DAX FUNDAMENTALS PLAYBOOK

Complete Guide to Data Analysis Expressions (Wrong Measures Intentionally to challenge your brain and create problem solving skills)

Correct Measures will be in Correct Measures csv file

Focus: DAX syntax, functions, patterns, and Australian business calculations

Duration: 60 minutes of pure DAX learning

What is DAX

Definition: DAX (Data Analysis Expressions) is a formula language for Power BI, Excel Power Pivot, and Analysis Services.

Key Differences from Excel

Feature	Excel	DAX
Scope	Cell references (A1:A100)	Entire tables/columns
Context	Single cell	Filter + Row context
Performance	Slow on 1M+ rows	Handles billions
Relationships	Manual VLOOKUP	Automatic via model

Basic Syntax

```
-- Measure format
Measure Name = Expression

-- Example
Total Revenue = SUM(FactSales[Revenue])

-- With comments
Total Revenue = SUM(FactSales[Revenue]) // Sum all revenue values
```

Table[Column] References

```
-- Correct
SUM(FactSales[Revenue])

-- Wrong (must specify table)
SUM([Revenue]) -- ERROR
```

Calculated Columns vs Measures

THE CRITICAL DIFFERENCE

Calculated Columns

When computed: At model refresh

Storage: Stored in memory

Context: Row context (each row independently)

```
-- Calculated Column in FactSales
Profit = FactSales[Revenue] - FactSales[Cost]

-- Result: Value stored for each row
-- Row 1: $100 - $60 = $40
-- Row 2: $150 - $90 = $60
-- Row 3: $200 - $120 = $80
```

Use for:

- Creating categories/groups
- Values needed in slicers
- Sorting columns

```
-- Age Group (must be calculated column for slicer)
Age Group = SWITCH(
    TRUE(),
    DimCustomer[Age] < 25, "18-24",
    DimCustomer[Age] < 35, "25-34",
    DimCustomer[Age] < 50, "35-49",
    "50+"
)
```

Measures

When computed: At query time (dynamic)

Storage: No storage (calculated on-the-fly)

Context: Filter context (responds to slicers)

```
-- Measure
Total Profit = SUM(FactSales[Revenue]) - SUM(FactSales[Cost])

-- Result: Changes with filters
-- No filter: $45M
-- State=NSW: $18M
-- Category=Dairy: $3.2M
```

Use for:

- Aggregations (SUM, COUNT, AVG)
- KPIs and metrics
- Time intelligence
- Percentages

Decision Matrix

Need	Use	
Value in slicer	✅ Calculated Column	
Aggregation that changes with filters	✅ Measure	
Grouping/categorization	✅ Calculated Column	
KPI that updates dynamically	✅ Measure	
Row-by-row calculation	✅ Calculated Column	
Percentage of total	✅ Measure	

Performance: Default to **Measures** (no storage, faster).

Evaluation Context - The Key Concept

What: Context determines which rows DAX considers when calculating.

Two Types:

1. **Filter Context** - Created by slicers, filters, visuals
2. **Row Context** - Created by iteration (SUMX, calculated columns)

Filter Context Example

```
Total Revenue = SUM(FactSales[Revenue])
```

Scenario 1: No filters

Context: All rows

Result: $45,000,000

Scenario 2: User selects State=NSW

Context: Only NSW rows

Result: $18,000,000

Scenario 3: State=NSW AND Category=Dairy

Context: NSW rows AND Dairy rows

Result: $3,200,000

Same measure, different results based on filter context.

Row Context Example

```
-- Calculated Column
Profit = FactSales[Revenue] - FactSales[Cost]

-- How it works:
-- Row 1: $100 - $60 = $40
-- Row 2: $150 - $90 = $60
-- Row 3: $200 - $120 = $80
```

Each row evaluated independently (row context).

