ADVANCED DIPLOMA IN DATA ANALYTICS

Comprehensive Study Book

2026 Edition

nasrin-a-

TABLE OF CONTENTS

UNIT I – Overview of Data Analytics

1.1 Introduction to Data Analytics

1.2 Need of Data Analytics

1.3 Business Analytics vs Data Analytics

1.4 Types / Categories / Models of Data Analytics

1.5 Data Scientist vs Data Engineer vs Data Analyst

1.6 Role of Data Analyst

1.7 Data Analytics in Practice

1.8 The Six-Step Data Analysis Process

1.9 Tools and Technologies in Data Analytics

1.10 Getting Started: Roadmap for the Aspiring Data Analyst

UNIT II – Excel for Data Analysis

2.1 Introduction to Microsoft Excel

2.2 The Excel Interface and Workbook Structure

2.3 Cell Referencing and Excel Tables

2.4 Essential Formulas and Functions

2.5 Conditional Aggregation – SUMIF, COUNTIF, AVERAGEIF

2.6 Lookup and Reference Functions

2.7 Text Functions for Data Cleaning

2.8 Date and Time Functions

2.9 Logical and Conditional Functions

2.10 Statistical Functions and the Data Analysis ToolPak

2.11 Data Sorting, Filtering and Advanced Filter

2.12 Data Validation

2.13 Conditional Formatting

2.14 PivotTables and PivotCharts

2.15 Data Visualisation with Charts and Sparklines

2.16 What-If Analysis

2.17 Power Query – Data Import and Transformation

2.18 Named Ranges and Formula Auditing

nasrin-a-mahin

2.19 Financial Functions

2.20 Excel Best Practices for Data Analysts

UNIT III – SQL for Data Analysis

3.1 Introduction to SQL

3.2 SQL Command Categories: DDL, DQL, DML, DCL, TCL

3.3 SQL Data Types

3.4 Core DML Operations: SELECT, INSERT, UPDATE, DELETE

3.5 Schema Changes: ALTER, DROP, TRUNCATE

3.6 SQL Clauses: WHERE, HAVING, ORDER BY, GROUP BY

3.7 SQL Operators: BETWEEN, IN, LIKE, IS NULL

3.8 SQL Aggregate Functions

3.9 SQL Joins

3.10 SQL Subqueries

3.11 CASE Statements

3.12 NULL Handling Functions: COALESCE, IFNULL, NULLIF

3.13 String Functions

3.14 Date and Time Functions

3.15 UNION, UNION ALL, INTERSECT, EXCEPT

3.16 Common Table Expressions (CTEs)

3.17 Window Functions

3.18 Views, Indexes and Query Performance

UNIT IV – Statistics for Data Analysis

4.1 Introduction to Statistics

4.2 Types of Data

4.3 Measures of Central Tendency

4.4 Measures of Dispersion

4.5 Measures of Shape: Skewness and Kurtosis

4.6 Measures of Relationship: Covariance and Correlation

4.7 Probability Theory Fundamentals

4.8 Probability Distributions

4.9 Statistical Inference and Sampling

4.10 Hypothesis Testing

4.11 Confidence Intervals

4.12 Common Statistical Tests

nasrin-a-mahin

UNIT V – Data Analysis with Python

5.1 Introduction to Python for Data Analysis

5.2 Python Basics – Data Types, Variables, Operators

5.3 NumPy – Arrays and Numerical Computing

5.4 NumPy – Array Operations and Indexing

5.5 NumPy – Mathematical and Statistical Functions

5.6 Pandas – Series and DataFrames

5.7 Pandas – Reading and Writing Data

5.8 Pandas – Data Selection and Indexing

5.9 Pandas – Data Cleaning and Handling Missing Values

5.10 Pandas – Data Transformation

5.11 Pandas – GroupBy and Aggregation

5.12 Pandas – Merge, Join, and Concatenate

5.13 Pandas – Pivot Tables and Reshaping

5.14 Data Visualization with Matplotlib

5.15 Advanced Visualization with Seaborn

5.16 Python Best Practices for Data Analysis

UNIT VI – Data Visualization

6.1 Introduction to Data Visualization

6.2 Principles of Effective Data Visualization

6.3 Basic Chart Types

6.4 Advanced Chart Types

6.5 Specialized and Domain-Specific Charts

6.6 Choosing the Right Chart Type

6.7 Color Theory in Data Visualization

6.8 Design Principles – Data-Ink Ratio and Chart Junk

6.9 Dashboard Design Best Practices

6.10 Interactive Visualizations

6.11 Data Storytelling

6.12 Common Visualization Mistakes to Avoid

nasrin-a-mahin

6.13 Visualization Tools and Software

6.14 Accessibility in Data Visualization

UNIT VII – R Programming for Data Analysis

7.1 Introduction to R

7.2 R Data Types and Structures

7.3 Control Structures and Functions

7.4 Data Manipulation with dplyr

7.5 Data Visualization with ggplot2

7.6 Statistical Analysis in R

7.7 R Markdown and Reporting

UNIT VIII – Power BI for Business Intelligence

8.1 Introduction to Power BI

8.2 Power BI Components

8.3 Connecting & Importing Data

8.4 Data Transformation with Power Query

8.5 Data Modelling & Relationships

8.6 DAX – Data Analysis Expressions

8.7 Building Reports & Dashboards

8.8 Power BI Service & Sharing

8.9 Best Practices

UNIT IX – Tableau for Data Visualization

9.1 Introduction to Tableau

9.2 Tableau Products

9.3 Connecting to Data Sources

9.4 Tableau Workspace & Interface

9.5 Building Basic Visualizations

9.6 Calculated Fields & Parameters

9.7 Dashboards & Stories

9.8 Advanced Features

9.9 Best Practices

UNIT X – Machine Learning for Data Analytics

10.1 Introduction to Machine Learning

10.2 Types of Machine Learning

10.3 The Machine Learning Workflow

nasrin-a-mahin

10.4 Data Preprocessing for ML

10.5 Supervised Learning – Regression

10.6 Supervised Learning – Classification

10.7 Unsupervised Learning – Clustering

10.8 Unsupervised Learning – Dimensionality Reduction

10.9 Model Evaluation Metrics

10.10 Train-Test Split and Cross-Validation

10.11 Overfitting, Underfitting, and the Bias-Variance Tradeoff

10.12 Regularization Techniques

10.13 Feature Engineering

10.14 Machine Learning with Python – Scikit-Learn

10.15 Real-World ML Applications

10.16 ML Best Practices and Common Pitfalls

nasrin-a-mahin

UNIT I: Overview of Data Analytics

1.1 Introduction to Data Analytics

We live in the age of data. Every click, swipe, transaction, sensor reading, and social media post generates data — and the volume is growing exponentially. Organisations that can harness this data effectively gain a significant competitive advantage: they understand their customers better, operate more efficiently, reduce risk, and innovate faster.

Data analytics is the discipline of transforming raw data into meaningful, actionable insights. It sits at the intersection of mathematics, statistics, computer science, and domain expertise, combining rigorous quantitative techniques with business understanding to drive smarter decisions.

> *Data Analytics is the systematic process of collecting, cleaning, transforming, analysing, and interpreting data in order to discover useful information, identify patterns, and support evidence-based decision-making.*

Modern data analytics goes far beyond simply producing reports. It encompasses a wide spectrum of activities — from exploratory data analysis and statistical modelling to machine learning, real-time dashboards, and predictive systems — all with the goal of converting raw data into value.

Why Data Analytics Matters

Consider a few real-world examples of data analytics in action:

- Netflix analyses viewing patterns of 260 million subscribers to recommend content and decide which original shows to commission — reducing churn and saving billions in marketing costs.
- Hospitals use predictive analytics on patient records to identify high-risk patients before they deteriorate, enabling earlier intervention and saving lives.
- Retailers like Amazon analyse purchasing behaviour, browsing history, and demographics to personalize product recommendations — a feature estimated to drive 35% of their revenue.
- Sports teams use performance analytics to make data-driven decisions on player selection, training programmes, and in-game tactics.
- Credit card companies process millions of transactions per second, using anomaly detection algorithms to flag fraudulent activity in real time.

In each case, data analytics is not just a technical exercise — it is a business capability that directly drives outcomes.

Data vs Information vs Insights vs Knowledge

It is important to distinguish between these related but distinct concepts:

Concept	Definition	Example
Data	Raw, unprocessed facts and figures with no context	"42,000"
Information	Data organized and presented with context	"42,000 units sold in Q3"
Insight	Meaningful conclusion drawn from analysis of information	"Sales rose 18% due to the summer campaign"
Knowledge	Accumulated insights used to make better future decisions	"Seasonal campaigns in Q3 consistently outperform Q4 discounts"
Wisdom	Applying knowledge to achieve long-term strategic objectives	"Invest in Q3 campaigns every year as the core growth strategy"

1.2 Need for Data Analytics

The need for data analytics has never been greater. Organisations across every sector are drowning in data but starving for insights. The following forces are driving the demand for skilled data analysts:

1. The Data Explosion

The amount of data generated globally is staggering. As of 2025, the world creates approximately 2.5 quintillion bytes of data every single day. This data comes from:

- Social media: over 500 million tweets, 4 billion Facebook interactions, and 100 billion WhatsApp messages per day
- IoT devices: smart sensors, connected machinery, and wearable devices continuously streaming data
- E-commerce: every product search, click, cart addition, and purchase creates a data trail
- Business transactions: ERP systems, CRMs, financial systems, and supply chain platforms
- Scientific research: genomics, climate science, physics experiments, and medical imaging

Without analytics, this data is worthless. With analytics, it becomes a strategic asset.

2. Competitive Pressure

Companies that use data analytics systematically outperform those that do not. McKinsey research shows data-driven organisations are 23 times more likely to acquire customers, 6 times more likely to retain them, and 19 times more likely to be profitable. This creates enormous pressure on businesses to build analytics capabilities or risk being left behind.

3. Key Business Benefits of Data Analytics

- **Improved Decision-Making:** Replace gut-feel with evidence. Data analytics provides objective, quantified support for decisions — reducing bias, uncertainty, and the cost of mistakes.
- **Better Customer Understanding:** Segment customers by behaviour, predict their needs, personalise interactions, and identify at-risk customers before they churn.
- **Operational Efficiency:** Identify bottlenecks, waste, and inefficiency in processes. Optimise resource allocation, scheduling, and supply chain operations.
- **Risk Management:** Detect fraud, assess credit risk, monitor compliance, and anticipate market disruptions using historical patterns and predictive models.
- **Effective Marketing:** Measure campaign ROI, identify the most profitable customer segments, and optimise marketing spend across channels.
- **Product Innovation:** Discover unmet customer needs, test product ideas with A/B experiments, and use usage data to continuously improve products.
- **Revenue Growth:** Identify cross-sell and upsell opportunities, optimise pricing dynamically, and forecast demand to prevent stockouts or overproduction.

4. The Data Analytics Lifecycle

Data analytics follows a structured lifecycle. Understanding this cycle helps analysts plan their work systematically:

1. Business Understanding: Define the problem, objectives, and success criteria clearly before touching any data.
2. Data Acquisition: Identify and collect relevant data from internal systems, external APIs, surveys, or publicly available sources.
3. Data Preparation: Clean, transform, and structure the data — removing errors, handling missing values, and engineering features.
4. Exploratory Data Analysis: Investigate the data through summary statistics, distributions, and visualisations to understand its characteristics and generate hypotheses.
5. Modelling and Analysis: Apply statistical techniques, machine learning models, or business logic to extract patterns and test hypotheses.
6. Visualisation and Communication: Present findings using charts, dashboards, and narratives tailored to the audience.
7. Deployment and Monitoring: Integrate insights or models into business processes and track their ongoing performance.

nasrin-a-mahin

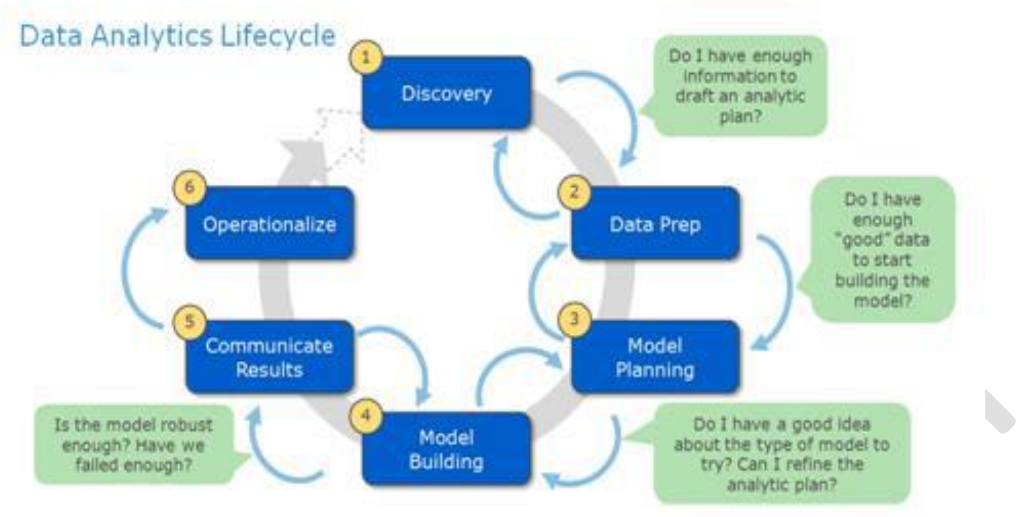


Fig1.1: Data analytics Lifecycle

1.3 Business Analytics vs Data Analytics

The terms "Business Analytics" and "Data Analytics" are often used interchangeably, but they have distinct emphases worth understanding.

