

Author: Sudhanshu Mukherjee

Date: September 09, 2025

- Install GitHub Copilot and GitHub Co Pilot Chat
- Make sure .pbip is enabled -> options -> Feature Preview
- Power BI Studio, Power Query / M Language, TMDL
- Create a GitHub repository
- Clone it locally
- Open PBI Desktop and Save Power BI Project (.pbip)* not .pbix
- Move to a folder and start working

Using global_sales.xlsx

1. Essential DAX Measures

Prompt:

Create core DAX measures for this 51K+ orders dataset:

1. Total Sales, Total Profit, Total Orders (distinct count)
2. Profit Margin % = Profit/Sales
3. Average Order Value = Sales/distinct Order count
4. Year-over-Year Sales Growth %
5. Top 10 Customers by Sales (measure)
6. Top 10 Products by Profit (measure)

Include proper number formatting and error handling

Result

```
// Core DAX Measures for Orders Dataset

// 1. Total Sales
Total Sales =
SUM('Orders'[Sales])

// 2. Total Profit
Total Profit =
SUM('Orders'[Profit])

// 3. Total Orders (distinct count)
Total Orders =
DISTINCTCOUNT('Orders'[Order ID])
```

```

// 4. Profit Margin %
Profit Margin % =
DIVIDE([Total Profit], [Total Sales], 0)

// 5. Average Order Value
Average Order Value =
DIVIDE([Total Sales], [Total Orders], 0)

// 6. Year-over-Year Sales Growth %
YoY Sales Growth % =
VAR CurrentYear = YEAR(TODAY())
VAR SalesThisYear = CALCULATE([Total Sales], YEAR('Orders'[Order Date]) =
CurrentYear)
VAR SalesLastYear = CALCULATE([Total Sales], YEAR('Orders'[Order Date]) =
CurrentYear - 1)
RETURN DIVIDE(SalesThisYear - SalesLastYear, SalesLastYear, 0)

// 7. Top 10 Customers by Sales (measure)
Top 10 Customers by Sales =
IF(
    RANKX(
        ALL('Orders'[CustomerID]),
        CALCULATE([Total Sales])
    ) <= 10,
    [Total Sales],
    BLANK()
)

// 8. Top 10 Products by Profit (measure)
Top 10 Products by Profit =
IF(
    RANKX(
        ALL('Orders'[ProductID]),
        CALCULATE([Total Profit])
    ) <= 10,
    [Total Profit],
    BLANK()
)

```

2. Data Modeling and Relationships

Prompt:

Create Power BI data model relationships and structure for this dataset:

Tables:

- Orders (51K+ rows): Row ID, Order ID, Order Date, Ship Date, Customer ID, Customer Name, Product ID, Product Name, Category, Sub-Category, Region, Country, State, City, Sales, Profit, Quantity, Discount, etc.
- Returns (1K+ rows): Returned, Order ID, Market
- People (14 rows): Person, Region

Requirements:

1. Define all table relationships with proper cardinality and filter directions
2. Identify primary keys and foreign keys for each table
3. Create relationship structure that supports:
 - Sales analysis by geography, product, customer
 - Returns analysis linked to original orders
 - Sales team performance by region
4. Recommend any additional calculated tables needed (like Date table)
5. Specify cross-filter directions for optimal performance
6. Include data model best practices for this structure

Result:

```
### Power BI Data Model Structure & Relationships

#### 1. **Table Relationships, Cardinality, and Filter Directions**

| From Table | From Column | To Table | To Column | Cardinality |
| Cross-filter Direction |
|-----|-----|-----|-----|-----|
|-----|
| Orders | Customer ID | Customers* | Customer ID | Many-to-One |
| Single | | | | |
| Orders | Product ID | Products* | Product ID | Many-to-One |
| Single | | | | |
| Orders | Order ID | Returns | Order ID | One-to-One |
| Single | | | | |
| Orders | Region | People | Region | Many-to-One |
| Single | | | | |
| Orders | Order Date | Date | Date | Many-to-One |
| Single | | | | |

\*If you do not have separate Customers/Products tables, use the columns in
Orders.

#### 2. **Primary Keys & Foreign Keys**
```

