



Skill Navora

TRACK 3

Power BI

ETL with Power Query, star-schema modelling, DAX measures and time intelligence, interactive reports, case studies, publishing, security and Copilot.

6 lessons

Version 2026.10

Updated October 2026

Created by **Varun Kumar**

www.skillnavora.in

Contents

Every entry is a clickable link. The PDF also has bookmarks for each lesson and section.

- [01 Getting Started with Power BI & Your First HR Report](#)
- [02 Power Query, Star Schema & Data Modelling \(Adventure Cycles\)](#)
- [03 DAX Fundamentals: Measures, Filter Context & CALCULATE](#)
- [04 DAX Time Intelligence, Ranking, Parameters & Dynamic Measures](#)
- [05 Visuals, Filters, Drill-through, Tooltips, Buttons & Report Design](#)
- [06 Power BI Service, Connectivity Modes, Security & Capstone](#)

Getting Started with Power BI & Your First HR Report

Understand what Power BI is and how its parts fit together, install Power BI Desktop, tour the three views, follow the ETL and analytics stages, and build a complete first report on HR data: cards, bar and column charts, a donut, slicers, a year column with DAX, groups and small multiples.

● Power BI class 1

✦ Power BI Desktop 2026 (Copilot, Fabric, PBIP)

Beginner

🕒 120 min

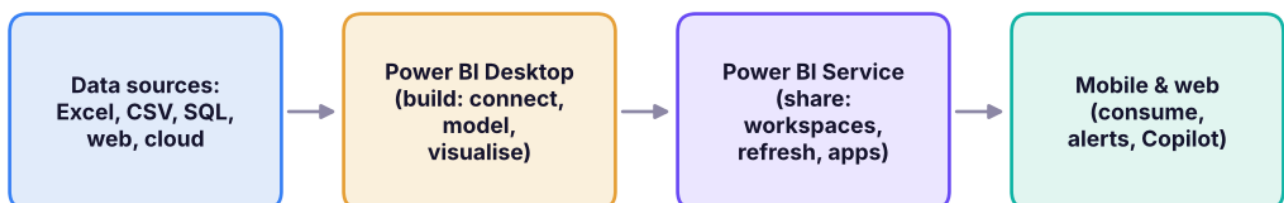
By the end of this lesson you can

- Describe Power BI Desktop, the Service and Mobile, and how data flows between them
- Explain ETL and the stages of the data-analytics process, and map them to Power BI tools
- Navigate Report, Table and Model views, and the Fields, Visualizations and Filters panes
- Load a CSV, check data types and build cards, bar/column charts, a matrix, a donut and slicers
- Add a calculated column with DAX (YEAR), create data groups, and use small multiples

What is Power BI?

Power BI is Microsoft's business-intelligence platform: it connects to data, cleans and models it, and turns it into interactive reports and dashboards that people can share. In 2026 it is part of **Microsoft Fabric**, Microsoft's all-in-one analytics platform, and it is one of the most requested tools in analyst job posts, alongside Excel, SQL and Tableau.

Power BI architecture



The Power BI components. You build in Desktop, publish to the Service, and consume on the web or mobile.

Component	What it is	Who uses it
Power BI Desktop	Free Windows application to build data models and reports (<code>.pbix</code>)	Analysts, developers
Power BI Service (app.powerbi.com)	Cloud site to publish, share, schedule refresh, create dashboards and apps	Analysts and report consumers
Power BI Mobile	iOS/Android apps to view reports and get alerts	Managers on the go
Power BI Report Builder	Paginated, print-ready reports (invoices, statements)	Report developers
On-premises data gateway	Secure bridge so the cloud can refresh data from your own servers	Admins
Power BI Report Server	Self-hosted option for organisations that cannot use the cloud	IT
Copilot	AI assistant that writes DAX, builds report pages and summarises insights (needs a Fabric/Premium capacity)	Everyone

TIP Report vs dashboard

In Power BI a **report** is one or more pages of interactive visuals built on a data model. A **dashboard** (Service only) is a single page of **pinned tiles** from different reports. Many teams use reports only, or share them as an **app**.

Licences in brief

Licence	Use
Power BI Desktop	Free: build and save locally
Free (Service)	Personal use in "My workspace"; cannot share with others
Pro	Publish and share with other Pro users (about US\$14 per user per month)
Premium Per User / Fabric capacity	Larger models, AI features, share with free users

(Prices change; check Microsoft's current pricing page.)

Installing Power BI Desktop

STEP BY STEP

Install

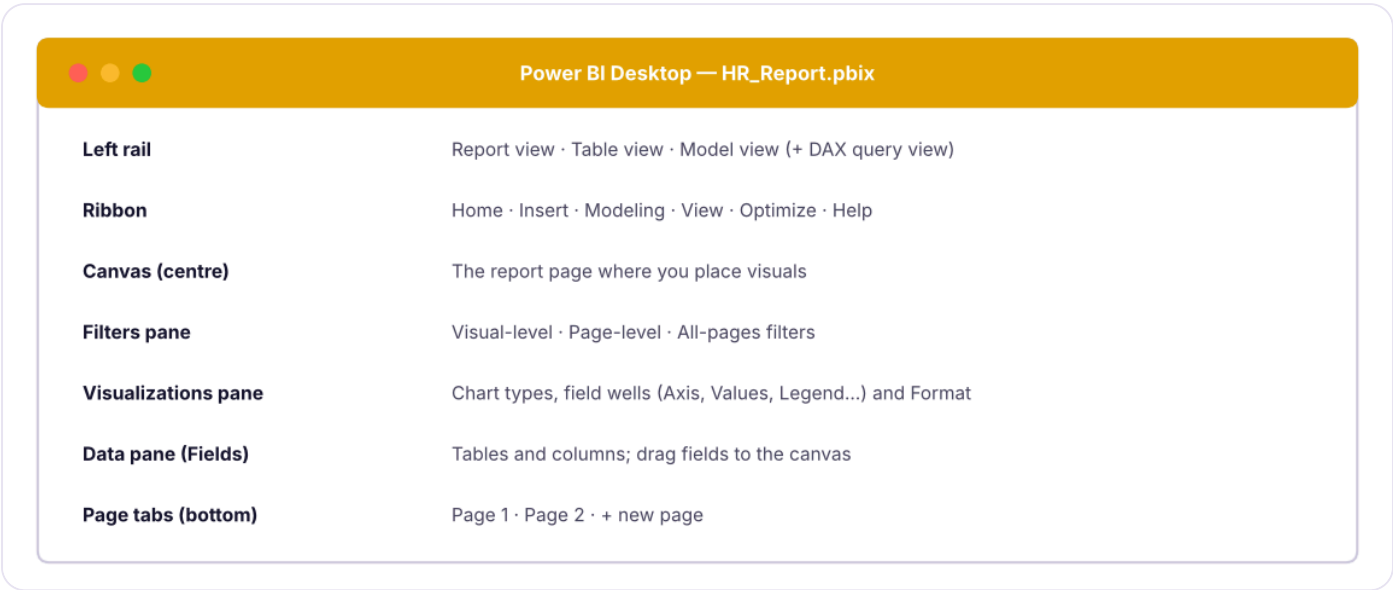
1. **Easiest:** open the **Microsoft Store**, search "Power BI Desktop" and click Install (it updates itself every month).

2. Or download it from the Microsoft Power BI website → Download → Power BI Desktop.

3. Requirements: Windows 10/11 (64-bit), 4 GB+ RAM (8 GB+ recommended). **There is no Mac version:** Mac users can use the Service in a browser, a Windows virtual machine, or Windows 365.

4. Start the app and sign in with a work or school account (optional for building locally).

The three views

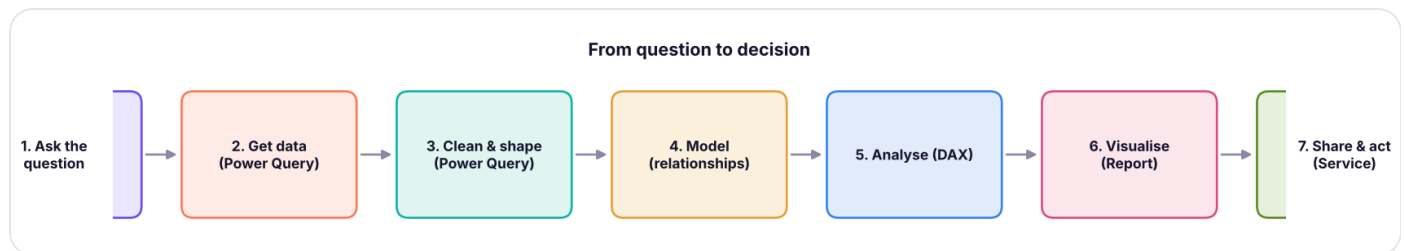


The Power BI Desktop window: left rail with three views, canvas in the middle, panes on the right.

View	Icon	Use it to
Report	bar chart	Design the report pages
Table (Data)	grid	Inspect the data and create calculated columns and measures
Model	diagram	See and edit relationships between tables
DAX query view	{ }	Run DAX queries and test measures

ETL and the analytics process

ETL = **E**xtract (get data from sources), **T**ransform (clean and reshape it), **L**oad (store it in the model). In Power BI, Extract and Transform are done in **Power Query**, and Load puts the result into the data model. (You met Power Query in Excel Lesson 12: it is the same engine.)



The data analytics stages, mapped to Power BI.

Always start with the **question**: "Why are people leaving?" decides which data and visuals you need. The worst Power BI reports are tours of every chart type with no question.

Your first report: HR attrition

CASE STUDY The brief

HR wants a one-page report on the company's 1,500 employees: how many there are, how many left (attrition), where attrition is highest, and whether overtime, job satisfaction or income play a role. Use `hr_employees.csv`.

The data (first rows):

Python

```
import pandas as pd
hr = pd.read_csv("hr_employees.csv", parse_dates=["hire_date"])
print(hr.shape)
hr.head(5)
```

OUTPUT

(1500, 15)

	emp_id	gender	age	department	job_title	education	hire_date	years_at_c
0	E00001	Male	39	Operations	Operations Manager	Bachelor's	2025-05-26	0
1	E00002	Female	45	IT	Tester	Bachelor's	2020-10-17	5
2	E00003	Male	41	IT	Tester	Bachelor's	2019-05-06	6
3	E00004	Male	21	R&D	Data Scientist	Bachelor's	2025-08-23	0
4	E00005	Male	41	Sales	Sales Executive	Master's	2015-08-24	10

5 rows × 15 columns

Step 1: get the data

STEP BY STEP Load the CSV

1. Home → **Get data** → **Text/CSV** → choose `hr_employees.csv`.
2. The preview shows the columns and detected data types. Click **Transform Data** (not Load) to check types in Power Query.
3. Confirm: `age`, `years_at_company`, `monthly_income`, `job_satisfaction`, `performance_rating` are whole numbers; `hire_date` is **Date**; the rest are text.
4. Home → **Close & Apply**.

In the Data pane you now see the table `hr_employees` with a Σ next to numeric fields (they can be summed) and no symbol next to text fields.

Step 2: KPI cards

Drag fields onto the canvas, then choose the **Card** visual:

- **Employees** = Count of `emp_id`
- **Attrition (leavers)**: use a measure (Lesson 3 explains measures): `Leavers = CALCULATE ( COUNTROWS ( hr_employees ), hr_employees[attrition] = "Yes" )`
- **Attrition %** = `DIVIDE ( [Leavers], COUNTROWS ( hr_employees ) )`, formatted as a percentage
- **Average monthly income** = Average of `monthly_income`

Python

```
n = len(hr); leavers = (hr.attrition == "Yes").sum()
print(f"Employees      : {n:,}")
print(f"Leavers         : {leavers:,}")
print(f"Attrition %        : {leavers / n:.1%}")
print(f"Avg income (₹)     : {hr.monthly_income.mean():,.0f}")
```

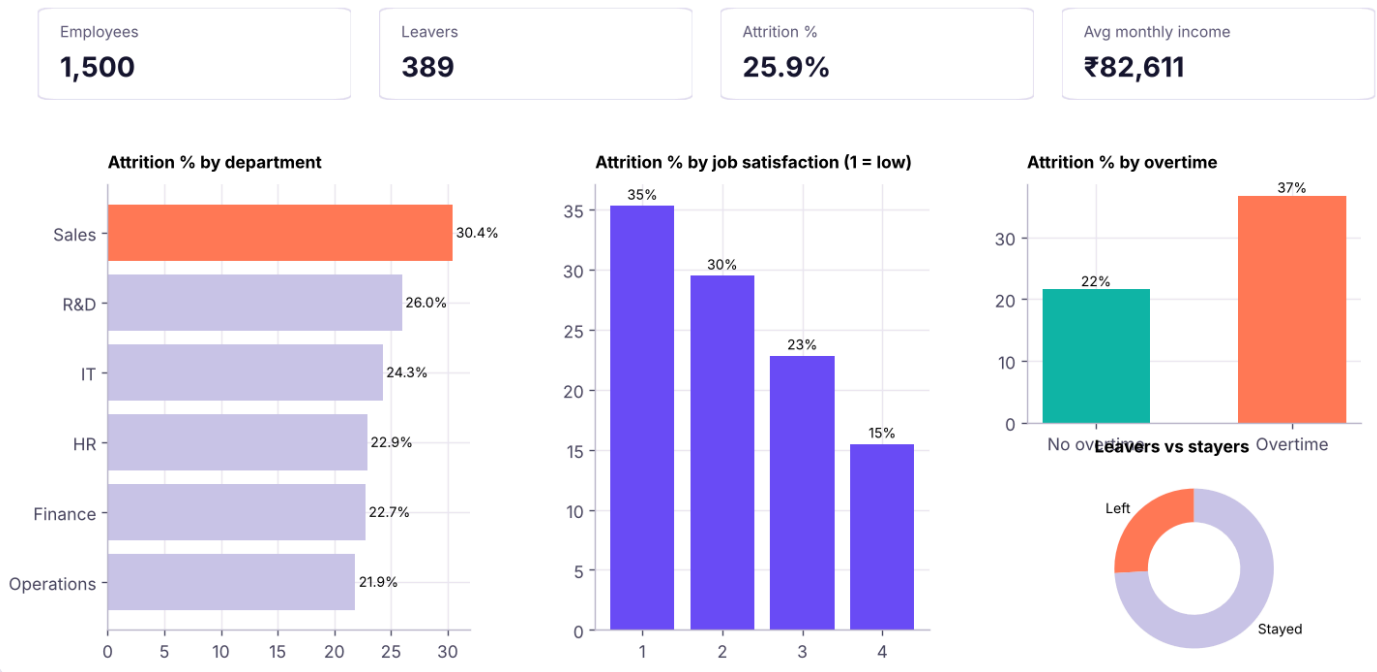
OUTPUT

```
Employees      : 1,500
Leavers         : 389
Attrition %     : 25.9%
Avg income (₹) : 82,611
```

Step 3: bar, column and donut charts

- **Clustered bar chart:** Y-axis department, X-axis = Attrition % → ranks departments.
- **Clustered column chart:** X-axis job_satisfaction, Y-axis Attrition %.
- **Donut chart:** Legend attrition, Values = count of emp_id.
- **Matrix:** Rows department, Columns overtime, Values Attrition % (turns into a heat map with conditional formatting, Lesson 5).
- **Slicers:** gender, education, city.

HR Attrition Overview



The finished HR page, drawn from the real data: KPI cards, attrition by department, by satisfaction and overtime, and the leaver/stayer donut.

Step 4: a calculated column with DAX

We want to see joining trends by year, but the table has only hire_date. Add a **calculated column**:
Table view (or Modeling) → **New column**:

DAX

```
Hire Year = YEAR ( hr_employees[hire_date] )
```

A calculated column is evaluated **row by row when the data loads** and stored in the model, just like a new column in Excel. Now a line/column chart of **headcount by hire year** works:

Python

```
hr["hire_year"] = hr.hire_date.dt.year
by_year = hr.groupby("hire_year").agg(hired=("emp_id", "count"), left=("left", "sum"))
by_year["attrition_%"] = (by_year.left / by_year.hired * 100).round(1)
by_year.tail(8)
```

OUTPUT

	hired	left	attrition_%
hire_year			
2019	82	16	19.5
2020	85	20	23.5
2021	120	22	18.3
2022	136	35	25.7
2023	196	60	30.6
2024	234	71	30.3
2025	284	93	32.7
2026	64	17	26.6

8 rows × 3 columns

Step 5: data groups and bins

Groups combine values into custom categories, such as the "age band" below, without changing the source data:

- **Bins** (numbers): right-click the field → **New group** → Group type *Bin* → bin size 5 → creates age (bins).
- **List groups** (text): select several values → right-click → **Group** → rename, for example group *Mumbai* and *Pune* as "Maharashtra".

