BI Desktop

File Home Insert Modeling View Help Format Data / Drill Table tools Measure tools

Undo Paste Cut Copy Format painter Get data Excel Power BI datasets SQL Enter data Recent sources Transform Refresh data New visual Text box More visuals New measure Quick measure Publish

1 Total Costs =  
2 SUMX( Sales, Sales[Order Quantity] \* Sales[Total Unit Cost] )

Date	Total Sales	Total Sales Iteration
16/08/2019	59,764.00	59,764.00
18/08/2019	22,069.80	22,069.80
22/08/2019	5,259.50	5,259.50
24/08/2019	26,498.50	26,498.50
28/08/2019	18,090.00	18,090.00
29/08/2019	21,064.80	21,064.80
01/09/2019	12,830.50	12,830.50
05/09/2019	6,150.60	6,150.60
09/09/2019	12,944.40	12,944.40
10/09/2019	9,708.30	9,708.30
12/09/2019	9,112.00	9,112.00
<b>Total</b>	<b>2,357,227.50</b>	<b>2,357,227.50</b>

Total Sales by C

0.24M (10.26%)

0.7... (2...)

Product Name: Multiple selections

Customer Names: ☐ 21st Ltd ☐ 3LAB, Ltd ☐ Amylin Group

Autoplay ☐ Skip lead-in ☐

[Bookmark](#)

AI Toolkit

DAX doesn't work unless your data model and relationships are set up correctly  
Duration: 07:34

The only way to learn is to practice  
Duration: 03:55

Let's Review

### DAX Basics

Calculated columns vs measures  
Duration: 08:47

Understanding context & how the DAX calculation engine works  
Duration: 09:27

Evaluation or initial context  
Duration: 03:52

Aggregations or filter context  
Duration: 05:51

Iterations or row context  
Duration: 08:11

Understand context in calculated columns (context transition)  
Duration: 07:55

How the data models iterates w/DAX (filter propagation)  
Duration: 04:10

Let's Review

### Getting started with DAX

Aggregation Functions - SUM/AVERAGE/MIN/MAX  
Duration: 04:58

More Aggregators - COUNT/COUNTA/COUNTBLANK/COUNTROWS/DISTINCTCOUNT  
Duration: 06:12

Iteration Functions - SUMX/AVERAGEX/MINX/MAXX  
Duration: 04:48

Measure branching methodology  
Duration: 07:47

How to use combinations of DAX

# 41 TOPICS TO FOCUS ON WHEN LEARNING DAX

- Introduction to DAX
- Key Concepts in DAX
- Data Types and Operators
- Naming Conventions
- Context in DAX
- Basic Aggregation Functions
- Count Functions
- Iterating Functions
- Measure Branching
- Data Types and Conversion
- Error Handling in DAX
- Logical Functions
- Division in DAX
- Time Intelligence Functions
- Table Functions
- Quick Measures
- Data Model and Relationships
- CALCULATE Function
- Filter Context

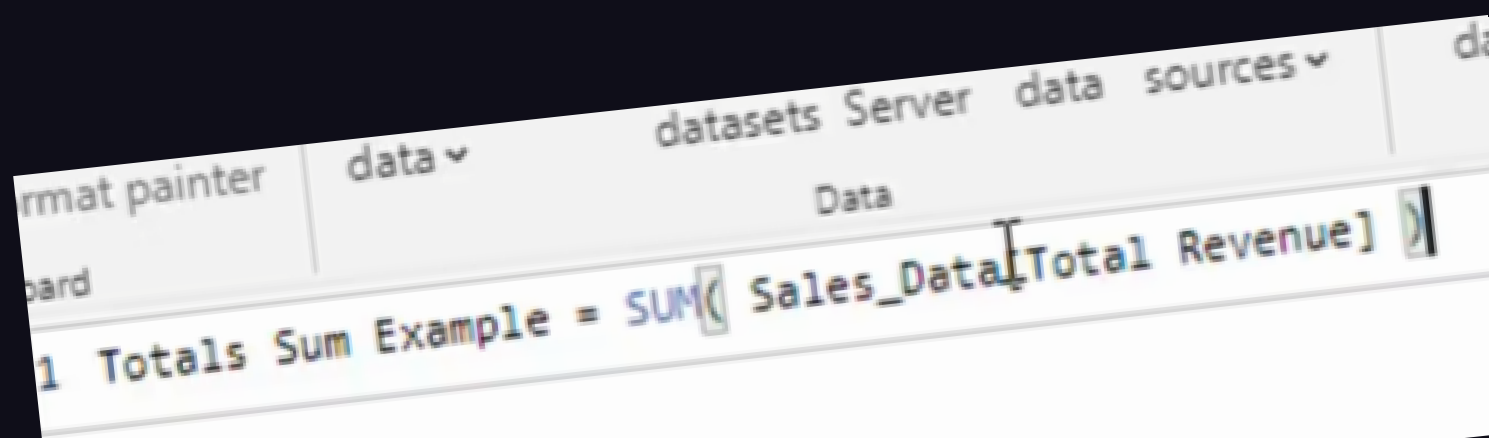
- ALL Functions
- VALUES Function
- SELECTEDVALUE Function
- RANKX Function
- TOPN Function
- Measure Groups
- Formatting DAX Code
- Using Variables in DAX
- Writing Comments
- Context Transition
- Filter Propagation
- Handling Totals
- Virtual Tables
- Scenario Analysis
- Advanced Analytics
- Best Practices for DAX
- Error Handling Best Practices
- Combining DAX Functions
- Evaluating DAX Formulas
- Context in Calculated Columns
- CALCULATE and Context
- Time Intelligence in Practice

# 1

# Introduction to DAX

## What is DAX and its importance in Power BI

- **DAX (Data Analysis Expressions)** is a powerful language designed for data modeling and analysis in Power BI.
- It is more **scalable and efficient** than traditional Excel formulas, allowing for advanced data manipulations and calculations.
- DAX enables quick creation of complex analytical solutions with relatively simple code.
- Its integration with Power BI significantly enhances the tool's ability to handle large datasets and perform in-depth analysis.