- ****Orders:****
 - PK: Row ID (or Order ID if unique)
 - FK: Customer ID, Product ID, Region, Order Date
- ****Returns:****
 - PK: Order ID
 - FK: Order ID (links to Orders), Market
- ****People:****
 - PK: Person
 - FK: Region
- ****Date Table (Recommended):****
 - PK: Date
 - FK: Order Date (in Orders)

3. ****Relationship Structure Supports:****

- ****Sales analysis by geography, product, customer:****
 - Orders linked to geography columns (Region, Country, State, City), Product ID, Customer ID
- ****Returns analysis linked to original orders:****
 - Returns linked to Orders via Order ID
- ****Sales team performance by region:****
 - Orders linked to People via Region

4. ****Additional Calculated Tables Needed****

- ****Date Table:****
- Use DAX:

****Date = CALENDAR(MIN(Orders[Order Date]), MAX(Orders[Order Date]))****

Add columns: Year, Month, Quarter, Week, etc.

- ****Customers Table:**** (if not present, create from Orders)
- ****Products Table:**** (if not present, create from Orders)

5. ****Cross-filter Directions****

- Use ****Single**** direction for all relationships for optimal performance and to avoid ambiguity.
- Only use ****Both**** if you need bi-directional filtering for specific analysis (e.g., People ↔ Orders for team performance).

6. ****Data Model Best Practices****

- Use surrogate keys (Row ID) for Orders if possible.
- Hide foreign key columns in fact tables.
- Use star schema: Fact table (Orders), dimension tables (Date, Customers, Products, People).
- Avoid unnecessary bi-directional relationships.

- Create calculated columns/tables for time intelligence (Date table).
- Ensure all relationships are active and properly documented.

****Diagram Example:****

```
Orders (Fact)
  |--< Customers (Dim)
  |--< Products (Dim)
  |--< Date (Dim)
  |--< People (Dim)
  |--1 Returns (Fact)
```

Note:

Since your current data has everything in one Orders table, we need to create dimension tables first.

Go to Data view -> Click New Table -> Paste this DAX

```
Date =
ADDCOLUMNS(
    CALENDAR(DATE(2012,1,1), DATE(2015,12,31)),
    "Year", YEAR([Date]),
    "Month", MONTH([Date]),
    "Quarter", QUARTER([Date]),
    "MonthName", FORMAT([Date], "MMMM"),
    "WeekdayName", FORMAT([Date], "dddd")
)
```

```
Customers =
DISTINCT(
    SELECTCOLUMNS(
        Orders,
        "Customer ID", Orders[Customer ID],
        "Customer Name", Orders[Customer Name],
        "Segment", Orders[Segment]
    )
)
```

```
Products =
SUMMARIZE(
    Orders,
```

```
Orders[Product ID],  
"Category", FIRSTNONBLANK(Orders[Category], Orders[Category]),  
"Sub-Category", FIRSTNONBLANK(Orders[Sub-Category], Orders[Sub-  
Category]))  
)
```

3. Conditional Formatting Rules

Prompt:

Create comprehensive conditional formatting rules for Power BI reports using global sales data with these columns: Sales, Profit, Profit Margin %, Quantity, Discount, YoY Growth %, Region, Category, Customer Segment.

Requirements:

1. Create conditional formatting expressions for sales performance tables:
 - Red for negative profit or losses
 - Green for high performance (>15% profit margin or >20% growth)
 - Yellow/Orange for moderate performance (5-15% profit margin, 0-20% growth)
 - Gray for below target performance
2. Create data bars and color scales for:
 - Sales amounts (gradient from light to dark blue)
 - Profit margins (red-yellow-green scale)
 - Quantity sold (size-based formatting)
3. Include formatting for tables, matrices, and cards
4. Specify exact color codes, threshold values, and formatting rules

Result

Step 1: Create Your Table

1. **Create a Table visual**
2. **Add these fields:**
 - Customer Name (or Product Name)
 - Total Sales
 - Total Profit
 - Profit Margin %

