**Section A: Creating a Calculated Column**

**🔍 Objective:**

Create a **calculated column** named Late Shipment that identifies whether an order was shipped after its due date. Label it as either **"Late Shipment"** or **"On Time"**.

**🔧 Steps to Follow:**

**✅ Step 1: Open Data View**

* Launch Power BI Desktop.
* Open your .pbix file.
* Click on the **Data view** (table icon on the left pane).

**✅ Step 2: Select the Table**

* From the Fields pane, click on the table that contains ShipDate and DueDate (e.g., FactInternetSales).

**✅ Step 3: Add a Calculated Column**

* Go to the **Table Tools** or **Modeling** tab in the top ribbon.
* Click on **"New column"**.

**✅ Step 4: Write the DAX Formula**

Paste the following formula in the formula bar:

Late Shipment =

IF (

'FactInternetSales'[ShipDate] > 'FactInternetSales'[DueDate],

"Late Shipment",

"On Time"

)

**✅ Step 5: Validate the Column**

* Press **Enter**.
* Verify that the column is correctly populated with "Late Shipment" and "On Time".
* Optional: Click on the dropdown filter to inspect distribution.

**➕ Bonus Practice**

**▶ Create Another Column for Purchase Category:**

If Sales amount is greater than 500 then purchase is "Premium Purchase" or else "Standard Purchase"

**▶ Visualize Your Work:**

* Add a **Table visual** with:
  + OrderDate, ShipDate, DueDate, Late Shipment, Purchase Category
* Add a **Slicer** on Late Shipment to analyze on-time vs late orders.

**🤔 Key Concepts Recap:**

* **Calculated Columns**:
  + Stored in Power BI's data model.
  + Calculated at **refresh time**.
  + Use DAX just like Measures but are materialized into memory.
* **Difference from Measures**:
  + Measures are computed **on the fly** in visuals.
  + Measures do not consume memory unless used in a visual.

**SECTION B: Power BI Practice – Building a Calendar Table Using DAX**

**🔍 Objective:**

Create a dynamic **Calendar Table** using CALENDARAUTO() in DAX, enrich it with additional attributes (year, month, month name, and month-year), and prepare it for time intelligence analysis.

**🔧 Steps to Follow:**

**✅ Step 1: Create Calendar Table**

* Go to the **Modeling** tab.
* Click on **New Table**.
* Enter the following DAX code:

Calendar Table =

ADDCOLUMNS(

CALENDARAUTO(),

"Year", YEAR([Date]), [Here create other columns like Month, Days etc ]

)

**✅ Step 2: Understand CALENDARAUTO()**

* It scans all date columns in your model.
* Finds the **earliest and latest** date.
* Returns a full continuous date range.
* Accepts a number argument to define fiscal year start:
  + CALENDARAUTO(3) → Fiscal year starts in **April**.

**✅ Step 3: Sort Month Name**

* Select **Month Name** column.
* Go to **Column tools** → **Sort by Column** → choose **Month**.

**✅ Step 4: Mark as Date Table**

* Click the three dots ... next to the Calendar Table.
* Choose **Mark as Date Table**.
* Select the **Date** column.

**✅ Step 5: Build Relationship**

* Open **Model view**.
* Connect Calendar Table[Date] → FactInternetSales[OrderDate] (or similar).

**📆 Why This Is Important**

* Enables **Time Intelligence Functions** like:
  + YTD, QTD, MTD
  + YOY, MOM, Rolling averages
* Ensures **continuous date values** for accurate analysis
* Allows **custom fiscal calendars**

**✅ Step 1: Create Calendar Table**

* Go to the **Modeling** tab.
* Click on **New Table**.
* Enter the following DAX code:

Calendar Table =

ADDCOLUMNS(

CALENDARAUTO(),

"Year", YEAR([Date]),

"Month", MONTH([Date]),

"Month Name", FORMAT([Date], "MMMM"),

"Month Year", FORMAT([Date], "MMM/yyyy")

)

**✅ Step 2: Understand CALENDARAUTO()**

* It scans all date columns in your model.
* Finds the **earliest and latest** date.
* Returns a full continuous date range.
* Accepts a number argument to define fiscal year start:
  + CALENDARAUTO(3) → Fiscal year starts in **April**.