Data Analytics

> *Data Analytics focuses on examining datasets to draw conclusions about the information they contain. It is primarily concerned with what the data says — finding patterns, testing hypotheses, and building models. The output is analytical findings: correlations, predictions, classifications, or anomalies.*

- Primarily technical in nature — requires statistical, programming, and data wrangling skills
- Works with structured, semi-structured, and unstructured data from any source
- Tools: Python, R, SQL, Excel, statistical software
- Output: Models, reports, dashboards, statistical findings
- Typical roles: Data Analyst, Data Scientist, Statistical Analyst

Business Analytics

> *Business Analytics applies data analysis specifically to business decisions. It is more applied and business-context-driven — asking "what should we do about this?" rather than simply "what does the data show?". It sits closer to management and strategy.*

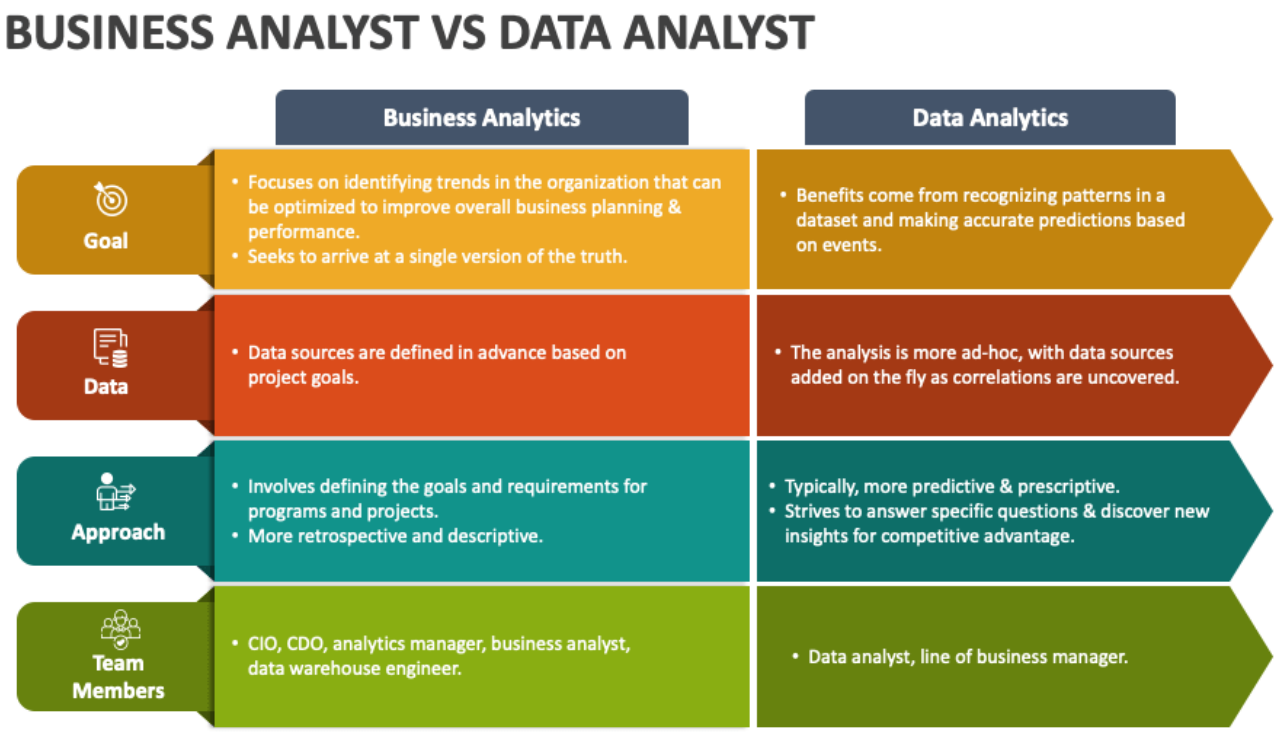
- Combines analytical skills with strong business domain knowledge

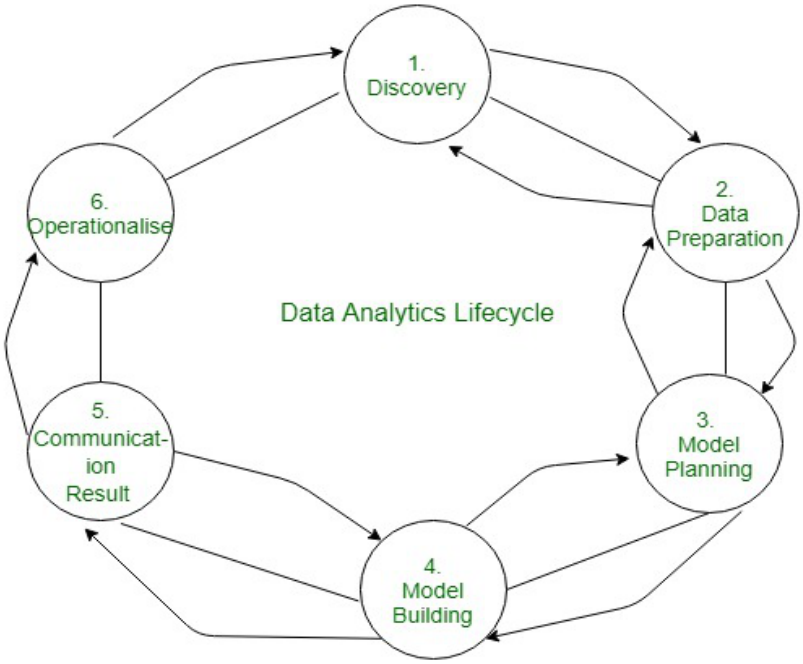
- Focused on business KPIs, metrics, and strategic objectives
- Tools: Power BI, Tableau, Excel, SQL, business intelligence platforms
- Output: Business recommendations, strategy documents, executive dashboards
- Typical roles: Business Analyst, BI Analyst, Analytics Manager

Side-by-Side Comparison

Dimension	Data Analytics	Business Analytics
Primary Focus	Data patterns and statistical findings	Business decisions and strategic insights
Orientation	Technical / quantitative	Applied / business-context-driven
Key Questions	"What does the data show?"	"What should we do about it?"
Core Skills	Statistics, programming, data engineering	Business acumen, communication, BI tools
Typical Output	Models, visualisations, statistical reports	Recommendations, dashboards, strategies
Tools Used	Python, R, SQL, Excel	Power BI, Tableau, Excel, SAP
Audience	Technical peers, data teams	Executives, business stakeholders

nasrin-a-m





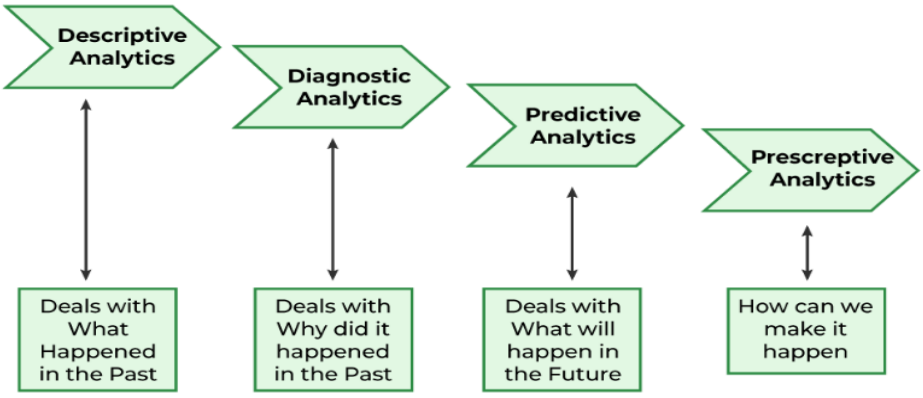
1.4 Types / Categories / Models of Data Analytics

Data analytics can be categorised into four types arranged along a spectrum of complexity and value.

Each type answers a progressively more powerful business question.

Type	Key Question	Complexity	Value
Descriptive	"What happened?"	Low	Moderate
Diagnostic	"Why did it happen?"	Medium	Good

Predictive	"What will happen?"	High	High
Prescriptive	"What should we do?"	Very High	Very High



1. Descriptive Analytics — What Happened?

Descriptive analytics is the foundation of all data analysis. It summarises historical data to provide a clear picture of what has occurred over a given time period. This is the most common form of analytics in business today.

Descriptive analytics does not explain causes or predict the future — it simply describes the past and present state of the business through aggregations, summaries, and visualisations.

- Monthly/quarterly/annual sales reports and revenue summaries
- Website traffic dashboards showing page views, bounce rates, and session duration
- Customer demographics reports — age distribution, geographic spread, purchase frequency
- Operational KPI dashboards — production volumes, quality defect rates, employee headcount
- Financial statements — profit and loss, balance sheet, cash flow analysis

Key techniques: summary statistics (mean, median, mode, standard deviation), frequency distributions, histograms, bar charts, pivot tables, and aggregation queries in SQL.

2. Diagnostic Analytics — Why Did It Happen?

Diagnostic analytics goes a step deeper, investigating the causes behind outcomes identified by descriptive analytics. It answers the "why" questions that naturally follow descriptive findings.

For example, if descriptive analytics reveals that sales fell 15% in Q3, diagnostic analytics investigates which products, regions, customer segments, or time periods drove the decline — and what factors correlate with the drop.

- **Drill-down analysis:** Decomposing a top-line metric into its constituent parts to isolate the source of a change

- **Root cause analysis:** Systematically identifying the underlying cause of a problem rather than treating symptoms
- **Correlation analysis:** Identifying statistical relationships between variables
- **Data discovery:** Exploring datasets to uncover unexpected patterns or anomalies
- **Data mining:** Applying algorithms to large datasets to find hidden relationships

3. Predictive Analytics — What Will Happen?

Predictive analytics uses historical data, statistical models, and machine learning algorithms to forecast future outcomes. It moves from understanding the past to anticipating the future — enabling proactive rather than reactive decision-making.

Predictive analytics does not provide certainty about future events — it produces probability-weighted forecasts. The more high-quality data available, the more accurate the predictions tend to be.

- Customer churn prediction: Which customers are likely to cancel their subscription in the next 90 days?
- Demand forecasting: How much inventory will we need next month by product and region?
- Credit risk scoring: What is the probability this loan applicant will default?
- Predictive maintenance: When is this machine component likely to fail, and when should we schedule maintenance?
- Sales forecasting: What revenue can we expect next quarter based on current pipeline and historical patterns?
- Medical diagnosis support: What is the likelihood a patient has a particular condition based on symptoms and test results?

Key techniques: Linear and logistic regression, decision trees, random forests, gradient boosting, neural networks, time series forecasting (ARIMA, Prophet), and survival analysis.

Key tools: Python (scikit-learn, statsmodels, Prophet), R (caret, forecast), Azure ML, Amazon SageMaker.

4. Prescriptive Analytics — What Should We Do?

Prescriptive analytics is the most advanced form. It not only predicts what will happen but recommends specific actions to take in order to achieve the best possible outcome. It combines predictive models with optimisation algorithms, business rules, and simulation to prescribe decisions.

> *Prescriptive analytics effectively closes the decision loop — it takes the "what will happen" from predictive analytics and converts it into "here is what you should do, and here is why, and here is what each option will cost/yield."*

- Dynamic pricing: Adjust product prices in real time based on demand, competition, inventory, and customer segment

nasim-a-mahin

- Supply chain optimisation: Determine the optimal sourcing, routing, and inventory strategy to minimise cost while meeting service levels
- Treatment optimisation in healthcare: Recommend the best treatment plan for a patient given their medical history, genetics, and current condition
- Portfolio optimisation in finance: Determine the optimal asset allocation to maximise risk-adjusted returns
- Marketing budget allocation: Prescribe how to distribute marketing spend across channels to maximise revenue

Key techniques: Linear programming, integer programming, simulation (Monte Carlo), reinforcement learning, decision analysis, and multi-objective optimisation.

1.5 Data Scientist vs Data Engineer vs Data Analyst

These three roles are often confused because they all work with data. However, they have distinct focuses, skill sets, and outputs. Understanding the differences helps organisations build effective data teams and helps individuals plan their careers.

Data Analyst

> *The Data Analyst bridges the gap between raw data and business decisions. Their primary job is to understand what the data says and communicate it clearly to stakeholders.*

- **Core Focus:** Querying, cleaning, analysing, and visualising data to answer specific business questions
- **Typical Tasks:** Writing SQL queries, building Excel/Power BI/Tableau dashboards, performing exploratory analysis, creating reports
- **Technical Skills:** SQL (essential), Excel, Python or R (intermediate), data visualisation tools, basic statistics
- **Soft Skills:** Communication, business acumen, storytelling with data, attention to detail
- **Education:** Bachelor's degree in Mathematics, Statistics, Business, Economics, or related field
- **Salary Range (India):** ₹4–12 LPA entry to mid-level; ₹12–25 LPA senior level
- **Typical Output:** Dashboards, reports, data summaries, business recommendations

Data Engineer

> *The Data Engineer builds and maintains the data infrastructure that makes analysis possible. They are the architects and plumbers of the data world — ensuring that clean, reliable data flows from source systems to where analysts and scientists can use it.*

- **Core Focus:** Building, scaling, and maintaining data pipelines, warehouses, and infrastructure
- **Typical Tasks:** ETL pipeline development, data warehouse design, database administration, streaming data processing
- **Technical Skills:** Python, Scala, Java, SQL, Spark, Kafka, Airflow, cloud platforms (AWS/Azure/GCP), database management
- **Soft Skills:** Systems thinking, problem-solving, attention to performance and scalability

- **Education:** Bachelor's or Master's degree in Computer Science, Software Engineering, or related field
- **Salary Range (India):** ₹6–15 LPA mid-level; ₹15–35 LPA senior level
- **Typical Output:** Data pipelines, data warehouses, data lakes, real-time streaming systems

Data Scientist

> *The Data Scientist applies advanced statistical modelling and machine learning to solve complex, open-ended problems. They often work on the cutting edge — developing novel models, running experiments, and creating the predictive and prescriptive systems that give organisations a competitive advantage.*

- **Core Focus:** Building predictive models, applying machine learning, conducting experiments, and generating novel insights
- **Typical Tasks:** Feature engineering, model training and evaluation, A/B testing, NLP, computer vision, deployment of ML models
- **Technical Skills:** Python/R (advanced), machine learning (scikit-learn, TensorFlow, PyTorch), SQL, statistics, cloud ML platforms
- **Soft Skills:** Scientific curiosity, hypothesis-driven thinking, communication of complex findings
- **Education:** Master's or PhD in Mathematics, Statistics, Computer Science, or quantitative discipline is common
- **Salary Range (India):** ₹8–20 LPA mid-level; ₹20–50 LPA senior/lead level
- **Typical Output:** ML models, research findings, experimental results, AI-powered products

Skills Comparison Table

Skill Area	Data Analyst	Data Engineer	Data Scientist
SQL	Expert	Expert	Proficient
Python / R	Intermediate	Advanced	Expert
Statistics	Intermediate	Basic	Expert
Machine Learning	Basic	Basic	Expert
Data Pipelines/ETL	Basic	Expert	Intermediate
Data Visualisation	Expert	Basic	Intermediate
Cloud Platforms	Basic	Expert	Intermediate
Business Acumen	Expert	Intermediate	Intermediate
Communication	Expert	Intermediate	Intermediate

1.6 Role of a Data Analyst

A Data Analyst is responsible for collecting, processing, and performing statistical analysis on data to help organisations make better decisions. They serve as a critical link between raw data and business strategy, translating numbers into stories that non-technical stakeholders can act on.