## 2

# Key Concepts in DAX

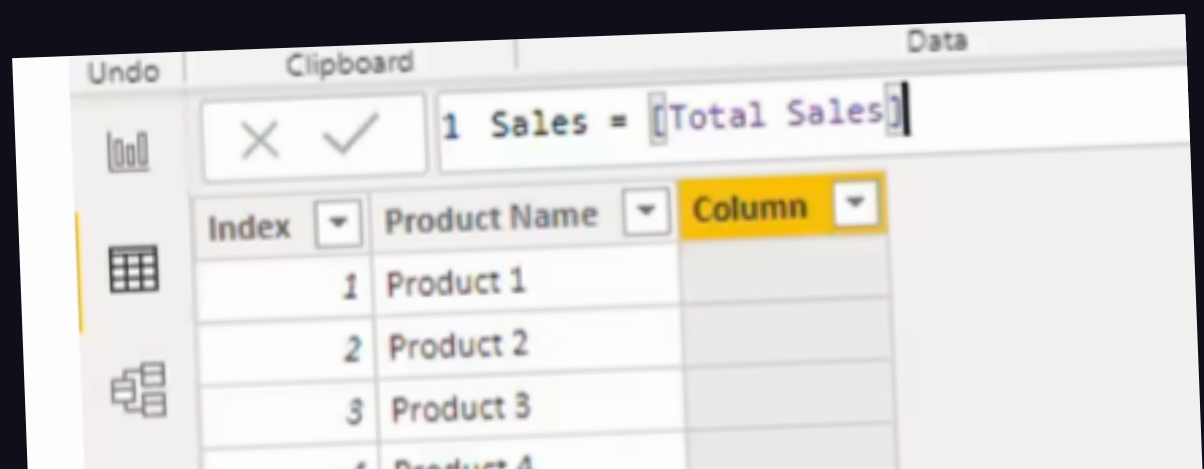
## Understanding Context in DAX

- **Context** is critical in DAX as it defines the environment in which formulas are evaluated, affecting the results.
- There are two main types of context: **row context** (related to individual rows) and **filter context** (related to filtered data sets).
- Proper understanding of context helps in accurately writing and troubleshooting DAX formulas.
- Mastery of context is essential for **optimizing performance** and achieving correct data insights.

### 3

# Calculated Columns vs Measures

- **Calculated columns** are computed during data load and are stored in the data model, often used for intermediate calculations.
- **Measures** are dynamic calculations evaluated at query time and provide flexibility and efficiency for reporting.
- Measures generally consume **less memory** and adapt better to report interactions compared to calculated columns.
- Utilizing measures for most calculations leverages DAX's strengths and improves report performance.



The screenshot shows a software interface with a formula bar at the top and a data table below it. The formula bar contains the text "1 Sales = [Total Sales]". The data table has three columns: "Index", "Product Name", and "Column". The "Column" column is highlighted in yellow. The table contains four rows of data.

Index	Product Name	Column
1	Product 1	
2	Product 2	
3	Product 3	
4	Product 4	



# 4

# Data Types and Operators

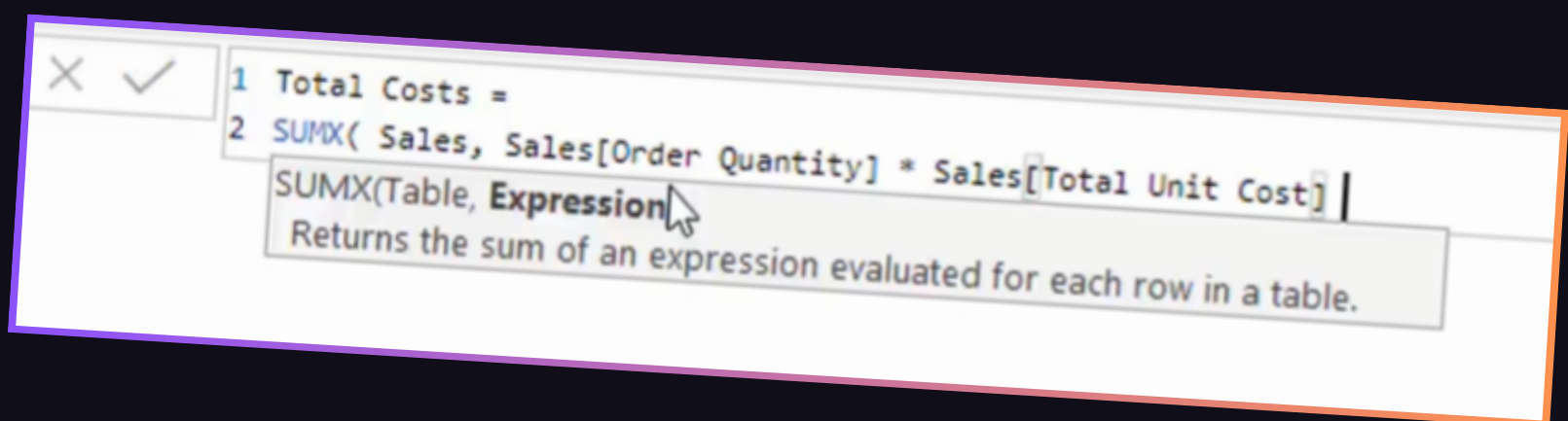
## Importance of Data Types in DAX Calculations

- Ensuring correct **data types** is crucial for accurate calculations and prevents formula errors in DAX.
- Numeric data types are necessary for performing aggregations and mathematical operations.
- Text data types are used for string manipulations and categorizing data for analysis.
- Regularly checking and adjusting data types in the **query editor** helps maintain data integrity and calculation reliability.

# 5

## Common Operators Used in DAX

- DAX employs standard mathematical operators (+, -, \*, /) for performing basic arithmetic calculations.
- Comparison operators (e.g., =, <>, >, <, >=, <=) are used in logical expressions to compare values.
- The **& operator** concatenates text strings, allowing the combination of multiple text values into one.
- Logical operators (AND, OR, NOT) enable complex conditional statements within DAX formulas.



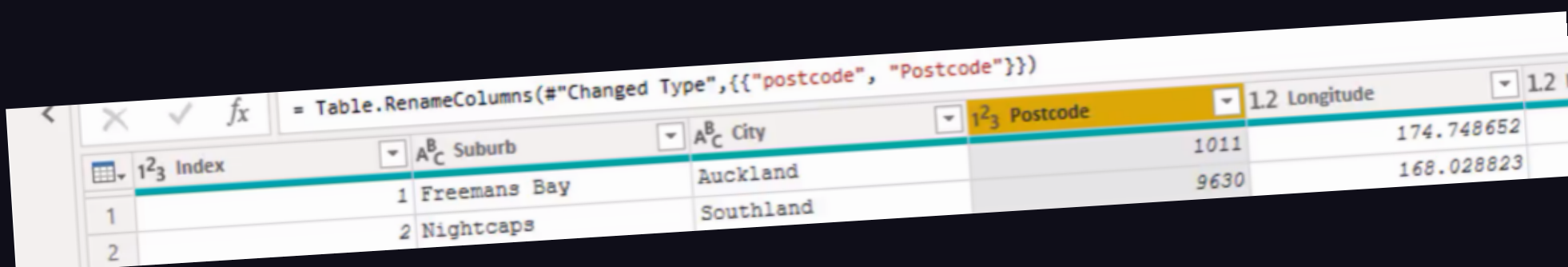


# 6

## Naming Conventions

### Best Practices for Naming Conventions in DAX

- Use clear, **descriptive names** for tables, columns, and measures to enhance readability and maintainability of DAX code.
- Avoid using underscores, abbreviations, and special characters in names to prevent confusion.
- Consistent naming conventions help make the code more understandable, especially in collaborative environments.
- Proper naming conventions future-proof reports by making them easier to update and expand over time.