**✅ Step 3: Sort Month Name**

* Select **Month Name** column.
* Go to **Column tools** → **Sort by Column** → choose **Month**.

**✅ Step 4: Mark as Date Table**

* Click the three dots ... next to the Calendar Table.
* Choose **Mark as Date Table**.
* Select the **Date** column.

**✅ Step 5: Build Relationship**

* Open **Model view**.
* Connect Calendar Table[Date] → FactInternetSales[OrderDate] (or similar).

**📆 Why This Is Important**

* Enables **Time Intelligence Functions** like:
  + YTD, QTD, MTD
  + YOY, MOM, Rolling averages
* Ensures **continuous date values** for accurate analysis
* Allows **custom fiscal calendars**

**SECTION C: Power BI Practice – Creating Basic Measures Using DAX**

**🔍 Objective:**

Learn how to build basic **measures** using DAX to perform aggregations like total sales and count of transactions.

**🔧 Steps to Follow:**

**✅ Step 1: Create Total Sales Amount Measure**

* Go to the **Modeling** tab.
* Click on **New Measure**.
* Enter the following DAX formula:

Total Sales Amount = [DAX Formula]

Press **Enter** or click the check mark to validate the measure.

**✅ Step 2: Create a Measure to Count Orders**

There are two ways to count sales:

**Option A: Count Rows**

Number of Sales = ???

**Option B: Count Specific Column**

Number of Sales = ???

* Use either method depending on whether your data structure requires row counting or unique identifier counting.

**✅ Step 3: Add Measures to Report**

* Add **Total Sales Amount** and **Number of Sales** to a Card or Table visual.
* Format the visuals using standard Power BI features.

**✅ Step 4: Use Comments in DAX**

* **Single-line comment**:

-- This is a comment

* **Multi-line comment**:

/\*

This is a multi-line

comment in DAX

\*/

Use comments to explain your logic or to temporarily disable code.

**🤔 Key Concepts Recap:**

* **Measures**:
  + Are not stored in memory until used.
  + Calculated **on the fly** when used in visuals.
  + Use DAX expressions and support complex logic.
* **Best Practices**:
  + Keep naming clear (e.g., Total Sales Amount instead of SumSales)
  + Store measures in meaningful tables or create a **Measures table** for organization.

**🔍 Objective:**

Organize your DAX measures more efficiently by creating a **dedicated Measures Table**, ensuring a clean, readable, and professional Power BI data model.

**🔧 Steps to Follow:**

**✅ Step 1: Create a Measures Table**

* Go to the **Home** tab.
* Click **Enter Data**.
* In the dialog box:
  + Rename the table to #Measures (include the # to sort it at the top).
  + You don’t need to enter any data.
* Click **Load**.

💡 *Why #Measures?*  
The # prefix ensures the table appears at the **top** of the Fields list in Power BI for quick access.

**✅ Step 2: Move Existing Measures**

* Select each measure (e.g., Total Sales Amount, Number of Sales).
* In the **Measure Tools** tab, find the **Home Table** dropdown.
* Change it to #Measures.

**✅ Step 3: Remove Unused Placeholder Column**

* Expand the #Measures table.
* Right-click on the default column (usually Column1).
* Choose **Delete from model**.

🧠 *After deletion*, the table icon changes to a **calculator icon** (🔢), indicating it's a **measure-only table**.