Context Transition

Problem: Measures need filter context, not row context

```
-- This FAILS in calculated column
Profit = [Total Revenue] - [Total Cost] -- ERROR
```

Solution: CALCULATE transitions row → filter context

```
-- This WORKS
Profit = CALCULATE([Total Revenue] - [Total Cost])
```

Filter Context vs Row Context

Filter Context

Created by:

- Slicers
- Visual filters
- Row/column headers in matrix
- CALCULATE function

Affects: Which rows are visible to measure

```
-- Filter context from slicer
Total Revenue = SUM(FactSales[Revenue])

-- User selects State=NSW
-- DAX thinks: "Show me NSW rows only, sum those"
```

Row Context

Created by:

- Calculated columns
- Iterator functions (SUMX, FILTER)

Affects: One row at a time

```
-- Row context in SUMX
Total Profit = SUMX(
    FactSales,
    FactSales[Revenue] - FactSales[Cost]
)

-- DAX thinks:
-- "Row 1: Revenue=$100, Cost=$60, Profit=$40"
-- "Row 2: Revenue=$150, Cost=$90, Profit=$60"
-- "Sum: $100"
```

Combining Both

```
Total Profit = SUMX(
 FactSales, -- Iterate rows (row context)
 FactSales[Revenue] - FactSales[Cost]
)

-- If user filters State=NSW:
-- 1. Filter context limits to NSW rows
-- 2. SUMX iterates those NSW rows
-- 3. Calculates Profit per row
-- 4. Sums results
```

Core DAX Functions

Aggregation, Arithmetic & Conditional Logic

1

Aggregation Functions

```
-- SUM
Total Revenue = SUM(FactSales[Revenue])
Total Cost = SUM(FactSales[Cost])

-- AVERAGE
Avg Transaction Value = AVERAGE(FactSales[Revenue])

-- COUNT Functions
Transaction Count = COUNT(FactSales[SalesID])
Product Count = COUNTA(DimProduct[ProductName])
Row Count = COUNTROWS(FactSales)
Unique Customers = DISTINCTCOUNT(FactSales[CustomerID])

-- MIN / MAX
First Order Date = MIN(FactSales[OrderDate])
Last Order Date = MAX(FactSales[OrderDate])
```

2

Arithmetic Calculations

```
-- Gross Profit
Gross Profit =
VAR Revenue = SUM(FactSales[Revenue])
VAR Cost = SUM(FactSales[Cost])
RETURN Revenue - Cost

-- Australian GST Calculations
Revenue Ex GST = DIVIDE(
    SUM(FactSales[Revenue]),
    1.1 -- Remove GST
)

GST Amount = [Total Revenue] - [Revenue Ex GST]

-- GST % Check
GST % Check = DIVIDE([GST Amount], [Total Revenue])
-- Result: 0.0909 (10/110)
```

3

Percentages and Ratios

```
-- Gross Margin %
Gross Margin % = DIVIDE(
    [Gross Profit],
    [Total Revenue]
)

-- Revenue % of Grand Total
Revenue % of Total = DIVIDE(
    [Total Revenue],
    CALCULATE(
        [Total Revenue],
        ALL(DimProduct) -- Remove product filter
    )
)

-- YoY Growth %
YoY Growth % = DIVIDE(
    [Total Revenue] - [Revenue PY],
    [Revenue PY]
)
```

4

Conditional Logic

```
-- IF Function
Performance Status = IF(
    [Gross Margin %] >= 0.35, "Excellent",
    IF([Gross Margin %] >= 0.30, "Good",
    IF([Gross Margin %] >= 0.25, "Fair",
    "Poor"
)))

-- SWITCH Function (Cleaner)
Performance Status = SWITCH(
    TRUE(),
    [Gross Margin %] >= 0.35, "Excellent",
    [Gross Margin %] >= 0.30, "Good",
    [Gross Margin %] >= 0.25, "Fair",
    "Poor" -- Default
)
```