The screenshot shows a DAX formula bar with the formula: `= Table.RenameColumns(#"Changed Type",{{"postcode", "Postcode"}})`. Below the formula bar is a table with the following data:

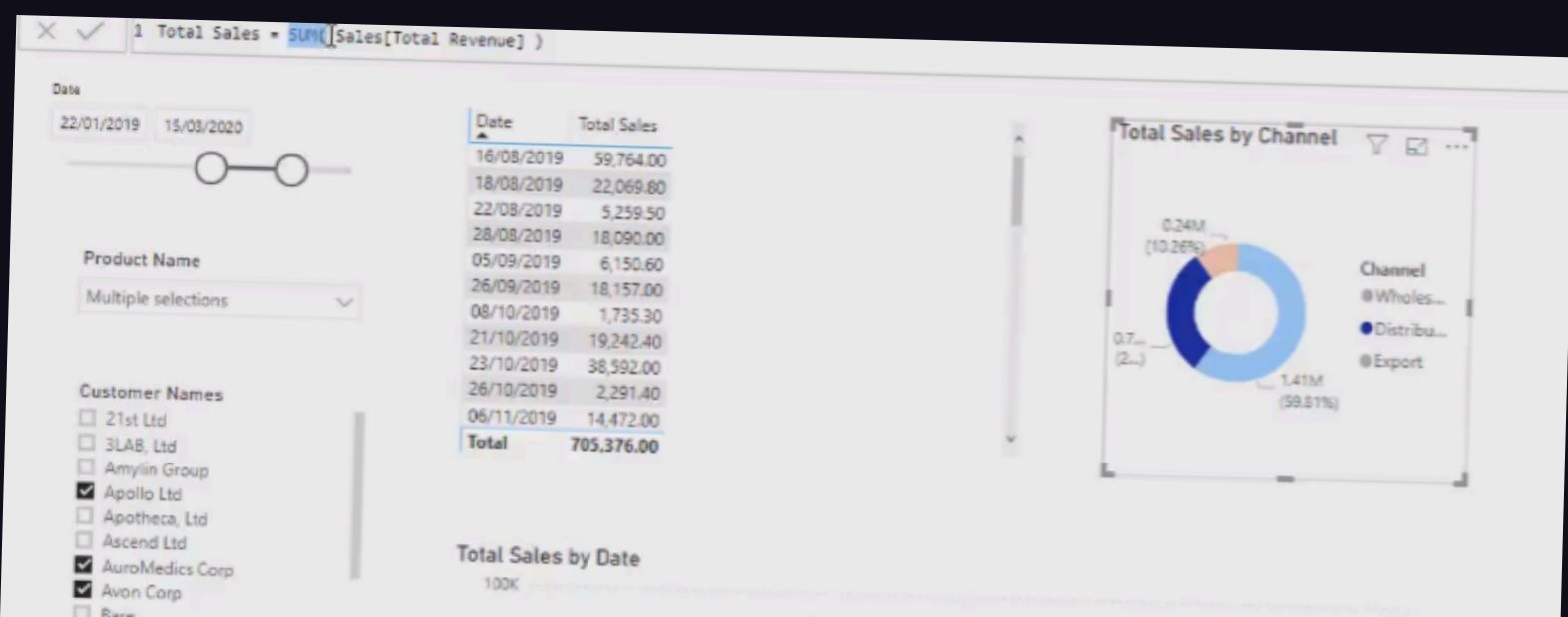
1 <sup>2</sup> Index	A <sup>B</sup> Suburb	A <sup>B</sup> City	1 <sup>2</sup> Postcode	1.2 Longitude
1	Freemans Bay	Auckland	1011	174.748652
2	Nightcaps	Southland	9630	168.028823

# 7

## Context in DAX

### Evaluation Context and Initial Context

- **Evaluation context** determines how DAX formulas calculate values based on the current filters and selections.
- **Initial context** is established by the report's filters and slicers, affecting the data that formulas operate on.
- Understanding both types of context is crucial for writing accurate and efficient DAX formulas.
- Context transitions, caused by functions like **CALCULATE**, alter the initial context to produce the desired calculations.



## Leverage CALCULATE

- The **CALCULATE function** is central for modifying filter context. It is powerful for creating dynamic measures.

## Iterators

- Functions ending in X (like SUMX) apply **row context** automatically and are necessary for more complex row-level calculations.

## Context Transition

- Understand when and how to use **context transition** to effectively manage row-level and aggregate calculations.

# 8

## Basic Aggregation Functions

SUM, AVERAGE, MIN, and MAX

- The **SUM** function adds up all the values in a column, providing a total.
- **AVERAGE** calculates the mean of a set of values within a column.
- **MIN** returns the smallest value, while **MAX** returns the largest value in a column.
- These basic aggregation functions are foundational for performing essential data analysis tasks in DAX.