**🤔 Why Use a Measures Table?**

| **Benefit** | **Explanation** |
| --- | --- |
| ✅ Cleaner Model | Prevents measures from being scattered across unrelated tables. |
| ✅ Easier Navigation | All logic-related calculations are in one place. |
| ✅ Professional Practice | Aligns with industry best practices for maintainability and collaboration. |
| ✅ Faster Development | Quickly find and reuse measures in your development workflow. |

**📌 Best Practice Tip:**

When creating **new measures**, always ensure the **Home Table** is set to #Measures. This keeps your model consistent and avoids confusion.

**SECTION E: Power BI Practice – Understanding the CALCULATE Function**

**🔍 Objective:**

Learn how to use the powerful CALCULATE() function in DAX to **modify filter context** and create dynamic measures.

**📘 What Is CALCULATE?**

* CALCULATE() is **the only DAX function** that can modify filter context.
* Syntax:

DAX

CopyEdit

CALCULATE(<expression>, <filter1>, <filter2>, ...)

* First argument: a valid DAX expression (like a SUM or a Measure).
* Remaining arguments: filters that modify the context in which the expression is evaluated.

**🧪 Example: Total Sales for Blue Products**

**✅ Step 1: Load the Product Table**

* Go to **Home** → **Transform Data** → Power Query.
* Re-import or enable the Product table from your source.
* Ensure it has a **ProductKey**.
* In FactInternetSales, make sure ProductKey is present.

**✅ Step 2: Create Relationship**

* Go to **Model view**.
* Create a relationship:
  + Product[ProductKey] → FactInternetSales[ProductKey]
* This enables filtering between tables.

**✅ Step 3: Create a Measure Using CALCULATE()**

* Go to **Modeling** → **New Measure**.
* Enter this formula:

DAX

CopyEdit

Total Sales Amount (Blue) =

CALCULATE(

[Total Sales Amount],

'Product'[Color] = "Blue"

)

**✅ Step 4: Visualize Results**

* Add a **Table visual** with:
  + Product[Color]
  + [Total Sales Amount]
  + [Total Sales Amount (Blue)]

**✅ Step 5: Understand the Outcome**

* [Total Sales Amount] shows the sales grouped by actual color.
* [Total Sales Amount (Blue)] overrides each row’s context to show **only blue** sales.

**⚠️ Caution**

* CALCULATE() can override existing filters.
* Without care, it may produce misleading results (e.g., all rows showing the same value).
* For precise control, combine it with FILTER() (covered in next section).

**🧠 Why This Is Important**

* CALCULATE() is the **gateway to advanced DAX**.
* Enables scenarios like:
  + Sales this year vs. last year
  + Sales for specific regions
  + Conditional aggregations

**SECTION F: Power BI Practice – Using FILTER and KEEPFILTERS with CALCULATE**

### 🔍 Objective:

Understand how to control and refine filter behavior using FILTER() and KEEPFILTERS() within CALCULATE().

### 🔧 Key Concepts:

* CALCULATE() overrides default filter context.
* To **limit or preserve** that override, use FILTER() or KEEPFILTERS().

### 🧪 Step-by-Step Examples:

#### ✅ 1. Using FILTER() inside CALCULATE

Total Sales Amount Using FILTER =

CALCULATE(

[Total Sales Amount],

FILTER(

'Product',

'Product'[Color] = "Blue"

)

)

* This filters the **entire Product table** to only include Blue.
* Only Blue color retains value; all others return blank.

#### ✅ 2. Using KEEPFILTERS() inside CALCULATE

Total Sales Amount KeepFilters =

CALCULATE(

[Total Sales Amount],

KEEPFILTERS('Product'[Color] = "Blue")

)

* Retains existing filters and applies the Blue color condition only where relevant.
* Preferred for **performance** and **filter context preservation**.

### 🔎 When to Use FILTER() vs. KEEPFILTERS()

* Use **FILTER()**:
  + When comparing **measures** inside CALCULATE().
  + When filtering based on **expressions** (e.g., Sales > 100).
* Use **KEEPFILTERS()**:
  + When you want to **retain existing filters**.
  + More efficient and lightweight than FILTER().