For flexible logic, a calculated column is more transparent:

DAX

```
Age Band =
SWITCH (
    TRUE (),
    hr_employees[age] < 25, "Under 25",
    hr_employees[age] < 35, "25-34",
    hr_employees[age] < 45, "35-44",
    "45 and over"
)
```

Python

```
bands = pd.cut(hr.age, [0, 24, 34, 44, 120], labels=["Under 25", "25-34", "35-44", "45 and over"])
hr.groupby(bands, observed=False).agg(employees=("emp_id", "count"), attrition_pct=("left", lambda s: round(s.mean() * 100, 1)))
```

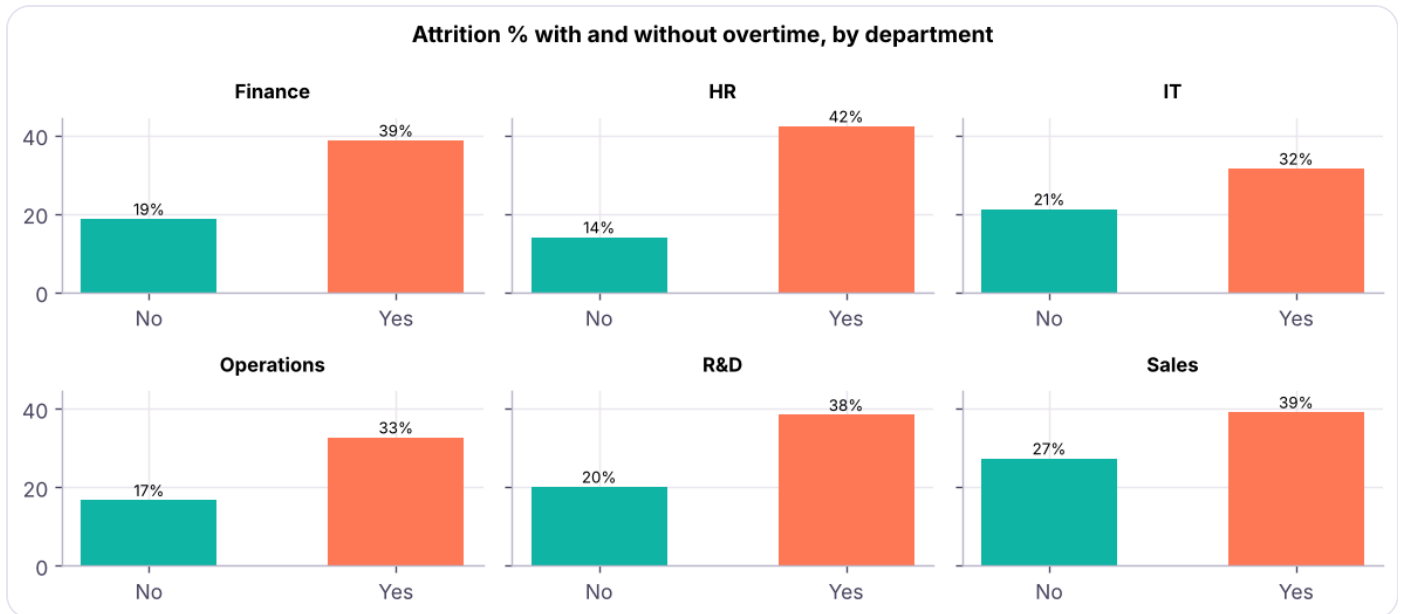
OUTPUT

	employees	attrition_pct
age		
Under 25	148	35.1
25-34	624	26.9
35-44	574	23.5
45 and over	154	22.1

4 rows × 2 columns

Step 6: small multiples

Small multiples split one visual into a grid of identical mini-charts by a field. Drag department into the visual's **Small multiples** well: you get one small column chart of attrition by satisfaction for every department. This lets you compare patterns across groups at a glance without a cluttered legend.



Small multiples: attrition % by overtime for each department. The same pattern repeats everywhere, which says overtime is a company-wide issue, not a one-department issue.

What does the report tell us?

Python

```
t = hr.groupby("overtime").left.agg(["count", "mean"]); t["mean"] = (t["mean"] * 100).round(1)
t.columns = ["employees", "attrition_%"]
print(t.to_string()); print()
low = hr[hr.job_satisfaction ≤ 2].left.mean() * 100; high = hr[hr.job_satisfaction ≥ 3].left.mean() * 100
print(f"Attrition for satisfaction 1-2: {low:.1f}% vs 3-4: {high:.1f}%")
```

OUTPUT

	employees	attrition_%
overtime		
No	1067	21.6
Yes	433	36.7

Attrition for satisfaction 1-2: 32.5% vs 3-4: 19.4%

Insight sentences for a manager: *overtime workers leave far more often than others; low satisfaction scores go with higher attrition; Sales has the highest attrition among departments.* Notice the careful wording: these are **associations** in the data, not proof of cause. (The ML track shows how to test them properly.)

Good habits from day one

1. **Name things clearly:** Attrition %, not Measure 3; rename pages ("HR Overview").

2. **Put measures in a dedicated table** later (Lesson 3) and format them (percentage, thousands separator).
3. **Check data types** in Power Query before loading.
4. **One idea per page**; the title should say the insight.
5. **Save often** (.pbix), and use clear version names.
6. **Do not use pie charts** with many slices; prefer bars (ranked) and use donuts only for 2–3 parts.

2026 UPDATE Power BI in 2026

- **Copilot** can create a report page from a prompt ("Build an attrition overview for HR"), write DAX measures and summarise a page. It needs a paid Fabric/Premium capacity and your data to be well-modelled and named.
- **Power BI Project (PBIP)** format stores the report as text files, so teams can use **Git** and code review.
- **TMDL** (Tabular Model Definition Language) and **DAX query view** make models easier to script and test.
- **Direct Lake** (Fabric) reads Delta tables directly with import-like speed. (See Lesson 6.)

Summary

KEY TAKEAWAYS

- Power BI = Desktop (build) + Service (share) + Mobile (consume), now inside Microsoft Fabric.
- ETL: **Extract and Transform in Power Query, Load into the model**; analytics stages run from question to decision.
- Three views: **Report, Table, Model**; build with the Fields, Visualizations and Filters panes.
- A **card** shows a KPI; bar/column charts compare categories; a donut shows 2–3 parts; a matrix is a pivot table; slicers filter.
- A DAX **calculated column** (YEAR, SWITCH) is computed row by row at load time; **groups/bins** categorise without changing the data; **small multiples** repeat a chart per category.

Quick quiz

6 questions · tap an answer to check it

1. In Power BI, where do you perform the Extract and Transform steps of ETL?

A Report view

B Power Query

C The Service

D The Model view

Answer: B. Power Query connects to sources and transforms data; Close & Apply loads it into the model.

2. Which view shows relationships between tables?

A Report view

B Table view

C Model view

D Filters pane

Answer: C. Relationships are created and edited in Model view.

3. Power BI Desktop is available for:

A Windows and Mac

B Windows only

C Mac only

D Browsers only

Answer: B. Desktop is Windows-only; Mac users use the Service or a Windows virtual machine.

4. What does a calculated column do?

- A Runs only when a slicer changes
- B Computes a value for every row when the data is loaded and stores it in the model**
- C Replaces a measure
- D Changes the source file

Answer: B. Calculated columns are row-by-row and stored; measures calculate on the fly.

5. Which visual repeats the same chart for every value of a field?

- A Matrix
- B Slicer
- C Small multiples**
- D Card

Answer: C. Small multiples create a grid of mini-charts split by a category.

6. Which licence do you need to share a report with colleagues who also have Pro licences?

- A None, Desktop is enough
- B Free Service licence
- C Power BI Pro (or Premium Per User)**
- D Gateway licence

Answer: C. Sharing in the Service requires Pro for creators and viewers (unless the workspace is on Premium/Fabric capacity).

EXERCISE Build the HR page

Build the page shown above in Power BI Desktop with `hr_employees.csv`. Then add: (1) a slicer for `city` and `gender`; (2) a matrix of `attrition %` by `job_title` and `education`; (3) an `Age Band` column and a column chart of `attrition %` by band; (4) a title text box that states your main finding.

Show solution (worked answer)

(1) Slicer visual, fields `city`, `gender` (Format → Slicer settings → Style: Tile or Dropdown). (2) Matrix: Rows `job_title`, Columns `education`, Values `Attrition %` (turn on Cell elements → Background colour for a heat map). (3) Use the SWITCH formula in this lesson; chart X-axis `Age Band`, Y-axis `Attrition %`; sort the axis in the right order using a sort-by-column. (4) Insert → Text box, for example "Overtime is linked to about 15 more attrition points company-wide." Verify the claim against your visuals before you publish.

INTERVIEW CORNER Questions you may be asked

Q: What is the difference between Power BI Desktop and Service? Desktop is the free authoring tool for models and reports on Windows; the Service is the cloud platform to publish, share, refresh on a schedule, build dashboards and apps, and apply security.

Q: What is ETL and where is it done in Power BI? Extract, Transform, Load: Power Query connects to sources and applies transformation steps; the result is loaded into the VertiPaq in-memory model.

Q: Calculated column vs measure? A calculated column is evaluated per row at refresh time and stored (uses memory); a measure is evaluated at query time in the current filter context (uses CPU). Use measures for aggregations such as sums and ratios.

Q: Report vs dashboard? A report is multi-page and interactive, built on one model. A dashboard is a single page of pinned tiles from several reports, available only in the Service.

Power Query, Star Schema & Data Modelling (Adventure Cycles)

Prepare and model data the professional way. Clean data in Power Query, load a star schema (fact and dimension tables), create one-to-many relationships, understand filter direction, build and mark a date table, create hierarchies, and avoid the most common modelling mistakes, using the Adventure Cycles case study.

● Power BI class 3

✦ Power BI Desktop 2026

Intermediate

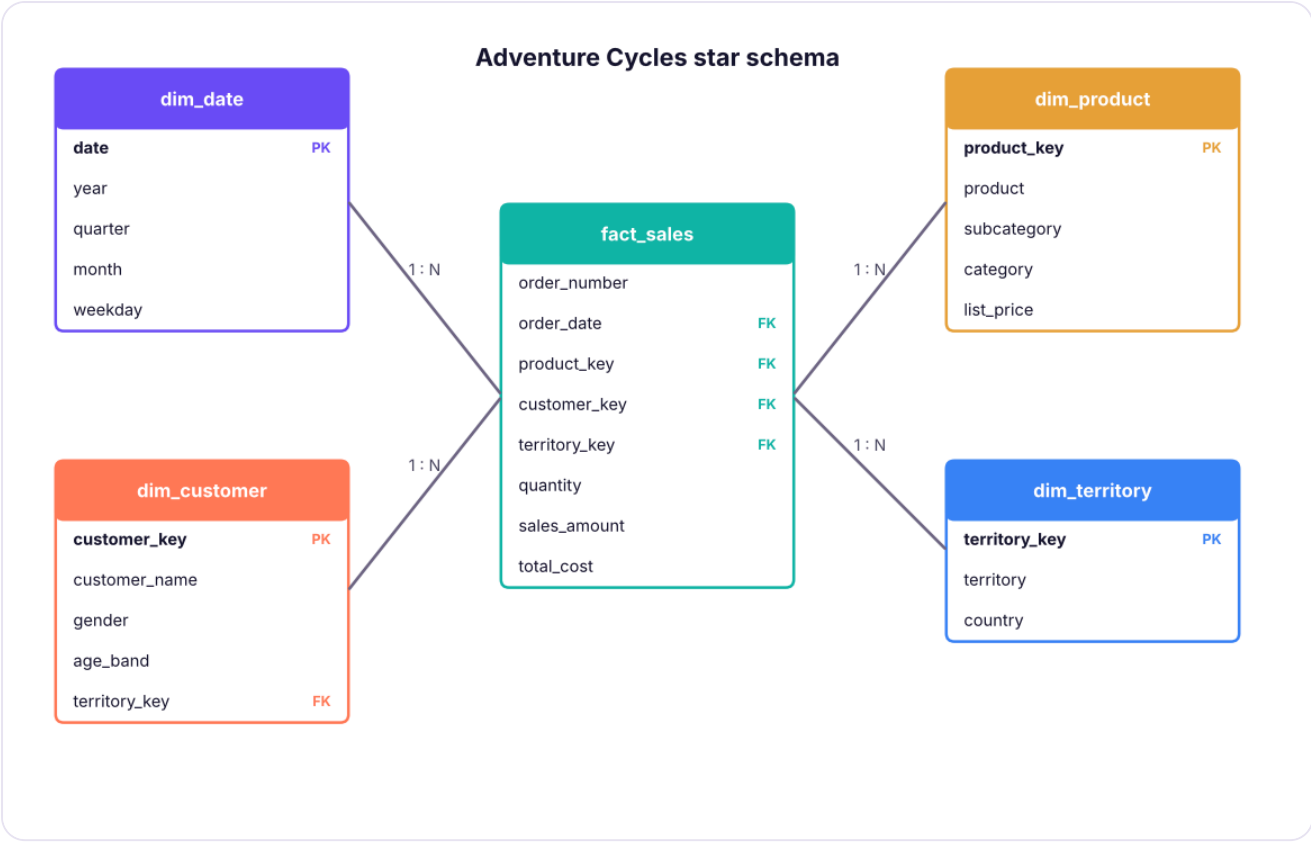
🕒 140 min

By the end of this lesson you can

- Apply common Power Query steps and read the Applied Steps and M code
- Explain fact tables, dimension tables and the star schema, and why they beat one wide table
- Create relationships with the right cardinality and filter direction, and explain how filters flow
- Build a date table, mark it as a date table and add hierarchies
- Diagnose common modelling problems (many-to-many, ambiguous paths, missing dates)

Why modelling matters more than visuals

Beautiful visuals on a bad model give wrong or slow answers. In Power BI the **data model** decides what numbers mean: how tables relate, which table filters which, and how fast DAX runs. The golden rule: **model as a star schema**.



A star schema: one central fact table of events/numbers surrounded by descriptive dimension tables. Filters flow from dimensions (one side) to the fact (many side).

	Fact table	Dimension table
Describes	Events or transactions (a sale, a click, a shipment)	The "who, what, where, when" of those events
Rows	Many (millions)	Fewer (thousands)
Columns	Keys + numeric measures (quantity, amount, cost)	Keys + descriptive attributes (name, category, city)
Changes	Grows constantly	Changes rarely
Example	fact_sales	dim_product , dim_customer , dim_date

Why a star beats one big flat table: smaller file (text stored once), faster visuals, one place to fix a name, correct filtering across several facts, and simpler DAX. A **snowflake** (dimensions normalised into sub-dimensions) works but adds joins; prefer a star in Power BI.

The Adventure Cycles dataset

A fictional bicycle company sells bikes, components, clothing and accessories in India, the UAE and Singapore. The model has one fact table and four dimensions (plus the date table):

Python

```
import pandas as pd
D = "adventure_cycles/"
fact = pd.read_csv(D + "fact_sales.csv", parse_dates=["order_date"]); prod = pd.read_csv(D
+ "dim_product.csv")
cust = pd.read_csv(D + "dim_customer.csv"); terr = pd.read_csv(D + "dim_territory.csv");
dates = pd.read_csv(D + "dim_date.csv", parse_dates=["date"])
for name, df in [("fact_sales", fact), ("dim_product", prod), ("dim_customer", cust),
("dim_territory", terr), ("dim_date", dates)]:
    print(f"{name:14s} {df.shape[0]:>6,} rows × {df.shape[1]} columns")
fact.head(4)
```

OUTPUT

fact_sales 9,000 rows × 9 columns
dim_product 45 rows × 6 columns
dim_customer 1,200 rows × 5 columns
dim_territory 6 rows × 3 columns
dim_date 1,096 rows × 6 columns

	order_number	order_date	product_key	customer_key	territory_key	quantity	ur
0	SO50387	2023-01-01	11	103	1	3	11
1	SO55797	2023-01-01	32	798	1	3	20
2	SO51236	2023-01-01	9	750	5	1	12
3	SO55082	2023-01-01	13	759	1	1	75

4 rows × 9 columns

Power Query: prepare each table

Open **Transform Data**. Typical steps for each table (all recorded in **Applied Steps**):

Table	Steps
fact_sales	Set types (order_date Date; quantity Whole number; amounts Fixed decimal number), remove blank rows, add Profit = [sales_amount] - [total_cost] as a custom column, keep only needed columns
dim_product	Trim/clean text, set list_price and standard_cost as decimals, remove duplicates on product_key
dim_customer	Replace M/F with Male/Female, remove duplicates on customer_key
dim_territory	Check there is one row per territory_key
dim_date	Set date to Date; sort month by month_no later in the model

Query settings (right pane) shows each step; click any step to see the data at that point, and delete a step to undo it. Rename queries (Sales, Product...) because those names become table names in the model.

Common transformations (from class and beyond)

Goal	Where	Notes
Promote headers, set types	Home	Always check types: wrong types break relationships
Remove duplicates / blank rows	Home → Remove rows	Needed for dimension keys
Split column by delimiter	Transform → Split Column	e.g. IN-2025-00042
Replace values, trim, clean, format	Transform	
Unpivot columns	Transform → Unpivot	Turn Jan, Feb, Mar columns into rows (Excel Lesson 12)
Merge queries	Home → Merge	A join; use to bring a category into the fact (but prefer relationships)
Append queries	Home → Append	Stack monthly files
Group by	Transform → Group By	Pre-aggregate large tables
Conditional / Custom column	Add Column	M formulas
Reference vs Duplicate	Right-click query	<i>Reference</i> creates a new query that starts from the output of another (changes flow through); <i>Duplicate</i> copies the steps
Parameters	Home → Manage Parameters	A variable such as a file path or a start date, used in queries (Lesson 4)
Disable load	Right-click query	Keep helper queries out of the model

What is "query folding"?