Step 2: Apply Conditional Formatting to Profit Column

1. **Select the table visual**
2. **Format pane** (paint brush icon) → **Values**

3. **Find "Total Profit"** in the list
4. **Conditional formatting** → **Background color** → **Turn ON**
5. **Format style:** Select **"Rules"**

Step 3: Set Up the Color Rules

Rule 1 – Negative Profit (Red):

- **If value:** `is less than` **0**
- **Color:** Red `#D32F2F`

Rule 2 – High Performance (Green):

- **If value:** `is greater than or equal to` **500** (adjust based on your data)
- **Color:** Green `#388E3C`

Rule 3 – Moderate Performance (Yellow):

- **If value:** `is greater than or equal to` **100** and `is less than` **500**
- **Color:** Yellow `#FBC02D`

Default Color: Light Gray `#F5F5F5`

Step 4: Apply Rules

- Click **"Apply"** or **"OK"**

🎨 Alternative: Use Profit Margin % Instead

If you want to base colors on Profit Margin % (more meaningful):

Apply to Profit Margin % Column:

1. **Select Profit Margin % column** in the table
2. **Conditional formatting** → **Background color** → **ON**
3. **Format style:** "Rules"

Rules for Profit Margin %:

- **Excellent (Green):** `>= 15%` → `#388E3C`
- **Good (Light Green):** `>= 10%` and `< 15%` → `#66BB6A`
- **Fair (Yellow):** `>= 5%` and `< 10%` → `#FBC02D`
- **Poor (Orange):** `>= 0%` and `< 5%` → `#FFA726`
- **Loss (Red):** `< 0%` → `#D32F2F`

🚀 Even Simpler: Color Scale Method

For gradual color transitions:

1. **Select Profit Margin % column**
2. **Conditional formatting** → **Background color** → **ON**
3. **Format style:** **"Color scale"**
4. **Set colors:**
 - **Minimum:** Red `#D32F2F`
 - **Center:** Yellow `#FBC02D` (set to 10%)
 - **Maximum:** Green `#388E3C`

🇮🇹 Quick Data Bars (No Measure Needed)

For Sales column:

1. **Select Total Sales column**
2. **Conditional formatting** → **Data bars** → **Turn ON**
3. **Positive bar color:** Dark Blue `#0D47A1`
4. **Show bar only:** OFF (shows both value and bar)

Step 5: Matrix with Advanced Formatting

Create Category Performance Matrix:

1. **Matrix visual**
2. **Rows:** Category, Sub-Category
3. **Columns:** Region
4. **Values:** Total Sales, Profit Margin %

Apply Multiple Conditional Formats:

1. **Sales:** Data bars (blue gradient)
2. **Profit Margin %:** Color scale (red-yellow-green)
3. **Background:** Based on performance vs benchmarks

Step 6: Testing Your Conditional Formatting

4. Custom Color Themes

Prompt:

Generate a comprehensive Power BI custom theme JSON file for a professional business dashboard with these requirements:

1. Create a corporate color scheme with:
 - Primary brand color: Professional blue (#2E5BBA or similar)
 - Secondary colors: Gray scale for backgrounds and text

- Accent colors: Green for positive values, red for negative, orange for warnings
 - Data visualization palette: 10+ distinct colors for charts and graphs
2. Include theme settings for:
 - Background colors (report, visual, card backgrounds)
 - Text colors (titles, labels, data labels)
 - Grid lines and borders
 - Button and slicer styling
 - Chart colors (bars, lines, areas, pie slices)
 3. Ensure accessibility with:
 - High contrast ratios for text readability
 - Color-blind friendly palette
 - Clear visual hierarchy
 4. Provide JSON structure for:
 - Data colors array (for automatic chart coloring)
 - Visual styles (cards, tables, charts)
 - Text classes (title, subtitle, body)
 - Background themes
 5. Include both light and dark theme variations
 6. Add hover states and selection colors
 7. Make it suitable for executive dashboards and professional presentations

Output as complete JSON theme file ready to import into Power BI.