### ✅ 3. Advanced Use Case: Filter by Measure (e.g., SalesAmount > 100)

Orders > 100 =

CALCULATE(

[Number of Sales],

FILTER(

'FactInternetSales',

'FactInternetSales'[SalesAmount] > 100

)

)

* Cannot use [Total Sales Amount] > 100 directly (Power BI does not allow filtering on a measure).

### ✅ 4. Challenge Example: Late Orders Value

Late Orders Value =

CALCULATE(

[Total Sales Amount],

???)

)

Number of Late Orders =

CALCULATE(

???

)

### 🧠 Summary

* CALCULATE() alters context—powerful but requires caution.
* FILTER() creates a new filter context → heavy but flexible.
* KEEPFILTERS() keeps the existing filters → efficient and preferred.
* Choose based on use case and performance considerations.

**SECTION G: Power BI Practice – Understanding Iterator Functions (e.g., SUMX)**

### 🔍 Objective:

Learn how to use iterator functions like SUMX to perform row-by-row calculations and aggregate results efficiently.

### 🔧 Key Concepts:

* Iterator functions perform row-wise operations and then aggregate the result.
* Common iterator functions: SUMX, COUNTX, AVERAGEX, etc.

### 🧪 Step-by-Step Example Using SUMX

#### ✅ Step 1: Create a New Measure

Iterate SUMX =

SUMX(

'FactInternetSales',

'FactInternetSales'[SalesAmount] \* 2

)

* This multiplies each row’s SalesAmount by 2, then sums the results.

#### ✅ Step 2: Add to a Visual

* Create a **Table visual**.
* Add Iterate SUMX next to Total Sales Amount to compare.
* Confirm that the value is exactly double.

### 🤔 Why Use Iterator Functions?

* Enable **custom row-level logic** in aggregations.
* Useful when you need calculations like:
  + Multiplying two columns.
  + Conditional aggregation.
  + Dynamic logic across rows.

**SECTION H: Power BI Practice – Using RELATED and RELATEDTABLE Functions**

### 🔍 Objective:

Learn how to retrieve related data from other tables using RELATED() and RELATEDTABLE() functions, and apply them in calculated columns and iterator functions.

### 🧪 Example 1: Create a Related Column for Age Buckets

Age Bucket = RELATED('DimCustomer'[Age Bucket])

* Ensure there's an active relationship.
* RELATED must be used on the many-side of the relationship.

### 🧪 Example 2: Use RELATED Inside an Iterator (SUMX)

Sales for Age 50+ =

SUMX(

FILTER(

'FactInternetSales',

RELATED('DimCustomer'[Age Bucket]) = "50+"

),

'FactInternetSales'[SalesAmount]

)

### 🧪 Example 3: Use RELATEDTABLE with COUNTX

Sales Count per Customer =

COUNTX(

RELATEDTABLE('FactInternetSales'),

'FactInternetSales'[SalesOrderNumber]

)

### 🧠 Summary:

* RELATED() works like Excel’s VLOOKUP — fetches related value for each row.
* RELATEDTABLE() returns a **table** of related rows — must be used inside an iterator.
* Both require **active relationships** to function correctly.
* Prefer RELATED for simple lookups, and RELATEDTABLE for aggregation-based use cases.

**SECTION I: Power BI Practice – Advanced Filter Removal Using ALL and REMOVEFILTERS**

### 🔍 Objective:

Learn to override or ignore filters using ALL() and REMOVEFILTERS() inside DAX calculations.

### 🔧 Key Concepts:

* ALL() removes filters from one or more columns/tables and returns all rows.
* REMOVEFILTERS() is more readable and intended to replace ALL() in most filter-clearing use cases.

### 🧪 Example 1: Use ALL() to Calculate Overall Total

Total Sales Overall =

CALCULATE(

[Total Sales Amount],

ALL('Product'[Color])

)

* Use case: Show the same total across all product colors regardless of filter selection.