When the source is a database (SQL Server, Snowflake), Power Query **translates your steps into SQL** and lets the database do the work, instead of downloading everything first. A step that cannot be translated "breaks folding" and everything after it runs locally. Check by right-clicking a step → *View Native Query*. Keep foldable steps (filter, select columns, merge, group) **before** non-foldable ones.

Power Query M

```
let
    Source = Csv.Document(File.Contents("C:\Data\fact_sales.csv"), [Delimiter="," ,
    Encoding=65001]),
    Promoted = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),
    Typed = Table.TransformColumnTypes(Promoted, {"order_date", type date},
    {"quantity", Int64.Type},
    {"sales_amount", Currency.Type}, {"total_cost", Currency.Type})),
    Profit = Table.AddColumn(Typed, "profit", each [sales_amount] - [total_cost],
    Currency.Type)
in
    Profit
```

Relationships

A **relationship** links a column in one table to a column in another so that filters can flow between them. In **Model view**, drag dim_product[product_key] onto fact_sales[product_key]. Power BI usually detects relationships on same-named columns (Home → Manage relationships → Autodetect).

Cardinality

Type	Meaning	Use
One-to-many (1:*)	A key is unique in one table, repeated in the other	The standard: dimension → fact
One-to-one (1:1)	Unique on both sides	Splitting a wide table
Many-to-many (*:*)	Duplicates on both sides	Avoid; use a bridge table

Check uniqueness before creating a one-to-many: if the "one" side has duplicates Power BI will complain.

Python

```
checks = {"dim_product[product_key]": prod.product_key, "dim_customer[customer_key]":
cust.customer_key, "dim_territory[territory_key]": terr.territory_key, "dim_date[date]":
dates.date}
for name, col in checks.items(): print(f"{name:30s} unique: {col.is_unique}")
orphans = {"product_key": ~fact.product_key.isin(prod.product_key), "customer_key":
~fact.customer_key.isin(cust.customer_key), "territory_key":
~fact.territory_key.isin(terr.territory_key)}
for k, m in orphans.items(): print(f"fact rows with no matching {k}: {m.sum()}")
```

OUTPUT

```

dim_product[product_key]      unique: True
dim_customer[customer_key]    unique: True
dim_territory[territory_key]  unique: True
dim_date[date]                unique: True
fact rows with no matching product_key: 0
fact rows with no matching customer_key: 0
fact rows with no matching territory_key: 0

```

Unique keys on the "one" side and no orphans on the "many" side: the model is safe.

Filter direction



Filters flow along relationships from the one side to the many side (single direction). Selecting Bikes in dim_product filters fact_sales, which then filters the other dimensions only if bi-directional filtering is on.

Setting	Behaviour	Advice
Single (dimension → fact)	A selection in a dimension filters the fact	Default and best
Both (bi-directional)	Filters also flow back from fact to dimension	Use only when needed (many-to-many, "show only customers who bought"); can cause ambiguity and slowness
Active vs inactive relationship	Only one active path between two tables; an inactive one can be switched on in DAX with USERELATIONSHIP	e.g. Order Date and Ship Date both link to the date table

Seeing the model work

Total sales by product category and by country, using the relationships (a join in pandas):

Python

```
m = fact.merge(prod, on="product_key").merge(terr, on="territory_key")
m["profit"] = m.sales_amount - m.total_cost
out = m.pivot_table(index="category", columns="country", values="sales_amount",
aggfunc="sum", margins=True, margins_name="Total") / 1e6
out.round(1).rename_axis("Sales (₹ million)")
```

OUTPUT

country	India	Singapore	UAE	Total
Sales (₹ million)				
Accessories	6.7	1.7	1.3	9.8
Bikes	320.6	74.8	71.3	466.8
Clothing	6.0	1.3	1.1	8.5
Components	73.5	18.8	17.2	109.6
Total	406.9	96.7	91.0	594.6

5 rows × 4 columns

In Power BI you get this with a matrix: Rows category, Columns country, Values SUM(sales_amount). No join code needed: the relationships do it.

The date table

Time intelligence (year-to-date, same period last year, Lesson 4) requires a **proper date table**: one row per day with **no gaps**, covering all dates in the data, and **marked as a date table**.

DAX

```
Date =  
VAR MinDate = DATE ( YEAR ( MIN ( fact_sales[order_date] ) ), 1, 1 )  
VAR MaxDate = DATE ( YEAR ( MAX ( fact_sales[order_date] ) ), 12, 31 )  
RETURN  
ADDCOLUMNS (   
    CALENDAR ( MinDate, MaxDate ),  
    "Year",      YEAR ( [Date] ),  
    "Quarter",   "Q" & QUARTER ( [Date] ),  
    "Month No",  MONTH ( [Date] ),  
    "Month",     FORMAT ( [Date], "mmm" ),  
    "Year-Month", FORMAT ( [Date], "yyyy-mm" ),  
    "Weekday",   FORMAT ( [Date], "ddd" )  
)
```

STEP BY STEP Set it up

1. Model → **New table** → paste the DAX above (or load dim_date.csv).
2. Select the table → **Table tools** → **Mark as date table** → choose the Date column. Power BI validates that dates are unique and continuous.
3. Create the relationship Date[Date] → fact_sales[order_date] (one-to-many).
4. **Sort Month by Month No** (select Month → Column tools → Sort by column) so April does not come before January.
5. Hide the raw order_date in the fact table so report authors use the date table.

Python

```
print("Fact order dates :", fact.order_date.min().date(), "to",  
      fact.order_date.max().date())  
print("Date table      :", dates.date.min().date(), "to", dates.date.max().date(), "|  
rows:", len(dates))  
full = pd.date_range(dates.date.min(), dates.date.max())  
print("Continuous (no gaps):", len(full) == len(dates) and (dates.date.sort_values().values  
= full.values).all())  
dates.head(4)
```

OUTPUT

```
Fact order dates : 2023-01-01 to 2025-12-30  
Date table      : 2023-01-01 to 2025-12-31 | rows: 1096  
Continuous (no gaps): True
```

	date	year	quarter	month_no	month	weekday
0	2023-01-01	2023	Q1	1	Jan	Sun
1	2023-01-02	2023	Q1	1	Jan	Mon
2	2023-01-03	2023	Q1	1	Jan	Tue
3	2023-01-04	2023	Q1	1	Jan	Wed

4 rows × 6 columns

WATCH OUT Do not rely on auto date/time

Power BI can create hidden date tables automatically ("Auto date/time"). They bloat the model and cannot be customised or shared. Turn it off (File → Options → Data Load → Time intelligence) and always use your own marked date table.

Hierarchies

A **hierarchy** lets users drill down through levels with one click. In the Data pane, right-click category in `dim_product` → **Create hierarchy**, then drag subcategory and product onto it. Visuals with a hierarchy get drill-down buttons (arrows at the top-right of the visual): from category → sub-category → product. The date table's Year → Quarter → Month → Date is the classic one.

Common modelling problems

Symptom	Likely cause	Fix
Totals repeat the same number for every category	No relationship between the table you filter and the fact	Create the relationship
"A circular dependency was detected" / ambiguous paths	Two paths between two tables	Make one inactive, or remove the bi-directional filter
Many-to-many warning	Duplicate keys on both sides	Build a clean dimension with unique keys, or a bridge table
Blank row ("(Blank)") appears in visuals	Fact rows whose key is missing in the dimension (orphans)	Add the missing rows to the dimension, or fix keys
YTD / last-year measures give wrong or blank results	Date table not marked, has gaps, or not related	Fix the date table
Slow model, big file	Wide tables, high-cardinality text columns, auto date/time	Star schema, remove unused columns, reduce precision, disable auto date/time
Months sort alphabetically	Missing sort-by-column	Sort Month by Month No

TIP Best-practice checklist

1. Star schema with surrogate integer keys.
2. One-to-many, single-direction relationships.
3. One marked date table.
4. Hide keys and technical columns from report view.
5. Disable auto date/time.
6. Remove columns you do not use.
7. Name tables and columns for business users.
8. Put measures in a separate _Measures table (Lesson 3).

DO IT WITH AI Do it with AI

Ask Copilot or any assistant: *"Review this list of tables and columns and propose a star schema with a fact table and dimensions, the keys, and the relationships"* and paste your column lists. Compare its answer with the checklist above, and verify key uniqueness yourself in Power Query (Column profile → *Distinct equals Count*).

Summary

KEY TAKEAWAYS

- Model as a **star schema**: facts (events and numbers) surrounded by dimensions (descriptions), joined by **one-to-many, single-direction** relationships.
- Power Query prepares each table: types, cleaning, unpivot, merge/append, group, parameters; **query folding** pushes work to the source.
- Filters flow from the **one side** (dimension) to the **many side** (fact); use bi-directional filters sparingly; one active relationship per table pair, others inactive (USERRELATIONSHIP).
- A continuous, **marked date table** is required for time intelligence; sort month names by month number; turn off auto date/time.
- Hierarchies enable drill-down; most "wrong totals" come from missing relationships, duplicate keys or orphans.

Quick quiz

6 questions · tap an answer to check it

1. In a star schema, which table typically contains the numeric measures such as quantity and sales amount?

A Dimension table

B Fact table

C Date table

D Parameter table

Answer: B. Facts hold events and measures; dimensions hold descriptions.

2. A one-to-many relationship between dim_product and fact_sales means:

- A** A product appears once in sales
- B** Each product_key appears once in dim_product and many times in fact_sales
- C** Both tables have unique keys
- D** The relationship is bidirectional

Answer: B. The "one" side must have unique keys.

3. Why is a bi-directional relationship risky?

- A** It is not allowed
- B** It can create ambiguous filter paths and slow performance
- C** It deletes data
- D** It only works with dates

Answer: B. Use it only when you must (for example many-to-many scenarios).

4. What does "Mark as date table" require?

- A** A column of text
- B** A unique, continuous Date column with no gaps
- C** A relationship to every table
- D** A measure

Answer: B. Time-intelligence functions need a valid contiguous date table.

5. Months show as Apr, Aug, Dec, Feb... in alphabetical order. Fix?

- A** Rename the months
- B** Use a pie chart
- C** Sort the Month column by the Month No column
- D** Delete the date table

Answer: C. Sort by column controls the display order.

6. What does Power Query's "query folding" do?

- A** Compresses the PBIX file
- B** Translates steps into the source system's query (e.g. SQL) so the database does the work
- C** Hides the query
- D** Merges two queries

Answer: B. Folding improves refresh performance for database sources.

EXERCISE Model it yourself

Load the five Adventure Cycles CSV files, set types, create the four relationships, mark `dim_date` as the date table, hide key columns, create a Category → Subcategory → Product hierarchy, and build a matrix of sales by category (rows) and year (columns).

Show solution (worked answer)

In Power Query set `order_date` and `date` to Date. In Model view drag `product_key`, `customer_key`, `territory_key` from each dimension to the fact, and `date` to `order_date`. Table tools → Mark as date table → `date`. Right-click the key columns → Hide in report view. Create the hierarchy by right-clicking `category`. In Report view add a Matrix with Rows `dim_product[category]`, Columns `dim_date[year]`, Values Sum of `sales_amount`. Compare with the pandas pivot: Bikes should be about ₹46.7 crore of the total ₹59.5 crore.

INTERVIEW CORNER Questions you may be asked

Q: What is a star schema and why use it in Power BI? One fact table joined to several dimension tables with one-to-many relationships. It compresses well, filters predictably, and makes DAX simple and fast.

Q: How do filters propagate in a Power BI model? From the one side to the many side along active relationships, in the filter direction. Bi-directional relationships also filter from many back to one.

Q: Why do you need a date table? Built-in time-intelligence DAX (YTD, SAMEPERIODLASTYEAR) needs a continuous, marked date table; it also gives you year/quarter/month columns, fiscal calendars and holidays.

Q: How do you handle two dates in the fact table (order date and ship date)? Role-playing dimension: one date table with one active relationship (order date) and one inactive (ship date) activated with `USERELATIONSHIP` in specific measures, or two copies of the date table.

DAX Fundamentals: Measures, Filter Context & CALCULATE

Learn DAX, the formula language of Power BI. Understand calculated columns vs measures, row context vs filter context, write core aggregations and iterators (SUMX), master CALCULATE and FILTER, ALL/REMOVEFILTERS and ALLSELECTED, handle division safely, use variables, and build a library of business measures.

● Power BI classes 2-3

✦ Power BI Desktop 2026

Intermediate

🕒 150 min

By the end of this lesson you can

- Explain the difference between a calculated column and a measure, and when to use each
- Describe row context and filter context and predict what a measure returns in a visual
- Write aggregation measures and iterators (SUM, COUNTROWS, DISTINCTCOUNT, SUMX, AVERAGEX)
- Use CALCULATE with filter arguments, ALL/REMOVEFILTERS and ALLSELECTED to build ratios and percent-of-total
- Use variables, DIVIDE, IF/SWITCH and display folders to write clean, reusable measures

What is DAX?

DAX (Data Analysis Expressions) is the formula language used in Power BI, Excel's Data Model and Analysis Services. It looks like Excel formulas, but it works on **columns and tables** and recalculates for whatever the user selects in a visual. DAX is the skill that separates a report builder from a Power BI developer.

	Calculated column	Measure
Evaluated	Row by row when data is refreshed	On the fly , for the current filter selection
Stored in the model	Yes (uses memory)	No (only the formula)
Result depends on slicers?	No	Yes
Use for	New attributes to slice by (Age Band, Hire Year, Profit per row)	Totals, ratios, KPIs, growth
Appears in	Rows, columns, slicers, filters	Values area, cards, tooltips

Calculated column vs measure

Calculated column

- Profit = [sales_amount] - [total_cost]
- Stored on every fact row
- Same value whatever you select
- Good for slicing and filtering

Measure

- Total Profit = SUM(sales) - SUM(cost)
- Nothing stored per row
- Changes with slicers and visuals
- Good for numbers shown in visuals

The same table, two ways. A calculated column adds a value to every row. A measure returns one number computed from whichever rows are currently filtered.

TIP Rule of thumb

If you want to **put it on an axis or slicer**, make a column. If you want to **put it in the Values well**, make a measure. In a star schema, almost every number is a measure.

DAX syntax in five minutes

DAX

```
Total Sales = SUM ( fact_sales[sales_amount] )
```

- `Total Sales` is the **measure name** (spaces allowed, no brackets when defining it).
- `=` starts the formula.
- `SUM` is a **function**; `fact_sales[sales_amount]` is a **fully qualified column** (table name, then column in square brackets). Always include the table name for columns; **never** include it for measures: `[Total Sales]`.
- Strings use double quotes; comments use `--` or `//` or `/* ... */`.
- Operators: `+` `-` `*` `/` , `&` (join text), `=` `<` `>` `<=` `>=` , `&&` (AND), `||` (OR), `IN { "A", "B" }`.
- Spaces and line breaks are ignored; format with **daxformatter.com** or the formatting button.

Creating measures

Home → **New measure** (or right-click a table). Put all measures in an empty table named `_Measures` (Home → Enter data → name `_Measures` → Load, then hide its default column) so they are easy to find. Use **display folders** (Model view → Properties) to group them.

Basic aggregations

DAX

```
Total Sales      = SUM ( fact_sales[sales_amount] )
Total Cost       = SUM ( fact_sales[total_cost] )
Total Profit     = [Total Sales] - [Total Cost]
Profit Margin %  = DIVIDE ( [Total Profit], [Total Sales] )
Units Sold       = SUM ( fact_sales[quantity] )
Orders           = DISTINCTCOUNT ( fact_sales[order_number] )
Order Lines      = COUNTROWS ( fact_sales )
Avg Order Value  = DIVIDE ( [Total Sales], [Orders] )      -- in this data every order has
one line, so Orders = Order Lines
Customers        = DISTINCTCOUNT ( fact_sales[customer_key] )
```

Check the values in Python; they are what a card visual with no filters would display:

Python

```
import pandas as pd
D = "adventure_cycles/"
f = pd.read_csv(D + "fact_sales.csv", parse_dates=["order_date"]); p = pd.read_csv(D +
"dim_product.csv"); t = pd.read_csv(D + "dim_territory.csv")
sales, cost = f.sales_amount.sum(), f.total_cost.sum()
print(f"Total Sales      : {sales:,.0f}")
print(f"Total Cost       : {cost:,.0f}")
print(f"Total Profit      : {sales - cost:,.0f}")
print(f"Profit Margin %   : {(sales - cost) / sales:.1%}")
print(f"Units Sold        : {f.quantity.sum():,}")
print(f"Orders             : {f.order_number.nunique():,}")
print(f"Order Lines         : {len(f):,}")
print(f"Avg Order Value     : {sales / f.order_number.nunique():,.0f}")
print(f"Customers           : {f.customer_key.nunique():,}")
```

OUTPUT

```
Total Sales      : 594,559,665
Total Cost       : 384,392,790
Total Profit      : 210,166,875
Profit Margin %   : 35.3%
Units Sold        : 22,703
Orders           : 9,000
Order Lines       : 9,000
Avg Order Value   : 66,062
Customers         : 1,200
```

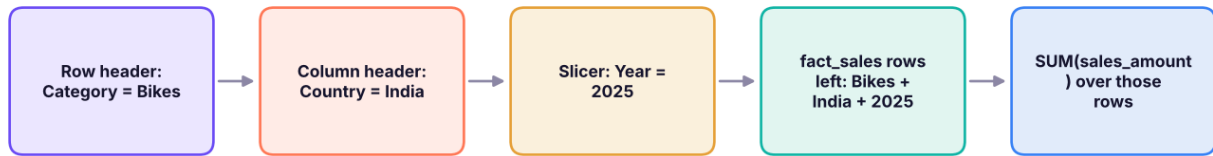
WATCH OUT DIVIDE, not /

[A] / [B] returns an error or infinity when B is 0 or blank. DIVIDE ([A], [B]) returns BLANK (or an alternate result: DIVIDE ([A], [B], 0)). Always use DIVIDE for ratios.