# 9

## Count Functions

COUNT, COUNTA, COUNTBLANK, COUNTROWS, and DISTINCTCOUNT

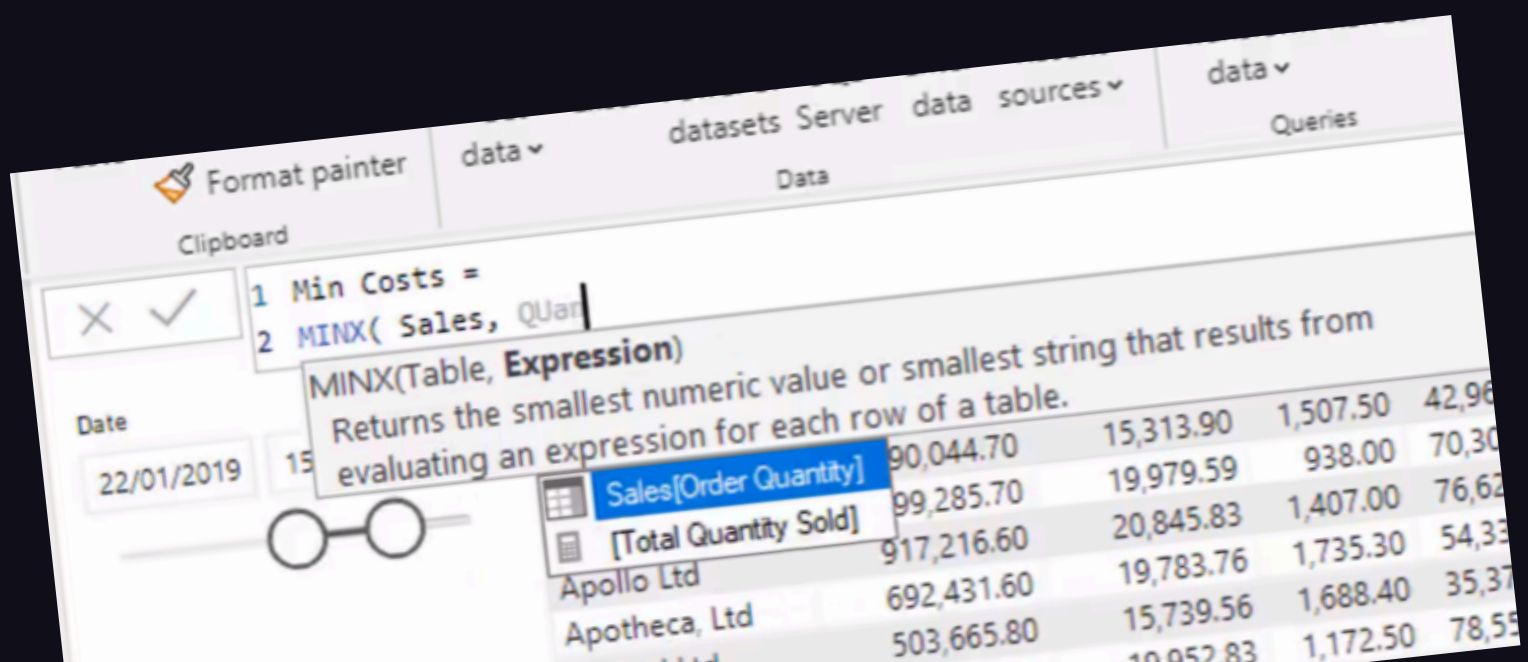
- **COUNT** counts the number of non-blank values in a column, while **COUNTA** counts all values, including blanks.
- **COUNTBLANK** specifically counts the number of blank values in a column.
- **COUNTROWS** counts the number of rows in a table, and **DISTINCTCOUNT** returns the number of unique values in a column.
- These functions are used to understand data distribution and handle missing data.

# 10

## Iterating Functions

SUMX, AVERAGEX, MINX, and MAXX

- **SUMX** iterates over a table and sums the results of an expression evaluated for each row.
- **AVERAGEX** calculates the average of an expression evaluated for each row in a table.
- **MINX** returns the smallest value, and **MAXX** returns the largest value from an expression evaluated across a table.
- Iterating functions allow for complex, row-by-row calculations, providing flexibility and precision in data analysis.



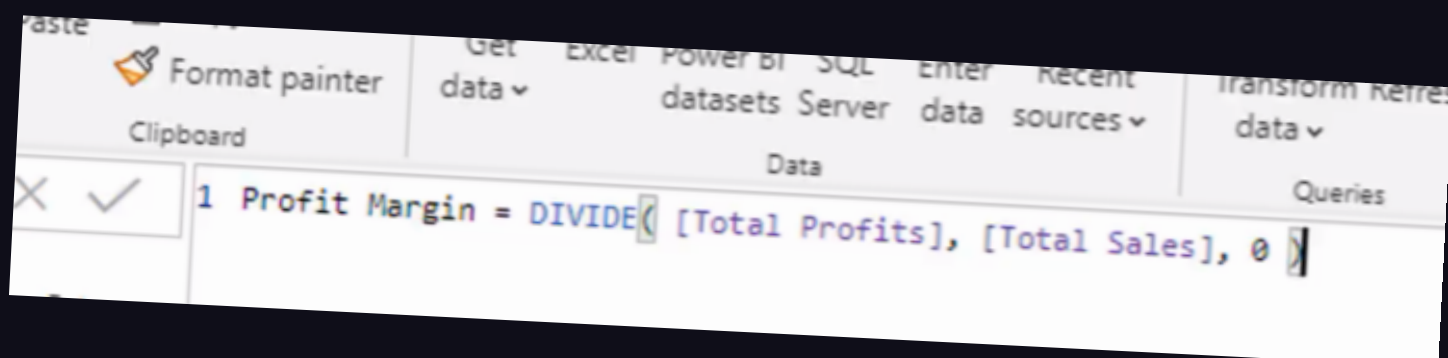


# 11

## Measure Branching

How to Branch Out from Initial Measures

- **Measure branching** involves creating new measures based on existing ones to build complex calculations.
- This technique allows for modular and reusable DAX code, improving maintainability.
- Start with simple measures and incrementally build more complex ones by referencing the initial measures.
- Measure branching enhances the **scalability and readability** of your DAX code, making it easier to manage and update.



# Data Types and Conversion

## Data Type Conversion Functions

- Functions like **VALUE**, **FORMAT**, and **CONVERT** allow you to change data types within DAX formulas.
- These functions are useful when dealing with mixed data types or when you need to format values for presentation.
- Understanding data type conversion helps in maintaining consistency and accuracy in calculations.
- Proper use of conversion functions ensures that your data is correctly interpreted and processed by DAX.

# 13

## Error Handling in DAX

Using BLANK, ISBLANK, and IFERROR

- The **BLANK** function represents missing or undefined values in DAX calculations.
- **ISBLANK** checks if a value is blank and returns TRUE if it is, otherwise FALSE.
- **IFERROR** provides a way to handle errors by returning a specified value if an error occurs in a calculation.
- Effective error handling ensures that your DAX formulas are robust and can handle unexpected data scenarios gracefully.

# 14

## Logical Functions

### IF and SWITCH Functions

- The **IF** function performs conditional checks, returning different results based on whether a condition is true or false.
- **SWITCH** evaluates an expression against multiple values, returning a corresponding result for the first match found.
- These functions enable complex conditional logic, making your DAX formulas more dynamic and adaptable.
- Use logical functions to implement decision-making processes directly within your DAX calculations.



# Division in DAX

## DIVIDE Function

- The **DIVIDE** function performs division while handling potential division-by-zero errors gracefully.
- It takes two arguments: the numerator and the denominator, with an optional argument for an alternate result if division by zero occurs.
- This function ensures your calculations remain error-free and reliable when dividing values.
- Using DIVIDE is preferred over the simple division operator (/) due to its built-in error handling capabilities.

# Time Intelligence Functions

Overview and Examples of DATEADD and Other Time Functions

- Time intelligence functions like **DATEADD** allow for calculations over date ranges, such as comparing periods or calculating year-over-year growth.
- Functions such as **SAMEPERIODLASTYEAR**, **PARALLELPERIOD**, and **TOTALYTD** provide powerful tools for time-based analysis.
- These functions help in creating dynamic time-based calculations, essential for financial and operational reporting.
- Understanding time intelligence functions is crucial for accurate trend analysis and forecasting in Power BI.