### 🧪 Example 2: Use REMOVEFILTERS() for Clarity

Total Sales No Filter =

CALCULATE(

[Total Sales Amount],

REMOVEFILTERS('Product'[Color])

)

* This is functionally equivalent to ALL, but often easier to read.

### 🧪 Example 3: Use in Percentage Calculation

% Sales by Color =

DIVIDE(

[Total Sales Amount],

CALCULATE([Total Sales Amount], REMOVEFILTERS('Product'[Color]))

)

* Shows percentage contribution by each color to total sales.

### 🧠 Summary:

* Use ALL() or REMOVEFILTERS() to calculate grand totals or ignore filters in KPIs.
* REMOVEFILTERS() is generally preferred for clarity.
* Great for comparing context-based results vs overall performance.

**SECTION J: Power BI Practice – Using SELECTEDVALUE and VALUES**

### 🔍 Objective:

Understand how SELECTEDVALUE() and VALUES() work to capture or inspect user selection and unique values from filters or visuals.

### 🔧 Key Concepts:

* SELECTEDVALUE() returns a single value if exactly one value is selected; otherwise returns blank (or a default you define).
* VALUES() returns a one-column table of unique values from the column in the current filter context.

### 🧪 Example 1: Basic SELECTEDVALUE()

Selected Product Color = SELECTEDVALUE('Product'[Color], "Multiple")

* Returns the selected product color, or "Multiple" if more than one is selected.

### 🧪 Example 2: Use SELECTEDVALUE() in Titles

Dynamic Title = "Sales Report for " & SELECTEDVALUE('Product'[Color], "All Colors")

* Great for dynamic labels and titles in visuals.

### 🧪 Example 3: Inspecting Filter Context with VALUES()

Unique Colors = CONCATENATEX(VALUES('Product'[Color]), 'Product'[Color], ", ")

* Shows all unique colors currently in context, separated by commas.

### 🧠 Summary:

* Use SELECTEDVALUE() when expecting a single user selection (ideal for slicers).
* Use VALUES() to fetch all distinct values from a column in the current context.
* Combine with CONCATENATEX() for display, or IN operator for filtering logic.

**SECTION K: Power BI Practice – Understanding CALCULATETABLE**

### 🔍 Objective:

Learn how CALCULATETABLE() works to return filtered tables and how it's used in advanced DAX scenarios like virtual relationships and ranking.

### 🔧 Key Concepts:

* CALCULATETABLE() evaluates a table expression in a **modified filter context**.
* It is often used in **virtual relationships**, **ranking**, and **time intelligence** scenarios.
* Unlike CALCULATE(), it **returns a table**, not a scalar value.

### 💡 Syntax:

CALCULATETABLE(<table>, <filter1>, <filter2>, ...)

### 📊 Example 1: Filtered Table of Sales Over 1000

Sales Over 1000 =

CALCULATETABLE(

'FactInternetSales',

'FactInternetSales'[SalesAmount] > 1000

)

* Returns only rows from FactInternetSales where SalesAmount > 1000.
* Can be used inside COUNTROWS, SUMX, SELECTCOLUMNS, etc.

### 🔹 Use with COUNTROWS

Count Sales Over 1000 =

COUNTROWS(

CALCULATETABLE(

'FactInternetSales',

'FactInternetSales'[SalesAmount] > 1000

)

)

* Counts number of transactions with SalesAmount greater than 1000.

### 🔹 Example with RELATEDTABLE logic (Virtual Relationship)

Customers with Sales Over 1000 =

CALCULATETABLE(

VALUES('DimCustomer'[CustomerKey]),

'FactInternetSales'[SalesAmount] > 1000

)

* Returns distinct customer keys where sales exceeded 1000.
* Can be used with TREATAS() to simulate a virtual relationship.