Filter context: the heart of DAX

When you put a measure in a visual, DAX evaluates it **once per cell**, with a **filter context** made of everything that filters the data for that cell: row and column headers, slicers, page/report filters, and cross-filtering from other visuals.

Filter context of one matrix cell



Filter context for one cell of a matrix (Category = Bikes, Country = India, Year = 2025 slicer). DAX applies all of them to the fact table before it sums.

Same measure, different context in each cell. The matrix below is exactly what Power BI shows for **Total Sales** with Category on rows and Country on columns:

Python

```

m = f.merge(p, on="product_key").merge(t, on="territory_key")
m["year"] = m.order_date.dt.year
print((m.pivot_table(index="category", columns="country", values="sales_amount",
aggfunc="sum") / 1e6).round(1).to_string())
print("\nSame measure, with a Year = 2025 slicer:")
print((m[m.year == 2025].pivot_table(index="category", columns="country",
values="sales_amount", aggfunc="sum") / 1e6).round(1).to_string())
  
```

OUTPUT

country	India	Singapore	UAE
category			
Accessories	6.7	1.7	1.3
Bikes	320.6	74.8	71.3
Clothing	6.0	1.3	1.1
Components	73.5	18.8	17.2

Same measure, with a Year = 2025 slicer:

country	India	Singapore	UAE
category			
Accessories	2.2	0.7	0.4
Bikes	102.3	24.6	22.0
Clothing	2.0	0.3	0.4
Components	22.9	7.1	5.1

Row context

A **row context** exists when DAX goes through a table row by row: in a **calculated column**, and inside **iterator** functions (`SUMX` , `AVERAGEX` , `FILTER` ...). In row context, column references mean "this row's value".

DAX

```
-- Calculated column on fact_sales: row context gives each row's own values
Line Profit = fact_sales[sales_amount] - fact_sales[total_cost]
```

Row context does **not** automatically filter other tables; **CALCULATE** converts a row context into a filter context ("context transition").

Iterators: SUMX, AVERAGEX, MAXX

`SUM ( column )` adds one column. When you need to **calculate per row first** and then aggregate, use an iterator: `SUMX ( table, expression )`.

DAX

```
-- Revenue at list price (unit_price x quantity), calculated row by row
List Value = SUMX ( fact_sales, fact_sales[quantity] * fact_sales[unit_price] )

-- Average margin per order line (not the same as total profit / total sales!)
Avg Line Margin % = AVERAGEX ( fact_sales, DIVIDE ( fact_sales[sales_amount] -
fact_sales[total_cost], fact_sales[sales_amount] ) )

-- Most profitable single line
Max Line Profit = MAXX ( fact_sales, fact_sales[sales_amount] - fact_sales[total_cost] )
```

Python

```
f["line_profit"] = f.sales_amount - f.total_cost
list_value = (f.quantity * f.unit_price).sum()
avg_line_margin = ((f.sales_amount - f.total_cost) / f.sales_amount).mean()
print(f"List Value          : {list_value:,.0f}")
print(f"Realised/List       : {f.sales_amount.sum() / list_value:.1%} (average price
realisation)")
print(f"Avg Line Margin % : {avg_line_margin:.1%} vs overall margin
{(f.sales_amount.sum() - f.total_cost.sum()) / f.sales_amount.sum():.1%}")
print(f"Max Line Profit    : {f.line_profit.max():,.0f}")
```

OUTPUT

```
List Value          : 617,201,300
Realised/List       : 96.3% (average price realisation)
Avg Line Margin % : 34.5% vs overall margin 35.3%
Max Line Profit    : 174,060
```

The average of line margins (34.5%) differs slightly from the overall margin (35.3%) because large lines count more in the total than in a simple average. Choose the definition the business wants, and

name the measure clearly.

CALCULATE: change the filter context

`CALCULATE ( expression, filter1, filter2, ... )` evaluates the expression in a **modified** filter context. It is the most important DAX function.

DAX

```
Bike Sales    = CALCULATE ( [Total Sales], dim_product[category] = "Bikes" )
India Sales   = CALCULATE ( [Total Sales], dim_territory[country] = "India" )
Bike Sales India 2025 =
    CALCULATE ( [Total Sales], dim_product[category] = "Bikes", dim_territory[country] =
"India", dim_date[year] = 2025 )
High Value Lines = CALCULATE ( COUNTROWS ( fact_sales ), fact_sales[sales_amount] > 200000
)
```

Multiple filter arguments are combined with **AND**. A simple column filter like `dim_product[category] = "Bikes"` **replaces** any existing filter on that column.

Python

```
bike = f.merge(p, on="product_key").query("category == 'Bikes'").sales_amount.sum()
print(f"Bike Sales          : {bike:,.0f}")
india_2025_bikes = m[(m.category == "Bikes") & (m.country == "India") & (m.year ==
2025)].sales_amount.sum()
print(f"Bike Sales India 2025: {india_2025_bikes:,.0f}")
print(f"High Value Lines    : {(f.sales_amount > 200000).sum():,.0f}")
```

OUTPUT

```
Bike Sales          : 466,754,465
Bike Sales India 2025: 102,271,520
High Value Lines    : 797
```

Filter functions: ALL, REMOVEFILTERS, ALLSELECTED, ALLEXCEPT, KEEPFILTERS

Function	Effect	Typical use
<code>ALL (table or column) / REMOVEFILTERS (...)</code>	Ignores filters on the table/column	Grand totals, denominators
<code>ALLSELECTED (...)</code>	Ignores filters from the visual's own rows/columns but keeps slicers and page filters	"% of visible total"
<code>ALLEXCEPT (table, col1, ...)</code>	Removes all filters on the table except those columns	Subtotals per group
<code>VALUES (column)</code>	The distinct values visible in the current context	Counting / looping over what is filtered
<code>SELECTEDVALUE (column, alt)</code>	The single selected value (or alternate)	Titles, dynamic measures
<code>KEEPFILTERS (filter)</code>	Intersects with, rather than replaces, existing filters	Advanced

Percent of total

DAX

```
Sales % of Total =  
DIVIDE ( [Total Sales], CALCULATE ( [Total Sales], REMOVEFILTERS ( dim_product ) ) )  
  
Sales % of Visible Total =  
DIVIDE ( [Total Sales], CALCULATE ( [Total Sales], ALLSELECTED ( dim_product ) ) )  
  
Share Within Country =  
DIVIDE ( [Total Sales], CALCULATE ( [Total Sales], ALLEXCEPT ( dim_territory, dim_territory[country] ) ) )
```

Python

```
cat = m.groupby("category").sales_amount.sum()  
print((100 * cat / cat.sum()).round(1).rename("Sales % of Total").to_string())
```

OUTPUT

```
category  
Accessories      1.6  
Bikes            78.5
```

Clothing	1.4
Components	18.4

FILTER: row-by-row conditions

`FILTER ( table, condition )` is an iterator that returns the rows where the condition is TRUE. Use it when a filter needs a **measure or several columns**; for a simple single-column comparison, the plain `column = value` form inside `CALCULATE` is faster.

DAX

```
-- Sales from products with a list price above 1,00,000
Premium Sales = CALCULATE ( [Total Sales], FILTER ( dim_product, dim_product[list_price] >
100000 ) )

-- Number of products whose total sales beat 1 crore
Strong Products = COUNTROWS ( FILTER ( VALUES ( dim_product[product] ), [Total Sales] >
10000000 ) )
```

Python

```
prem = f.merge(p, on="product_key").query("list_price > 100000").sales_amount.sum()
ps = f.groupby("product_key").sales_amount.sum()
print(f"Premium Sales : {prem:,.0f}")
print(f"Strong Products: {(ps > 10_000_000).sum()} (products with sales above ₹1 crore)")
```

OUTPUT

```
Premium Sales : 303,639,995
Strong Products: 20 (products with sales above ₹1 crore)
```

Variables and readability

`VAR ... RETURN` stores intermediate results: easier to read, easier to debug, and faster (each variable is evaluated once).

DAX

```
Profit Margin vs Overall =
VAR CurrentMargin = DIVIDE ( [Total Profit], [Total Sales] )
VAR OverallMargin = CALCULATE ( DIVIDE ( [Total Profit], [Total Sales] ), REMOVEFILTERS ( )
)
RETURN
CurrentMargin - OverallMargin
```



Logical and text measures

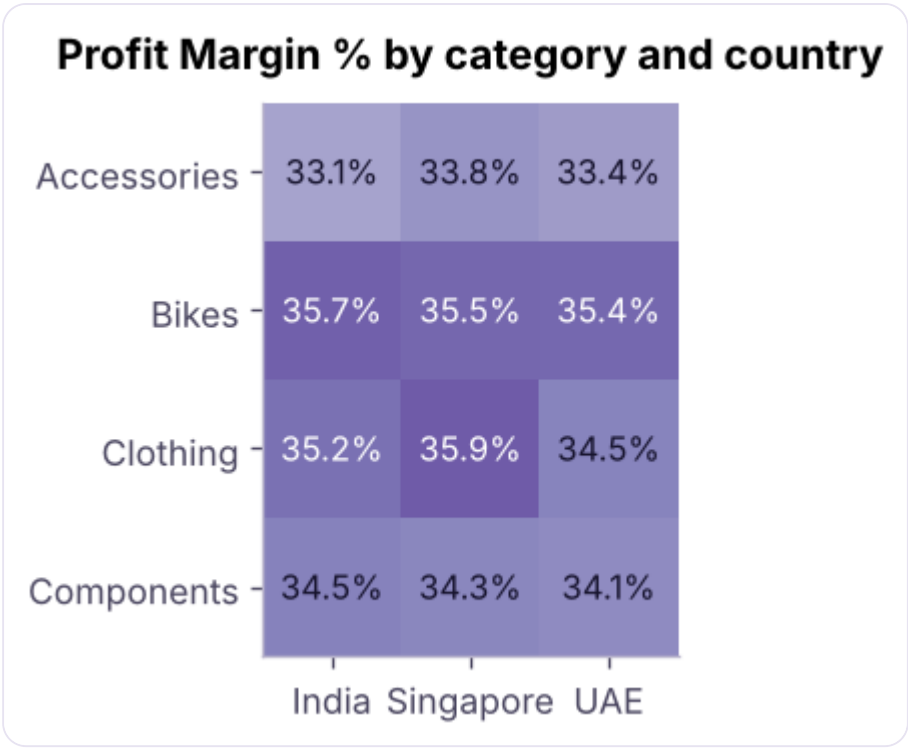
DAX

```
Margin Status =  
SWITCH (  
    TRUE (),  
    ISBLANK ( [Profit Margin %] ), "No data",  
    [Profit Margin %] ≥ 0.4,      "Excellent",  
    [Profit Margin %] ≥ 0.3,      "Good",  
    "Review"  
)  
  
Selected Country Title =  
"Sales for " & SELECTEDVALUE ( dim_territory[country], "all countries" )
```

Function	Purpose
IF (test, a, b) , SWITCH (expr, v1, r1, ...) , SWITCH (TRUE (), cond1, r1, ...)	Branching
ISBLANK , COALESCE (x, 0) , BLANK ()	Handle empty values
HASONE (col) , ISFILTERED (col)	Test the current context
FORMAT (value, "0.0%") , CONCATENATEX (table, col, ", ")	Text



Measures in action: a profit-margin matrix



Profit margin % by category and country: the matrix you would build with the Profit Margin % measure and conditional formatting (darker = higher).

Common DAX mistakes

Mistake	Fix
Using a calculated column where a measure is needed (results do not respond to slicers)	Create a measure
Dividing with <code>/</code>	<code>DIVIDE</code>
Writing <code>SUM ([Measure])</code>	<code>SUM</code> takes columns; use <code>SUMX (table, [Measure])</code> for measures over rows
Averaging ratios (<code>AVERAGE</code> of a margin column)	Compute <code>DIVIDE (SUM numerator, SUM denominator)</code> unless you really want the average of ratios
Using <code>FILTER (ALL (table), ...)</code> when a column filter works	Prefer simple column filters inside <code>CALCULATE</code>
Forgetting that a measure's result depends on context; testing in a card	Test in a table or matrix with the fields you care about
Hard-coding values (<code>"Bikes"</code>) in many measures	Use a parameter table or <code>SELECTEDVALUE</code>

DO IT WITH AI Do it with AI

Copilot and chat assistants write decent DAX, but they invent column names and often misuse context. A reliable prompt gives the **exact table and column names**, the **relationships**, and the **expected result**: *"Tables: fact_sales(order_date, product_key, sales_amount, total_cost), dim_product(product_key, category), dim_date(date). Write a DAX measure for each category's share of total sales that ignores the category filter but respects year slicers. Explain the filter context."* Then test the measure in a table next to your own pandas check.

Summary

KEY TAKEAWAYS

- **Calculated columns** run row by row at refresh and are stored; **measures** run at query time in the current **filter context**.
- Filter context = row/column headers + slicers + filters + cross-filtering; measures are re-evaluated for every cell.
- Row context exists in calculated columns and iterators (`SUMX` , `AVERAGEX` , `FILTER`); `CALCULATE` converts it to filter context.
- `CALCULATE ( expression, filters )` changes the filters; `REMOVEFILTERS` / `ALL` , `ALLSELECTED` , `ALLEXCEPT` remove them for percent-of-total calculations.
- Use `DIVIDE` , variables (`VAR ... RETURN`), `SWITCH ( TRUE (), ... )` , clear names, a `_Measures` table and display folders.

Quick quiz

6 questions · tap an answer to check it

1. Which is evaluated for every cell of a matrix based on slicers and row/column headers?

A A calculated column

B A measure

C A relationship

D A parameter

Answer: B. Measures respond to filter context; calculated columns are fixed at refresh.

2. Which function lets you compute a total that ignores the Category filter (for a denominator)?

A SUM

B CALCULATE ([Total Sales], REMOVEFILTERS (dim_product[category]))

C FILTER ([Total Sales])

D COUNTROWS

Answer: B. REMOVEFILTERS (or ALL) clears the filter inside CALCULATE.

3. Which is the safest way to compute a margin?

A [Profit] / [Sales]

B DIVIDE ([Profit], [Sales])

C IF ([Sales] = 0, 0, [Profit]) / [Sales]

D SUM ([Profit]) / SUM ([Sales])

Answer: B. DIVIDE handles zero and blank denominators gracefully.

4. `SUMX ( fact_sales, fact_sales[quantity] * fact_sales[unit_price] )` is needed because:

- A SUM cannot add numbers
- B The product must be calculated per row before adding**
- C SUMX is faster than SUM
- D It creates a calculated column

Answer: B. SUM works on one existing column; SUMX evaluates an expression per row.

5. What is row context?

- A The current row being evaluated in a calculated column or iterator**
- B The filters from slicers
- C The relationship direction
- D A table filter

Answer: A. Row context refers to "this row" during row-by-row evaluation.

6. What does `CALCULATE ( [Total Sales], dim_product[category] = "Bikes" )` return in a visual that shows each country?

- A Total sales for all products
- B Bike sales for each country in the visual**
- C Bike sales for all countries on every row
- D An error

Answer: B. The Category filter is replaced by Bikes, while the Country filter from the visual still applies.

EXERCISE Write the measures

Using the Adventure Cycles model, write DAX for: (1) Profit and margin % for **Clothing** only; (2) each country's share of total sales (works in a visual with country on rows); (3) the number of customers who bought more than one product line; (4) a title measure that says "Top category: ..." for the category with the highest sales in the current context.

Show solution (worked answer)

DAX

```
-- (1)
Clothing Profit = CALCULATE ( [Total Profit], dim_product[category] = "Clothing" )
Clothing Margin = DIVIDE ( [Clothing Profit], CALCULATE ( [Total Sales],
dim_product[category] = "Clothing" ) )
-- (2)
Country Share = DIVIDE ( [Total Sales], CALCULATE ( [Total Sales], REMOVEFILTERS (
dim_territory ) ) )
-- (3)
Multi-line Customers =
COUNTROWS ( FILTER ( VALUES ( fact_sales[customer_key] ), CALCULATE ( COUNTROWS (
fact_sales ) ) > 1 ) )
-- (4)
Top Category =
VAR t = ADDCOLUMNS ( VALUES ( dim_product[category] ), "@s", [Total Sales] )
VAR top1 = TOPN ( 1, t, [@s], DESC )
RETURN "Top category: " & MAXX ( top1, dim_product[category] )
```

Verify each in a table visual next to a pandas check.

