# DAX expressions (core KPIs & useful measures)

I’ve grouped them by purpose and included short notes on assumptions (table/column names).  
**Assumptions** (adjust names to match your model):

* Sales Transactions table = Sales with columns: Quantity, UnitPrice, DiscountAmount, CostPerUnit, TransactionDateTime, PromotionID
* Products table = Products with SKU, StandardCost, ListPrice, MinReorderLevel
* Inventory table = Inventory with StoreID, SKU, InventoryDate, ClosingStock
* Calendar table = Calendar with Date (marked as Date table)
* Promotions table = Promotions with PromotionID, StartDate, EndDate, Budget

If your table/column names differ, replace them accordingly.

**Revenue & Sales**

-- 1) Gross Sales (Quantity \* UnitPrice)

Gross Sales = SUMX( Sales, Sales[Quantity] \* Sales[UnitPrice] )

-- 2) Total Discount

Total Discount = SUM( Sales[DiscountAmount] )

-- 3) Net Sales (after discounts, before tax/shipping)

Net Sales = [Gross Sales] - [Total Discount]

-- 4) Units Sold

Units Sold = SUM( Sales[Quantity] )

**Cost & Margin**

-- 5) Cost of Goods Sold (COGS) using CostPerUnit if available

COGS = SUMX( Sales, Sales[Quantity] \* COALESCE( Sales[CostPerUnit], RELATED(Products[StandardCost]) ) )

-- 6) Gross Profit

Gross Profit = [Net Sales] - [COGS]

-- 7) Gross Margin %

Gross Margin % =

DIVIDE( [Gross Profit], [Net Sales], 0 )

**Time Intelligence (requires Calendar[Date] relationship to Sales TransactionDate)**

-- 8) Sales YTD

Sales YTD =

CALCULATE( [Net Sales],

DATESYTD( Calendar[Date], "03/31" ) -- adjust fiscal year end if needed, or remove 2nd arg for calendar YTD

)

-- 9) Sales Last Year

Sales LY =

CALCULATE( [Net Sales],

SAMEPERIODLASTYEAR( Calendar[Date] )

)

-- 10) Sales MoM % Change (month-over-month %)

Sales MoM % =

VAR ThisMonth = [Net Sales]

VAR PrevMonth = CALCULATE( [Net Sales], PARALLELPERIOD( Calendar[Date], -1, MONTH ) )

RETURN DIVIDE( ThisMonth - PrevMonth, PrevMonth, 0 )

**Inventory KPIs**

-- 11) Average Daily Sales (useful for Days of Supply) -- for selected SKU / Store

Avg Daily Sales =

VAR Days =

MAX( 1, COUNTROWS( VALUES( Calendar[Date] ) ) ) -- number of days in current filter context

VAR TotalUnits = [Units Sold]

RETURN DIVIDE( TotalUnits, Days, 0 )

-- 12) Days of Supply = ClosingStock / AvgDailySales

Days of Supply =

VAR ClosingStock = SUM( Inventory[ClosingStock] )

VAR AvgDaily = [Avg Daily Sales]

RETURN IF( AvgDaily > 0, DIVIDE( ClosingStock, AvgDaily ), BLANK() )

-- 13) Inventory Turnover (COGS / Avg Inventory)

-- NOTE: You need AvgInventory (e.g., average ClosingStock over period). Below computes avg closing stock in filter context.

Average Inventory = AVERAGEX( VALUES( Inventory[InventoryDate] ), SUM( Inventory[ClosingStock] ) )

Inventory Turnover =

DIVIDE( [COGS], [Average Inventory], BLANK() )

**Stock health & Reorder**

-- 14) % of SKUs below MinReorderLevel (across current store/context)

SKUs Below Reorder % =

VAR TotalSKUs = DISTINCTCOUNT( Inventory[SKU] )

VAR SKUsBelow =

CALCULATE(

DISTINCTCOUNT( Inventory[SKU] ),

FILTER(

SUMMARIZE( Inventory, Inventory[SKU], "Avail", SUM( Inventory[ClosingStock] ) ),

[Avail] <= RELATED( Products[MinReorderLevel] )

)

)

RETURN DIVIDE( SKUsBelow, TotalSKUs, 0 )

-- 15) Reorder Recommended (flag measure for a selected SKU/store)

Reorder Recommended =

VAR Avail = SUM( Inventory[ClosingStock] )

VAR MinRL = SELECTEDVALUE( Products[MinReorderLevel], 0 )

RETURN IF( Avail <= MinRL, 1, 0 )

**Promotions & Lift**

-- 16) Promo Sales (sales lines tied to a promotion)