### 🔹 Ranking Use Case (Top N customers)

Top Customers Table =

TOPN(

5,

CALCULATETABLE(

SUMMARIZE(

'DimCustomer',

'DimCustomer'[CustomerKey],

"TotalSales", SUM('FactInternetSales'[SalesAmount])

),

[TotalSales]

),

[TotalSales], DESC

)

* Creates a virtual table of top 5 customers by total sales.

### 🧩 Summary:

* CALCULATETABLE() is the **table-returning version** of CALCULATE().
* It enables **advanced filtering**, **virtual tables**, and **complex relationship modeling**.
* Best used inside iterator functions or table-based constructs (e.g., COUNTROWS, SUMMARIZE, TOPN).

**SECTION L: Power BI Practice – Virtual Relationships with TREATAS**

### 🔍 Objective:

Use TREATAS() to create virtual relationships between tables without physically modeling them.

### 🔧 Key Concepts:

* TREATAS() treats a **column or table** as if it is related to another.
* Especially useful when you **can’t or don’t want** to create a physical relationship.

### 🧪 Example: Virtual Join Between Unrelated Tables

Total Sales by Category =

CALCULATE(

[Total Sales Amount],

TREATAS(

VALUES('CategoryFilter'[Category]),

'Product'[Category]

)

)

* Applies filter from CategoryFilter table to Product table using category names.

### 🧠 Summary:

* Enables virtual filter propagation between tables.
* Useful for scenarios with disconnected slicers or many-to-many relationships.
* Clean and powerful alternative to bidirectional or complex relationships.

**Power BI Practice – Calculation Groups (SECTION M)**

### 🔍 Objective:

Learn how to simplify and standardize multiple DAX measures using **Calculation Groups** via **Tabular Editor**.

### 🔧 Key Concepts:

* **Calculation Groups** allow you to apply logic (like time intelligence or formatting) across many measures **without duplicating code**.
* Created using **Tabular Editor**, not directly inside Power BI UI.
* Each Calculation Group contains **Calculation Items**.

### 🧪 Step-by-Step: Create a Time Intelligence Calculation Group

#### ✅ Step 1: Open Tabular Editor

* In Power BI Desktop, go to **External Tools** → Open **Tabular Editor**.

#### ✅ Step 2: Create a Calculation Group

* In the Explorer panel, right-click on **Tables** → **Create New** → **Calculation Group**.
* Name it Time Intelligence.

#### ✅ Step 3: Add Calculation Items

Right-click on Time Intelligence → **New Calculation Item**.

Repeat to add the following:

Name: YTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESYTD('Calendar Table'[Date]))

Name: QTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESQTD('Calendar Table'[Date]))

Name: MTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESMTD('Calendar Table'[Date]))

🧠 SELECTEDMEASURE() dynamically references the current measure in a visual.

#### ✅ Step 4: Save and Refresh

* Click **Save** (Ctrl+S) in Tabular Editor.
* Return to Power BI and **refresh** the model.

#### ✅ Step 5: Use in Report

* Add a Matrix or Table visual.
* Place the **Time Intelligence Calculation Group** in **columns**.
* Place any measure (e.g., Total Sales Amount) in **values**.

Now each time calculation (YTD, QTD, MTD) appears automatically for all selected measures.

### 🔧 Bonus: Formatting with Calculation Groups

You can also create a group for **Measure Formatting** (e.g., dynamic % vs. absolute formatting):

Name: As Percentage

Expression: SELECTEDMEASURE() \* 1

Format String Expression: "0.00%"

Name: As Currency

Expression: SELECTEDMEASURE()

Format String Expression: "$#,0"

📌 Requires Power BI Premium or Premium Per User for format string expression support.

### 🧠 Summary:

* Calculation Groups improve **scalability** and **maintainability** of DAX code.
* Enable reusable logic across visuals without hardcoding variants.
* Created using **Tabular Editor**, not native Power BI UI.
* Leverage SELECTEDMEASURE() to dynamically apply transformations.