Q: What is the difference between a calculated column and a measure? A column is computed per row at refresh and stored; a measure is computed at query time for the current filter context and returns a single value.

Q: Explain filter context and row context. Filter context is the set of filters (axes, slicers, filters) applied to the model when a measure is evaluated. Row context is the "current row" while iterating a table (calculated columns, iterators). CALCULATE turns row context into filter context.

Q: What does CALCULATE do? Evaluates an expression in a modified filter context; its filter arguments add, replace or remove filters.

Q: ALL vs ALLSELECTED? ALL removes all filters from the specified table/column (grand total); ALLSELECTED removes only the filters coming from the visual's own rows and columns but keeps outer filters such as slicers, giving a total of what is visible.

Q: SUM vs SUMX? SUM adds a column; SUMX evaluates an expression for each row of a table and sums the results (needed for products of columns, conditional logic per row).

DAX Time Intelligence, Ranking, Parameters & Dynamic Measures

Make dashboards answer "compared with when?". Build year-to-date, month-to-date, same-period-last-year, month-on-month and year-on-year measures, rolling averages, rankings with RANKX and TOPN, what-if parameters, field parameters and dynamic measure switching, and know visual calculations and calculation groups.

● Power BI classes 3–4

✦ Power BI Desktop 2026 (field parameters, calculation groups, visual calculations)

Advanced

🕒 150 min

By the end of this lesson you can

- Explain what time intelligence needs (a marked date table) and use TOTALYTD, DATESYTD, SAMEPERIODLASTYEAR, DATEADD and PARALLELPERIOD
- Build YoY %, MoM %, YTD, rolling 3-month and running-total measures
- Rank items with RANKX and show the Top N with TOPN, including an "Others" bucket
- Create what-if parameters and dynamic measure switching with field parameters
- Describe calculation groups, visual calculations and DAX query view, and when to use them

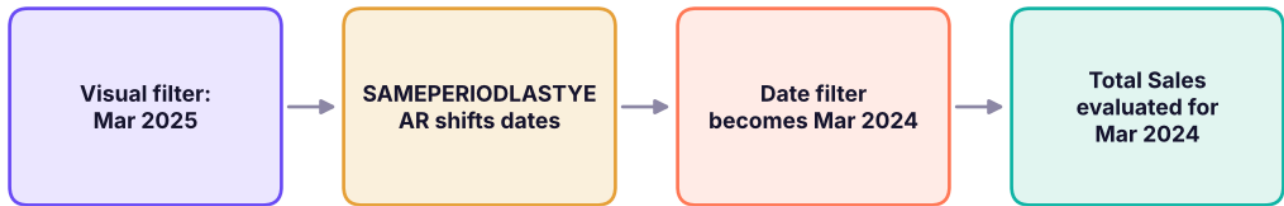
What time intelligence needs

Time-intelligence functions move or expand the **date filter**: "this year to date", "the same period last year". They only work when:

1. You have a **date table** with every date (Lesson 2),
2. it is **marked as a date table**, and
3. it is **related** to the fact table's date column.

Always use the **date table's** `Date` column inside time functions, never the fact table's date.

How CALCULATE + time intelligence works



Time intelligence changes the date filter, then CALCULATE re-evaluates the measure.
SAMEPERIODLASTYEAR shifts the selected dates back exactly one year.

Core measures

DAX

```
Total Sales = SUM ( fact_sales[sales_amount] )

-- Cumulative totals
Sales YTD = TOTALYTD ( [Total Sales], dim_date[date] )
Sales QTD = TOTALQTD ( [Total Sales], dim_date[date] )
Sales MTD = TOTALMTD ( [Total Sales], dim_date[date] )
-- Indian financial year ending 31 March:
Sales FYTD = TOTALYTD ( [Total Sales], dim_date[date], "31-3" )

-- Previous periods
Sales LY   = CALCULATE ( [Total Sales], SAMEPERIODLASTYEAR ( dim_date[date] ) )
Sales PM   = CALCULATE ( [Total Sales], DATEADD ( dim_date[date], -1, MONTH ) )
Sales PQ   = CALCULATE ( [Total Sales], DATEADD ( dim_date[date], -1, QUARTER ) )

-- Growth
YoY % = DIVIDE ( [Total Sales] - [Sales LY], [Sales LY] )
MoM % = DIVIDE ( [Total Sales] - [Sales PM], [Sales PM] )
YoY Δ = [Total Sales] - [Sales LY]
```

Function	Returns	Notes
TOTALYTD (expr, dates, [year_end])	Year-to-date total	Shortcut for CALCULATE (expr, DATESYTD (dates))
DATESYTD / DATESQTD / DATESMTD	The date set from the period start to the last visible date	Use inside CALCULATE
SAMEPERIODLASTYEAR (dates)	The selected dates shifted back one year	Needs continuous dates
DATEADD (dates, n, interval)	Shift by n days, months, quarters or years	Month/quarter shifts handle month-end well
PARALLELPERIOD (dates, n, interval)	The full period shifted (full month, quarter, year)	
DATESBETWEEN (dates, start, end)	Custom ranges	
DATESINPERIOD (dates, end, -n, interval)	Rolling window ending at end	Rolling 3/12 months
PREVIOUSMONTH / PREVIOUSQUARTER / PREVIOUSYEAR	Whole previous period	
EOMONTH , EDATE , DATEDIFF	Date arithmetic	

Let's compute what these measures return for our data (monthly sales, year-to-date, last year and YoY):

Python

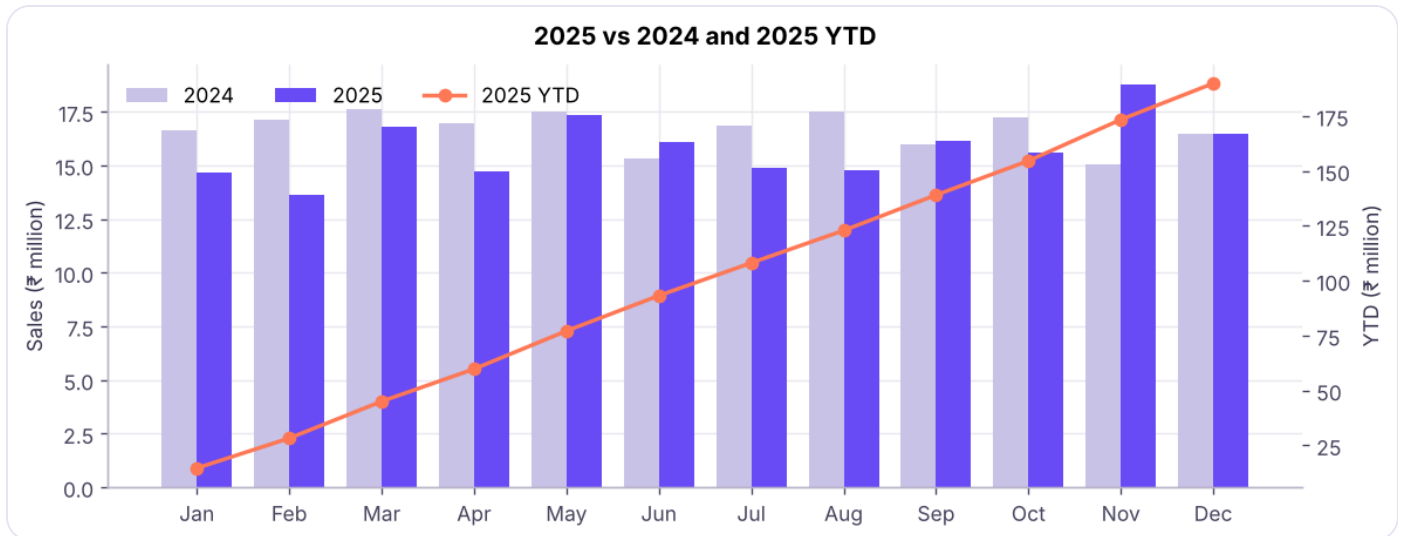
```
import pandas as pd
D = "adventure_cycles/"
f = pd.read_csv(D + "fact_sales.csv", parse_dates=["order_date"]); p = pd.read_csv(D + "dim_product.csv")
f["ym"] = f.order_date.dt.to_period("M")
mth = f.groupby("ym").sales_amount.sum().to_frame("Total Sales")
mth["Sales PM"] = mth["Total Sales"].shift(1)
mth["MoM %"] = (mth["Total Sales"] / mth["Sales PM"] - 1) * 100
mth["Sales LY"] = mth["Total Sales"].shift(12)
mth["YoY %"] = (mth["Total Sales"] / mth["Sales LY"] - 1) * 100
mth["Sales YTD"] = mth.groupby(mth.index.year)["Total Sales"].cumsum()
out = mth[mth.index.year == 2025].copy()
out["Total Sales"] = (out["Total Sales"] / 1e6).round(1); out["Sales LY"] = (out["Sales LY"] / 1e6).round(1); out["Sales YTD"] = (out["Sales YTD"] / 1e6).round(1)
out[["Total Sales", "Sales LY", "YoY %", "MoM %", "Sales YTD"]].round(1).rename_axis("2025 month")
```

OUTPUT

	Total Sales	Sales LY	YoY %	MoM %	Sales YTD
2025 month					
2025-01	14.6	16.7	-12.1	-11.2	14.6
2025-02	13.6	17.1	-20.5	-6.9	28.3
2025-03	16.8	17.6	-4.8	23.1	45.1
2025-04	14.7	17.0	-13.2	-12.1	59.8
2025-05	17.4	17.5	-1.0	17.7	77.2
2025-06	16.1	15.3	5.0	-7.4	93.2
2025-07	14.9	16.9	-11.5	-7.2	108.1
2025-08	14.8	17.5	-15.6	-1.0	122.9
2025-09	16.2	16.0	1.0	9.4	139.1
2025-10	15.6	17.3	-9.7	-3.5	154.7
2025-11	18.8	15.1	24.6	20.5	173.4
2025-12	16.5	16.5	0.0	-12.2	189.9

12 rows × 5 columns

(Values in ₹ million.) In a Power BI matrix with `dim_date[month]` on rows and these measures in the Values area you would see exactly this.



Monthly sales for 2025 against the same month in 2024, with year-to-date sales on the right axis. This is the classic "this year vs last year" chart.

COMMON MISTAKES Common time-intelligence bugs

- **Date table not marked / has gaps:** SAMEPERIODLASTYEAR returns blank or wrong values.
- **Using the fact table's date column** in the function: use `dim_date[date]`.
- **Comparing partial periods:** the current month has data for 10 days, last year's month has 31, so YoY looks terrible. Compare "to date" with `DATESYTD` and `SAMEPERIODLASTYEAR`, or restrict to complete months.
- **Fiscal year:** pass "31-3" as the year-end argument to `TOTALYTD`, and create Fiscal Year and Fiscal Quarter columns in the date table.
- **Showing a "last year" value in months with no data** (future months): wrap with `IF ([Total Sales] <> BLANK (), ... )`.

Rolling windows and running totals

DAX

```
-- Rolling 3-month sales ending at the last visible date
Sales Rolling 3M =
CALCULATE ( [Total Sales], DATESINPERIOD ( dim_date[date], MAX ( dim_date[date] ), -3,
MONTH ) )

-- Rolling 3-month average of monthly sales (average over the months in the window)
Avg Monthly Sales R3M =
DIVIDE ( [Sales Rolling 3M], 3 )      -- valid once three full months exist in the window

-- Running total across all dates up to the current one
Sales Running Total =
CALCULATE ( [Total Sales], FILTER ( ALL ( dim_date[date] ), dim_date[date] ≤ MAX (
dim_date[date] ) ) )
```

In SQL this was `SUM() OVER (ORDER BY ... ROWS BETWEEN ...)` (SQL Lesson 12). DAX does the same thing by redefining the date filter.

Python

```
mth["Rolling 3M"] = mth["Total Sales"].rolling(3).sum()
mth["Rolling 3M avg"] = mth["Total Sales"].rolling(3).mean()
(mth[["Total Sales", "Rolling 3M", "Rolling 3M avg"]].tail(6) /
1e6).round(1).rename_axis("month (₹ million)")
```

OUTPUT

	Total Sales	Rolling 3M	Rolling 3M avg
month (₹ million)			
2025-07	14.9	48.3	16.1
2025-08	14.8	45.8	15.3
2025-09	16.2	45.8	15.3
2025-10	15.6	46.5	15.5
2025-11	18.8	50.5	16.8
2025-12	16.5	50.9	17.0

6 rows × 3 columns

Ranking: RANKX and TOPN

DAX

```
-- Rank of each product category by sales within the current context
Category Rank = RANKX ( ALL ( dim_product[category] ), [Total Sales], , DESC, DENSE )

-- Show only the top 5 products: sales of a product if it is in the top 5, else blank
Top 5 Product Sales =
VAR top5 = TOPN ( 5, ALL ( dim_product[product] ), [Total Sales], DESC )
RETURN CALCULATE ( [Total Sales], KEEPFILTERS ( top5 ) )

-- Everything outside the top 5, as one "Others" value
Others Sales = [Total Sales] - [Top 5 Product Sales]
```

For simple Top N visuals, you do not need DAX: in the **Filters pane**, choose the visual's field → Filter type **Top N** → Show items Top 5 → By value Total Sales → Apply filter. Use DAX when the Top N must respond to slicers in a custom way or you need an "Others" bar.

Python

```
prod_sales = f.merge(p, on="product_key").groupby(["category",
"product"]).sales_amount.sum().sort_values(ascending=False)
top5 = prod_sales.head(5)
print((top5 / 1e6).round(1).to_string())
print(f"\nTop 5 total : {top5.sum() / 1e6:,.1f} M | Others: {(prod_sales.sum() -
top5.sum()) / 1e6:,.1f} M | Top 5 share: {top5.sum() / prod_sales.sum():.1%}")
```

OUTPUT

category	product	
Bikes	Road Bike E-331	51.3
	Mountain Bike C-245	42.4
	Touring Bike A-507	39.7
	Mountain Bike D-662	36.2
	Mountain Bike B-168	35.9

Top 5 total : 205.4 M | Others: 389.2 M | Top 5 share: 34.5%

Parameters

What-if parameter

Modeling → New parameter → Numeric range: name Price Change %, from -20 to +20, increment 1, default 0, and tick **Add slicer**. Power BI creates a table and a measure Price Change % Value. Use it in your own measure to simulate:

DAX

```
Simulated Sales = [Total Sales] * ( 1 + [Price Change % Value] / 100 )  
Simulated Profit = [Simulated Sales] - [Total Cost]
```

Dragging the slicer instantly shows what profit would be with a 5% price increase (ignoring demand changes: always say so on the page).

Python

```
base_sales, base_cost = f.sales_amount.sum(), f.total_cost.sum()  
rows = [(chg, base_sales * (1 + chg / 100) / 1e6, (base_sales * (1 + chg / 100) -  
base_cost) / 1e6) for chg in (-10, -5, 0, 5, 10)]  
pd.DataFrame(rows, columns=["Price change %", "Simulated sales (₹ M)", "Simulated profit (₹  
M)"]).round(1)
```

OUTPUT

	Price change %	Simulated sales (₹ M)	Simulated profit (₹ M)
0	-10	535.1	150.7
1	-5	564.8	180.4
2	0	594.6	210.2
3	5	624.3	239.9
4	10	654.0	269.6

5 rows × 3 columns

Power Query parameters

Home → Manage parameters creates parameters used inside queries, for example StartDate or a ServerName. Use them to switch between a test and a production database, or to filter large tables at load time. In the Service, parameters can be changed in dataset settings.

Dynamic measure switching

A very popular pattern: a slicer that lets users choose which measure the chart shows (Sales, Profit, Orders).

The modern way: Field parameters. Modeling → **New parameter** → **Fields**: add Total Sales, Total Profit, Orders; tick **Add slicer**. Power BI creates a table with the three measures; use that

table's field on a chart's axis or values and the slicer swaps the measure. A field parameter can also switch **axis** fields (Category, Country, Year).

The classic way: a disconnected table + SWITCH.

DAX

```
-- Table "Measure Selector" (Enter data): columns Measure Name, Sort
Selected Measure =
SWITCH (
    SELECTEDVALUE ( 'Measure Selector'[Measure Name], "Sales" ),
    "Sales", [Total Sales],
    "Profit", [Total Profit],
    "Orders", [Orders]
)
```

Format strings can be dynamic too (Power BI 2024+ "dynamic format string"): thousands for sales, percentage for margin.

KPI visuals and forecasting

- **KPI visual:** needs a value (indicator), a trend axis (date) and a target (for example last year's sales). Colours show good/bad.
- **Gauge:** current vs target; min/max/target set in the format pane.
- **Forecasting:** in a **line chart** with a date axis, Analytics pane → **Forecast** → set forecast length (for example 6 months), confidence interval (95%) and seasonality (12 points). Power BI uses exponential smoothing (ETS). It is a quick what-comes-next view, not a replacement for proper forecasting models (ML track).
- **Analytics pane** also offers constant lines, average, min/max, percentile lines, trend lines, and **anomaly detection**.