5

Error Handling

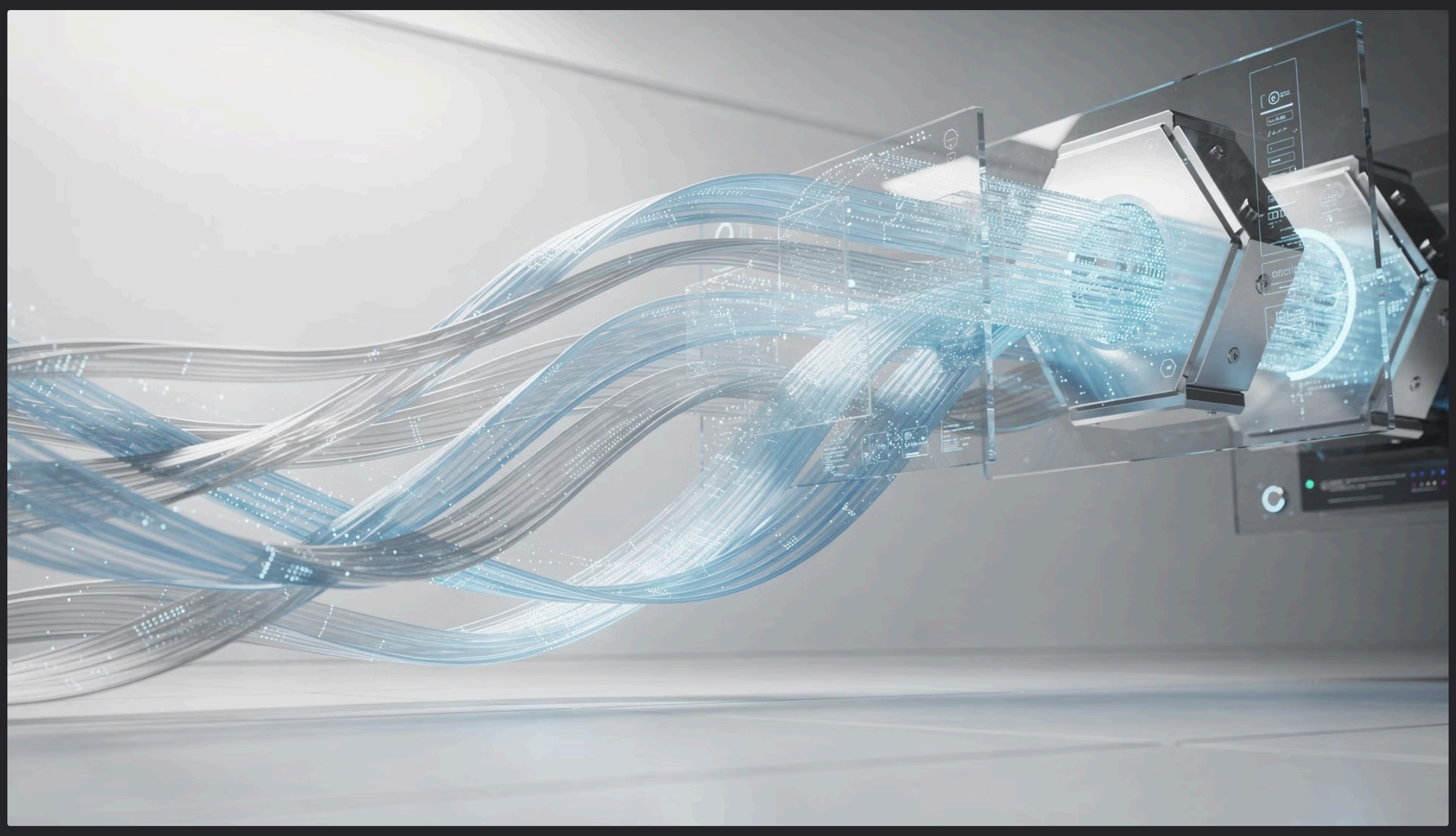
```
-- DIVIDE Function (Always use this!)
Margin % = DIVIDE(
    SUM(FactSales[Revenue]) - SUM(FactSales[Cost]),
    SUM(FactSales[Cost]),
    0 -- Alternate result if division by zero
)

-- IFERROR Function
Budget Variance % = IFERROR(
    DIVIDE(
        [Total Revenue] - [Budget Revenue],
        [Budget Revenue]
    ),
    BLANK()
)
```

Always use DIVIDE, never / operator - DIVIDE handles division by zero gracefully

CALCULATE - The Most Powerful Function

Modifies filter context before evaluating expression

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



01

Basic Usage

```
-- NSW revenue only (ignores State slicer)
NSW Revenue = CALCULATE(
    [Total Revenue],
    DimStore[State] = "NSW"
)
```

02

Multiple Filters (AND logic)

```
-- NSW AND Dairy
NSW Dairy Revenue = CALCULATE(
    [Total Revenue],
    DimStore[State] = "NSW",
    DimProduct[Category] = "Dairy"
)
```