✅ **Next Practice**: Use Calculation Groups to switch between different KPI views (e.g., Actuals, Budget, Forecast).

### 🧪 Section M: Create a Time Intelligence Calculation Group

#### ✅ Step 1: Open Tabular Editor

* In Power BI Desktop, go to **External Tools** → Open **Tabular Editor**.

#### ✅ Step 2: Create a Calculation Group

* In the Explorer panel, right-click on **Tables** → **Create New** → **Calculation Group**.
* Name it Time Intelligence.

#### ✅ Step 3: Add Calculation Items

Right-click on Time Intelligence → **New Calculation Item**.

Repeat to add the following:

Name: YTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESYTD('Calendar Table'[Date]))

Name: QTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESQTD('Calendar Table'[Date]))

Name: MTD

Expression: CALCULATE(SELECTEDMEASURE(), DATESMTD('Calendar Table'[Date]))

🧠 SELECTEDMEASURE() dynamically references the current measure in a visual.

#### ✅ Step 4: Save and Refresh

* Click **Save** (Ctrl+S) in Tabular Editor.
* Return to Power BI and **refresh** the model.

#### ✅ Step 5: Use in Report

* Add a Matrix or Table visual.
* Place the **Time Intelligence Calculation Group** in **columns**.
* Place any measure (e.g., Total Sales Amount) in **values**.

Now each time calculation (YTD, QTD, MTD) appears automatically for all selected measures.

### 🔧 Bonus: Formatting with Calculation Groups

You can also create a group for **Measure Formatting** (e.g., dynamic % vs. absolute formatting):

Name: As Percentage

Expression: SELECTEDMEASURE() \* 1

Format String Expression: "0.00%"

Name: As Currency

Expression: SELECTEDMEASURE()

Format String Expression: "$#,0"

📌 Requires Power BI Premium or Premium Per User for format string expression support.

### 🧠 Section N: Dynamic KPI Switching Using Calculation Groups

KPI switching allows users to toggle between Actuals, Budget, Forecast, etc., in a visual using a slicer.

#### ✅ Step 1: Create a New Calculation Group

* In Tabular Editor, right-click **Tables** → **Create New** → **Calculation Group**.
* Name it KPI Selector.

#### ✅ Step 2: Add Calculation Items:

Name: Actual

Expression: [Actual KPI]

Name: Budget

Expression: [Budget KPI]

Name: Forecast

Expression: [Forecast KPI]

Replace [Actual KPI], [Budget KPI], and [Forecast KPI] with actual measure names in your model.

#### ✅ Step 3: Save and Refresh

* Press **Ctrl+S** to save.
* Go back to Power BI and **refresh** the model.

#### ✅ Step 4: Build Report

* Add a Matrix or Card visual.
* Place KPI Selector in **columns** or **slicer**.
* Add a base KPI measure (like [Actual KPI]) to **values**.

🧠 SELECTEDMEASURE() is not needed in this case because each item refers directly to a specific KPI.

### 🔄 Best Practices:

* Use KPI switching when you have **multiple parallel metrics** (e.g., Actual vs. Target).
* Combine with **Time Intelligence** Calculation Group for powerful multi-dimensional analysis.
* Name Calculation Items clearly for end-user slicer usability.

✅ **Next Practice**: Use Calculation Groups to apply dynamic unit conversions (e.g., Units → 000s → Millions).

### 📐 Section O: Dynamic Unit Conversion Using Calculation Groups

Sometimes, reports require displaying values in different units like units, thousands (K), or millions (M).

#### ✅ Step 1: Create a Calculation Group

* Open **Tabular Editor** → Right-click **Tables** → **New Calculation Group**.
* Name it Unit Scale.

#### ✅ Step 2: Add Calculation Items

Add the following items:

Name: Units

Expression: SELECTEDMEASURE()

Format String Expression: "#,0"

Name: Thousands (K)

Expression: SELECTEDMEASURE() / 1000

Format String Expression: "#,0,K"

Name: Millions (M)

Expression: SELECTEDMEASURE() / 1000000

Format String Expression: "#,0,,M"

🧠 The **Format String Expression** dynamically formats the visual labels to reflect scale.