Promo Sales =

CALCULATE( [Net Sales], NOT( ISBLANK( Sales[PromotionID] ) ) )

-- 17) Non-Promo Sales

Non Promo Sales =

CALCULATE( [Net Sales], ISBLANK( Sales[PromotionID] ) )

-- 18) Promotion Lift % -- (PromoSales - Baseline) / Baseline

-- Baseline approach example: average non-promo sales for same sku/category in comparable prior period

Promotion Lift % =

VAR PromoTotal = [Promo Sales]

VAR Baseline = CALCULATE( [Non Promo Sales], SAMEPERIODLASTYEAR( Calendar[Date] ) ) -- simple baseline; replace with better baseline if available

RETURN DIVIDE( PromoTotal - Baseline, Baseline, 0 )

**Customer measures (if Customer table exists)**

-- 19) Active Customers (customers with a purchase in last N days, default 90)

Active Customers (90d) =

VAR Cutoff = TODAY() - 90

RETURN

CALCULATE(

DISTINCTCOUNT( Sales[CustomerID] ),

FILTER( Sales, Sales[TransactionDateTime] >= Cutoff )

)

-- 20) Average Order Value (AOV)

Average Order Value =

DIVIDE( [Net Sales], DISTINCTCOUNT( Sales[TransactionID] ), 0 )

**Operational & Quality checks**

-- 21) Stockout Rate = % of store-SKU-date combos with zero available stock

Stockout Rate =

VAR TotalSnapshots = COUNTROWS( Inventory )

VAR ZeroStock = CALCULATE( COUNTROWS( Inventory ), Inventory[ClosingStock] = 0 )

RETURN DIVIDE( ZeroStock, TotalSnapshots, 0 )

-- 22) Margin per Unit (useful in visuals)

Margin per Unit =

DIVIDE( [Gross Profit], [Units Sold], 0 )

**Tips & best practices**

1. Use SUMX when per-row arithmetic is required (e.g., Quantity \* UnitPrice).
2. Prefer COALESCE or IF to fallback to related product costs if Sales[CostPerUnit] is null.
3. Mark your Calendar table as the model date table and create a single direction relationship from Calendar[Date] to transaction date (or use inactive relationships + USERELATIONSHIP where necessary).
4. For performance, keep grain consistent — line-level sales measures work best when model relationships are optimized and you avoid iterating huge tables unnecessarily.
5. For period comparisons prefer DATESINPERIOD, SAMEPERIODLASTYEAR, or PARALLELPERIOD depending on business need.
6. Validate measures with small sample slices before applying complex filters.

# Prepare & load data for Power BI without writing SQL

Below are, copy-pasteable ways to prepare, transform, and combine your 8 datasets using tools and languages that don’t require SQL (Power Query / M, Excel/CSV, connectors, APIs, simple Python/R where convenient). Each item is a self-contained recipe you can run now in Power BI (Power Query), Excel, or a scripting environment.

**1) Ingest CSV / Excel / Google Sheets**

* Place Sales.csv, Products.xlsx, Inventory.csv, etc. in a folder or OneDrive/SharePoint.
* In Power BI Desktop: **Get Data → Folder** (or Excel / Web → Google Sheets link / SharePoint Folder).
* Power Query will combine files in a folder automatically; use **Transform Data** to shape.

Short tip: Put all daily inventory CSVs in a folder and use Folder connector to append them automatically.

**2) Use Power Query (M) to join Sales (lines) to Products (master) — M snippet**

(Load both files via Get Data → Transform Data, then in Advanced Editor paste/adjust this.)

let

// Load Sales and Products from existing queries named "Sales" and "Products"

SalesSource = Sales,

ProductsSource = Products,

// Ensure keys are same type

SalesTyped = Table.TransformColumnTypes(SalesSource, {{"SKU", type text}, {"TransactionDateTime", type datetime}}),

ProductsTyped = Table.TransformColumnTypes(ProductsSource, {{"SKU", type text}}),

// Merge (left join) Products into Sales

Merged = Table.NestedJoin(SalesTyped, "SKU", ProductsTyped, "SKU", "ProductDetails", JoinKind.LeftOuter),

// Expand selected product columns (StandardCost, Category, Brand)

Expanded = Table.ExpandRecordColumn(Merged, "ProductDetails", {"ProductName","Category","StandardCost","ListPrice"}, {"ProductName","Category","StandardCost","ListPrice"}),

// Add GrossSales and NetSales columns

AddedGross = Table.AddColumn(Expanded, "GrossSales", each [Quantity] \* [UnitPrice], type number),

AddedNet = Table.AddColumn(AddedGross, "NetSales", each [GrossSales] - (if [DiscountAmount] = null then 0 else [DiscountAmount]), type number)

in

AddedNet

Use this to create a clean, denormalized Sales table for Power BI visuals — no SQL needed.