03

Removing Filters

```
-- Total revenue across ALL states
Total Revenue All States = CALCULATE(
    [Total Revenue],
    ALL(DimStore[State])
)

-- Usage: % of total
Revenue % of Total = DIVIDE(
    [Total Revenue],
    CALCULATE([Total Revenue], ALL(DimProduct))
)
```

04

Australian FY Example

```
-- Current FY Revenue
Current FY Revenue =
VAR FYStart = IF(
    MONTH(TODAY()) >= 7,
    DATE(YEAR(TODAY()), 7, 1),
    DATE(YEAR(TODAY()) - 1, 7, 1)
)
VAR FYEnd = DATE(YEAR(FYStart) + 1, 6, 30)
RETURN
CALCULATE(
    [Total Revenue],
    DimDate[Date] >= FYStart,
    DimDate[Date] <= FYEnd
)
```

FILTER Function

What: Creates filtered table for use in CALCULATE

Basic Filter

```
-- High margin products (>35%)
High Margin Revenue = CALCULATE(
    [Total Revenue],
    FILTER(
        DimProduct,
        DimProduct[MarginPercent] > 0.35
    )
)
```

Multiple Conditions

```
-- Premium products
Premium Revenue = CALCULATE(
 [Total Revenue],
 FILTER(
 DimProduct,
 DimProduct[MarginPercent] > 0.35
 && DimProduct[UnitPrice] > 50
 )
)
```

ALL, ALLEXCEPT, ALLSELECTED

Controlling Filter Context

ALL - Remove All Filters

```
-- Grand total (ignores all product filters)
Grand Total Revenue = CALCULATE(
    [Total Revenue],
    ALL(DimProduct)
)

-- % of total
Revenue % = DIVIDE(
    [Total Revenue],
    CALCULATE([Total Revenue], ALL(DimProduct))
)
```

ALLEXCEPT - Keep Some Filters

```
-- Remove all filters EXCEPT category
Revenue % of Category =
VAR CategoryTotal = CALCULATE(
    [Total Revenue],
    ALLEXCEPT(DimProduct, DimProduct[Category])
)
RETURN DIVIDE([Total Revenue], CategoryTotal)

-- Example:
-- Dairy total: $8.5M
-- Milk: $2.8M / $8.5M = 32.9%
```

ALLSELECTED - Respect Visual Filters

```
-- % of what user sees
Revenue % of Visible = DIVIDE(
    [Total Revenue],
    CALCULATE(
        [Total Revenue],
        ALLSELECTED(DimProduct)
    )
)

-- If user filters to Dairy only:
-- Denominator = Dairy total, not all products
```

Comparison Table

Function	Effect	Use Case
ALL	Ignores ALL filters	Grand total, % of dataset
ALLEXCEPT	Keeps specific filters	% of category/state
ALLSELECTED	Respects visual filters	% of what's visible

Boolean Filters

AND Logic

```
-- Multiple filters (implicit AND)
NSW Dairy = CALCULATE(
    [Total Revenue],
    DimStore[State] = "NSW",
    DimProduct[Category] = "Dairy"
)
```

OR Logic

```
-- IN operator
East Coast Revenue = CALCULATE(
    [Total Revenue],
    DimStore[State] IN {"NSW", "VIC", "QLD"}
)
```

NOT Logic

```
-- Exclude Dairy
Non-Dairy Revenue = CALCULATE(
 [Total Revenue],
 DimProduct[Category] <> "Dairy"
)
```

Time Intelligence

Australian Financial Year (July 1 - June 30)

YTD (Year-to-Date)

```
Revenue YTD = TOTALYTD(
[Total Revenue],
DimDate[Date],
"6-30" -- FY ends June 30
)

-- Example:
-- Current date: December 15, 2024
-- FY24/25 started: July 1, 2024
-- YTD: July 1 - December 15 (5.5 months)
```