#### ✅ Step 3: Save and Refresh

* Save your work (Ctrl+S), return to Power BI, and **refresh**.

#### ✅ Step 4: Add to Report

* Drag Unit Scale into a slicer.
* Add any base numeric measure (e.g., [Total Sales]) to a Matrix or Card.

Now users can toggle between **Units**, **K**, and **M** scales instantly.

💡 This technique enhances report **usability**, especially when working with high-volume data or when displaying financial reports in different contexts.

✅ **Next Practice**: Conditional formatting of measures using SELECTEDMEASURENAME() or ISSELECTEDMEASURE().

### 🧠 ****Section P: Conditional Formatting with Calculation Groups****

Enable **dynamic color or style changes** in visuals based on selected measures or values.

#### ✅ Use Case:

You want to **highlight KPIs** differently when different measures are selected via a slicer.

#### ✅ Step 1: Create a Calculation Group

* In **Tabular Editor**, right-click Tables → **New Calculation Group**.
* Name it: Measure Styling.

#### ✅ Step 2: Add Conditional Format Calculation Items

plaintext

CopyEdit

Name: Highlight KPI

Expression:

VAR \_value = SELECTEDMEASURE()

RETURN

IF(\_value > 1000000, \_value, BLANK())

Format String Expression: "#,0"

This will only show values greater than 1 million; others will be blank, simulating a highlight.

#### ✅ Step 3: Use SELECTEDMEASURENAME() for Custom Formatting

Example for applying different styles to different KPIs:

plaintext

CopyEdit

Name: Custom Format

Expression: SELECTEDMEASURE()

Format String Expression:

SWITCH(

SELECTEDMEASURENAME(),

"Total Sales", "$#,0,,M",

"Profit Margin", "0.00%",

"Units Sold", "#,0",

"#,0"

)

#### ✅ Step 4: Save & Refresh Power BI

* Press **Ctrl+S** in Tabular Editor → return to Power BI → click **Refresh**.
* Add Measure Styling to a visual to apply formatting dynamically.

### 💡 Why This Matters:

* You can control **visibility**, **appearance**, or **highlight logic** without changing visuals.
* This is useful for **executive dashboards** where KPIs may need different scales and formatting.

### Section P: SWITCH vs IF – Dynamic Measure Switching

Use the SWITCH() function for simplified logic over multiple IF statements.

#### ✅ Step 1: Create a Manual Table for Selection

* In Power BI → **Enter Data**
* Name the table: Data Type
* Add column: Type with values:
  + Percentage
  + Number

#### ✅ Step 2: Create a Slicer

* Drag the Type column to the report canvas.
* Change visual to **Slicer**.

#### ✅ Step 3: Create Measures

Currency Conversion % =

VAR SalesAll = CALCULATE([Currency Conversion], ALL('Sales Territory'))

RETURN DIVIDE([Currency Conversion], SalesAll)

Switched =

SWITCH(

SELECTEDVALUE('Data Type'[Type]),

"Percentage", [Currency Conversion %],

"Number", [Currency Conversion]

)

Optional IF version:

Switched IF =

IF(

SELECTEDVALUE('Data Type'[Type]) = "Percentage",

[Currency Conversion %],

IF(

SELECTEDVALUE('Data Type'[Type]) = "Number",

[Currency Conversion]

)

)

#### ✅ Step 4: Use in Report

* Replace [Currency Conversion] in visuals with Switched.
* Toggle between Number and Percentage using slicer.

⚠️ Formatting mixed output (like % and numbers) can be tricky. Consider using neutral formatting (e.g., 2 decimal places).

💡 SWITCH() is preferred when checking **multiple conditions** — easier to read and manage than deeply nested IF() blocks.