Core Responsibilities

1. Data Collection and Acquisition

- Identify relevant data sources — internal databases, third-party APIs, publicly available datasets, surveys, and web scraping
- Write SQL queries to extract specific subsets of data from relational databases
- Import data from diverse formats: CSV, Excel, JSON, XML, databases, cloud storage
- Assess data quality and completeness at the point of collection
- Document data sources, structures, and any known quality issues

2. Data Cleaning and Preparation

- Handle missing values: impute using mean/median/mode, forward-fill, or flag and remove depending on context
- Remove or resolve duplicate records
- Standardise formats: dates, currencies, units of measurement, categorical labels
- Detect and investigate outliers — distinguish data errors from genuine extreme values
- Transform and reshape data: pivot, melt, merge, aggregate as required
- Validate cleaned data against business rules and expected ranges

3. Exploratory Data Analysis (EDA)

- Compute descriptive statistics: mean, median, mode, standard deviation, skewness, kurtosis
- Examine distributions using histograms, box plots, and density plots
- Identify correlations between variables using scatter plots and correlation matrices
- Detect patterns, seasonality, and trends in time-series data
- Formulate hypotheses based on observed patterns for further testing

4. Statistical Analysis and Modelling

- Apply hypothesis tests (t-tests, chi-squared, ANOVA) to determine statistical significance
- Build regression models to quantify relationships between variables
- Perform segmentation and clustering to group similar customers or behaviours
- Apply time-series forecasting for demand planning or trend projection
- Use A/B testing frameworks to evaluate the impact of changes and experiments

5. Data Visualisation and Reporting

- Choose the right chart type for each insight: bar for comparisons, line for trends, scatter for correlations, pie for proportions
- Build interactive dashboards in Power BI, Tableau, or Excel that stakeholders can explore independently
- Design reports that lead the reader logically from problem to finding to recommendation
- Tailor the level of technical detail to the audience — executive summaries vs analytical deep-dives
- Refresh dashboards automatically when underlying data is updated

nasrin-a-mahin

6. Communication and Storytelling

- Present findings in plain language — avoid jargon when communicating with non-technical stakeholders
- Use the "So What?" test for every finding: always connect the data to a business implication
- Structure presentations with a clear narrative: context → finding → implication → recommendation
- Prepare written reports with executive summaries and appendices for technical detail
- Handle questions and challenges from stakeholders with data-backed responses

7. Collaboration and Ethics

- Work closely with Data Engineers to understand data infrastructure and request new data pipelines
- Collaborate with Data Scientists to support advanced modelling projects with clean, well-understood datasets
- Partner with business stakeholders to define analytical questions and validate that findings make business sense
- Uphold data ethics: handle sensitive and personal data with care, comply with GDPR and other privacy regulations
- Maintain data documentation so that analysis is reproducible and auditable

Essential Skills for a Data Analyst

Skill Category	Specific Skills
Technical — Core	SQL (joins, aggregations, window functions), Excel (formulas, PivotTables, Power Query)
Technical — Programming	Python (pandas, numpy, matplotlib, seaborn) or R (dplyr, ggplot2)
Technical — Visualisation	Power BI, Tableau, or Google Data Studio; chart design principles
Statistical	Descriptive stats, probability, hypothesis testing, regression, correlation
Analytical	Problem decomposition, pattern recognition, logical reasoning, critical thinking
Business	Understanding of KPIs, P&L, business processes, and industry context
Communication	Data storytelling, report writing, presentation, stakeholder management
Soft Skills	Attention to detail, intellectual curiosity, adaptability, teamwork

1.7 Data Analytics in Practice

Data analytics is not confined to a single industry — it has become a universal discipline that creates value wherever data exists. Understanding the breadth of applications provides context for why the skills covered in this diploma are so sought after.

Industry Applications

Healthcare

- Predictive readmission modelling: identify patients at high risk of returning to hospital within 30 days
- Clinical trial analysis: determine drug efficacy and safety from patient outcome data
- Fraud detection in insurance claims: flag suspicious billing patterns for investigation
- Resource optimisation: predict patient admissions to optimise staffing and bed allocation
- Personalised medicine: tailor treatment plans based on patient genomics and historical outcomes

Finance and Banking

- Credit scoring: assess the probability of loan default using applicant financial history
- Algorithmic trading: use statistical models and ML to execute trades at optimal times
- Fraud detection: flag unusual transaction patterns in real time across millions of daily transactions
- Customer lifetime value modelling: identify the most profitable customer segments
- Risk management: stress-test portfolios against economic scenarios

Retail and E-commerce

- Product recommendation engines: suggest relevant items based on browsing and purchase history
- Dynamic pricing: adjust prices in real time based on demand, competition, and inventory levels
- Customer churn prediction: identify customers likely to stop purchasing and trigger retention campaigns
- Inventory optimisation: forecast demand by product and location to prevent stockouts and overstock
- Market basket analysis: discover which products are frequently purchased together to inform store layout and promotions

Manufacturing and Supply Chain

- Predictive maintenance: use sensor data to predict equipment failures before they cause downtime
- Quality control: use computer vision and statistical process control to detect defects on production lines
- Supply chain optimisation: model the optimal sourcing, routing, and inventory strategy across the supply network
- Demand forecasting: predict production requirements weeks or months in advance

Marketing and Advertising

- Customer segmentation: group customers by behaviour, demographics, and value for targeted campaigns
- Marketing mix modelling: determine the contribution of each marketing channel to sales
- A/B testing: rigorously measure the impact of changes to websites, emails, or advertisements
- Social media analytics: track sentiment, reach, and engagement across platforms
- Attribution modelling: credit the right touchpoints in a customer's journey to conversion

Sports Analytics

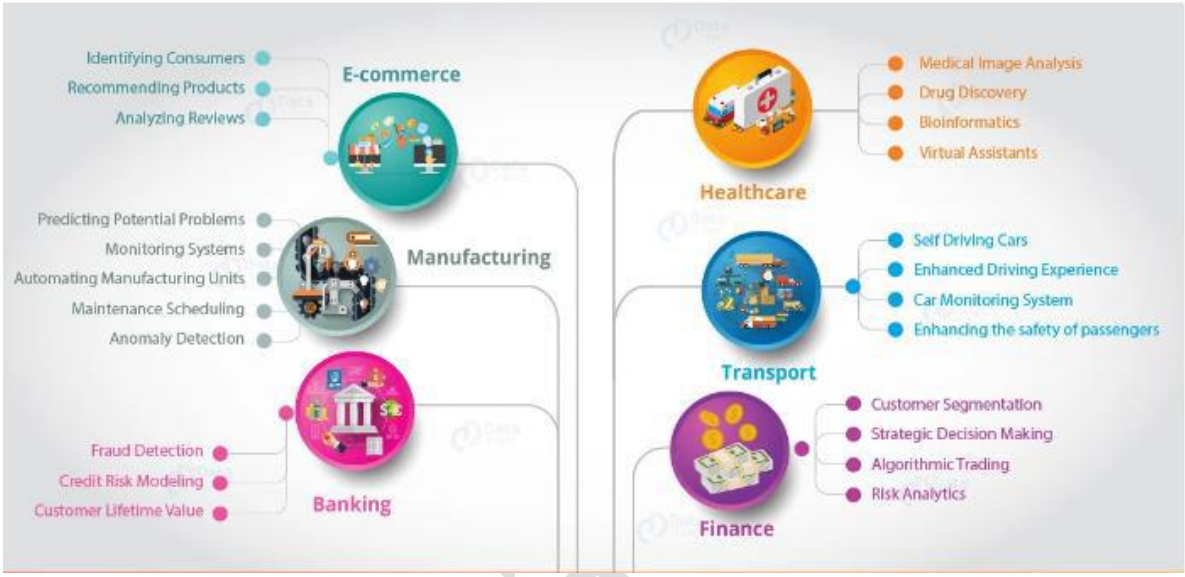
- Player performance analysis: quantify individual contributions beyond traditional statistics
- Injury risk modelling: track physiological indicators to identify players at elevated injury risk
- Opposition analysis: analyse competitors' tactical patterns to inform game plans

nasrin-a-mahin

- Fan engagement: personalise communications and offers to different fan segments

Government and Public Sector

- Crime prediction and prevention: identify high-risk times and locations for proactive policing
- Disease surveillance: detect outbreaks early using hospital admission data, social media, and search trends
- Traffic and urban planning: optimise signal timing, public transport routes, and city infrastructure
- Tax compliance: identify patterns of tax evasion for targeted audit selection



1.8 The Six-Step Data Analysis Process

Every data analysis project, regardless of industry or complexity, follows a broadly similar process. Understanding this framework ensures that analysts approach problems systematically, produce reliable results, and communicate findings effectively.

Step 1: Define the Problem

The most common mistake in data analysis is jumping straight to data collection without clearly defining the problem. A vague question produces vague answers. Invest time up front to articulate exactly what you are trying to understand or achieve.

- Identify the core business question or problem — be specific, not general
- Agree on the decision that will be made based on the analysis output
- Define the success criteria: what would a good answer look like?
- Understand the constraints: time, data availability, computational resources
- Document assumptions and scope boundaries clearly

Step 2: Collect the Data

Once the problem is defined, identify and gather the data needed to address it. Data can come from internal systems (databases, ERP, CRM), external sources (government data, market research, third-party APIs), or be generated through surveys and experiments.

- Ensure the data is relevant to the specific question being asked
- Gather from multiple sources to enrich the analysis where possible
- Document the origin, format, update frequency, and structure of each dataset
- Check for data access, licensing, and privacy considerations
- Collect a sample first to assess quality before full extraction

Step 3: Clean and Prepare the Data

Raw data is almost never ready for analysis. Data cleaning is the process of detecting and correcting errors, inconsistencies, and gaps in the data. Analysts typically spend 60–80% of their time on this step — reflecting its importance.

- Handle missing values: impute, flag, or remove depending on the extent and reason
- Remove duplicate records and resolve conflicting entries
- Standardise formats: dates, currencies, units, and categorical labels
- Detect and investigate outliers — distinguish errors from genuine extreme values
- Transform variables as needed: log transforms, binning, encoding categorical variables
- Validate the cleaned data against business rules and expected value ranges

Step 4: Analyse the Data

With clean data in hand, the analysis phase applies statistical and analytical techniques to uncover patterns, test hypotheses, and answer the question defined in Step 1. The techniques used depend on the type of analytics being performed.

- Compute descriptive statistics to understand distributions and central tendency
- Identify correlations, trends, seasonality, and anomalies
- Apply statistical tests to determine significance of findings
- Build models (regression, clustering, classification) where needed
- Validate findings against domain knowledge and business context
- Document every analytical decision and assumption for reproducibility

Step 5: Visualise and Communicate the Results

Findings that cannot be communicated effectively have no business value. Visualisation translates complex analytical outputs into intuitive, accessible formats that enable non-technical stakeholders to understand and act on the insights.

nasrin-a-mahin

- Choose chart types that best represent the data: bar for comparisons, line for trends, scatter for correlations
- Keep visualisations clean — remove chart junk, use consistent colours, and label clearly
- Structure the narrative: start with the key finding, then provide supporting evidence
- Build dashboards for ongoing monitoring; write reports for one-time deep dives
- Tailor complexity and terminology to the audience

Step 6: Interpret Results and Support Decisions

The final step converts analytical findings into business decisions and actions. This requires combining data insights with domain knowledge, business context, and stakeholder input to arrive at practical recommendations.

- Frame every finding in terms of its business implication — answer "so what?"
- Provide clear, specific recommendations — not just observations
- Quantify uncertainty and confidence where possible — avoid overconfidence in predictions
- Anticipate objections and alternative interpretations — address them proactively
- Establish a monitoring plan to track whether the recommended action achieved the expected outcome
- Feed lessons learned back into the process for continuous improvement

1.9 Tools and Technologies in Data Analytics

A modern data analyst is expected to be comfortable with a range of tools. The specific tools vary by organisation and role, but the following represent the core technology stack covered in this diploma and widely used in industry.

Category	Tools	Primary Use in Analytics
Spreadsheets	Microsoft Excel, Google Sheets	EDA, financial modelling, quick analysis, reporting
Query Languages	SQL (MySQL, PostgreSQL, SQL Server)	Data extraction, transformation, and aggregation from databases
Statistical Programming	Python, R	Statistical analysis, data manipulation, ML, automation
Business Intelligence	Power BI, Tableau, Looker	Interactive dashboards, self-service reporting, data storytelling
Data Engineering	Apache Spark, Airflow, dbt, Kafka	Building and orchestrating data pipelines and ETL processes
Cloud Platforms	AWS, Azure, Google Cloud	Scalable data storage, compute, and managed analytics services
Machine Learning	scikit-learn, TensorFlow, PyTorch	Predictive modelling, classification, NLP, computer vision
Version Control	Git, GitHub, GitLab	Tracking code changes, collaboration, and reproducibility
Databases	MySQL, PostgreSQL, MongoDB, Snowflake	Storing, querying, and managing structured and unstructured data

1.10 Getting Started: Roadmap for the Aspiring Data Analyst

The following roadmap provides a structured path for building the skills covered across this diploma programme. It reflects the logical order in which tools and concepts build on each other.

Step 1 — Build Statistical Foundations

Understand the mathematical and statistical concepts that underpin all data analysis: descriptive statistics, probability, distributions, hypothesis testing, and regression. These concepts appear throughout every other topic in this diploma.

Step 2 — Master Excel for Data Analysis

Excel is the universal tool of data analysis. Master formulas, PivotTables, Power Query, and charting. Excel skills are expected in virtually every data analyst role and serve as a practical foundation for more advanced tools.

Step 3 — Learn SQL for Data Extraction

SQL is the language of databases and the most universally required technical skill for data analysts. Learn how to write SELECT queries, filter and aggregate data, join multiple tables, and use subqueries and window functions.

Step 4 — Learn Python or R for Data Analysis

Python (with pandas, numpy, matplotlib, and seaborn) or R (with dplyr, ggplot2, and tidyr) enable automation, statistical analysis, and data manipulation at scale — going beyond what Excel and SQL alone can achieve.

Step 5 — Learn a BI / Visualisation Tool

Power BI and Tableau are the leading platforms for building interactive dashboards and reports. Master at least one to create the professional-quality visualisations expected by business stakeholders.

Step 6 — Work on Real Projects

Theoretical knowledge must be reinforced through practice. Use platforms like Kaggle, the UCI Machine Learning Repository, or real datasets from your work or local organisations to build end-to-end projects for your portfolio.

nasrin-a-mahin

Step 7 — Build a Portfolio and Get Certified

Document your projects on GitHub or a personal website. Pursue relevant certifications (Google Data Analytics, Microsoft Data Analyst, Tableau Desktop Specialist) to validate your skills to employers.

Where to Practice — Free Resources

- Kaggle (kaggle.com): Datasets, competitions, and free courses in Python, SQL, and ML
- SQL Bolt (sqlbolt.com): Interactive SQL lessons from basics to advanced
- Mode SQL Tutorial (mode.com/sql-tutorial): Real-world SQL practice with actual datasets
- Google Data Analytics Certificate (Coursera): Comprehensive beginner-level programme
- DataCamp (datacamp.com): Interactive Python, R, SQL, and Power BI courses
- Towards Data Science (towardsdatascience.com): Articles and tutorials from practitioners
- YouTube — Alex the Analyst, Ken Jee, Luke Barousse: Free data analyst career content

> *The best time to start is now. Pick one tool, one dataset, and one question — and work through the six-step analysis process end to end. That first complete project will teach you more than any number of tutorials watched passively.*

nasrin-a-nishin

nasrin-a-mahin

UNIT II: Excel for Data Analysis

2.1 Introduction to Microsoft Excel

Microsoft Excel is the most widely used data analysis tool in the world. Part of the Microsoft 365 suite, it provides a grid-based environment where data can be stored, manipulated, analysed, and visualised — all without writing a single line of code.

For a data analyst, Excel is the Swiss Army knife of the profession. It is the tool most employers expect mastery of at any level, and it remains indispensable for quick exploratory analysis, financial modelling, reporting, and dashboard creation.

> *Excel can handle up to 1,048,576 rows and 16,384 columns per worksheet — sufficient for the vast majority of business datasets. For larger datasets, Power Query, SQL, or Python are used to pre-aggregate before loading into Excel.*

Why Excel is Essential for Data Analysts

- No programming required — immediately productive for business users
- Hundreds of built-in functions covering maths, statistics, finance, text, and logic
- PivotTables for interactive summarisation without formulas
- Power Query for automated, repeatable data cleaning and transformation
- Charts and dashboards for professional data visualisation
- Widely understood by business stakeholders — easy to share findings
- Integrates seamlessly with SQL databases, Power BI, and Python

Excel Versions and Key Features by Version

Version	Year	Key Features Added
Excel 2016	2016	Power Query built-in, new chart types (Waterfall, Treemap, Sunburst)
Excel 2019	2019	IFS, SWITCH, TEXTJOIN, CONCAT, MAXIFS, MINIFS functions
Excel 365	2021+	XLOOKUP, XMATCH, dynamic arrays, LET, LAMBDA, FILTER, SORT, UNIQUE
Excel 2021	2021	Dynamic array functions, XLOOKUP, LET function

2.2 The Excel Interface and Workbook Structure

Understanding the Excel interface is the first step to working efficiently. Every element of the interface serves a specific purpose in the data analysis workflow.