**3) Build a Date/Calendar table in Power Query (M)**

Paste into Advanced Editor as a new blank query called Calendar and adjust dates:

let

StartDate = #date(2018,1,1),

EndDate = Date.From(DateTime.LocalNow()) + 365, // up to one year ahead

NumDays = Duration.Days(EndDate - StartDate),

Dates = List.Dates(StartDate, NumDays+1, #duration(1,0,0,0)),

TableDates = Table.FromList(Dates, Splitter.SplitByNothing(), {"Date"}, null, ExtraValues.Error),

WithFields = Table.TransformColumns(TableDates, {{"Date", each \_, type date}}),

AddDay = Table.AddColumn(WithFields, "Day", each Date.Day([Date]), Int64.Type),

AddMonth = Table.AddColumn(AddDay, "Month", each Date.Month([Date]), Int64.Type),

AddMonthName = Table.AddColumn(AddMonth, "MonthName", each Date.ToText([Date], "MMMM"), type text),

AddYear = Table.AddColumn(AddMonthName, "Year", each Date.Year([Date]), Int64.Type),

AddQuarter = Table.AddColumn(AddYear, "Quarter", each Date.QuarterOfYear([Date]), Int64.Type),

AddDayOfWeek = Table.AddColumn(AddQuarter, "DayOfWeek", each Date.DayOfWeek([Date])+1, Int64.Type),

AddIsWeekend = Table.AddColumn(AddDayOfWeek, "IsWeekend", each if Date.DayOfWeek([Date]) in {5,6} then true else false)

in

AddIsWeekend

Mark this Calendar table as the Date table in Power BI (Modeling → Mark as date table).

**4) Compute ClosingStock from Inventory movements without SQL**

If you have daily movement snapshots or line events, Power Query can aggregate:

let

Inv = Inventory, // raw inventory movements with columns: StoreID, SKU, InventoryDate, OpeningStock, StockReceived, StockSold, StockReturned, StockAdjusted

Typed = Table.TransformColumnTypes(Inv, {{"InventoryDate", type date}, {"OpeningStock", Int64.Type}, {"StockReceived", Int64.Type}, {"StockSold", Int64.Type}, {"StockReturned", Int64.Type}, {"StockAdjusted", Int64.Type}}),

AddedClosing = Table.AddColumn(Typed, "ClosingStock", each [OpeningStock] + [StockReceived] - [StockSold] + [StockReturned] - [StockAdjusted], Int64.Type),

// optionally group by StoreID, SKU, InventoryDate to ensure single row per snapshot

Grouped = Table.Group(AddedClosing, {"StoreID","SKU","InventoryDate"}, {{"ClosingStock", each List.Sum([ClosingStock]), type number}})

in

Grouped

This yields a daily closing stock per SKU per store ready for DaysOfSupply calculations (DAX or additional M).

**5) Promotion attribution without SQL — Power Query + M approach**

* Expand PromotionID in Sales (if list or code).
* Create a merged table of Sales × Promotions (left join) using PromotionID.
* Add IsPromo = if PromotionID = null then false else true.
* To measure baseline, compute non-promo rolling averages using DAX later — but you can approximate baseline in Power Query: group past N weeks of non-promo sales by SKU and compute average.

Simple M grouping to get non-promo averages:

let

SalesTbl = Sales, // with PromotionID and NetSales

NonPromo = Table.SelectRows(SalesTbl, each ([PromotionID] = null)),

// convert to date only

NonPromoDate = Table.TransformColumns(NonPromo, {{"TransactionDateTime", DateTime.Date}}),

Grouped = Table.Group(NonPromoDate, {"SKU","TransactionDateTime"}, {{"DailyNet", each List.Sum([NetSales]), type number}}),

// average per SKU across entire history

SKUAvg = Table.Group(Grouped, {"SKU"}, {{"AvgDailyNonPromoSales", each List.Average([DailyNet]), type number}})

in

SKUAvg

Join SKUAvg back to promotions to estimate lift.

**6) Create calculated columns / measures in Power Query (when DAX not desired)**

Although DAX is standard for measures, you can precompute many derived columns in Power Query (like GrossSales, NetSales, Margin per line). This reduces model complexity and is useful when you want static columns:

// Example: add MarginPerLine

Table.AddColumn(YourTable, "MarginPerLine", each ([NetSales] - ([Quantity] \* if [CostPerUnit] <> null then [CostPerUnit] else [StandardCost])), type number)

But note: heavy aggregation is best in DAX for performance; Power Query precomputations are fine for demo or small datasets.