Result

```
{
  "name": "Corporate Executive Dashboard Theme",
  "dataColors": [
    "#2E5BBA",
    "#1B9E77",
    "#D32F2F",
    "#FBC02D",
    "#388E3C",
    "#FFA726",
    "#7B1FA2",
    "#0288D1",
    "#F57C00",
    "#455A64",
    "#C2185B",
    "#0097A7"
  ],
  "background": "#F5F7FA",
  "foreground": "#222222",
  "tableAccent": "#2E5BBA",
```

```

"visualStyles": {
  "*": {
    "*": {
      "background": [{"color": {"solid": {"color": "#FFFFFF"}}}],
      "border": [{"color": {"solid": {"color": "#B0BEC5"}}}]
    }
  },
  "card": {
    "*": {
      "background": [{"color": {"solid": {"color": "#FFFFFF"}}}],
      "border": [{"color": {"solid": {"color": "#B0BEC5"}}}],
      "dropShadow": [{"show": true, "color": {"solid": {"color": "#00000020"}}}]
    }
  },
  "tableEx": {
    "*": {
      "background": [{"color": {"solid": {"color": "#FFFFFF"}}}],
      "border": [{"color": {"solid": {"color": "#B0BEC5"}}}]
    }
  },
  "slicer": {
    "*": {
      "background": [{"color": {"solid": {"color": "#FFFFFF"}}}],
      "border": [{"color": {"solid": {"color": "#B0BEC5"}}}]
    }
  },
  "textClasses": {
    "callout": {
      "fontSize": 45,
      "fontFace": "Segoe UI Light",
      "color": "#2E5BBA"
    },
    "title": {
      "fontSize": 20,
      "fontFace": "Segoe UI Semibold",
      "color": "#2E5BBA"
    },
    "header": {
      "fontSize": 16,
      "fontFace": "Segoe UI Semibold",
      "color": "#222222"
    },
    "label": {
      "fontSize": 13,
      "fontFace": "Segoe UI",
      "color": "#455A64"
    }
  }
}

```

```
}
```

5. Row Level Security Rules

Prompt:

Create comprehensive Row-Level Security (RLS) DAX rules for Power BI using this global sales dataset structure:

Tables:

- Orders: Customer ID, Customer Name, Region, Country, State, Sales, Profit, Salesperson ID, etc.
- People: Person (salesperson name), Region
- Returns: Order ID, Market
- User Roles to Implement:
 1. Sales Manager: Sees only their specific region's data
 2. Country Manager: Sees only their country's data
 3. Salesperson: Sees only their own sales records
 4. Regional Director: Sees multiple regions they manage
 5. CEO/Executive: Sees all data (no restrictions)

Result:

Implementation

Step 1: Create RegionAccess Table

1. **Go to Data view** in Power BI Desktop
2. **Click "New table"**
3. Copy-paste this DAX:

Regional Director

```
RegionAccess =  
DATATABLE(  
    "Username", STRING,  
    "Region", STRING,  
    {  
        {"john.director@company.com", "Central"},  
        {"john.director@company.com", "East"},
```

```

        {"mary.director@company.com", "West"},
        {"mary.director@company.com", "South"},
        {"sarah.director@company.com", "Central"},
        {"sarah.director@company.com", "West"}
    }
)

```

Country Managers Table

```

CountryAccess =
DATATABLE(
    "Username", STRING,
    "Country", STRING,
    {
        {"us.manager@company.com", "United States"},
        {"uk.manager@company.com", "United Kingdom"},
        {"canada.manager@company.com", "Canada"},
        {"france.manager@company.com", "France"},
        {"germany.manager@company.com", "Germany"},
        {"australia.manager@company.com", "Australia"}
    }
)

```

Sales Manager Table

```

RegionManager =
DATATABLE(
    "Username", STRING,
    "Region", STRING,
    {
        {"central.manager@company.com", "Central"},
        {"east.manager@company.com", "East"},
        {"west.manager@company.com", "West"},
        {"south.manager@company.com", "South"}
    }
)