## PRO TIPS

There is a new, more efficient way to complete time intelligence to explore

Minimize the complexity of DAX expressions by using **OFFSET**, leading to faster performance and easier automation. **OFFSET** ensure a standard, predictable operation across all time intelligence calculations, reducing the potential for errors and enhancing overall model performance.

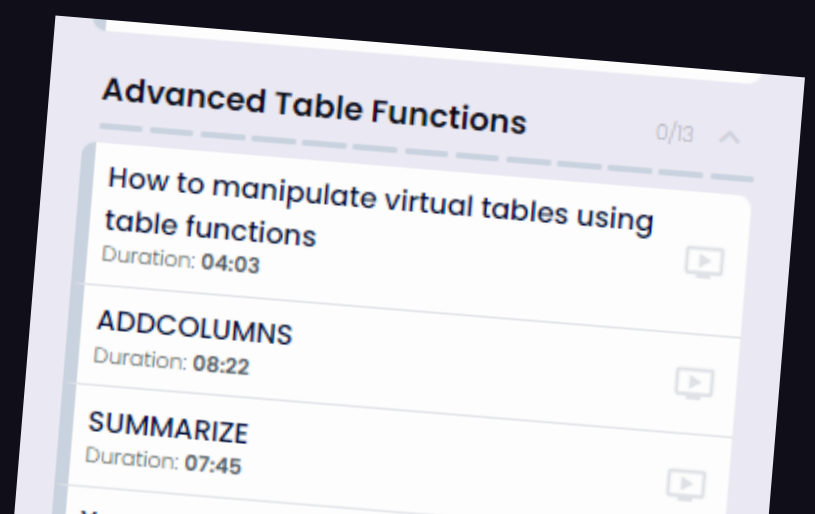
Replace complex standard time intelligence functions with **OFFSET** to manage time-related data more effectively. **OFFSET** turn a variety of time periods (like years, quarters, months) into a simple continuous series, making operations more transparent and easier to manage.



# Table Functions

Understanding and Using  
SUMMARIZECOLUMNS, GENERATE, and  
GROUPBY

- **SUMMARIZECOLUMNS** creates a summary table for the requested totals over a set of groups.
- **GENERATE** performs a cross join of two tables and evaluates an expression for each row in the first table.
- **GROUPBY** groups rows of a table by one or more columns, applying aggregation functions to each group.
- Table functions are fundamental for manipulating and summarizing data in more complex ways than standard aggregations allow



## PRO TIPS

There are many important table functions that can take your DAX up another level. Here are some other examples:

- **CALCULATETABLE** – Modifies the context of the data and returns a table based on specified filters.
- **FILTER** – Creates a subset of a table based on given conditions.
- **SUMMARIZE** – Generates a summary table grouped by specified columns.
- **CROSSJOIN** – Produces the Cartesian product of two or more tables.
- **EXCEPT** – Returns the rows from one table that are not present in another table.
- **TOPN** – Retrieves the top N rows from a table based on a specified order.

# Quick Measures

## Positives and Negatives of Using Quick Measures

- **Quick measures** allow for rapid creation of common calculations through a graphical interface without writing DAX code.
- They are useful for beginners or for quickly prototyping calculations.
- However, they can generate overly complex DAX code that may be inefficient or difficult to understand.
- Relying too much on quick measures can hinder learning and mastering DAX fundamentals

# Data Model and Relationships

Setting Up a Data Model and the Importance of Relationships

- A well-designed **data model** is essential for DAX to work effectively, enabling accurate and efficient calculations.
- Establishing relationships between tables allows for seamless data integration and context propagation.
- Best practices include using star schema designs and ensuring proper cardinality (one-to-many or many-to-one relationships).
- Correctly setting up your data model simplifies your DAX formulas and enhances performance

# CALCULATE Function

## Importance and Use of the CALCULATE Function

- The **CALCULATE** function changes the context in which a DAX expression is evaluated, making it one of the most powerful functions in DAX.
- It allows you to modify filter contexts to perform complex calculations, such as conditional aggregations and dynamic measures.
- Understanding how to use **CALCULATE** is critical for advanced data analysis and solving complex business scenarios.
- It integrates seamlessly with other DAX functions, expanding their capabilities and application



# Filter Context

Using FILTER and KEEPFILTERS Functions

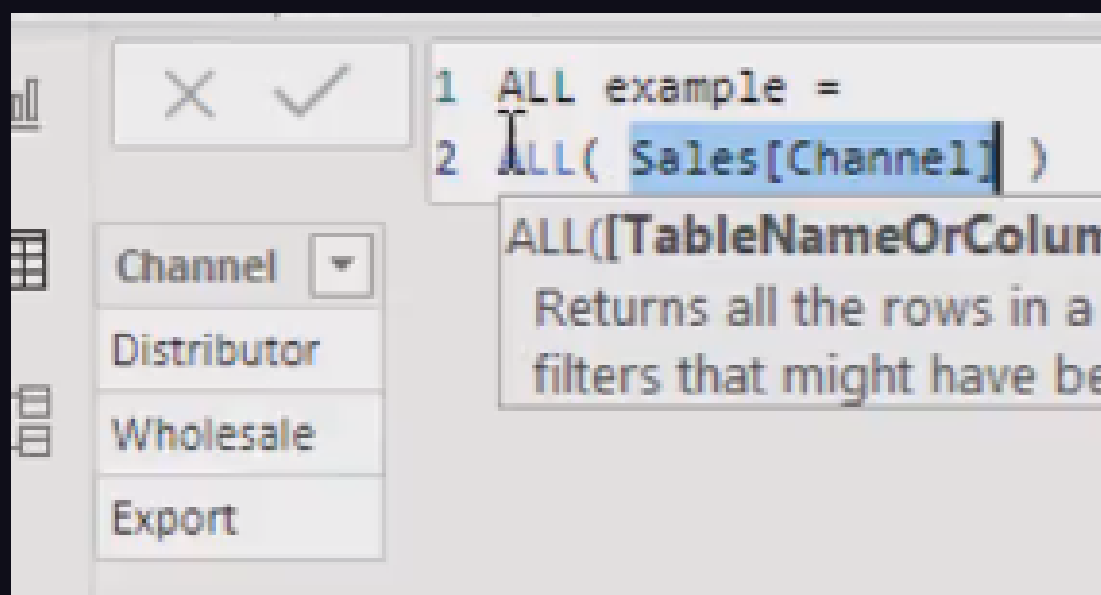
- The **FILTER** function creates a table that includes only rows that meet specified criteria, essential for applying complex filters.
- **KEEPFILTERS** modifies the filter context without removing existing filters, providing precise control over filter application.
- These functions are critical for creating accurate and detailed reports by managing how data is filtered and displayed.
- Mastery of filter functions is necessary for developing nuanced and targeted data insights.