Newer DAX features worth knowing

Feature	What it is
Calculation groups	A reusable "time-intelligence switcher": define Current, YTD, LY, YoY % once and apply them to every measure (created in Tabular Editor or in the model view in recent versions)
Visual calculations	DAX written inside a visual that can refer to other fields in the visual (running sum, moving average, % of parent) with simple functions such as RUNNINGSUM, MOVINGAVERAGE, PREVIOUS, FIRST
DAX query view	EVALUATE queries to test and debug measures; Copilot can write DAX here
User-defined functions (DAX UDFs)	Reusable functions defined with the FUNCTION keyword (preview)
INFO.* functions	Query model metadata (INFO.TABLES(), INFO.MEASURES())

DAX

```
-- Visual calculation (typed in the visual's formula bar)
Running Sales = RUNNINGSUM ( [Total Sales] )
Moving Avg 3 = MOVINGAVERAGE ( [Total Sales], 3 )
```

DO IT WITH AI Do it with AI

Copilot can write time-intelligence measures from a sentence ("Show sales year to date versus last year"). It works best if the date table is marked, columns and tables have clear names, and you add **descriptions** to measures (Model view → Properties). Check the generated DAX uses your dim_date[date] (not the fact table's date), and confirm one value by hand.

Summary

KEY TAKEAWAYS

- Time intelligence needs a **continuous, marked date table**, related to the fact; always reference the date table's column.
- **TOTALYTD**, **SAMEPERIODLASTYEAR**, **DATEADD**, **PARALLELPERIOD**, **DATESINPERIOD** build YTD, last-year, MoM, YoY and rolling windows; **DIVIDE (current - previous, previous)** gives growth.
- **RANKX** ranks, **TOPN** selects the top items (add an "Others" measure); the Filters-pane **Top N** is the no-code route.
- **What-if parameters** simulate scenarios; **field parameters** let users switch measures or axes; the disconnected table + **SWITCH** pattern does the same in older models.
- Know calculation groups, visual calculations and the DAX query view for advanced work.

Quick quiz

6 questions · tap an answer to check it

1. Which measure shows year-to-date sales?

A `SUM ( fact_sales[sales_amount] )`

B `TOTALYTD ( [Total Sales], dim_date[date] )`

C `SAMEPERIODLASTYEAR ( [Total Sales] )`

D `RANKX ( dim_date, [Total Sales] )`

Answer: B. `TOTALYTD` accumulates from the start of the year to the current date.

2. SAMEPERIODLASTYEAR returns blank for every month. What should you check first?

A The theme

B That the date table is continuous, marked as a date table and related to the fact

C The colour of the visual

D The number of slicers

Answer: B. Time intelligence relies on a valid date table.

3. Which formula gives month-on-month growth?

A `[Total Sales] - [Sales PM]`

B `DIVIDE ( [Total Sales] - [Sales PM], [Sales PM] )`

C `SUM ( [Sales PM] ) / [Total Sales]`

D `DATEADD ( [Total Sales], -1, MONTH )`

Answer: B. Growth rate = (current – previous) ÷ previous, using DIVIDE.

4. What does a field parameter let users do?

A Change the report theme

B Switch which measures or fields a visual shows with a slicer

C Refresh the data

D Create relationships

Answer: B. Field parameters create a slicer that swaps fields in visuals.

5. In the Indian financial year (April–March), how do you build YTD sales?

A TOTALYTD ([Total Sales], dim_date[date]) always works

B TOTALYTD ([Total Sales], dim_date[date], "31-3")

C DATESMTD

D PARALLELPERIOD (...)

Answer: B. The third argument sets the year-end date.

6. What is the correct way to compute "Others" after a Top 5?

A Add a second visual

B Total sales minus the Top 5 measure

C Use FILTER ALL

D Use PREVIOUSMONTH

Answer: B. Others = overall total – Top N total.

EXERCISE Time-intelligence measures

For Adventure Cycles write measures for: (1) sales for the last 12 months ending at the selected date; (2) YoY % by category; (3) the best month of each year; (4) a what-if parameter "Discount %" from 0 to 30 that reduces Total Sales and shows simulated profit margin.

Show solution (worked answer)

DAX

```
-- (1)
Sales L12M = CALCULATE ( [Total Sales], DATESINPERIOD ( dim_date[date], MAX (
dim_date[date] ), -12, MONTH ) )
-- (2) put dim_product[category] on rows and dim_date[year] on columns
YoY % = DIVIDE ( [Total Sales] - [Sales LY], [Sales LY] )
-- (3)
Best Month =
VAR t = ADDCOLUMNS ( VALUES ( dim_date[month] ), "@s", [Total Sales] )
RETURN MAXX ( TOPN ( 1, t, [@s], DESC ), dim_date[month] )
-- (4) parameter table 'Discount %' with measure [Discount % Value]
Simulated Margin = DIVIDE ( [Total Sales] * ( 1 - [Discount % Value] / 100 ) - [Total
Cost], [Total Sales] * ( 1 - [Discount % Value] / 100 ) )
```

Validate (2) and (3) against pandas group-bys.

INTERVIEW CORNER Questions you may be asked

Q: What do you need for time-intelligence functions to work? A date table with continuous dates covering the data, marked as a date table, related to the fact table by date.

Q: How would you calculate YoY growth? Sales LY = CALCULATE ([Total Sales], SAMEPERIODLASTYEAR (dim_date[date])) then YoY % = DIVIDE ([Total Sales] - [Sales LY], [Sales LY]).

Q: How can users choose which metric a chart shows? Field parameters (modern) or a disconnected table with a SWITCH (SELECTEDVALUE (...), ...) measure.

Q: How do you show Top N plus an "Others" bar? A Top N measure using TOPN/KEEPFILTERS and an Others measure = total – Top N.

Visuals, Filters, Drill-through, Tooltips, Buttons & Report Design

Turn a model into an interactive, well-designed report. Choose visuals, format them (cards, matrix, charts, gauge, KPI, maps, images), use visual, page and report filters, edit interactions, drill-down and drill-through, custom tooltips, bookmarks and buttons, conditional formatting, themes and design principles.

● Power BI classes 2–4

✦ Power BI Desktop 2026

Intermediate

🕒 150 min

By the end of this lesson you can

- Pick the right visual for each question and configure its field wells and formatting
- Apply visual, page and report-level filters, Top N, relative date filters and slicer sync
- Control cross-filtering with Edit interactions; add drill-down, drill-through and report-page tooltips
- Build navigation with buttons, bookmarks, page navigation and a selection pane
- Apply themes and conditional formatting, and follow layout, colour and accessibility rules

Choosing the visual

Visual chooser

Compare categories

Clustered bar/column (ranked bars for long names)

Trend over time

Line, area, or column for few periods

Part-to-whole

Stacked bar, 100% stacked, treemap, donut (≤ 3 parts)

Relationship

Scatter or bubble (with a play axis for time)

Single KPI

Card, multi-row card, KPI, gauge

Detail & totals

Table, matrix (pivot) with conditional formatting

Geography

Map, filled map, shape map, Azure map

Flow & contribution

Waterfall, funnel, ribbon, decomposition tree

Explain / AI

Key influencers, Smart narrative, Q&A, anomaly detection, Copilot narrative

Match the visual to the question.

Visual	Fields	Good for	Watch out
Card / multi-row card	Values	One or few headline numbers	Format display units (K, M, Cr) and add a clear label
Clustered bar / column	Axis, Values, Legend	Comparisons	Sort descending, start at zero, ≤ 15 bars
Stacked / 100% stacked	Axis, Legend, Values	Composition	Only the bottom segment is easy to compare
Line / area	X-axis (date), Y-axis, Legend	Trends	Limit to ~5 lines; use a continuous date axis
Line + clustered column	Shared axis, column values, line values	Two measures with different scales (secondary axis)	Label both axes
Waterfall	Category, Y-axis	Bridge from start to end (profit walk)	Needs an ordered category
Scatter / bubble	X, Y, Size, Details, Play axis	Relationships and outliers	Add Details so each point is an entity
Treemap	Group, Values	Hierarchical share of total	Many tiny rectangles are unreadable
Donut / pie	Legend, Values	2–3 parts	More than 4 slices = use a bar
Matrix	Rows, Columns, Values	Pivot tables with drill and subtotals	Use conditional formatting for heat maps
Table	Columns	Exact values, detail list	Keep it short; a table is not a dashboard
KPI / Gauge	Value, Target, Trend axis	Progress to a target	Gauges waste space; prefer KPI cards
Funnel	Stage, Values	Drop-off between stages	
Map / Filled map	Location, Size, Legend	Geography	Geocoding needs clean place names (City, State, Country)
Slicer	Field	Filter the page	Styles: list, dropdown, tiles, between (range), relative date
Decomposition tree	Analyze, Explain by	Drill into drivers	Interactive exploration

Visual	Fields	Good for	Watch out
Key influencers	Analyze, Explain by	AI finds what drives a metric	Needs a binary/numeric target
Smart narrative	Auto-text	Written summary that updates	Check the wording

Formatting essentials

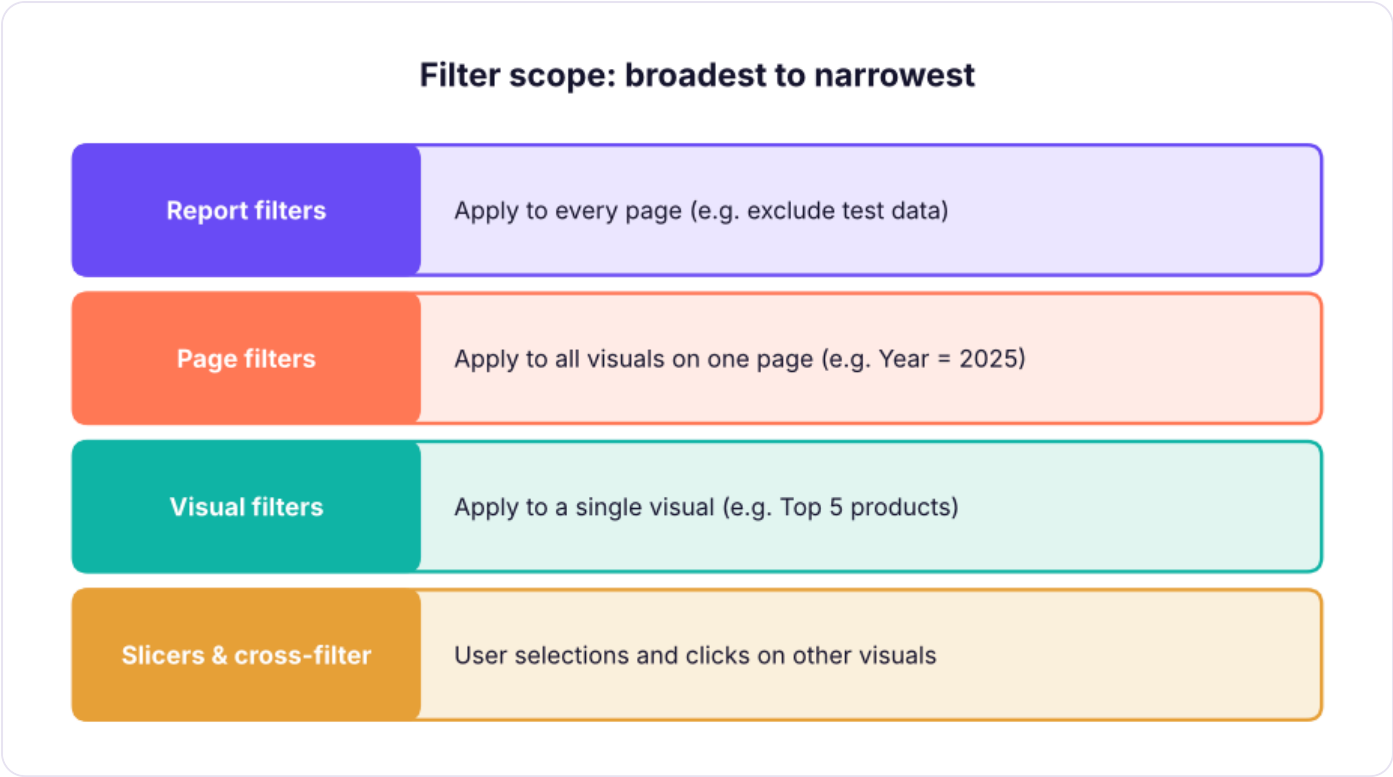
Select a visual → **Format visual** (paint-roller icon): **Visual** tab (axis, labels, colours, legend, gridlines, title) and **General** (title, size, position, background, border, alt text).

Task	Where
Display units and decimals	Format → Callout value / Data labels → Display units: Thousands, Lakhs (not built in; use Millions) or custom format strings
Sort	... menu → Sort axis (by value or by another field)
Colour a highlighted series	Format → Colors → per-category colours or conditional
Titles that state the insight	General → Title → fx (dynamic title from a measure)
Accessibility	General → Alt text; keep contrast high; do not rely on colour alone
Copy formatting	Home → Format painter
Align / distribute	Format → Arrange → Align and distribute; hold Shift to multi-select
Group visuals	Select several → Format → Group

TIP Dynamic titles

Create a measure `Title Sales = "Sales by " & SELECTEDVALUE ( dim_territory[country], "all countries" ) & " in " & SELECTEDVALUE ( dim_date[year], "all years" )` and bind it in Title → fx → *Field value*. The title now reflects the user's selections.

Filters: three levels



The three filter levels. Visual filters affect one visual, page filters one page, report filters every page.

Use the **Filters pane** (right side). Filter types:

Filter type	Behaviour
Basic	Tick values from a list
Advanced	"contains", "starts with", "is greater than", with AND/OR
Top N	Top/Bottom N items by a measure (even one not in the visual)
Relative date	"Last 6 months", "this year", "next 7 days"
Relative time	Last 4 hours etc.

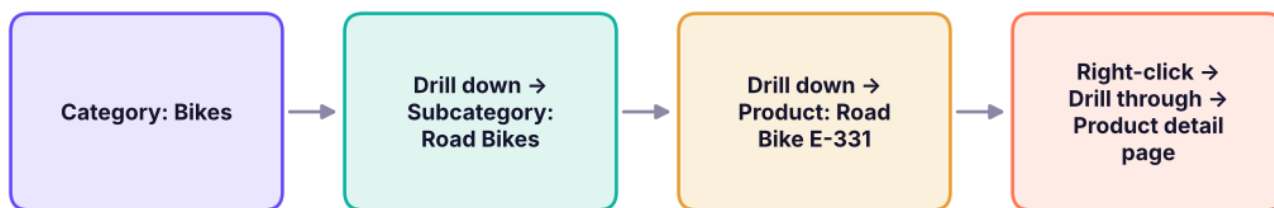
Tips: **Require single selection** in a slicer; **Sync slicers** across pages (View → Sync slicers); the **"Include" / "Exclude"** options on a selection filter in or out; lock the filters pane or hide it for consumers.

Edit interactions

By default clicking a bar **cross-filters** other visuals. Select the source visual → Format → **Edit interactions** → then on each target visual click the **Filter**, **Highlight** or **None** icon. Example: a KPI card for "overall sales" should ignore the clicks on the country chart (set to None), while the product chart should filter.

Drill-down and drill-through

Drill-down vs drill-through



Drill-down moves through a hierarchy inside one visual. Drill-through jumps to another page that is filtered by the selected item.

- **Drill-down:** needs a hierarchy (Lesson 2). Use the arrow buttons at the top of the visual: ↑ drill up, ↓ drill mode, ⇅ go to the next level in the hierarchy, ⤴ expand all.
- **Drill-through:** create a **detail page** → in the Visualizations pane, drag **product** into **Drill through** → **Add drill-through fields here** → Power BI adds a back button automatically. Users right-click a product in any visual → **Drill through** → **Product detail**. Use it for "customer 360", "order detail" and "product detail" pages. **Keep all filters** keeps slicer context.
- **Cross-report drill-through** links to a separate report in the Service.

Report-page tooltips

A **tooltip page** shows a mini-report when hovering over a data point.

STEP BY STEP Build one

1. Add a new page; Format page → Page information → **Allow use as tooltip** on; Page size → **Tooltip**.
2. Add visuals (a KPI, a small trend line, top 3 products) using fields that respond to the hovered item.
3. In the main visual: Format → General → **Tooltips** → Type **Report page** → choose the page.
4. Hide the tooltip page from navigation (right-click the tab → Hide page).

Buttons, bookmarks and navigation

Feature	What it does	Use
Buttons (Insert → Buttons)	Back, Bookmark, Page navigation, Q&A, Web URL, Apply all slicers, Clear all slicers	Menus, navigation bars, reset filter
Bookmarks (View → Bookmarks)	Save the current state (filters, slicer selections, visibility, spotlight) of a page	Story-telling views, toggle between visuals
Selection pane (View → Selection)	List, name, show or hide every object on a page	Required for bookmark toggles
Page navigator (Insert → Navigator)	Auto-generated tab buttons for pages or bookmarks	Menu
Conditional visibility	Bookmarks that show/hide visuals	"Chart ⇌ Table" toggle

STEP BY STEP Chart / table toggle with bookmarks

1. Place a column chart and a table on top of each other.
2. Selection pane: hide the table → View → Bookmarks → **Add** → name it "Chart view"; make the table visible and hide the chart → Add → "Table view". Untick **Data** in each bookmark's options so slicers are not reset.
3. Insert two buttons and set Action → **Bookmark** → choose the matching bookmark.