```

Phase 2

Step 1: Access Role Management

1. Modeling tab → Manage roles

Role 1 - Sales Manager

```
Orders[Region] = LOOKUPVALUE(
    RegionManager[Region],
    RegionManager[Username],
    USERPRINCIPALNAME()
)
```

Role 2 - Country Manager

```
Orders[Country] = LOOKUPVALUE(
    CountryAccess[Country],
    CountryAccess[Username],
    USERPRINCIPALNAME()
)
```

Role 3 - Regional Director

```
Orders[Region] IN
SELECTCOLUMNS(
    FILTER(
        RegionAccess,
        RegionAccess[Username] = USERPRINCIPALNAME()
    ),
    "Region", RegionAccess[Region]
)
```

Role 4 - CEO

Leave Blank

Test Setup:

1. **Modeling** → **View as**
2. **Select:** "Sales Manager role" | "Country Manager" | Regional Director Role | CEO
3. **Other user:** central.manager@company.com | us.manager@company.com | john.director@company.com | ceo@company.com
4. **Apply**

6. Custom Visual

Prompt:

Create comprehensive custom visual integration guide for Power BI with:

Specific Use Cases for Global Sales Data:

- Sales flow from regions to categories (Sankey)

Result

```
### Power BI Custom Visual Integration Guide
```

```
#### Use Case: Sales Flow from Regions to Categories (Sankey Diagram)
```

```
---
```

```
#### **1. Install the Sankey Custom Visual**
```

- Go to **Power BI Desktop > Visualizations pane**
- Click the **ellipsis (...) > Get more visuals**
- Search for **Sankey** (e.g., “Sankey Chart by Microsoft” or “Sankey Diagram”)
- Click **Add** to import the visual

```
---
```

```
#### **2. Prepare Your Data**
```

- Ensure your dataset has:
 - **Source**: Region (e.g., `Orders[Region]`)
 - **Destination**: Category (e.g., `Orders[Category]`)
 - **Weight/Value**: Sales (e.g., `Orders[Sales]`)
- If needed, create a summarized table:

```
SankeyData =  
SUMMARIZE(  
    Orders,  
    Orders[Region],  
    Orders[Category],  
    "TotalSales", SUM(Orders[Sales])  
)
```

```
---
```

```
#### **3. Add the Sankey Visual to Your Report**
```

- Drag the **Sankey** visual onto your report canvas
- Assign fields:
 - **Source**: `Region`

- ****Destination****: `Category`
- ****Weight****: `TotalSales` (or `Sales`)

****4. Customize the Visual****

- Use the ****Format**** pane to:
 - Adjust colors for accessibility and branding
 - Set node and link labels for clarity
 - Enable tooltips for additional details (e.g., show `TotalSales`, `Profit Margin %`)
 - Adjust link thickness based on sales volume

****5. Interactivity & Filtering****

- The Sankey visual supports cross-filtering:
 - Clicking a region or category highlights related flows
 - Use slicers for time, customer segment, or product filters

****6. Best Practices****

- Limit the number of nodes for readability (e.g., top 10 regions/categories)
- Use color coding consistent with your corporate theme
- Add explanatory text or legends for executive audiences
- Test with sample data to ensure flows are meaningful

****7. Export & Share****

- Save your report as `.pbix`
- Publish to Power BI Service for sharing and collaboration

****Summary:****

The Sankey visual is ideal for showing sales flow from regions to categories. Prepare your data, install the visual, assign fields, and format for clarity and accessibility. Use slicers and tooltips for deeper insights.

Let me know if you need DAX code for dynamic Sankey data or further customization tips!