**7) Use connectors & APIs (no SQL): Shopify, Stripe, Google Analytics, CSV, Excel, SharePoint, Azure Blob**

* Power BI has built-in connectors: **Web (API)**, **Google Sheets**, **SharePoint Folder**, **OneDrive for Business**, **Azure Blob Storage**, **Dynamics 365**, **Salesforce**, **Shopify** etc.
* Example: call a REST API returning JSON (Transactions) and expand the JSON into table rows — no DB needed.

Power Query sample to call an API (replace URL & headers):

let

Source = Json.Document(Web.Contents("https://api.example.com/transactions", [Headers=[Authorization="Bearer YOURTOKEN"]])),

Records = Source[transactions],

ToTable = Table.FromList(Records, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

Expanded = Table.ExpandRecordColumn(ToTable, "Column1", {"id","sku","quantity","unit\_price","timestamp"}, {"TransactionID","SKU","Quantity","UnitPrice","TransactionDateTime"})

in

Expanded

**8) Use small Python or R scripts (for heavy transforms) — still “no SQL”**

If you prefer code, Power BI allows **Python** and **R** scripts as data sources and transforms. Example: load multiple Excel sheets, pivot/unpivot, do advanced joins, then output a dataframe back to Power BI.

Python mini-example (run in Power BI Get Data → Python script):

import pandas as pd

# assume files are in a folder; use pandas to read and merge

sales = pd.read\_csv(r'C:\data\Sales.csv', parse\_dates=['TransactionDateTime'])

products = pd.read\_excel(r'C:\data\Products.xlsx', sheet\_name='Products')

sales = sales.merge(products[['SKU','StandardCost','Category']], on='SKU', how='left')

sales['GrossSales'] = sales['Quantity'] \* sales['UnitPrice']

sales['NetSales'] = sales['GrossSales'] - sales['DiscountAmount'].fillna(0)

result = sales

Power BI will import result as a table. (No SQL; just Python.)

**9) Automate ETL with Power Automate / Azure Data Factory (no SQL)**

* For scheduled refreshes, use OneDrive/SharePoint sync + Power BI scheduled refresh.
* Or use **Power Automate** to fetch CSVs from email attachments and save to OneDrive.
* For more robust orchestration (still without databases), use **Azure Data Factory** to copy files from SFTP → Blob → Power BI dataset refresh.

**10) Data quality & validation techniques (no DB)**

* Use Power Query steps to: remove duplicates (Table.Distinct), fill missing values (Table.FillDown or List.First), change types, trim texts.
* Create a **validation report** table in Power Query: rows with null keys, negative quantities, negative prices — present that as a report page for data cleansing before modeling.

Example M: find sales rows with missing SKU or zero Quantity:

let

SalesSrc = Sales,

Issues = Table.SelectRows(SalesSrc, each ([SKU] = null or [Quantity] = 0 or [UnitPrice] <= 0)),

CountIssues = Table.RowCount(Issues)

in

Issues

**11) Packaging for Power BI model (recommended final steps)**

1. Load transformed, denormalized Sales (with product columns) into the model.
2. Load Products, Inventory (daily snapshots), Calendar, Stores, Customers, Promotions, Suppliers.
3. Create relationships in the model view (one-to-many): Calendar[Date] → Sales[TransactionDate], Calendar[Date] → Inventory[InventoryDate], Products[SKU] → Sales[SKU], Products[SKU] → Inventory[SKU], Stores[StoreID] → Sales[StoreID].
4. Use DAX measures (from Answer 10) for aggregations — keeping raw numbers computed in Power Query where appropriate speeds up visuals for very large demo tables.
5. Turn on incremental refresh in Power BI Premium / Pro (if dataset is large) — this is a Power BI feature, not SQL.

**Quick checklist you can follow now (one-minute action plan)**

* Put source files (CSV/Excel) in a folder or OneDrive.
* Open Power BI Desktop → Get Data → Folder → select folder.
* In Power Query: apply the M snippets above to merge Sales→Products, compute Gross/Net, build Calendar.
* Load Inventory snapshots and use the ClosingStock computation.
* Create relationships, add DAX measures from Answer 10, build visuals.

If you want, I can:

* 1. Generate sample CSV files you can download and load into Power BI, or
  2. Paste a complete Power Query script that builds the whole denormalized model from example CSV filenames (Sales.csv, Products.csv, Inventory.csv, Stores.csv, Customers.csv, Suppliers.csv, Promotions.csv, Calendar).