Conditional formatting

Where	How
Table / matrix cells	Format → Cell elements → Background colour / Font colour / Data bars / Icons / Web URL → fx → <i>Gradient, Rules</i> or <i>Field value</i>
Column / bar colour	Format → Colors → fx → rules (green above target, red below)
Card text colour or title	fx on the callout/title
Based on a measure	Create a measure returning a hex colour: <code>Margin Colour = IF ([Profit Margin %] ≥ 0.35, "#1E8E5A", "#C0392B")</code> then Field value

Subcategory	Sales (₹ M)	Profit (₹ M)	Margin %
Mountain Bikes	161.9	54.9	33.9%
Road Bikes	155.7	56.1	36.0%
Touring Bikes	149.2	55.2	37.0%
Frames	76.1	27.3	35.8%
Wheels	33.4	10.5	31.3%
Helmets	8.2	2.7	33.1%
Jerseys	5.8	2.2	37.2%
Gloves	2.7	0.8	30.8%
Bottles	1.6	0.5	34.0%

A matrix with conditional formatting: a heat-map background on sales and traffic-light colours on margin (rules: $\geq 36\%$ green, otherwise amber/red).

Themes

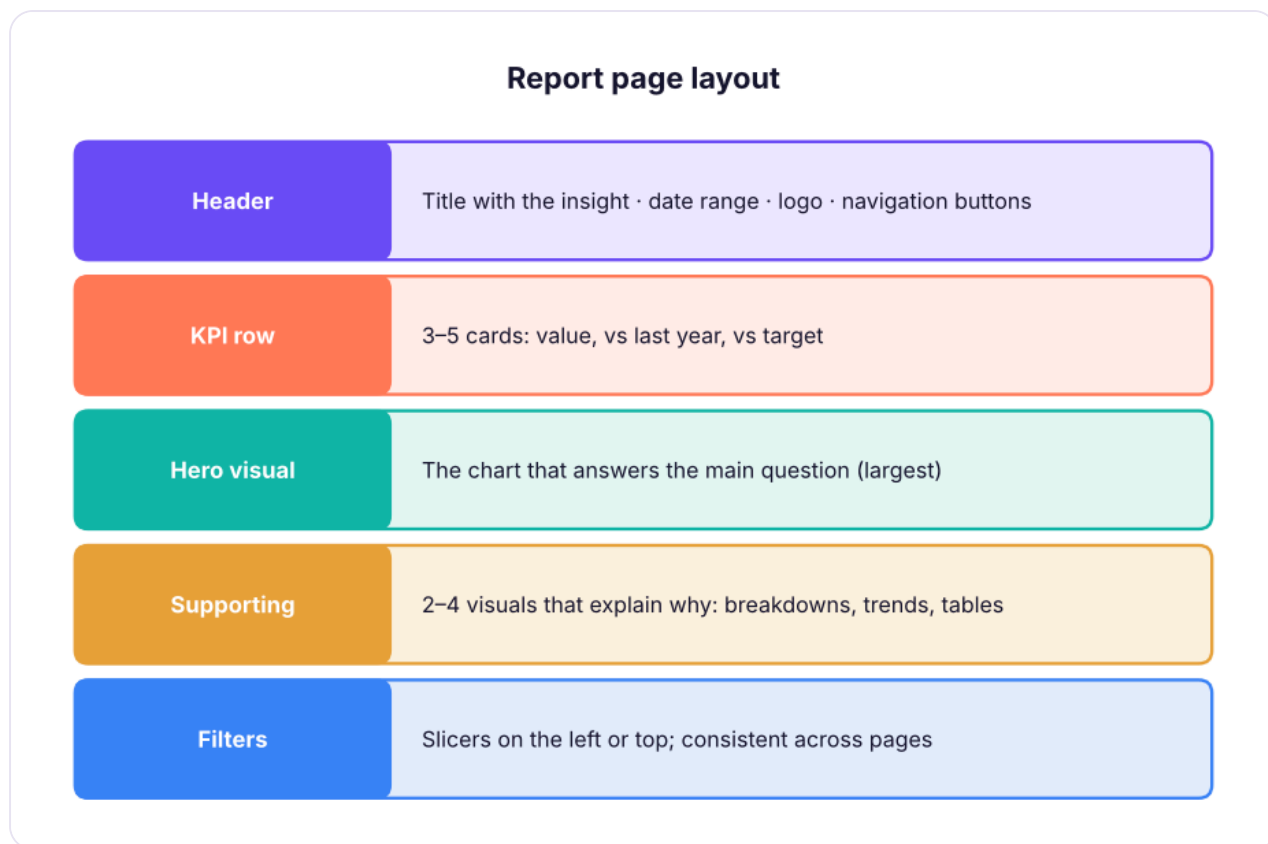
View → Themes offers built-in themes; you can download community themes or **Customize current theme** (colours, fonts, defaults). A **theme JSON file** (View → Themes → Browse for themes) applies one consistent look to every visual:

JSON

```
{
  "name": "SkillNavora",
  "dataColors": ["#6C4CF5", "#FF7A59", "#12B5A6", "#E9A23B", "#3B82F6", "#E04F8A"],
  "background": "#FFFFFF", "foreground": "#17142E", "tableAccent": "#6C4CF5",
  "textClasses": { "title": { "fontFace": "Segoe UI Semibold", "fontSize": 14, "color":
"#17142E" } }
}
```

Apply the company theme to every report so they look consistent, and edit colours in one place.

Report design principles



A layout that works: headline KPIs on top, the main insight chart large, supporting visuals below, filters on one side, consistent spacing.

1. **Start from the audience and decision**, not from the data: what will they do differently?
2. **Hierarchy**: most important top-left; big numbers first; details lower.
3. **Less is more**: 5–8 visuals per page; remove gridlines, borders and chart junk.
4. **Consistent colour logic**: one brand colour plus one highlight; red and green only for bad/good; the same colour always means the same thing.
5. **Align and space** visuals on a grid; same sizes for similar charts.
6. **Label for meaning**: insight titles, units (₹, %), clear axis titles, "as of" dates.
7. **Accessible**: contrast, alt text, tab order (Format → Tab order), do not use colour alone, test on mobile layout (View → Mobile layout).
8. **Performance**: fewer visuals, summarised data, no huge tables; check Optimize → Performance analyzer.

Q&A, images and mobile

- **Q&A visual**: users type "total sales by country in 2025" and get a chart. Improve it with **synonyms** (Model view → Linguistic schema / Q&A setup).

- **Images and logos:** Insert → Image (or text box). For images **in data** (product photos), set the column's **Data category** → **Image URL**.
- **Mobile layout:** View → Mobile layout; drag key visuals into a phone-shaped canvas.
- **Accessibility and exporting:** export to PDF or PowerPoint (File → Export), analyse in Excel, subscribe in the Service.

2026 UPDATE Copilot and AI visuals in 2026

Copilot can generate a report page from a prompt, suggest visuals for your fields, write summaries ("Summarise this page") and answer questions about the data. **Key influencers**, **Decomposition tree** and **Smart narrative** are built-in AI visuals; **anomaly detection** (Analytics pane) flags unusual spikes. Always verify AI-generated visuals and text against real numbers.

A mini case: the Adventure Cycles overview page

Plan the page before you build it:

Element	Fields / measure	Visual
Title	Dynamic title measure	Text box
KPI row	Total Sales , Total Profit , Profit Margin % , Orders , each with vs-last-year	4–5 cards
Hero	Monthly sales and profit margin	Line + column
Breakdown 1	Sales by category (Top N subcategories)	Bar
Breakdown 2	Sales by country	Map or bar
Drill-through	Product detail page	Table + KPIs
Filters	Year, Country, Category	Slicers

Adventure Cycles: sales overview 2023–2025

Total sales

₹59.5 Cr

Total profit

₹21.0 Cr

Profit margin

35.3%

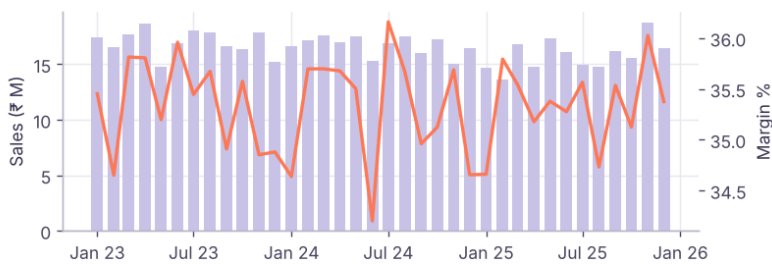
Orders

9,000

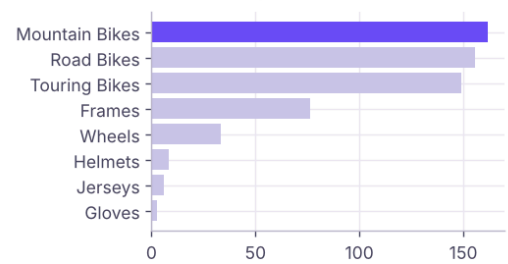
Customers

1,200

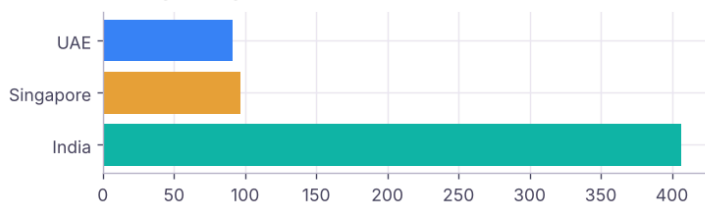
Monthly sales and margin



Top subcategories (₹ M)



Sales by country (₹ M)



Category mix



The Adventure Cycles overview page from the real data: KPI cards, monthly trend, subcategory ranking and country share.

Summary

KEY TAKEAWAYS

- Choose visuals from the question; format with insight titles, units, sorting and accessible colours.
- Filters work at **visual, page and report** level (Basic, Advanced, Top N, Relative date); use slicers, sync them across pages, and **Edit interactions** to control cross-filtering.
- **Drill-down** needs a hierarchy; **drill-through** jumps to a detail page; **tooltip pages** show mini-reports on hover.
- **Buttons + bookmarks + the Selection pane** build navigation and chart/table toggles.
- Use **conditional formatting** (rules, gradients, field value from a measure) and **themes** for consistency.
- Design for the decision: hierarchy, few visuals, consistent colour logic, alignment, labels, accessibility and mobile layout.

Quick quiz

6 questions · tap an answer to check it

1. Which filter level affects every page of a report?

- A Visual-level
- B Page-level
- C Report-level (All pages)**
- D Slicer-level

Answer: C. Report filters apply to all pages.

2. How do you jump from a product in a chart to a detail page for that product?

- A Drill-down
- B Drill-through (create a drill-through page with the product field)**
- C Add a slicer
- D Use a tooltip

Answer: B. Drill-through passes the selected value as a filter to another page.

3. What do bookmarks capture?

- A Only the page name
- B The state of a page: filters, slicers, visibility and more**
- C The data model
- D The theme

Answer: B. Bookmarks save a view that buttons can switch to.

4. Which feature stops a card from reacting when you click a bar chart?

- A Sync slicers
- B Edit interactions → set the card to None
- C Drill-through
- D Alt text

Answer: B. Edit interactions controls how visuals filter each other.

5. When is a donut chart appropriate?

- A With 12 categories
- B With 2–3 parts of a whole
- C To show a time trend
- D To compare 20 products

Answer: B. More than a few slices are hard to compare; use bars.

6. What is the benefit of a theme JSON file?

- A One consistent set of colours and fonts applied to all visuals and reusable across reports
- B It refreshes data
- C It creates relationships
- D It adds measures

Answer: A. Themes standardise the look and make branding quick.

EXERCISE Build the overview page

Using the Adventure Cycles model: (1) build the overview page above; (2) add a **Year** slicer and a **Country** slicer; (3) create a drill-through page for **Product** with sales, profit, margin and a monthly trend; (4) add a bookmarked toggle between "Sales" and "Profit" using a field parameter or bookmarks; (5) apply conditional formatting to the margin in a matrix and write a dynamic title.

Show solution (worked answer)

(1) Cards for the five measures; line + clustered column for monthly sales and margin; bar for top subcategories; bar for country; donut for category mix. (2) Slicer visuals with `dim_date[year]` (tile style) and `dim_territory[country]` (dropdown). (3) New page → drag `dim_product[product]` into Drill through; add a back button. (4) Modeling → New parameter → Fields → add `Total Sales`, `Total Profit`; use the parameter on the chart values; place its slicer as a toggle. (5) Matrix of `subcategory` × `country` with `Profit Margin %`; Cell elements → Background colour → Rules; title measure with `SELECTEDVALUE`.

INTERVIEW CORNER Questions you may be asked

Q: How do you design an effective Power BI dashboard? Start from the audience and decision; put headline KPIs top-left; use few visuals with insight titles; consistent colour and spacing; allow drill-down/through; test with users; check performance and accessibility.

Q: Drill-down vs drill-through? Drill-down navigates levels of a hierarchy within a visual; drill-through navigates to another page filtered by the selected item.

Q: What are the filter levels and how do slicers differ? Visual, page and report filters are set by the author in the Filters pane; slicers are on-canvas controls that users change, and they can be synced across pages.

Q: How would you let users toggle between a chart and a table? Two bookmarks that show/hide the visuals through the Selection pane, and two buttons linked to those bookmarks.

Power BI Service, Connectivity Modes, Security & Capstone

Take a report from Desktop to the people who need it. Understand Import, DirectQuery, Live connection, composite models and Direct Lake, publish to the Service, schedule refresh with a gateway, share through workspaces and apps, secure data with row-level security, tune performance, work with Copilot and Fabric, and complete the Adventure Cycles capstone.

● Power BI class 4

✦ Power BI Service & Microsoft Fabric 2026

Advanced

🕒 150 min

By the end of this lesson you can

- Compare Import, DirectQuery, Live connection, Composite and Direct Lake modes and choose one
- Publish a report, configure scheduled refresh and a gateway, and share via workspaces, apps and links
- Implement and test row-level security (static and dynamic) with DAX filters
- Use Performance Analyzer and best practices to speed up a model
- Describe Microsoft Fabric, Copilot and governance basics, and deliver an end-to-end capstone

Connectivity modes

How does Power BI get its data: **copy it** into the model (Import) or **ask the source each time** (DirectQuery)?

Import vs DirectQuery

Import mode

- Data is copied into the model (VertiPaq)
- Very fast visuals, full DAX
- Needs scheduled refresh; data is as old as the last refresh
- Size limits: 1 GB Pro per model (up to 10 GB+ in Premium)
- Default and best for most reports

DirectQuery

- No data stored; queries run on the source
- Near-real-time data, huge tables stay in the database
- Slower visuals; source must be fast
- Some DAX and Power Query features limited
- Use for very large or live operational data

Import copies data into Power BI's compressed in-memory model; DirectQuery sends a query to the source for every interaction.

Mode	Data stored in Power BI?	Refresh	Typical use
Import	Yes (compressed copy)	Scheduled (up to 8/day Pro, 48/day Premium)	Most reports; Excel, CSV, SQL
DirectQuery	No	Live per interaction	Large SQL databases, near-real-time dashboards
Live connection	No: connects to an existing semantic model (Analysis Services or a published Power BI dataset)	Live	One governed model shared by many reports
Composite model	Mix: some tables Import, some DirectQuery, plus other semantic models	Both	Large fact in DirectQuery, small dimensions imported (dual storage mode)
Direct Lake (Fabric)	Reads Delta tables in OneLake directly	Automatic on data change	Import-like speed with no copy; Fabric only

TIP A practical rule

Start with **Import**. Move to DirectQuery or Direct Lake only if data volume or freshness requirements force you to. Reduce the data first: only needed columns, aggregated tables, filtered date range.

Publishing and the Service

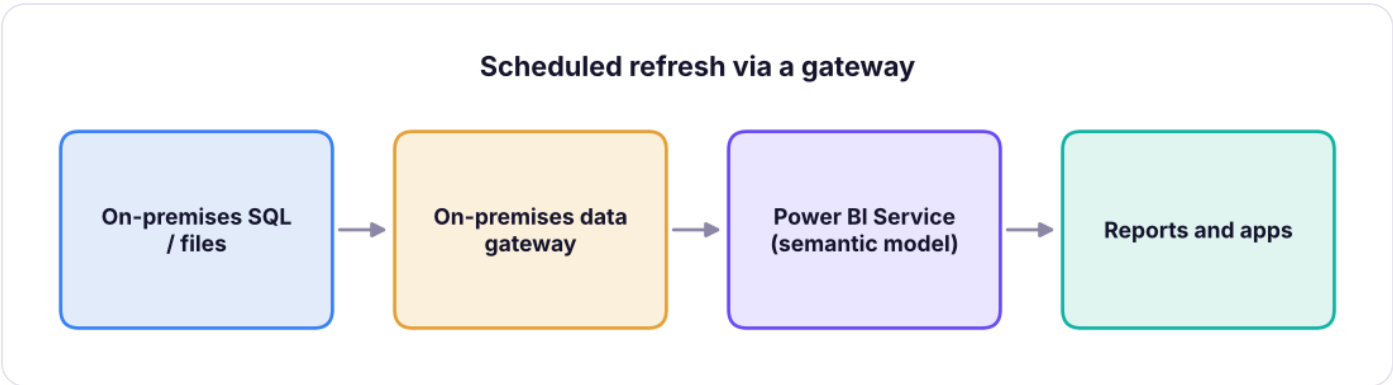
STEP BY STEP Publish a report

1. In Desktop: **Home** → **Publish** → sign in with a work/school account → choose a **workspace** (not "My workspace" if you want to share).
2. The Service now has a **semantic model** (the data and model) and a **report** (the visuals): the same objects in two items.
3. Open the report in the browser, check it, and **Share** with people or add it to an **app**.
4. Configure **scheduled refresh** (Settings → Semantic model → Scheduled refresh) with credentials; for on-premises sources install a **gateway**.