7. M-Language

Prompt:

Create M language code to import the Orders sheet from global_sales.xlsx and fix these data issues:

- Convert Order Date and Ship Date from Excel serial numbers (like 41121) to proper MM/DD/YYYY format
- Ensure Sales, Profit, Quantity, Discount are proper decimal number types
- Create a calculated column called 'Delivery Days' = Ship Date - Order Date
- Handle any null values in critical fields
- Set up proper data types for all columns

Result:

```
let
    // Import the Orders sheet
    Source =
Excel.Workbook(File.Contents("d:\Neokai\Github_PBI\AgenticPowerBI\global_sales.xlsx"), null, true),
    Orders_Sheet = Source{[Item="Orders",Kind="Sheet"]}[Data],

    // Promote headers
    PromotedHeaders = Table.PromoteHeaders(Orders_Sheet,
[PromoteAllScalars=true]),

    // Convert Excel serial dates to proper date format
    ConvertedDates = Table.TransformColumns(
        PromotedHeaders,
        {
            {"Order Date", each Date.From(Number.From(_)), type date},
            {"Ship Date", each Date.From(Number.From(_)), type date}
        }
    ),

    // Ensure numeric columns are proper decimal types
    ChangedTypes = Table.TransformColumnTypes(
        ConvertedDates,
        {
            {"Sales", type number},
            {"Profit", type number},
            {"Quantity", Int64.Type},
            {"Discount", type number}
        }
    ),
```



```

        // Handle nulls in critical fields (remove rows with nulls in Order
Date, Ship Date, Sales, Quantity)
        RemovedNulls = Table.SelectRows(
            ChangedTypes,
            each [Order Date] <> null and [Ship Date] <> null and [Sales] <>
null and [Quantity] <> null
        ),

        // Create Delivery Days column (difference between Ship Date and Order
Date)
        AddedDeliveryDays = Table.AddColumn(
            RemovedNulls,
            "Delivery Days",
            each Duration.Days([Ship Date] - [Order Date]),
            Int64.Type
        ),

        // Set proper data types for all columns (add more columns as needed)
        FinalTypes = Table.TransformColumnTypes(
            AddedDeliveryDays,
            {
                {"Order ID", type text},
                {"Order Date", type date},
                {"Ship Date", type date},
                {"Sales", type number},
                {"Profit", type number},
                {"Quantity", Int64.Type},
                {"Discount", type number},
                {"Delivery Days", Int64.Type}
                // Add other columns and types as needed
            }
        )
    in
        FinalTypes

```

8. Dynamic Titles and Labels

Prompt

Create simple dynamic titles and labels for Power BI reports using global sales data with these requirements:

1. Dynamic Report Titles:

- Show current date range being viewed (e.g., "Sales Report – January

2024")

- Update title based on selected filters (e.g., "Central Region Performance")
- Display current vs previous period (e.g., "Q4 2024 vs Q4 2023")

2. Smart Chart Titles:

- Bar chart titles that change with slicers (e.g., "Top Products in [Selected Region]")

Result

Dynamic Report Title

```
Dynamic Report Title =  
VAR MinDate = MIN('Orders'[Order Date])  
VAR MaxDate = MAX('Orders'[Order Date])  
VAR SelectedRegion = SELECTEDVALUE('Orders'[Region], "All Regions")  
RETURN  
"Sales Report: " &  
FORMAT(MinDate, "MMM YYYY") & " - " & FORMAT(MaxDate, "MMM YYYY") &  
" | " & SelectedRegion
```

Smart Cart Title for Products

```
Top Products Chart Title =  
VAR SelectedRegion = SELECTEDVALUE('Orders'[Region], "All Regions")  
RETURN  
"Top Products in " & SelectedRegion
```

Category Analysis Table

```
Category Analysis Title =  
VAR SelectedCategory = SELECTEDVALUE('Orders'[Category], "All Categories")  
VAR SelectedRegion = SELECTEDVALUE('Orders'[Region], "All Regions")  
RETURN  
SelectedCategory & " Performance in " & SelectedRegion
```

Data Freshness Indicator

```
Data Freshness Title =  
VAR LastOrderDate = MAX('Orders'[Order Date])  
VAR DaysOld = DATEDIFF(LastOrderDate, TODAY(), DAY)  
RETURN  
"Data as of: " & FORMAT(LastOrderDate, "MMM DD, YYYY") &  
" (" & DaysOld & " days ago)"
```

Create Dynamic Report Header:

For Bar Charts showing Top Products:

1. **Select your bar chart**
2. **Format pane** → **General** → **Title**
3. **Turn ON title**
4. **Title text:** Click **"fx"** → Select "Top Products Chart Title"

Add Region Slicer:

1. **Slicer visual**
2. **Add field:** Orders[Region]
3. **Format as dropdown or buttons**

Add Category Slicer:

1. **Another slicer visual**
2. **Add field:** Orders[Category]
3. **Format consistently**

9. Data Refresh Rules

Prompt

Create comprehensive data refresh automation rules and schedules for Power BI using global sales data with these requirements:

1. Automatic Refresh Scheduling:
 - Daily refresh at optimal times (6 AM, avoid peak hours)
 - Weekly full refresh for data model optimization
 - Real-time refresh for critical KPIs
 - Incremental refresh for large datasets (51K+ rows)
 - Holiday and weekend scheduling considerations
2. Refresh Failure Handling:
 - Automatic retry mechanisms (3 attempts with delays)
 - Error notification system (email alerts to admin/owners)
 - Fallback to cached data when source unavailable
 - Data quality validation after refresh
 - Rollback procedures for corrupted refreshes

Result

Power BI Data Refresh Automation Rules & Schedules

1. Automatic Refresh Scheduling

- ****Daily Refresh:****
 - Schedule at ****6:00 AM local time**** (before business hours, avoids peak usage).
 - Use Power BI Service > Dataset Settings > Schedule Refresh.
- ****Weekly Full Refresh:****
 - Schedule every ****Sunday at 6:00 AM**** for full data model optimization.
 - In Power BI Service, set a separate schedule for full refresh.
- ****Real-Time Refresh for Critical KPIs:****
 - Use ****DirectQuery**** or ****Streaming Dataset**** for dashboards with real-time KPIs.
 - Configure refresh intervals (e.g., every 5 minutes) for critical visuals.
- ****Incremental Refresh for Large Datasets:****
 - Enable ****Incremental Refresh**** in Power BI Desktop (Table > Incremental Refresh).
 - Set parameters:
 - ****RangeStart**** and ****RangeEnd**** for partitioning (e.g., last 12 months).
 - Daily refresh for new/changed data, weekly full refresh for all partitions.
- ****Holiday & Weekend Scheduling:****
 - Use Power BI Service API or Azure Automation to skip refreshes on holidays/weekends.
 - Maintain a calendar table with non-business days for reference.

2. Refresh Failure Handling

- ****Automatic Retry Mechanisms:****
 - Configure Power BI Service to retry failed refreshes up to ****3 times**** with 15-minute intervals.
- ****Error Notification System:****
 - Set up ****email alerts**** to dataset owners/admins on refresh failure (Power BI Service > Manage Alerts).
 - Integrate with Microsoft Power Automate for advanced notification

workflows.

- ****Fallback to Cached Data:****
 - If source is unavailable, retain last successful refresh data for reporting.
 - Notify users of stale data via dashboard banner or card.
- ****Data Quality Validation After Refresh:****
 - Use Power Query or DAX to validate key metrics (e.g., row counts, null checks).
 - Log validation results and alert if anomalies detected.
- ****Rollback Procedures for Corrupted Refreshes:****
 - Maintain backup of previous dataset version.
 - If corruption detected, revert to last known good dataset (manual or automated via API).

****Sample Power BI Service Refresh Schedule Setup****

- ****Daily:****
 - Time: 6:00 AM
 - Type: Incremental
- ****Weekly:****
 - Time: Sunday 6:00 AM
 - Type: Full
- ****Real-Time:****
 - DirectQuery/Streaming: Every 5 minutes

****Best Practices****

- Monitor refresh history regularly.
- Document refresh schedules and failure protocols.
- Use incremental refresh for performance and scalability.
- Communicate refresh status to business users.

****Summary:****

Automate refreshes with daily/weekly schedules, real-time updates for KPIs, and incremental refresh for large datasets. Implement robust failure handling, notifications, validation, and rollback procedures for reliable data delivery.

10. Report Documentation

Automated Report Distribution