Key Interface Elements

Element	Location	Purpose
Quick Access Toolbar	Top-left	Save, Undo, Redo and customisable shortcuts
Ribbon	Top	All commands organised into tabs: Home, Insert, Formulas, Data, Review, View
Name Box	Left of formula bar	Displays active cell address; type a name or address to navigate
Formula Bar	Below ribbon	Displays and edits the content or formula of the active cell
Column Headers	A–XFD across top	16,384 columns; click to select entire column
Row Numbers	1–1,048,576 down left	Over 1 million rows; click to select entire row
Worksheet Tabs	Bottom	Multiple sheets per workbook; right-click to rename, colour, move
Status Bar	Bottom	Shows Sum, Average, Count, Min, Max of selected range instantly

Navigating Efficiently — Essential Keyboard Shortcuts

Shortcut	Action
Ctrl + End	Jump to last used cell — find the extent of your data
Ctrl + Home	Jump to cell A1
Ctrl + Arrow Key	Jump to edge of data region in that direction
Ctrl + Shift + End	Select from current cell to last used cell
Ctrl + T	Convert selected range to a structured Table
Ctrl + Shift + L	Toggle AutoFilter dropdowns on/off
Alt + =	AutoSum the selected range
F4	Toggle reference type: A1 → $A$1 → A$1 → $A1 → A1
Ctrl + `	Toggle between showing values and formula text
Ctrl + 1	Open Format Cells dialog
Ctrl + D	Fill Down — copy cell above into selected cells
Ctrl + ;	Insert today's date
F2	Edit the active cell (enter Edit mode)

2.3 Cell Referencing and Excel Tables

Types of Cell References

Cell references determine how a formula behaves when copied to other cells. Mastering references is fundamental to writing correct, efficient formulas.

Reference Type	Syntax	Behaviour When Copied	Example Use
Relative	A1	Both row and column adjust	Simple arithmetic between adjacent cells

Absolute	$A$1	Neither row nor column adjusts	Fixed tax rate or conversion factor
Mixed — Lock Column	$A1	Column fixed, row adjusts	Multiply each row by a fixed column
Mixed — Lock Row	A$1	Row fixed, column adjusts	Headers in a lookup table
Cross-Sheet	Sheet2!A1	References another worksheet	Pull data from a summary sheet
3D Reference	Sheet1:Sheet3!A1	Same cell across multiple sheets	Sum same cell across 12 monthly sheets

> *Press F4 while editing a cell reference to cycle through all four reference types: A1 → $A$1 → A$1 → $A1 → A1. This is one of the most time-saving shortcuts in Excel.*

Excel Tables — Structured References

Converting a data range into an Excel Table (Ctrl+T) is best practice for all data analysis work. Tables provide automatic structure that makes formulas, PivotTables, and Power Query all work better.

- Tables auto-expand: adding a row below or column to the right automatically extends the table, including all formulas
- Structured references: =Table1[Sales] is self-documenting and always refers to the right column even if columns are rearranged
- Total Row: toggle on via Table Design tab to show configurable aggregations (sum, average, count, etc.) for each column
- AutoFilter built in: sort and filter controls appear on every column header automatically
- Named tables: give tables meaningful names (Table Design → Table Name) for use in formulas and Power Query
- Works with PivotTables: PivotTables based on a Table automatically update when data is added

2.4 Essential Formulas and Functions

Excel functions are predefined formulas that perform specific calculations. They follow the syntax: =FUNCTION_NAME(argument1, argument2, ...). Arguments can be values, cell references, ranges, or other functions (nested).

SUM, AVERAGE, COUNT Family

Function	Syntax	Description	Example
SUM	=SUM(range)	Adds all numbers in a range	=SUM(B2:B100)
AVERAGE	=AVERAGE(range)	Arithmetic mean of a range	=AVERAGE(B2:B100)
COUNT	=COUNT(range)	Count of cells containing numbers	=COUNT(B2:B100)
COUNTA	=COUNTA(range)	Count of non-empty cells (any type)	=COUNTA(B2:B100)
COUNTBLANK	=COUNTBLANK(range)	Count of empty cells	=COUNTBLANK(B2:B100)

MIN	=MIN(range)	Smallest value in range	=MIN(B2:B100)
MAX	=MAX(range)	Largest value in range	=MAX(B2:B100)
MEDIAN	=MEDIAN(range)	Middle value of sorted range	=MEDIAN(B2:B100)
MODE.SNGL	=MODE.SNGL(range)	Most frequently occurring value	=MODE.SNGL(B2:B100)
LARGE	=LARGE(range,k)	k-th largest value	=LARGE(B2:B100,3)
SMALL	=SMALL(range,k)	k-th smallest value	=SMALL(B2:B100,2)

Arithmetic Operators

Operator	Meaning	Example	Result
+	Addition	=A1+B1	Sum of A1 and B1
-	Subtraction	=A1-B1	Difference
*	Multiplication	=A1*B1	Product
/	Division	=A1/B1	Quotient (use IFERROR to handle divide-by-zero)
^	Exponentiation	=A1^2	A1 squared
&	Text concatenation	=A1&" "&B1	Joins text from A1 and B1 with a space
%	Percentage	=A1*15%	15% of A1

2.5 Conditional Aggregation – SUMIF, COUNTIF, AVERAGEIF

Conditional aggregation functions calculate totals, counts, or averages only for rows that meet specified criteria. These are among the most used functions in business data analysis.

Single-Condition Functions

Function	Syntax	Description
SUMIF	=SUMIF(range, criteria, sum_range)	Sum cells in sum_range where range meets criteria
COUNTIF	=COUNTIF(range, criteria)	Count cells in range that meet criteria
AVERAGEIF	=AVERAGEIF(range, criteria, avg_range)	Average cells in avg_range where range meets criteria

Examples:

```
=SUMIF(C2:C100, "North", D2:D100)       Sum sales where Region = "North"
=COUNTIF(B2:B100, ">1000")              Count rows where value exceeds 1000
=AVERAGEIF(C2:C100, "East", D2:D100)    Average sales for East region only
```

Multi-Condition Functions — SUMIFS, COUNTIFS, AVERAGEIFS

The "S" versions accept multiple criteria pairs — each criterion is a range followed by its condition:

```
=SUMIFS(sum_range, criteria_range1, criteria1, criteria_range2, criteria2, ...)
```

Examples:

```
=SUMIFS(D2:D100, C2:C100, "North", E2:E100, 2024)   Sales: North AND Year=2024
=COUNTIFS(B2:B100, ">=1000", C2:C100, "Active")     Count: value>=1000 AND Status=Active
=AVERAGEIFS(D2:D100, C2:C100, "East", F2:F100, "Q1") Avg sales: East AND Q1
```

> *Criteria can use wildcards: * (any characters), ? (single character). Examples: "North*" matches any text starting with North. "???" matches exactly 3 characters.*

2.6 Lookup and Reference Functions

Lookup functions retrieve data from one part of a dataset based on a matching value in another — essential for combining information from multiple tables.

VLOOKUP

VLOOKUP (Vertical Lookup) searches the leftmost column of a table for a value and returns a value from a specified column in the same row.

```
=VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])
```

Argument	Description	Example
lookup_value	Value to search for (employee ID, product code, etc.)	1023
table_array	Range containing the lookup table — always lock with $	$A$2:$D$100
col_index_num	Column number to return (1 = leftmost column of table_array)	3
range_lookup	FALSE = exact match (always use this for data lookups)	FALSE

```
=VLOOKUP(1023, $A$2:$D$100, 3, FALSE)   Returns column 3 value for ID 1023
```

Key limitation: VLOOKUP can only look to the right, and the lookup column must be the leftmost column of table_array.

HLOOKUP

HLOOKUP (Horizontal Lookup) works like VLOOKUP but searches across rows instead of columns — useful for transposed tables where categories are in rows.

```
=HLOOKUP(lookup_value, table_array, row_index_num, [range_lookup])
```

INDEX and MATCH — The Recommended Alternative

INDEX-MATCH is more powerful and flexible than VLOOKUP. It can look left, works when the lookup column is not leftmost, is not broken by inserting columns, and is generally faster on large datasets.

```
=INDEX(return_range, MATCH(lookup_value, lookup_range, 0))
```

How it works:

- MATCH finds the row number of the lookup_value within lookup_range (0 = exact match)
- INDEX returns the value at that row number within return_range

```
=INDEX($D$2:$D$100, MATCH(1023, $A$2:$A$100, 0))
```

This returns the value in column D where column A contains 1023 — and works regardless of column order.

Two-way lookup (row AND column):

```
=INDEX(B2:E10, MATCH("Product A", A2:A10, 0), MATCH("Mar", B1:E1, 0))
```

XLOOKUP — The Modern Standard (Excel 365 / 2021)

XLOOKUP replaces VLOOKUP, HLOOKUP, and INDEX-MATCH with a cleaner, more powerful single function.

```
=XLOOKUP(lookup_value, lookup_array, return_array, [if_not_found], [match_mode], [search_mode])
```

Argument	Description	Example Value
lookup_value	Value to search for	1023
lookup_array	Range to search in	A2:A100
return_array	Range to return from (can be multiple columns)	D2:D100
if_not_found	Return this if no match found	"Not Found"
match_mode	0=exact, -1=exact/smaller, 1=exact/larger, 2=wildcard	0
search_mode	1=first to last, -1=last to first (find most recent)	1

```
=XLOOKUP(1023, A2:A100, D2:D100, "Not Found")    Basic exact match

=XLOOKUP(1023, A2:A100, C2:F100, "Not Found")    Returns 4 columns at once
```

Other Reference Functions

Function	Syntax	Use Case
OFFSET	=OFFSET(ref, rows, cols, [h], [w])	Dynamic ranges that shift based on a formula
INDIRECT	=INDIRECT(ref_text)	Convert text string to a cell reference — =INDIRECT("A"&ROW())

CHOOSE	=CHOOSE(index, val1, val2,...)	Return a value from a list by position number
MATCH	=MATCH(val, range, 0)	Return the position of a value within a range
INDEX	=INDEX(range, row, [col])	Return value at a specific row/column position
ROW/COLUMN	=ROW() / =COLUMN()	Return the row or column number of a cell

2.7 Text Functions for Data Cleaning

Real-world data is messy. Text imported from external systems often contains extra spaces, inconsistent casing, mixed formats, and embedded numbers stored as text. These functions are the tools for cleaning it.

Case and Whitespace Functions

Function	Syntax	Example	Result
UPPER	=UPPER(text)	=UPPER("hello world")	HELLO WORLD
LOWER	=LOWER(text)	=LOWER("HELLO")	hello
PROPER	=PROPER(text)	=PROPER("john smith")	John Smith
TRIM	=TRIM(text)	=TRIM(" hello ")	hello (spaces removed)
CLEAN	=CLEAN(text)	=CLEAN(A1)	Removes non-printable characters

Extracting Text

Function	Syntax	Example	Result
LEFT	=LEFT(text, n)	=LEFT("AB1234", 2)	AB
RIGHT	=RIGHT(text, n)	=RIGHT("AB1234", 4)	1234
MID	=MID(text, start, n)	=MID("AB1234", 3, 4)	1234
LEN	=LEN(text)	=LEN("Hello")	5
FIND	=FIND(find, within)	=FIND("@","a@b.com")	2 (case-sensitive)
SEARCH	=SEARCH(find, within)	=SEARCH("A*","Amount")	1 (case-insensitive, wildcards ok)

Combining and Replacing Text

Function	Syntax	Example	Result
& operator	=text1&" "&text2	="Hello"&" "&"World"	Hello World

CONCAT	=CONCAT(t1,t2,...)	=CONCAT(A1," ",B1)	Joins without separator control
TEXTJOIN	=TEXTJOIN(delim,ignore,range)	=TEXTJOIN(", ",TRUE,A2:A5)	a, b, c, d
SUBSTITUTE	=SUBSTITUTE(text,old,new)	=SUBSTITUTE("Mr Smith","Mr","Ms")	Ms Smith
REPLACE	=REPLACE(text,start,n,new)	=REPLACE("AB123",3,3,"456")	AB456
TEXT	=TEXT(value,format)	=TEXT(44927,"DD/MM/YYYY")	01/01/2023
VALUE	=VALUE(text)	=VALUE("1,234")	1234 (numeric)

Practical Data Cleaning Workflow with Text Functions

8. Use TRIM on all imported text columns to remove leading/trailing spaces
9. Use PROPER or UPPER to standardise inconsistent name/category casing
10. Use SUBSTITUTE to strip unwanted characters (currency symbols, commas in numbers)
11. Use VALUE to convert text-formatted numbers to true numeric values
12. Use LEN + FIND to split "FirstName LastName" into separate columns
13. Use Flash Fill (Ctrl+E) for pattern-based splitting without formulas — Excel learns the pattern from one example

2.8 Date and Time Functions

Date and time data is extremely common in business datasets. Excel stores dates as serial numbers (1 = 1 January 1900) and times as decimal fractions of a day (0.5 = noon), enabling arithmetic directly on dates and times.

Current Date and Time

Function	Syntax	Description	Note
TODAY	=TODAY()	Returns today's date	Recalculates each time the file opens
NOW	=NOW()	Returns current date and time	Includes time component as decimal

Extracting Date and Time Components

Function	Example	Returns
YEAR	=YEAR(A2)	Year as a 4-digit number, e.g. 2024
MONTH	=MONTH(A2)	Month as a number 1–12
DAY	=DAY(A2)	Day of month as a number 1–31
WEEKDAY	=WEEKDAY(A2,2)	Day of week: Monday=1, Sunday=7 (with type 2)
WEEKNUM	=WEEKNUM(A2)	Week number of the year (1–53)
HOUR	=HOUR(A2)	Hour component (0–23)
MINUTE	=MINUTE(A2)	Minute component (0–59)
SECOND	=SECOND(A2)	Second component (0–59)

nasin-a-mahin

Date Arithmetic and Differences

Function	Syntax	Description	Example Result
Subtraction	=B2-A2	Days between two dates	=DATE(2024,12,31)-DATE(2024,1,1) → 365
DATEDIF	=DATEDIF(start,end,"Y")	Difference in years, months, or days	=DATEDIF(DOB,TODAY(),"Y") → age
DAYS	=DAYS(end,start)	Simple day count between dates	=DAYS(B2,A2)
NETWORKDAYS	=NETWORKDAYS(start,end)	Working days (Mon–Fri) between dates	Excludes weekends
NETWORKDAYS.INTL	=NETWORKDAYS.INTL(s,e,weekend)	Working days with custom weekend	e.g., Fri-Sat weekend
WORKDAY	=WORKDAY(start,n)	Date n working days from start	=WORKDAY(TODAY(),10)
EDATE	=EDATE(date,n)	Date n months from start	=EDATE(A2,3) → 3 months forward
EOMONTH	=EOMONTH(date,n)	Last day of month n months away	=EOMONTH(A2,0) → end of same month

2.9 Logical and Conditional Functions

Logical functions evaluate conditions and return different results based on TRUE or FALSE outcomes, enabling rule-based categorisation and error handling.