QTD (Quarter-to-Date)

```
Revenue QTD =
VAR LastDate = MAX(DimDate[Date])
VAR CurrentQuarter = MAX(DimDate[FY Quarter])
VAR QStart = SWITCH(
CurrentQuarter,
1, DATE(YEAR(LastDate), 7, 1), -- Jul
2, DATE(YEAR(LastDate), 10, 1), -- Oct
3, DATE(YEAR(LastDate) + 1, 1, 1), -- Jan
4, DATE(YEAR(LastDate) + 1, 4, 1) -- Apr
)
RETURN
CALCULATE(
[Total Revenue],
DimDate[Date] >= QStart,
DimDate[Date] <= LastDate
)
```

1 2 3

MTD (Month-to-Date)

```
Revenue MTD = TOTALMTD(
[Total Revenue],
DimDate[Date]
)
```

Year-over-Year (YoY) Comparisons

Prior Year Revenue

```
Revenue PY = CALCULATE(
    [Total Revenue],
    SAMEPERIODLASTYEAR(DimDate[Date])
)

-- Alternative
Revenue PY = CALCULATE(
    [Total Revenue],
    DATEADD(DimDate[Date], -1, YEAR)
)
```

YoY Growth %

```
YoY Growth % =
VAR Current = [Total Revenue]
VAR Prior = [Revenue PY]
RETURN DIVIDE(Current - Prior, Prior)

-- Example:
-- FY24/25: $45M
-- FY23/24: $41.5M
-- Growth: ($45M - $41.5M) / $41.5M = 8.4%
```

Rolling Averages

1 2 3

30-Day Moving Average

```
Revenue 30-Day Avg = AVERAGEX(
    DATESINPERIOD(
        DimDate[Date],
        LASTDATE(DimDate[Date]),
        -30,
        DAY
    ),
    [Total Revenue]
)
```

90-Day Moving Average

```
Revenue 90-Day Avg = AVERAGEX(
    DATESINPERIOD(
        DimDate[Date],
        LASTDATE(DimDate[Date]),
        -90,
        DAY
    ),
    [Total Revenue]
)
```

12-Month Moving Average

```
Revenue 12-Month Avg = AVERAGEX(
    DATESINPERIOD(
        DimDate[Date],
        LASTDATE(DimDate[Date]),
        -12,
        MONTH
    ),
    [Total Revenue]
)
```

Usage: Smooth daily volatility into trend lines

Advanced Patterns & Best Practices

1

RANKX - Ranking

```
-- Product revenue rank
Product Revenue Rank = RANKX(
    ALL(DimProduct[ProductName]),
    [Total Revenue],
    ,
    DESC, -- Highest = 1
    DENSE -- No gaps: 1,2,2,3
)

-- Rank within category
Product Rank in Category = RANKX(
    ALLEXCEPT(DimProduct, DimProduct[Category]),
    [Total Revenue],
    ,
    DESC,
    DENSE
)
```

2

TOPN - Top N Selection

```
-- Top 10 products revenue
Top 10 Revenue = CALCULATE(
    [Total Revenue],
    TOPN(
        10,
        ALL(DimProduct[ProductName]),
        [Total Revenue],
        DESC
    )
)
```

3

SUMX - Iterator Functions

```
-- Calculate profit (if not stored)
Total Profit = SUMX(
    FactSales,
    FactSales[Revenue] - FactSales[Cost]
)

-- Weighted average price
Avg Weighted Price = DIVIDE(
    SUMX(FactSales,
        FactSales[Quantity] * FactSales[UnitPrice]
    ),
    SUM(FactSales[Quantity])
)
```

Performance: SUM > SUMX (SUMX iterates row-by-row, slower)

4

Variables (VAR)

```
-- GOOD: Calculate once
Margin % =
VAR Revenue = [Total Revenue]
VAR Cost = [Total Cost]
VAR Profit = Revenue - Cost
RETURN DIVIDE(Profit, Revenue)

-- BAD: Calculates [Total Revenue] twice
Margin % = DIVIDE(
    [Total Revenue] - [Total Cost],
    [Total Revenue]
)
```

Benefits: Performance, Readability, Easier debugging

5

Budget Variance

```
-- Budget measures
Budget Revenue = SUM(DimBudget[BudgetRevenue])

Budget Variance $ = [Total Revenue] - [Budget Revenue]

Budget Variance % = DIVIDE(
 [Budget Variance $],
 [Budget Revenue]
)

Budget Status = SWITCH(
 TRUE(),
 [Budget Achievement %] >= 1.00, "
```