# 22

## ALL Functions

Understanding ALL, ALLEXCEPT, and ALLSELECTED

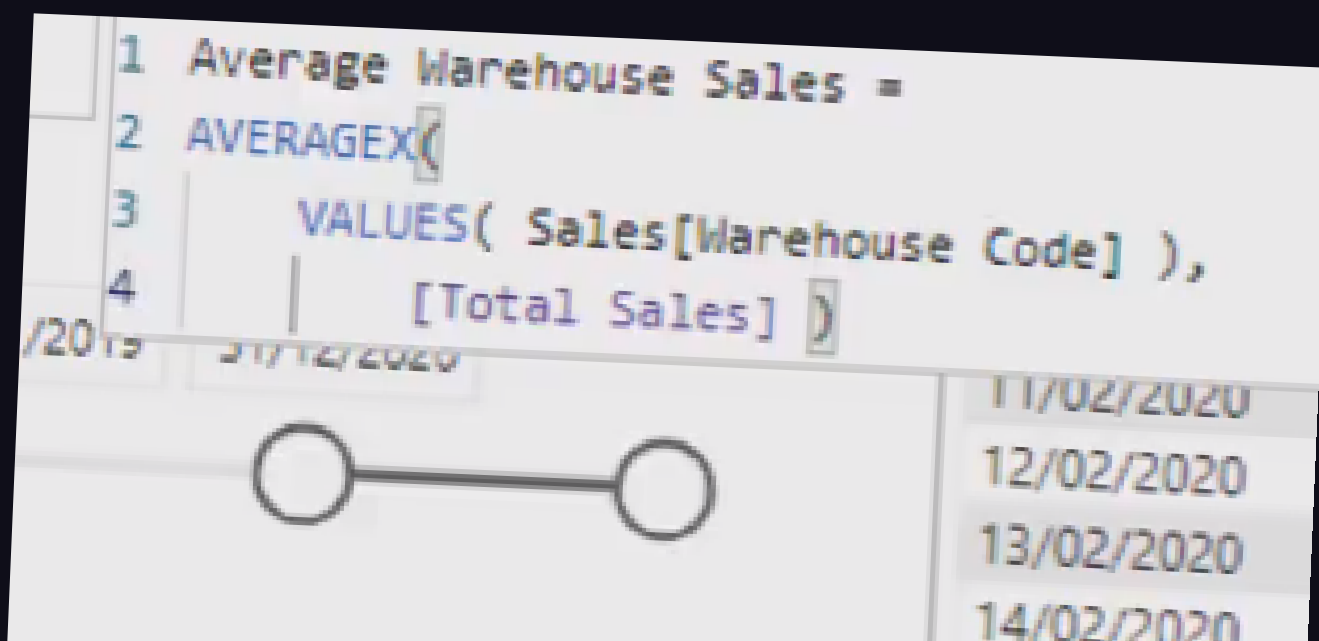
- **ALL** removes all filters from a table or column, useful for calculating totals and ignoring current context.
- **ALLEXCEPT** removes filters from all columns except the specified ones, offering selective filtering control.
- **ALLSELECTED** retains filters applied by the user while removing filters applied by visuals, useful for dynamic reporting.
- These functions are crucial for managing filter context and achieving precise, context-aware calculations.



# VALUES Function

## How to Use the VALUES Function

- The **VALUES** function returns a one-column table with distinct values from a specified column.
- It is useful for creating dynamic slicers and understanding unique entries within a column.
- **VALUES** can also be used to detect and handle many-to-many relationships by providing distinct values.
- This function plays a key role in conditional logic and dynamic reporting scenarios.



# SELECTEDVALUE

## Function

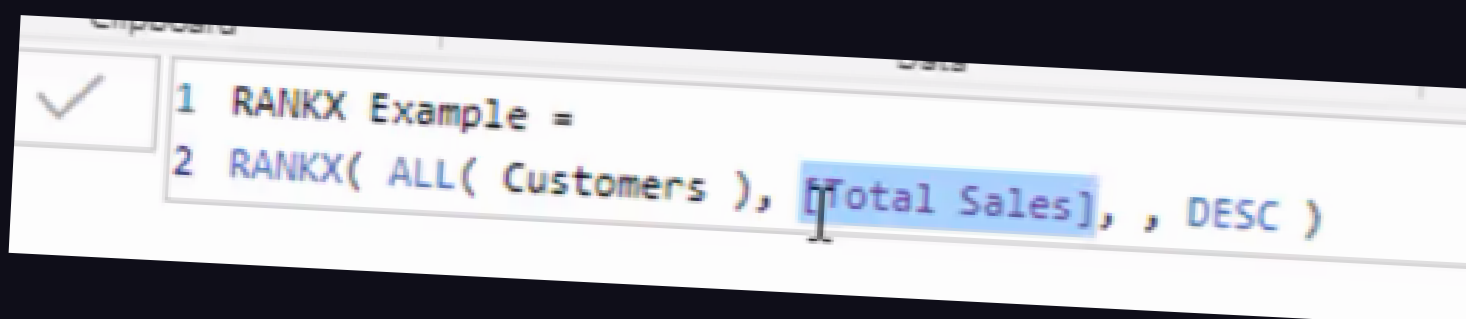
Differences Between VALUES and SELECTEDVALUE

- **SELECTEDVALUE** returns the value of a column if there's only one value in the current context, otherwise, it returns a specified alternate result.
- It simplifies scenarios where you need to get a single value from a column and handle cases with multiple values gracefully.
- This function is particularly useful for creating dynamic titles and labels in reports.
- **SELECTEDVALUE** is preferred over VALUES when you need a single value or a default result in case of multiple values.

# RANKX Function

Using RANKX for Ranking Data

- **RANKX** ranks items in a table based on the values in a specified expression.
- It is useful for creating rankings, such as top salespeople or best-performing products.
- The function can be customized to handle ties and ranking order (ascending or descending).
- Understanding RANKX helps in performing detailed competitive and performance analysis.

A screenshot of a code editor window showing two lines of SQL code. The first line is a comment: "1 RANKX Example =". The second line is the function call: "2 RANKX( ALL( Customers ), [Total Sales], , DESC )". The text "[Total Sales]" is highlighted in blue. A checkmark icon is visible in the top left corner of the code editor.

```
1 RANKX Example =  
2 RANKX( ALL( Customers ), [Total Sales], , DESC )
```

# TOPN Function

Differences Between RANKX and TOPN

- **TOPN** returns the top N rows of a table based on the values in a specified expression.
- It is useful for creating dynamic top lists, such as top 10 products or customers.
- Unlike **RANKX**, **TOPN** directly filters the table to the top N results, making it useful for focused analysis.
- **TOPN** can be combined with other functions to create advanced and interactive reports.