IF and Nested IF

```
=IF(logical_test, value_if_true, value_if_false)
```

Example — grade students:

```
=IF(A2>=90, "A", IF(A2>=75, "B", IF(A2>=60, "C", "F")))
```

Tip: Use IFS() for cleaner multi-condition logic (Excel 2019+):

```
=IFS(A2>=90,"A", A2>=75,"B", A2>=60,"C", TRUE,"F")
```

AND, OR, NOT

Function	Returns TRUE When	Example
AND(c1,c2,...)	ALL conditions are TRUE	=AND(A2>0, B2="Active")
OR(c1,c2,...)	ANY condition is TRUE	=OR(A2="North",A2="South")
NOT(condition)	Condition is FALSE	=NOT(ISBLANK(A2))

Combined example — flag rows where Sales > 5,000 AND Region = "North":

```
=IF(AND(D2>5000, C2="North"), "Target", "Other")
```

Error Handling Functions

Function	Syntax	Description
IFERROR	=IFERROR(formula, value_if_error)	Catches any error (#N/A, #DIV/0!, #REF!, etc.)
IFNA	=IFNA(formula, value_if_na)	Catches only #N/A errors — more precise
ISERROR	=ISERROR(value)	Returns TRUE if value is any error — use in IF
ISBLANK	=ISBLANK(A1)	Returns TRUE if cell is empty
ISNUMBER	=ISNUMBER(A1)	Returns TRUE if cell contains a number

```
=IFERROR(VLOOKUP(A2, Table, 2, 0), "Not Found")   Handle lookup errors gracefully

=IFNA(XLOOKUP(A2, B:B, C:C), "No Match")          Only catch #N/A
```

SWITCH — Clean Alternative to Nested IF

SWITCH is ideal when comparing a single value against multiple possible matches:

```
=SWITCH(A2, "N","North", "S","South", "E","East", "W","West", "Unknown")
```

2.10 Statistical Functions and the Data Analysis ToolPak

Core Statistical Functions

Function	Syntax	Description
STDEV.S	=STDEV.S(range)	Sample standard deviation (most common)
STDEV.P	=STDEV.P(range)	Population standard deviation
VAR.S	=VAR.S(range)	Sample variance
VAR.P	=VAR.P(range)	Population variance
QUARTILE.INC	=QUARTILE.INC(range,1)	Q1 (25th percentile)
PERCENTILE.INC	=PERCENTILE.INC(range,0.9)	90th percentile
RANK.EQ	=RANK.EQ(val,range,0)	Rank value in range (0=descending)

CORREL	=CORREL(arr1,arr2)	Pearson correlation coefficient
SKEW	=SKEW(range)	Skewness of a distribution
KURT	=KURT(range)	Kurtosis of a distribution
NORM.DIST	=NORM.DIST(x,mean,sd,TRUE)	Cumulative normal distribution probability
T.DIST.2T	=T.DIST.2T(x,df)	Two-tailed t-distribution probability

The Data Analysis ToolPak

Enable via: File → Options → Add-ins → Analysis ToolPak → Go. Adds a "Data Analysis" button to the Data tab providing one-click statistical analysis:

- Descriptive Statistics: Complete summary (mean, median, std dev, min, max, skewness, kurtosis) in one click
- Histogram: Frequency distribution with customisable bin ranges
- Correlation: Correlation matrix for multiple variables simultaneously
- Regression: Full linear regression output — coefficients, R-squared, ANOVA table, residuals, p-values
- t-Test: Paired samples, two-sample equal variance, two-sample unequal variance
- ANOVA: Single factor and two-factor analysis of variance
- Moving Average & Exponential Smoothing: Time-series smoothing tools
- Sampling: Extract a random sample from a dataset

2.11 Data Sorting, Filtering and Advanced Filter

Sorting Data

Sort via Data tab → Sort, or right-click a column header → Sort:

- Single-level sort: one column, ascending or descending
- Multi-level sort: Data → Sort → Add Level — e.g., sort by Region then by Sales within each region
- Custom sort order: sort by cell colour, font colour, or conditional formatting icon set
- Sort left to right: Options → Sort Left to Right (for transposed data)

> *Always convert data to a Table (Ctrl+T) before sorting. Table sorting is safer — it preserves all row data integrity and makes it easy to revert.*

AutoFilter

AutoFilter (Ctrl+Shift+L) adds dropdown arrows to column headers:

- Number Filters: Greater Than, Less Than, Between, Top 10, Above Average, Custom
- Text Filters: Contains, Does Not Contain, Begins With, Ends With
- Date Filters: Yesterday, This Week, Last Month, Between, Custom date range
- Filter by colour: filter rows by cell fill colour or font colour
- Multiple column filters: each column filter applies on top of existing filters (AND logic)

yasin-a-mahin

Advanced Filter

Data → Advanced: for complex filtering that AutoFilter cannot handle.

14. Create a criteria range on the same sheet — headers must exactly match data headers
15. The criteria range can contain multiple rows (each row = OR condition) and multiple columns (same row = AND condition)
16. Choose "Copy to another location" to extract results without disturbing the original data
17. Use formulas in the criteria range for computed conditions (e.g., =D2>AVERAGE($D$2:$D$100))

2.12 Data Validation

Data Validation (Data → Data Validation) restricts what values users can enter into cells — preventing data entry errors at source rather than detecting them later.

Validation Types

Type	Setting	Use Case
Whole Number	Between 1 and 100	Age, quantity, score
Decimal	Greater than 0	Price, percentage, measurement
List	=A1:A10 or "Yes,No,Maybe"	Standardise category entry via dropdown
Date	Between 01/01/2024 and 31/12/2024	Date range enforcement
Time	Between 09:00 and 17:00	Working hours restriction
Text Length	Between 10 and 13 characters	Phone number, reference code
Custom	=ISNUMBER(A1) or any formula	Complex business rules

Input Messages and Error Alerts

- Input Message (optional): shown as a tooltip when the cell is selected — guides the user on what to enter
- Error Alert — Stop: blocks invalid entry entirely (most restrictive)
- Error Alert — Warning: shows a warning but allows the user to proceed
- Error Alert — Information: notifies without blocking entry

Example — allow only dates in 2024 using a custom formula:

```
=AND(ISNUMBER(A1), YEAR(A1)=2024)
```

Circle Invalid Data

After setting up validation, use Data → Data Tools → Circle Invalid Data to highlight cells that already contain invalid values — useful for auditing existing datasets.

2.13 Conditional Formatting

Conditional Formatting automatically applies visual formatting (colour fills, font colours, icons, data bars) based on cell values, making patterns, trends, and outliers immediately visible without manual inspection.

Built-in Conditional Formatting Rules

Category	Examples
Highlight Cell Rules	Greater Than, Less Than, Between, Equal To, Text That Contains, Duplicate Values
Top/Bottom Rules	Top 10 Items, Top 10%, Bottom 10 Items, Below Average, Above Average
Data Bars	Proportional horizontal bars within cells — great for ranking comparisons at a glance
Colour Scales	2-colour or 3-colour gradient across range — creates a heat map effect
Icon Sets	Traffic lights (●●●), directional arrows (↑→↓), flags (⚑), stars (★★★) by threshold

Formula-Based Conditional Formatting — Most Powerful

Home → Conditional Formatting → New Rule → Use a formula. Select the entire data range first (e.g., A2:F100), then enter a formula that evaluates to TRUE/FALSE for the top-left cell of the range:

```
=$C2="Overdue"                    Highlight entire row where Status = Overdue
=$E2<AVERAGE($E$2:$E$100)         Highlight rows below average sales
=MOD(ROW(),2)=0                   Alternate row shading (zebra stripes)
=AND($C2="Active",$D2>TODAY())    Multiple conditions: active AND future date
```

- Lock the column with $ (e.g., $C2 not C2) so the condition checks the same column for all rows
- Manage rules: Home → Conditional Formatting → Manage Rules — change priority and stop-if-true behaviour
- Rules are evaluated top to bottom; tick "Stop If True" to prevent lower-priority rules from overriding

2.14 PivotTables and PivotCharts

PivotTables are the most powerful analysis feature in Excel. They enable interactive summarisation, cross-tabulation, and aggregation of large datasets — instantly and without writing a single formula.

Creating a PivotTable — Step by Step

18. Click anywhere inside your data range or Table
19. Go to Insert → PivotTable → New Worksheet (recommended)

20. In the PivotTable Fields pane, drag fields to four areas:
21. Filters: Creates a dropdown above the PivotTable to filter the entire report
22. Columns: Field values become column headers across the top
23. Rows: Field values become row labels down the left side
24. Values: Numeric field to aggregate — right-click to change from Sum to Count/Average/Max etc.

Key PivotTable Features

Feature	How to Access	What It Does
Grouping Dates	Right-click date field → Group	Auto-groups into Year, Quarter, Month — one click
Calculated Field	PivotTable Analyse → Fields, Items & Sets → Calculated Field	Add custom metrics: Profit Margin = Profit/Sales
Show Values As	Right-click Values field	% of Grand Total, Running Total, Rank, % of Parent Row
Slicers	PivotTable Analyse → Insert Slicer	Visual filter buttons — one slicer can control multiple PivotTables
Timelines	Insert → Timeline	Date filter with Month/Quarter/Year granularity slider
Conditional Formatting	Apply to Values area	Heat maps, data bars, and icons inside PivotTable
GETPIVOTDATA	Type = then click a PivotTable cell	Pull specific PivotTable values into formulas elsewhere
Refresh	Right-click → Refresh or Alt+F5	Update PivotTable after source data changes

PivotCharts

PivotCharts are dynamic charts linked directly to a PivotTable — they update automatically when filters or slicers change.

- Create: PivotTable Analyse → PivotChart, or Insert → PivotChart
- The PivotChart inherits all slicers and filters from the linked PivotTable
- Right-click chart → Change Chart Type to switch between chart styles without losing filter links
- Use PivotCharts as the core of interactive, self-service dashboards

2.15 Data Visualisation with Charts and Sparklines

Choosing the Right Chart Type

Chart Type	Best Used For	Excel Location
Clustered Column	Compare values across categories	Insert → Column
Stacked Column / Bar	Show composition and comparison together	Insert → Column/Bar
Line Chart	Trends and changes over time	Insert → Line
Pie / Doughnut	Part-to-whole proportions (2–5 segments max)	Insert → Pie

Scatter (XY) Plot	Correlation between two numeric variables	Insert → Scatter
Histogram	Distribution of a single numeric variable	Insert → Statistical → Histogram
Box & Whisker	Distribution, median, IQR, and outliers by group	Insert → Statistical
Waterfall	Cumulative gains and losses (P&L, cash flow)	Insert → Waterfall
Funnel	Stages in a process with decreasing values	Insert → Funnel
Treemap	Hierarchical proportions as nested rectangles	Insert → Hierarchy → Treemap
Combo Chart	Two chart types on one chart (bar + line)	Right-click → Change Chart Type → Combo
Map Chart	Geographic data by country or region	Insert → Maps (requires data connectivity)

Building a Professional Chart

25. Select your data range (always include headers)
26. Insert → Charts → choose chart type
27. Chart Design tab: Add Chart Element — add/edit title, axis labels, legend, data labels, gridlines
28. Format tab: Format individual elements by double-clicking (change colours, fonts, borders)
29. Right-click → Select Data: add/remove series or switch rows/columns
30. Chart Design → Chart Styles: apply a professional preset in one click
31. Move chart: Chart Design → Move Chart → New Sheet for a full-page chart

Sparklines

Sparklines are miniature charts inside a single cell — ideal for showing trends in dashboards without consuming significant space.

- Insert → Sparklines → Line, Column, or Win/Loss
- Select data range and the destination cells
- Sparkline tab: customise colours, highlight Max/Min points, and set axis options
- Group sparklines to apply consistent formatting across a row or column

2.16 What-If Analysis

What-If Analysis tools explore how changing input values affects formula results — essential for financial modelling, planning, and scenario analysis.

Goal Seek

Goal Seek finds the specific input value needed to achieve a desired formula result — working backwards from the answer to the input.

32. Set up a formula (e.g., =PMT(B1/12, B2, -B3) for monthly loan payment)
33. Data → What-If Analysis → Goal Seek
34. Set Cell: the formula cell | To value: your target | By changing cell: the input
35. Excel iterates to find the exact input that produces the target

Example: What interest rate gives a monthly payment of exactly £800?

Set Cell: payment formula | To value: 800 | By changing cell: interest rate cell

Scenario Manager

Scenario Manager lets you define, save, and switch between multiple named sets of input values.

36. Data → What-If Analysis → Scenario Manager → Add
37. Name your scenario (e.g., "Best Case") and select the changing cells
38. Enter the values for this scenario and add more scenarios (Base Case, Worst Case)
39. Summary: Click Summary to generate a scenario comparison table on a new sheet

Data Tables — Sensitivity Analysis

Data Tables show how a formula result changes across a range of input values — the most efficient way to do sensitivity analysis.

- One-Variable Data Table: vary one input (e.g., interest rate) across a range and see all resulting payments
- Two-Variable Data Table: vary two inputs simultaneously (e.g., rate AND term) — creates a matrix of results
- Set up: formula in corner cell, inputs in adjacent row/column, select full table range → Data → What-If → Data Table

2.17 Power Query – Data Import and Transformation

Power Query (Data → Get Data) is Excel's built-in ETL (Extract, Transform, Load) engine. It records every data transformation as an "Applied Step" — creating a fully reproducible, refreshable data preparation workflow that runs with a single click.

> *Power Query is transformative because it separates data preparation from analysis. Instead of manually cleaning data every time, you build the steps once and refresh automatically. This turns a 2-hour monthly task into a 2-second one.*

Connecting to Data Sources

- Excel file / Workbook: Get Data → From File → From Excel Workbook
- CSV / Text: Get Data → From File → From Text/CSV
- SQL Database: Get Data → From Database → choose server type and enter credentials
- Web page: Get Data → From Other Sources → From Web — scrape tables from any URL
- SharePoint / OneDrive: Get Data → From Online Services

easin-a-mahin

- Folder: Get Data → From File → From Folder — combine all files in a folder automatically

Common Power Query Transformations

Transformation	How to Access	Purpose
Remove Columns	Right-click column header → Remove	Keep only columns needed for analysis
Change Type	Data Type dropdown in column header	Ensure correct types (Date, Whole Number, Text, Decimal)
Remove Duplicates	Home → Remove Rows → Remove Duplicates	Deduplicate rows based on selected columns
Filter Rows	Dropdown arrow → Number/Text Filters	Include or exclude rows by condition
Replace Values	Home → Replace Values	Standardise inconsistent entries (e.g., "UK" → "United Kingdom")
Split Column	Home → Split Column → By Delimiter	Split "First Last" into two columns
Merge Columns	Select columns → Add Column → Merge Columns	Combine columns with a separator
Unpivot Columns	Select columns → Transform → Unpivot	Convert wide format to long/tidy format
Group By	Home → Group By	Aggregate: sum Sales by Region
Merge Queries	Home → Merge Queries	JOIN two tables by a key column (like SQL JOIN)
Append Queries	Home → Append Queries	Stack tables vertically (like SQL UNION ALL)
Add Custom Column	Add Column → Custom Column	New column with M language formula
Conditional Column	Add Column → Conditional Column	IF/THEN logic without M code
Fill Down	Transform → Fill → Down	Propagate values to fill null gaps

The M Language

Every Power Query transformation generates M language code in the background. View and edit it via View → Advanced Editor. Example:

```
let
    Source   = Excel.Workbook(File.Contents("sales.xlsx"), null, true),
    Sheet    = Source{[Name="Data"]}[Data],
    Filtered = Table.SelectRows(Sheet, each [Sales] > 1000),
    Sorted   = Table.Sort(Filtered, {{"Date", Order.Descending}})
in  Sorted
```

Refreshing Data

- Manual refresh: Data → Refresh All, or right-click query → Refresh
- Automatic refresh on file open: Data → Queries & Connections → right-click → Properties → Refresh data when opening the file
- Scheduled refresh: available when the workbook is published to SharePoint or Power BI

2.18 Named Ranges and Formula Auditing

Named Ranges

Naming a range replaces cryptic cell addresses with meaningful names, making formulas self-documenting and less error-prone.