Workspaces, apps and sharing

Concept	Meaning
Workspace	A shared container for semantic models, reports, dashboards. Roles: Admin, Member, Contributor, Viewer
App	A packaged, read-only, curated set of reports/dashboards published to a wider audience (best way to distribute)
Share link	Give a person or group access to one report (they need appropriate licence)
Dashboard	One page of pinned tiles from several reports; supports alerts and mobile
Subscriptions & alerts	Email snapshots on a schedule; data alerts on card/gauge thresholds
Export / Analyze in Excel / Embed	PDF/PowerPoint export, live pivot tables on the model, secure embedding in portals
Deployment pipelines	Move content Dev → Test → Prod with controlled changes
Endorsement, sensitivity labels, lineage view	Governance: certified datasets, data classification, impact analysis

Refresh and gateways



A scheduled refresh for on-premises data goes through a gateway installed inside the company network.

- **Cloud sources** (Azure SQL, SharePoint Online, OneDrive files) refresh directly without a gateway.
- A **standard gateway** serves many users; a **personal gateway** is for one person's testing.
- **Incremental refresh** (right-click a table → Incremental refresh) loads only new or changed partitions (for example the last 7 days) instead of the whole fact table: essential for big tables. It needs `RangeStart` and `RangeEnd` parameters (Power Query).

- Refresh failures (expired credentials, changed schema, gateway offline) appear in the refresh history; enable failure notifications.

Row-level security (RLS)

RLS restricts the rows a user can see: a country manager sees only their country.

Static RLS

STEP BY STEP Create roles in Desktop

1. **Modeling** → **Manage roles** → **Create**. Name the role `India`.
2. Select the table `dim_territory` and add a DAX filter: `[country] = "India"`.
3. Create roles `UAE` and `Singapore` the same way. **Save**.
4. **View as** (Modeling → View as) → choose a role to test what that person sees.
5. Publish; in the Service: **Semantic model** → **Security** → add users or groups to each role.

Because filters flow from `dim_territory` to `fact_sales` through the relationship, the India role sees only India's sales everywhere:

Python

```
import pandas as pd
D = "adventure_cycles/"
f = pd.read_csv(D + "fact_sales.csv"); t = pd.read_csv(D + "dim_territory.csv"); m =
f.merge(t, on="territory_key")
role_view = m.groupby("country").agg(orders=("order_number", "count"), sales=
("sales_amount", "sum"))
role_view["sales"] = (role_view.sales / 1e6).round(1)
role_view.rename(columns={"sales": "sales (₹ M) visible to that role"}).assign(**{"all-
company total (₹ M)": round(f.sales_amount.sum() / 1e6, 1)})
```

OUTPUT

	orders	sales (₹ M) visible to that role	all-company total (₹ M)
country			
India	6135	406.9	594.6
Singapore	1487	96.7	594.6
UAE	1378	91.0	594.6

3 rows × 3 columns

Dynamic RLS

One role for everyone, driven by a **security table** mapping user emails to territories:

DAX

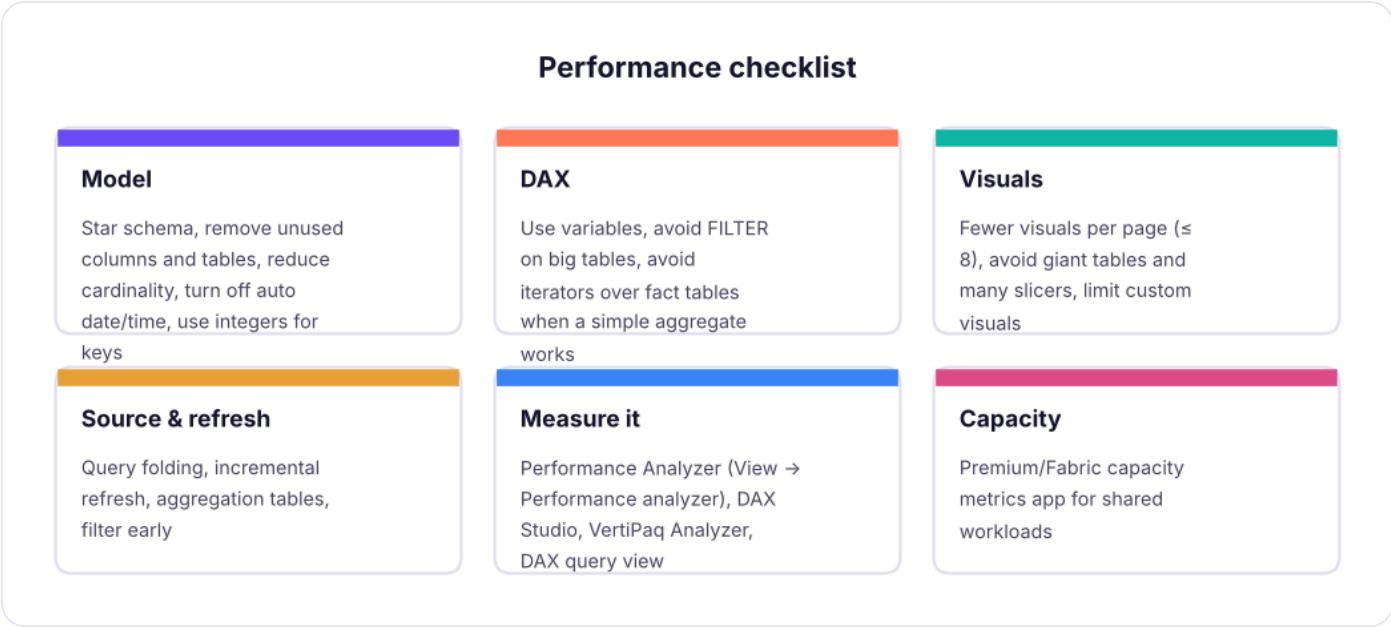
```
-- Table 'UserSecurity'(user_email, territory_key); role filter on dim_territory:
[territory_key]
    IN CALCULATETABLE ( VALUES ( UserSecurity[territory_key] ),
                        UserSecurity[user_email] = USERPRINCIPALNAME () )
```

USERPRINCIPALNAME() returns the signed-in user's email. Add a person to the table and they get access automatically: no role edits.

WATCH OUT RLS pitfalls

- RLS applies to **Viewers** only. Workspace Admins, Members and Contributors can see all data.
- RLS does not hide column names or the existence of tables; use **object-level security** for that.
- Filters must flow along relationships: apply RLS to a **dimension**, not the fact, where possible.
- RLS with DirectQuery depends on the source's behaviour; test with *View as*.
- Do not store secrets or personal data you do not need in the model. In India, handling personal data is governed by the DPDP Act 2023.

Performance tuning



Where Power BI report time goes, and where to look.

Performance Analyzer (View → Performance analyzer → Start recording → refresh visuals) shows, for each visual, the time spent in **DAX query**, **Visual display** and **Other**. If DAX dominates, optimise the measure or model; if display dominates, simplify the visual.

Practical wins:

Problem	Fix
File is huge	Remove columns you do not use; store IDs as integers; split date and time; use Whole number/Fixed decimal rather than text
Slow card or matrix	Rewrite measures with variables; avoid nested FILTER (ALL (bigTable))
Slow refresh	Query folding, incremental refresh, filter at source
Slow DirectQuery	Aggregations table (Import) in front of the big table; index the source
Many identical measures	Calculation groups or one base measure referenced by others

Microsoft Fabric, Copilot and AI

Microsoft Fabric unifies data engineering, data factory, data warehouse, data science, real-time analytics and Power BI over one data lake called **OneLake** (Delta/Parquet files). Power BI's semantic models can use **Direct Lake** over lakehouse tables. For data professionals: learn SQL (this track), Python (next track), and then Fabric notebooks, pipelines and lakehouses.

Fabric item	Role
Lakehouse / Warehouse	Stores data (files + SQL endpoint)
Dataflow Gen2	Power Query in the cloud
Notebook	Python/Spark for data science and engineering
Pipeline	Orchestrates data movement
Semantic model	The Power BI model (formerly "dataset")
Eventstream / Real-time	Streaming data

Copilot in Power BI (needs a Fabric or Premium capacity): *create a report page* from a prompt, *suggest measures* and *write DAX*, *summarise* a page or report, and *answer questions* about the data. **Prep data for AI** (add descriptions, synonyms, verified answers) improves the answers. Treat it as a fast assistant, not an authority.

Governance and good practice

1. **One certified semantic model per subject area**, many thin reports on top (live connection).
2. Name items consistently (Sales - Overview, Sales_Model); keep an **owner** and a description.
3. Version control: **PBIP** + Git, deployment pipelines, a test workspace.
4. Document measures (description property) and keep a measure dictionary.
5. Limit **who can export** data; use sensitivity labels.
6. Monitor usage (usage metrics report) and remove unused reports.
7. Train users: the best dashboard is useless if nobody trusts or understands it.

Capstone: Adventure Cycles performance review

CASE STUDY The brief

The sales director wants a 3-page Power BI report for the Adventure Cycles leadership team and country managers: **(1) Executive summary, (2) Products and customers, (3) Country detail (drill-through)**. It must show YTD and vs-last-year numbers, let managers filter by year, country and category, and each country manager must see **only their own country**.

Deliverables checklist

Step	What to produce
1. Requirements	Audience, questions, KPIs, refresh frequency, security
2. Data	Load the five CSV tables; clean in Power Query
3. Model	Star schema, relationships, marked date table, hierarchies (Lessons 1–2)
4. Measures	Sales, Profit, Margin %, Orders, AOV, YTD, LY, YoY %, Rank, Top 5 + Others (Lessons 3–4)
5. Report	Three pages, theme, tooltips, drill-through, bookmarks, mobile layout (Lesson 5)
6. Security	RLS roles by country, tested with <i>View as</i>
7. Publish	Workspace, scheduled refresh, app, subscriptions (this lesson)
8. Documentation	One-page "how to read this report" and a measure dictionary

The analysis the report must support: verify your page numbers against this independent calculation.

Python

```
m = f.merge(t, on="territory_key").merge(pd.read_csv(D + "dim_product.csv"),
on="product_key"); m["profit"] = m.sales_amount - m.total_cost
m["order_date"] = pd.to_datetime(m.order_date); m["year"] = m.order_date.dt.year
kpi = m.groupby("year").agg(sales=("sales_amount", "sum"), profit=("profit", "sum"),
orders=("order_number", "nunique"))
kpi["margin_%"] = (kpi.profit / kpi.sales * 100).round(1); kpi["YoY_sales_%"] =
(kpi.sales.pct_change() * 100).round(1)
kpi[["sales", "profit"]] = (kpi[["sales", "profit"]] / 1e6).round(1)
kpi
```

OUTPUT

	sales	profit	orders	margin_%	YoY_sales_%
year					
2023	204.2	72.2	3004	35.4	NaN
2024	200.4	70.8	3022	35.3	-1.9
2025	189.9	67.2	2974	35.4	-5.2

3 rows × 5 columns

Python

```
c = m.groupby("country").agg(sales=("sales_amount", "sum"), profit=("profit", "sum"))
c["margin_%"] = (c.profit / c.sales * 100).round(1); c["share_%"] = (c.sales /
c.sales.sum() * 100).round(1); c[["sales", "profit"]] = (c[["sales", "profit"]] /
1e6).round(1)
bike_share = (m[m.category == "Bikes"].sales_amount.sum() / m.sales_amount.sum() *
100).round(1)
print(c.to_string()); print(f"\nBikes are {bike_share}% of total sales; Components
{m[m.category == 'Components'].sales_amount.sum() / m.sales_amount.sum() * 100:.1f}%.")
```

OUTPUT

	sales	profit	margin_%	share_%
country				
India	406.9	144.1	35.4	68.4
Singapore	96.7	34.1	35.2	16.3
UAE	91.0	32.0	35.1	15.3

Bikes are 78.5% of total sales; Components 18.4%.

KEY TAKEAWAYS

What a good leadership summary says

1. **Sales are shrinking, not growing:** ₹204.2 M (2023) → ₹200.4 M (2024, -1.9%) → ₹189.9 M (2025, -5.2%), while the margin holds steady at about 35%. The problem is **volume or demand**, not pricing or cost.
2. **Bikes** generate 78.5% of sales and **India** 68.4%, so any decline in Indian bike sales moves the whole company: a concentration risk.
3. Singapore (16.3% of sales) and the UAE (15.3%) earn the same ~35% margin as India, so growth there is as profitable as at home.
4. Recommended next steps: find where the 2025 decline comes from (by category, country and month, using the YoY measures from Lesson 4), set recovery targets per country, and track them monthly in the report.

(Verify each statement against the tables above before you present it; the numbers, not the template, are the story.)

Summary

KEY TAKEAWAYS

- **Import** is fast and the default; **DirectQuery** keeps data in the source (slower, near-live); **Live connection** reuses a shared semantic model; **Composite** mixes modes; **Direct Lake** reads Fabric Delta tables directly.
- Publish to a **workspace**, schedule refresh (gateway for on-premises, incremental refresh for big facts), distribute through **apps**, and use subscriptions and alerts.
- **RLS** filters rows by role (static) or by the signed-in user (dynamic, `USERPRINCIPALNAME()`); it applies to Viewers and must be tested with *View as*.
- Tune performance with a star schema, lean columns, efficient DAX and **Performance Analyzer**.
- Fabric unifies the data stack; Copilot speeds up building; governance (certified models, pipelines, labels) keeps it trustworthy.

Quick quiz

6 questions · tap an answer to check it

1. Which mode stores a compressed copy of the data inside the Power BI model?

A Import

B DirectQuery

C Live connection

D Streaming

Answer: A. Import loads the data into the in-memory VertiPaq model.

2. When is an on-premises data gateway needed?

- A For any Excel file
- B To schedule refresh from data sources inside a company network**
- C To publish reports
- D To use slicers

Answer: B. Gateways let the cloud service reach on-premises sources.

3. A user with the Admin role in the workspace is also in an RLS role. What do they see?

- A Only the role's rows
- B All data, because RLS does not apply to Admin, Member and Contributor roles**
- C Nothing
- D Only their own data

Answer: B. RLS applies to Viewers; test with View as.

4. What does incremental refresh do?

- A Refreshes slicers only
- B Refreshes only new or changed partitions of a large table**
- C Deletes old data
- D Converts the model to DirectQuery

Answer: B. It speeds up refresh by loading only recent data.

5. Which tool shows time spent in DAX vs visual display?

A Model view

B Performance Analyzer

C Selection pane

D Bookmarks

Answer: B. Performance Analyzer breaks down each visual's load time.

6. Which function enables dynamic row-level security by user?

A TODAY ()

B SELECTEDVALUE ()

C USERPRINCIPALNAME ()

D RELATED ()

Answer: C. It returns the logged-in user's email for comparison with a security table.

EXERCISE Deploy and secure

(1) Create RLS roles for the three countries and test each with *View as*. (2) Create a dynamic RLS design using a `UserSecurity` table (write the table and the DAX filter). (3) List five changes you would make to a 2 GB model that refreshes in 45 minutes. (4) Write the checklist you would give a colleague before publishing a report to leadership.

Show solution (worked answer)

(1) Manage roles → filter `dim_territory[country]`; View as each role and confirm the totals equal the pandas numbers above. (2) Table: `user_email`, `territory_key`; filter on `dim_territory: [territory_key] IN CALCULATETABLE ( VALUES ( UserSecurity[territory_key] ), UserSecurity[user_email] = USERPRINCIPALNAME ( ) )`. (3) Remove unused columns; star schema and integer keys; incremental refresh on the fact; push transformations to the source (query folding); aggregate to the needed grain; disable auto date/time. (4) Numbers reconciled with source; filters and slicers tested; titles state insights; RLS tested; refresh scheduled and tested; owner and description added; performance checked; accessibility and mobile layout checked; app published with notes.

INTERVIEW CORNER Questions you may be asked

Q: Import vs DirectQuery? Import copies and compresses data into the model (fast, needs refresh, size limits); DirectQuery queries the source live (fresh, handles huge data, slower, DAX limitations).

Q: What is row-level security and how do you implement it? Restricting the rows a user can see: define roles with DAX filters on tables (static), or use `USERPRINCIPALNAME` and a security table (dynamic); assign users in the Service; test with *View as*.

Q: How do you improve a slow Power BI report? Measure with Performance Analyzer; fix the model (star schema, fewer columns), optimise DAX, simplify visuals, use aggregations/incremental refresh, and reduce data volume.

Q: What is a semantic model and why use live connections? The data model (tables, relationships, measures); multiple reports can connect live to one governed model so everyone uses the same definitions.