# Measure Groups

## Creating and Organizing Measure Groups

- **Measure groups** help organize and categorize measures within a Power BI model, improving manageability.
- Grouping measures makes it easier to navigate and maintain complex models with many calculations.
- Best practices include logically grouping related measures, such as sales measures or financial metrics.
- Measure groups enhance productivity and collaboration by keeping the model well-structured.

# Formatting DAX Code

Best Practices for Formatting DAX Formulas

- **Proper formatting** improves the readability and maintainability of DAX code, making it easier to understand and debug.
- Use indentation and line breaks to separate different parts of the formula clearly.
- Add **comments** to explain complex logic and document your thought process.
- Consistent formatting practices help in collaborative environments and future-proof your reports.

# Using Variables in DAX

## Simplifying Formulas with Variables

- **Variables in DAX**, declared using the VAR keyword, store intermediate calculations and make complex formulas easier to read.
- They improve performance by reducing redundant calculations within a formula.
- **Variables** enhance the clarity of DAX code by breaking down complex logic into manageable parts.
- Using variables allows for better debugging and maintenance of DAX formulas.

```
1 High Margin, Export Quantities Sold =  
2 VAR HighMarginProducts = FILTER( Products, [Profit Margin] > 0.32 ) //This is our high  
3 VAR ExportChannel = FILTER( Sales, Sales[Channel] = "Export" )  
4  
5 RETURN  
6 CALCULATE( [Total Quantity Sold], HighMarginProducts, ExportChannel )
```

# Writing Comments

Adding Comments to DAX Code for Clarity

- Comments, added with double forward slashes (`//`), help explain the purpose and logic of DAX formulas.
- They make the code easier to understand for others and for future reference.
- Well-commented code aids in troubleshooting and maintaining formulas.
- Including comments is a best practice, especially in complex DAX expressions, to document your thought process.

# Context Transition

Transitioning from Row Context to Filter Context

- **Context transition** occurs when functions like **CALCULATE** change the row context to a filter context.
- Understanding context transition is crucial for writing accurate DAX formulas, particularly in measures.
- It allows for complex calculations that aggregate data based on specific conditions.
- Mastering **context transition** helps in creating dynamic and responsive DAX calculations.

# Filter Propagation

How Filters Propagate Through Relationships in a Data Model

- Filters applied to one table can propagate to related tables through established relationships.
- Properly setting up relationships ensures that filters work as expected, enabling accurate data analysis.
- Understanding **filter propagation** helps in debugging and optimizing your data model.
- Effective use of **filter propagation** allows for complex, multi-table analyses within Power BI.

# Handling Totals

## Fixing Errors in Totals and Common Issues

- Errors in totals often arise from incorrect context or filter propagation in DAX formulas.
- Techniques such as using **CALCULATE** or adjusting filter contexts can resolve these issues.
- Ensuring consistent context across detailed and total rows is key to accurate calculations.
- Properly handling totals ensures that your reports display correct and reliable information.



# Virtual Tables

## Creating and Using Virtual Tables with Table Functions

- Virtual tables in DAX are created using functions like **SUMMARIZE**, **ADDCOLUMNS**, and **FILTER**, allowing for complex data manipulations without modifying the underlying data.
- They are essential for performing advanced calculations and creating dynamic, context-specific data sets.
- **Virtual tables** can be used within measures to aggregate and filter data in sophisticated ways.
- Understanding how to create and use virtual tables enhances the flexibility and power of your DAX formulas.

# Scenario Analysis

Applying DAX in Different Analytical Scenarios

- **Scenario analysis** involves using DAX to model and compare different business scenarios, such as budget vs. actual performance.
- Functions like **SWITCH** and **IF** can be used to create dynamic scenarios based on user inputs or conditions.
- This analysis helps in decision-making by providing insights into potential outcomes and trends.
- Mastery of scenario analysis in DAX enables the creation of versatile and interactive reports that adapt to various business needs.

# Advanced Analytics

Techniques for Advanced Calculations in Power BI

- Advanced DAX techniques include the use of complex functions, such as RANKX, TOPN, and advanced time intelligence functions.
- **Combining multiple DAX functions** allows for deep insights and sophisticated analyses, such as churn analysis and customer segmentation.
- Advanced calculations often involve iterative functions and context manipulation to achieve precise results.
- Proficiency in advanced analytics enables the development of high-value, data-driven solutions for complex business problems.

# Best Practices for DAX

General Best Practices for Using DAX Effectively

- Following best practices such as using variables, writing clean and readable code, and optimizing performance is essential.
- Regularly reviewing and refactoring your DAX formulas helps maintain code quality.
- Documenting your DAX code with comments and adhering to naming conventions ensures maintainability.
- Implementing best practices enhances the efficiency and reliability of your DAX solutions.

# Combining DAX Functions

Using Combinations of Functions for Complex Analysis

- Combining multiple DAX functions enables the creation of sophisticated and nuanced calculations.
- Examples include nesting functions like **CALCULATE**, **FILTER**, and **SUMX** to achieve specific results.
- Understanding how different functions interact allows for more flexible and powerful DAX formulas.
- Combining functions effectively can address complex analytical needs and provide deeper insights.

```
1 COUNTRONS w/virtual tables =  
2 COUNTRONS(  
3     CALCULATETABLE( Sales,  
4     | TOPN( 5, ALL( Products ), [Total Sales], DESC ) ) )
```

Date	Total Transaction
------	-------------------

# Evaluating DAX Formulas

Techniques for Auditing and Validating DAX Formulas

- Regular auditing and validation of DAX formulas ensure their accuracy and performance.
- Techniques include using the Performance Analyzer in Power BI and step-by-step debugging.
- Documenting assumptions and testing formulas with sample data helps identify issues.
- Effective evaluation practices maintain the integrity and reliability of your DAX calculations.

# Context in Calculated Columns

Specific Considerations for Context in  
Calculated Columns

- **Calculated columns operate in row context**, affecting how formulas are evaluated.
- Understanding the differences in context between calculated columns and measures is crucial.
- **Calculated columns** can impact performance, so use them judiciously.
- Context considerations ensure that calculated columns provide accurate and expected results.



# CALCULATE and Context

## Detailed Use of CALCULATE in Changing Context

- The **CALCULATE function modifies filter context**, enabling complex calculations.
- It allows for dynamic adjustments to the data being analyzed, such as conditional aggregations.
- Mastering **CALCULATE** is essential for advanced data modeling and analysis in DAX.
- Understanding its impact on context is key to leveraging its full potential.