Creating Named Ranges — Three Methods

Method 1 — Name Box (fastest):

40. Select the range you want to name
41. Click the Name Box (left of formula bar — shows current cell address)
42. Type the name and press Enter

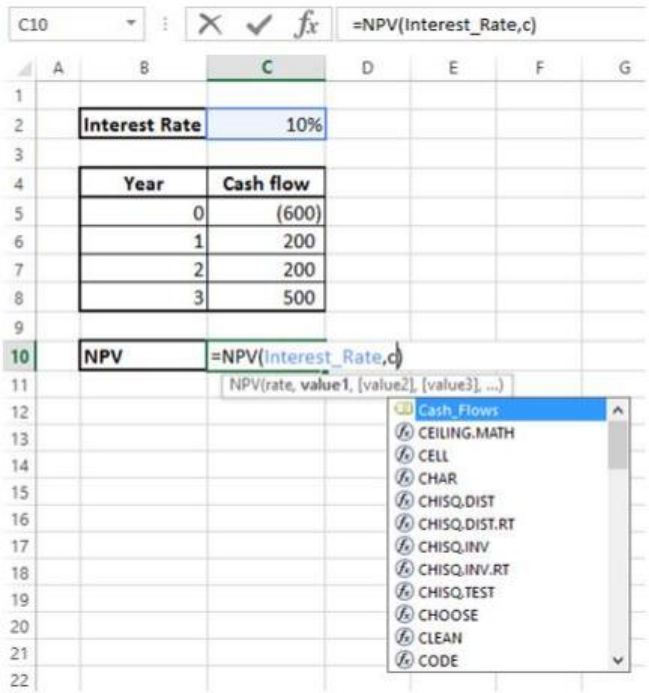


Fig 2.1 — Creating a Name using the Name Box

Method 2 — New Name Dialog:

43. Select the range
44. Formulas tab → Define Name → New Name dialog appears
45. Enter the name, verify the Refers to range, click OK

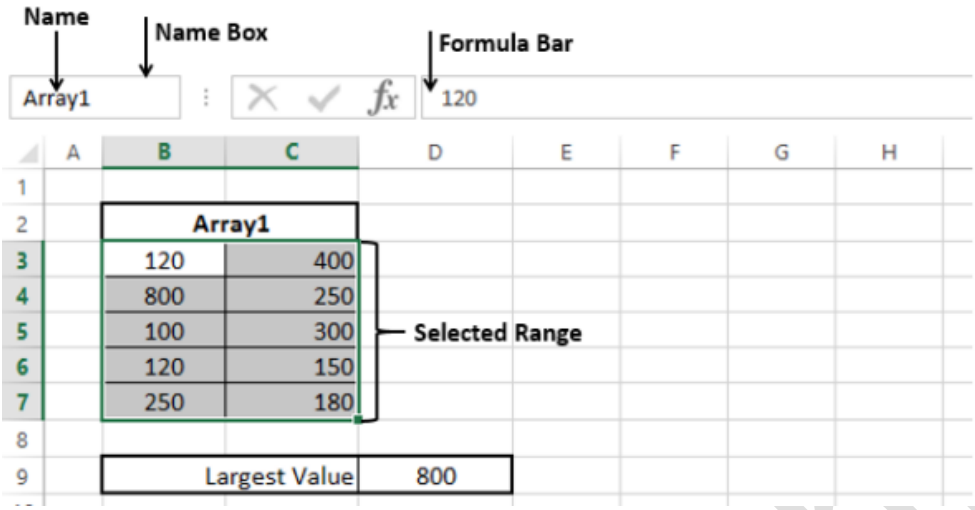


Fig 2.2 — New Name Dialog Box

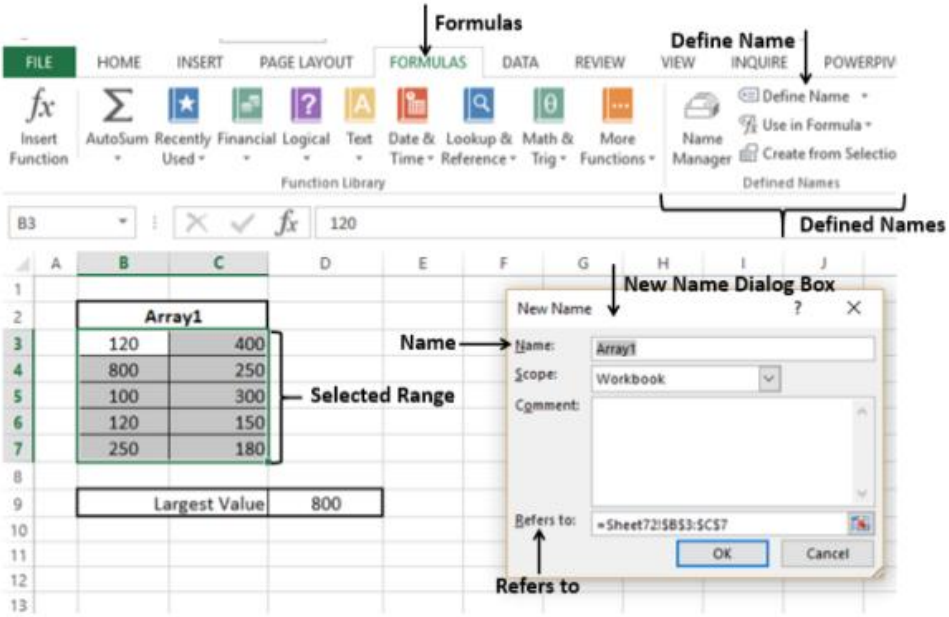


Fig 2.3 — Completed New Name Dialog

Method 3 — Create from Selection (when labels are adjacent):

46. Select data along with the adjacent row/column containing labels
47. Formulas → Create from Selection → tick Top Row or Left Column as appropriate
48. Excel creates names from the labels automatically

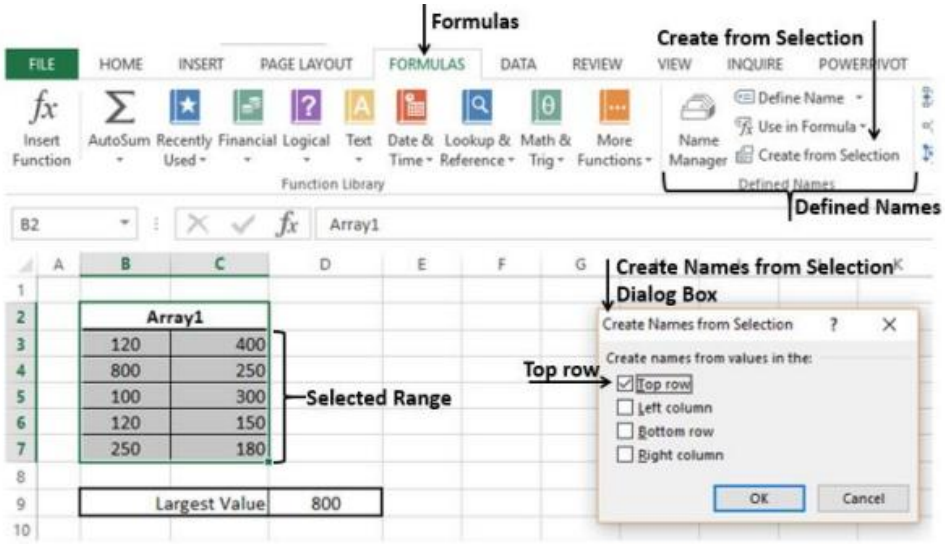


Fig 2.4 — Create Names from Selection Dialog

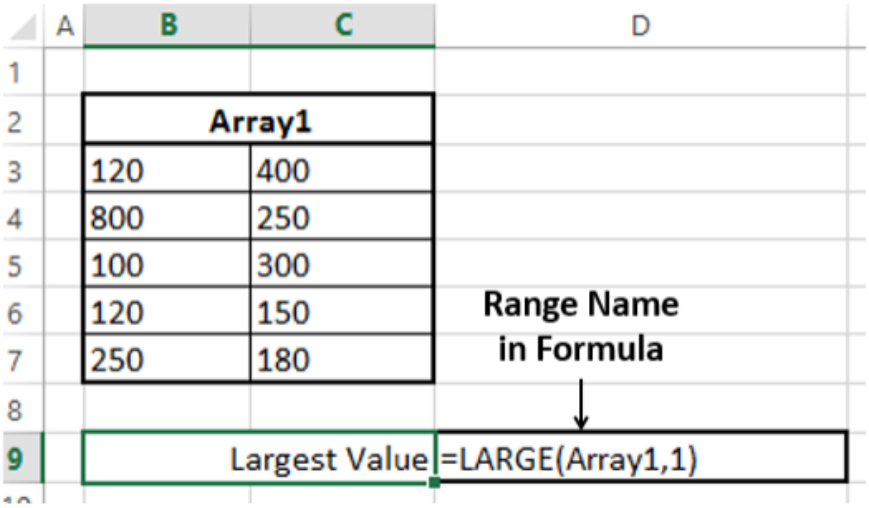


Fig 2.5 — Using a named range in a SUM formula

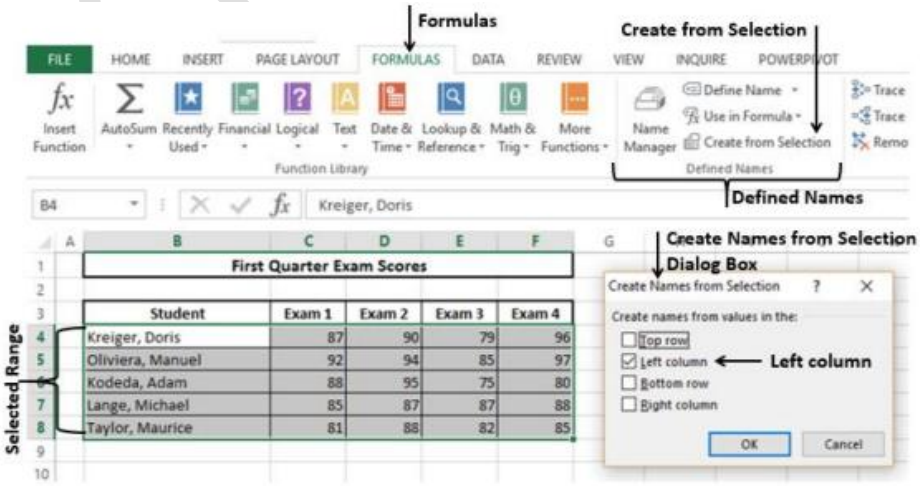


Fig 2.6 — Multiple named ranges created from selection

	A	B	C	D	E	F	G
1		First Quarter Exam Scores					
2							
3		Student	Exam 1	Exam 2	Exam 3	Exam 4	Total
4		Kreiger, Doris	87	90	79	96	=SUM(Kreiger_Doris)
5		Oliviera, Manuel	92	94	85	97	=SUM(Oliviera_Manuel)
6		Kodeda, Adam	88	95	75	80	=SUM(Kodeda_Adam)
7		Lange, Michael	85	87	87	88	=SUM(Lange_Michael)
8		Taylor, Maurice	81	88	82	85	=SUM(Taylor_Maurice)

Range Names in Formulas

Fig 2.7 — Named ranges applied in formulas

Creating Names for Constants

You can assign a name to a constant value without placing it in a cell — useful for fixed rates or thresholds used throughout the workbook:

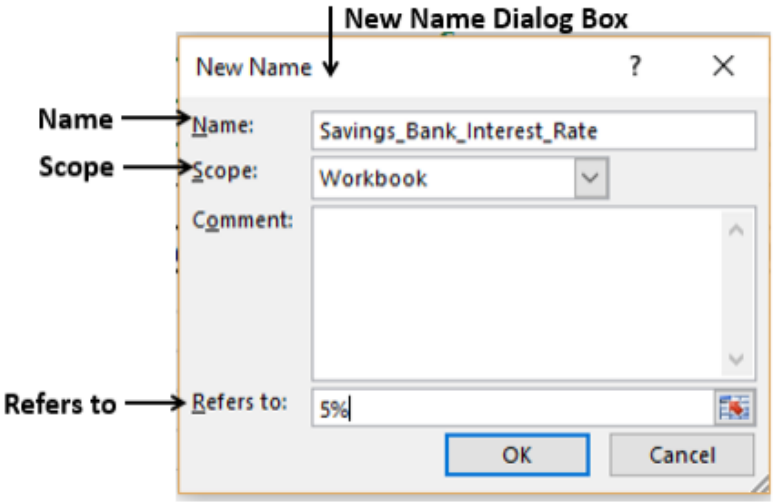


Fig 2.8 — Naming a constant in the New Name dialog

nasrin shin

Fig 2.9 — Constant name confirmed in Name Manager

Managing Names with Name Manager

Formulas → Name Manager shows all defined names with their values, references, scope, and comments.

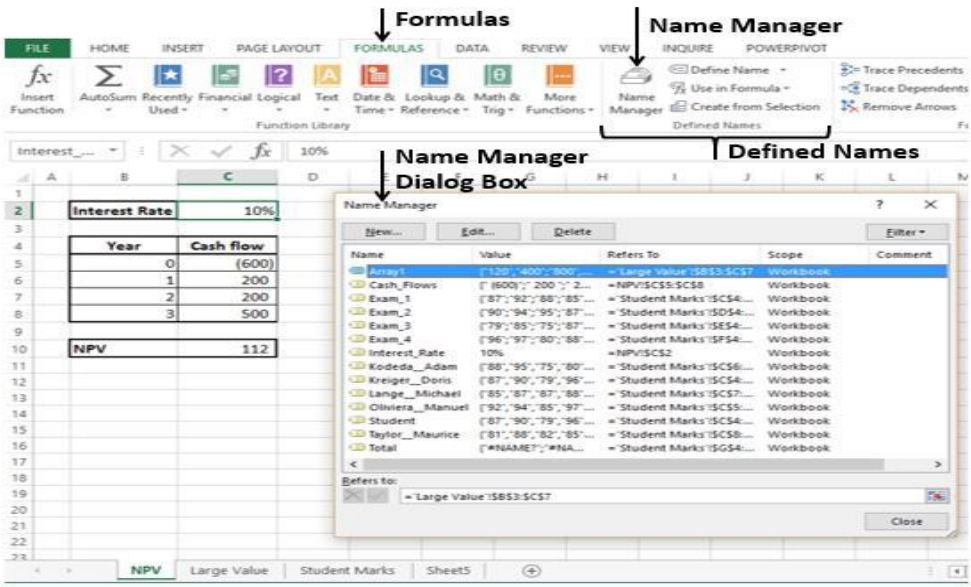


Fig 2.10 — The Name Manager dialog showing all defined names

- New: define a new name
- Edit: modify name, reference range, or comment
- Delete: remove selected names
- Filter: show names by scope or error status

Scope of a Name

A name can be scoped to the entire Workbook (accessible from any sheet) or to a specific Worksheet (only accessible on that sheet). Scope is set when creating the name and cannot be changed afterwards.