# ENTERPRISE DNA

## Check out 'Mastering DAX Calculations' to learn more

**Mastering DAX Calculations**

Level: Intermediate Total points: 1390 Duration: 12:17:13

Using variables to simplify formulas

EDNA AI Workout Ask & Learn

Untitled - Power BI Desktop

File Home Insert Modeling View Help Format Data / Drill Table tools Measure tools

Undo Paste Copy Format painter Get data Excel Power BI datasets SQL Enter data Recent sources Transform Refresh data New visual Text box More visuals New measure Quick measure measure Publish

1 Formula Example =  
2 VAR CustomerMin = 5  
3 VAR CustomerMax = 20  
4 VAR CustomerChannel =  
5  
6 CALCULATE( [Total Quantity],  
7 FILTER( Sales, Sales[Customer Name Index] > 5 && Sales[Customer Name Index] < 20 ),  
8 FILTER( Sales, Sales[Channel] = "Wholesale" ) )

AXW291	3756	579
FLR025	816	128
GUT930	1850	266
NXH382	1569	260
Total	7991	1233

Autoplay Skip lead-in

Bookmarks 2

AI Toolkit

Using variables to simplify formulas

Mastering DAX Calculations 0/4

Course Overview  
Course Index  
Let's get started  
Utilizing EDNA AI Ask & Learn

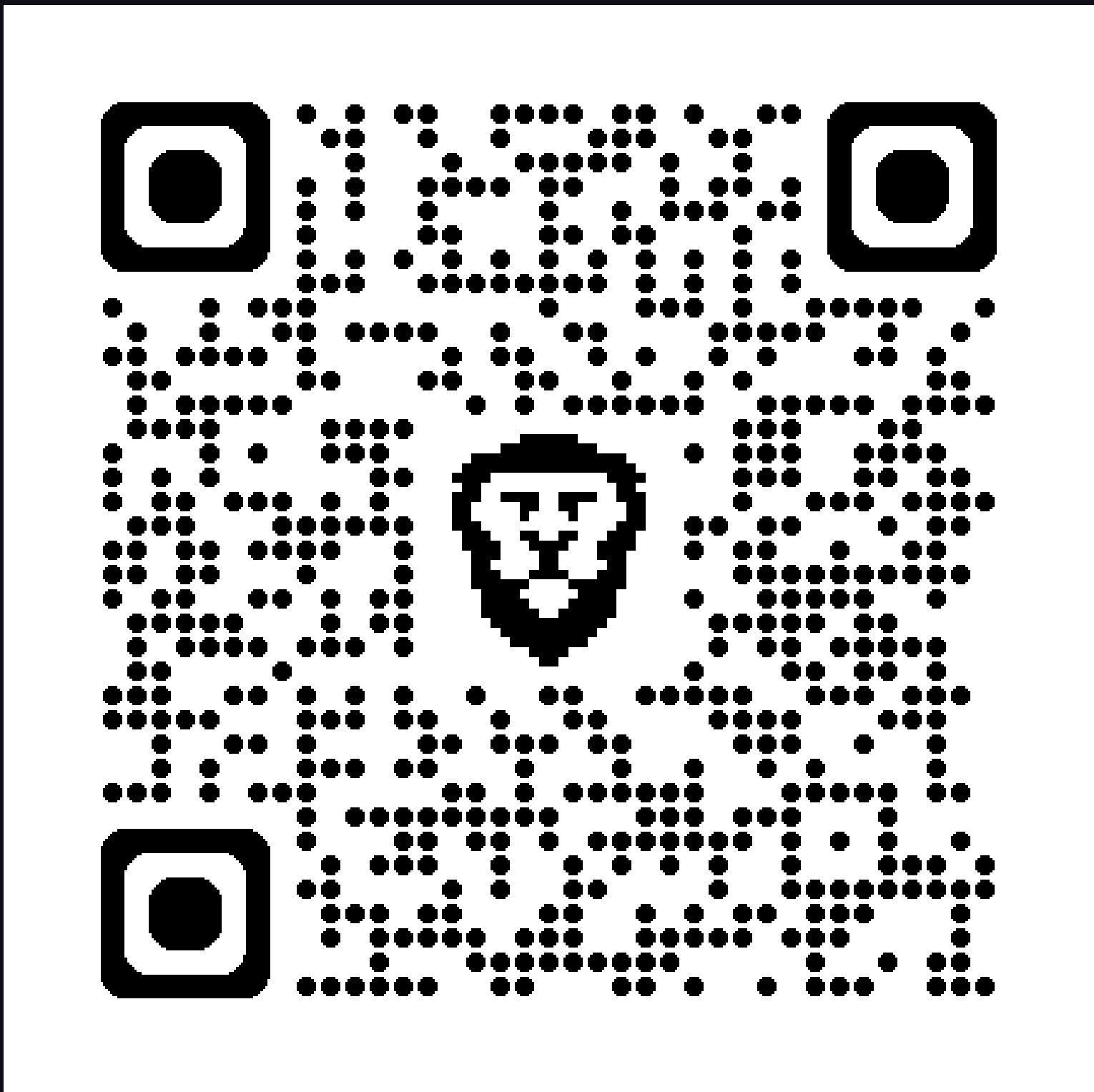
Introduction to DAX 0/4

Course resource pack download  
Guide to Mastering DAX in Power BI  
What is DAX & How to Learn to Master It? Duration: 09:18  
Data types and operators Duration: 05:18  
The importance of naming conventions Duration: 06:06  
Formula syntax Duration: 04:52  
Measure groups Duration: 04:11  
Formatting DAX code Duration: 06:54  
Using variables to simplify formulas Duration: 06:00  
Writing comments in your code Duration: 03:13  
Quick measure feature (positives vs negatives) Duration: 07:54  
DAX doesn't work unless your data model and relationships are set up correctly Duration: 07:34  
The only way to learn is to practice Duration: 03:55  
Let's Review

Course now available on **EDNA Learn**

\*This resource content was created from this comprehensive course within Enterprise DNA's learning platform EDNA Learn


This is fancy QR code. Give it a try.




# For an further deep dives into DAX topics check out

## Languages Advisor





### Languages Advisor



Get specialized AI advice on any tech language 

★★★★★



[Check out more AI Tools!](#) [Leave us feedback](#)

Advise on 

DAX

Can you give me a range of simple DAX functions that I can use  

Stop

GPT-3.5 Turbo | **GPT 4o** | GPT 4 Turbo | Gemini Pro  

#### Simple DAX Functions for Data Analysis

To enhance your data analysis capabilities with DAX (Data Analysis Expressions), here are some fundamental functions that you can apply to various scenarios. This guide covers basic aggregation, filtering, logical, text, and time intelligence functions to provide a solid foundation.

##### Aggregation Functions

Aggregation functions are essential for summarizing data in your Power BI reports.

- SUM: Calculates the sum of a column.

Total Sales = SUM(Sales[Amount])
- AVERAGE: Returns the average of a numeric column.

Average Sales = AVERAGE(Sales[Amount])
- COUNT: Counts the number of rows in a table.

Count of Sales = COUNT(Sales[Amount])
- COUNTROWS: Returns the number of rows in a table.

Get involved in the  
conversation



**LinkedIn**



**X (Twitter)**