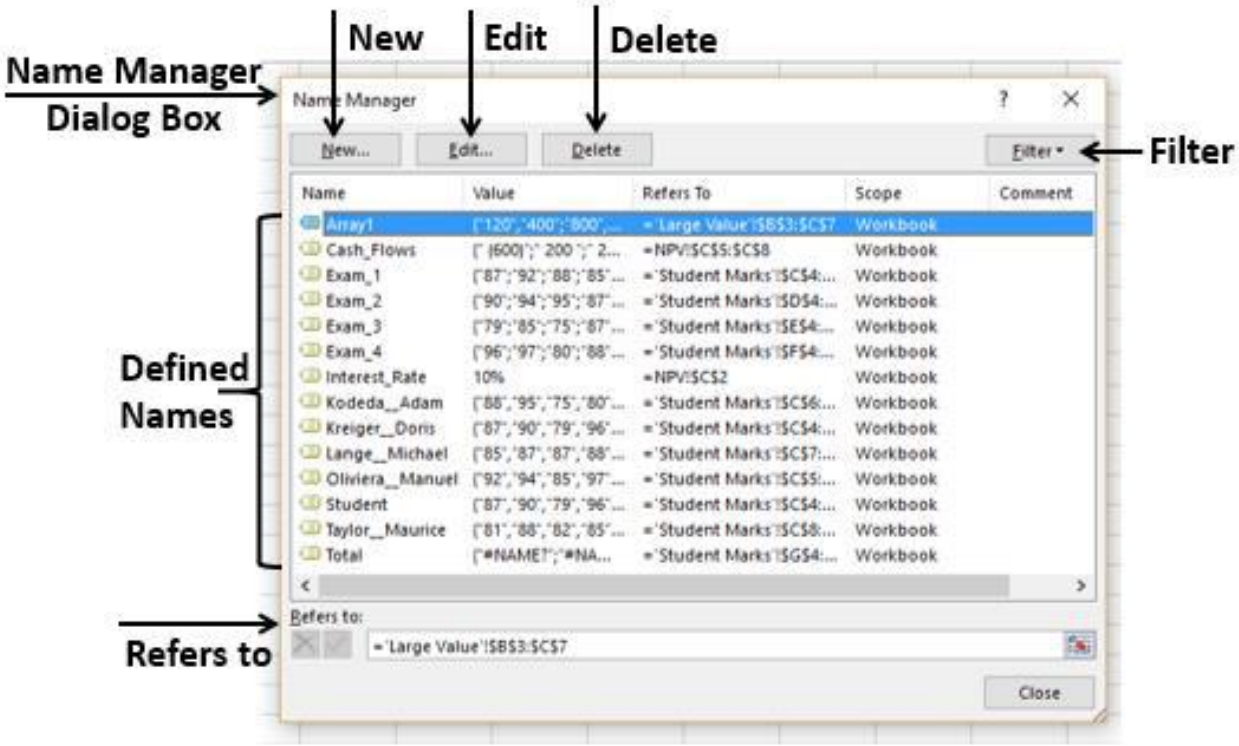


Fig 2.11 — Scope dropdown in the New Name dialog

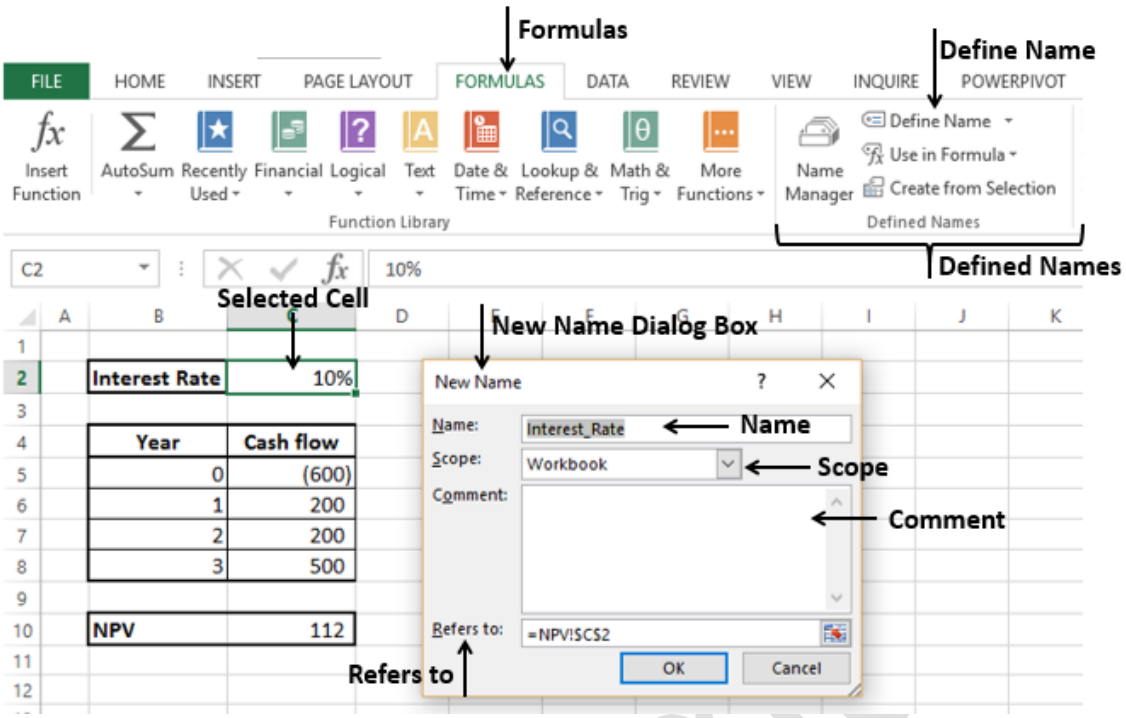


Fig 2.12 — Setting scope to a specific worksheet

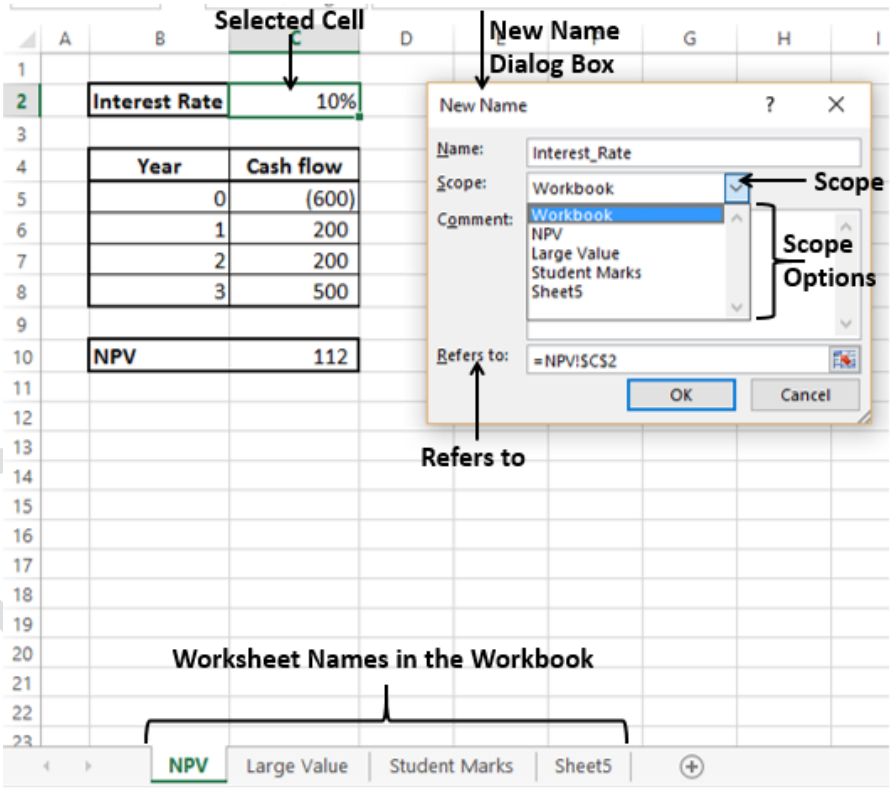


Fig 2.13 — Scope options dropdown list

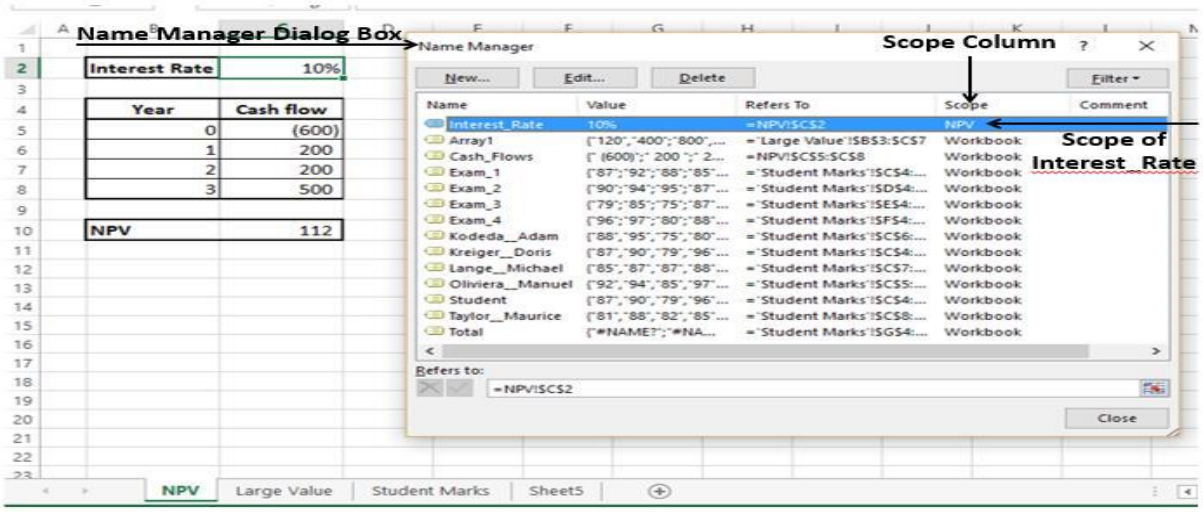


Fig 2.14 — Verifying worksheet scope in Name Manager

Deleting Names with Errors

Named ranges can develop errors when their referenced cells are deleted. Filter and remove them via Name Manager:

49. Formulas → Name Manager → Filter → Names with Errors
50. Select the errored names
51. Click Delete → OK to confirm

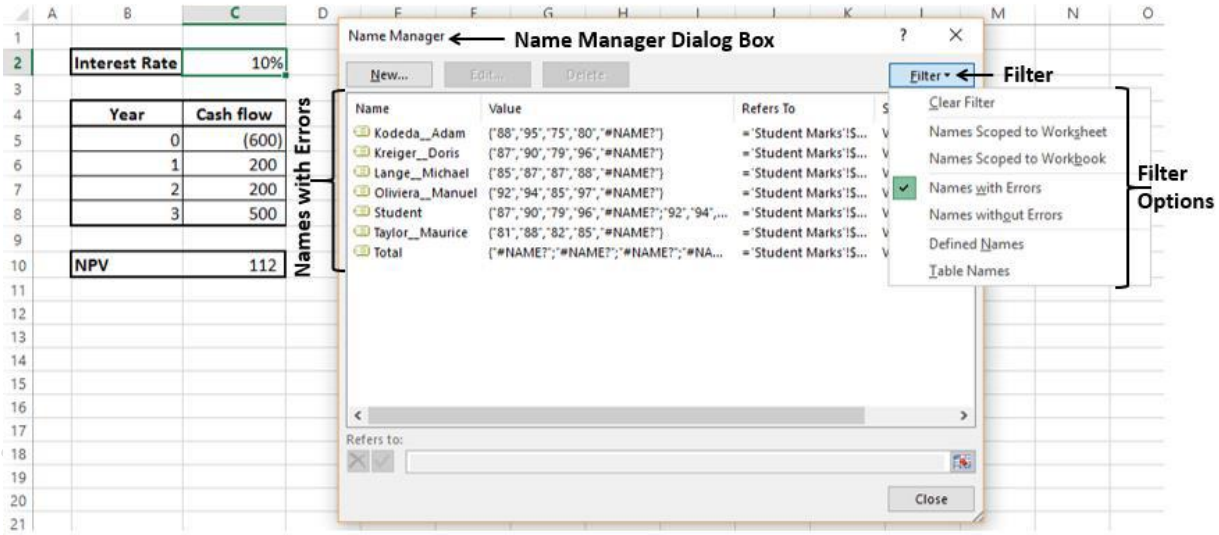


Fig 2.15 — Filtering names with errors in Name Manager

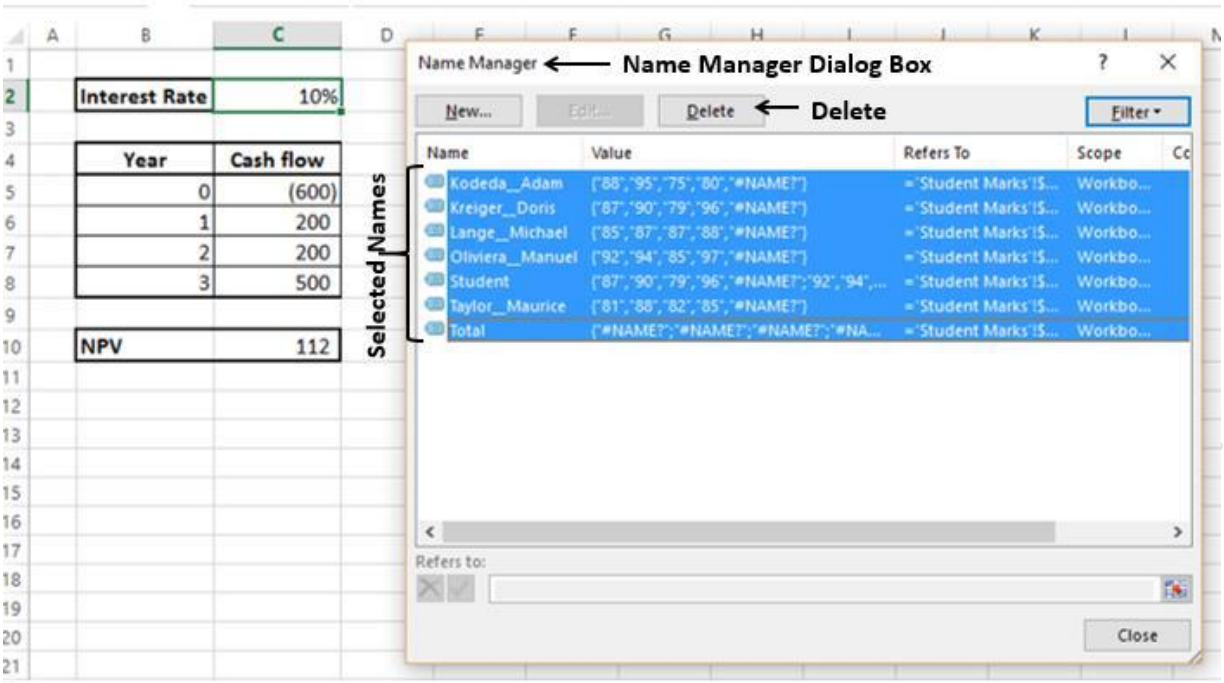


Fig 2.16 — Deleting selected error names

Editing Named Ranges

To edit a name, a reference range, or a comment:

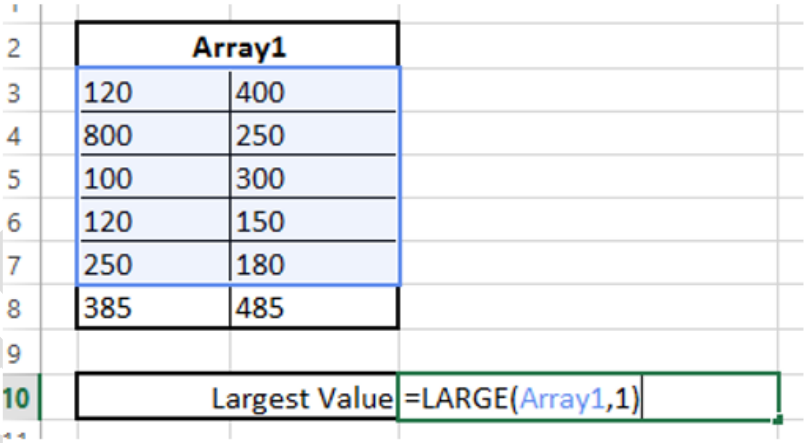


Fig 2.17 — Selecting the name to edit in Name Manager

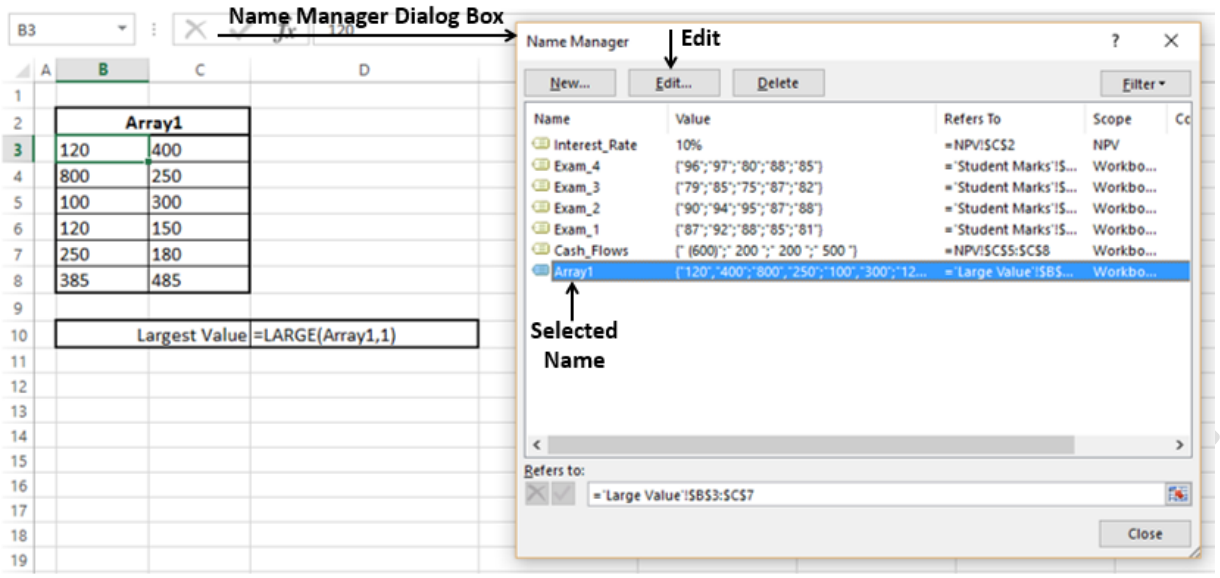


Fig 2.18 — Click Edit to open the Edit Name dialog

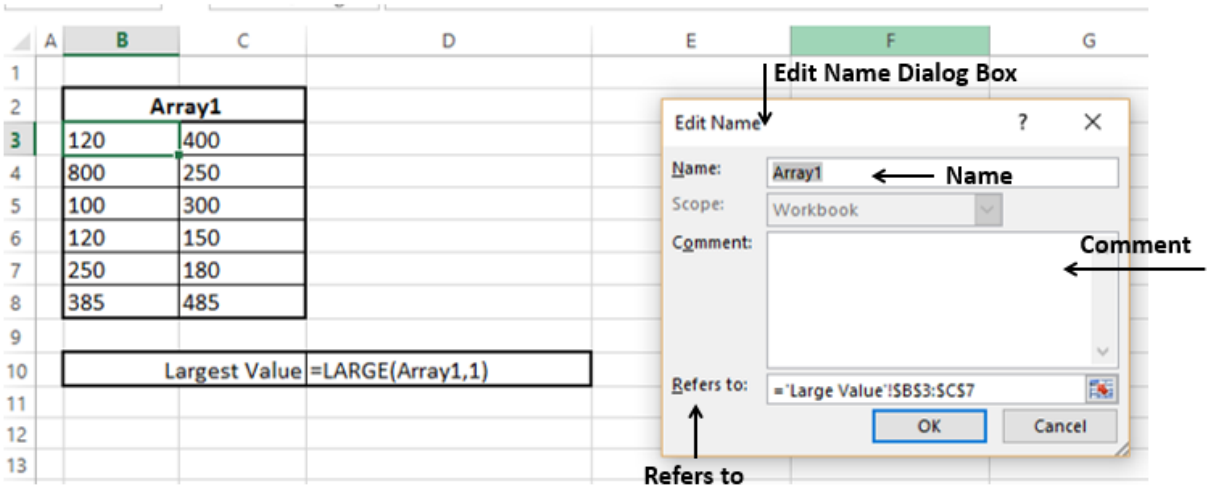


Fig 2.19 — The Edit Name dialog box

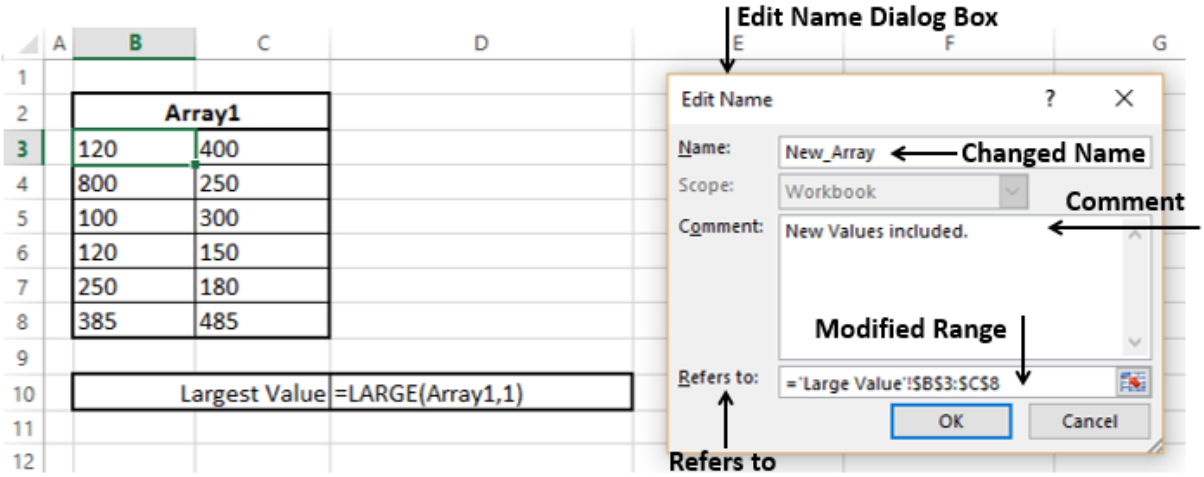


Fig 2.20 — Updating the Refers to range

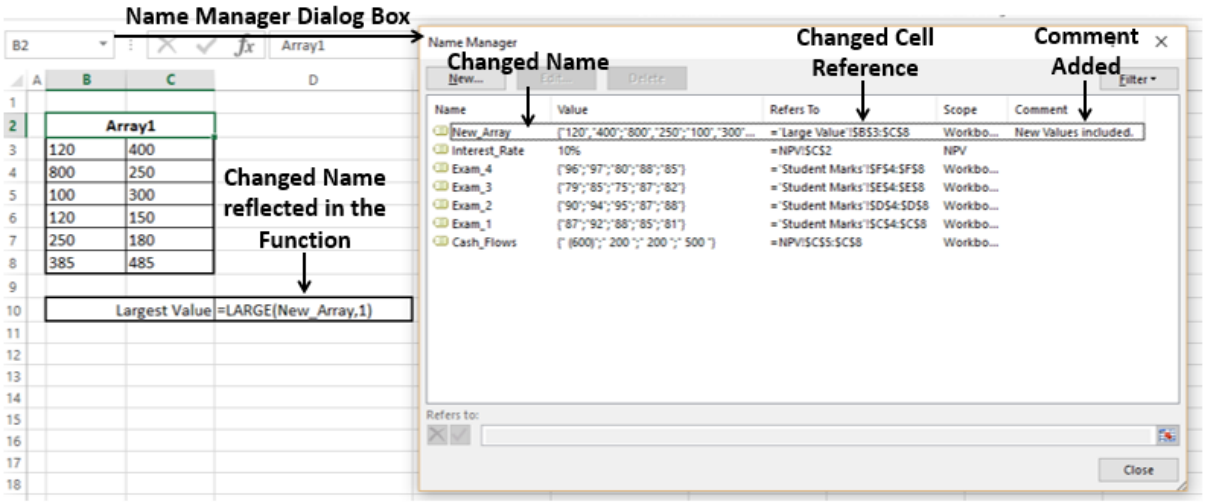


Fig 2.21 — Edited name confirmed in Name Manager

Applying Names to Existing Formulas

If formulas were written before names were defined, you can apply names retroactively:

	A	B	C
13			
14		Amount Borrowed	400000
15		Number of Months	180
16		Annual Interest Rate	0.06
17		Monthly Payment	=-PMT(C16/12,C15,C14)

Fig 2.22 — Formula using cell references before names applied

52. Define names using Create from Selection
53. Select the cell(s) containing the formula
54. Formulas → Define Name dropdown → Apply Names
55. Select the names to apply and click OK

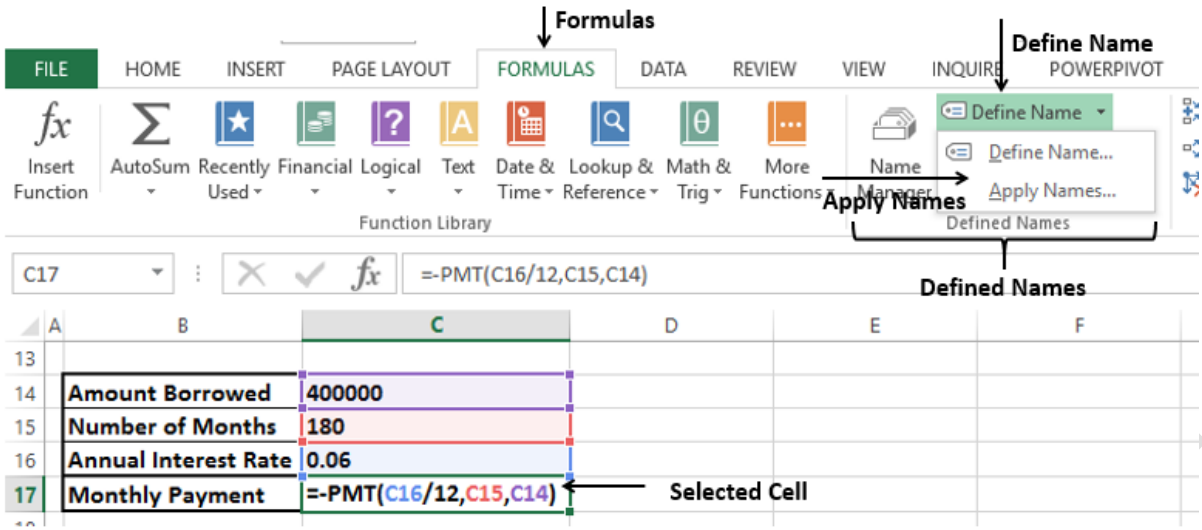


Fig 2.23 — Apply Names dropdown on the Formulas tab

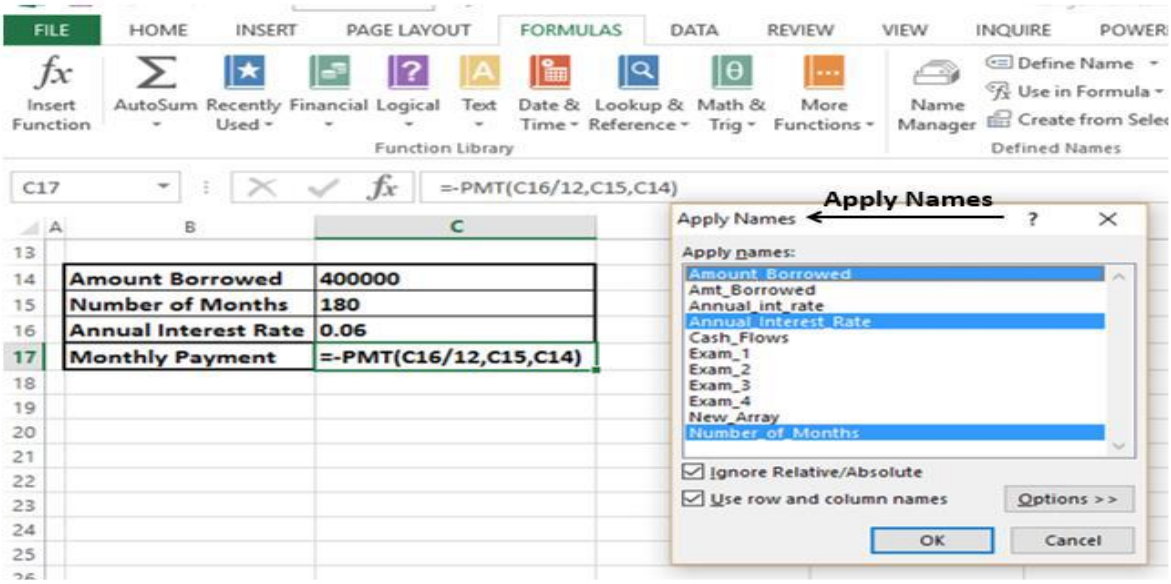


Fig 2.24 — Apply Names dialog — select names to substitute

Formula Auditing Tools

Formulas tab → Formula Auditing group provides tools to trace, understand, and debug formulas:

Tool	Shortcut	Purpose
Trace Precedents	Ctrl+[	Draws blue arrows to all cells that feed into the active formula
Trace Dependents	Ctrl+]	Shows which cells use the active cell in their formulas
Remove Arrows	—	Clears all trace arrows from the worksheet
Show Formulas	Ctrl+`	Toggles between formula text and calculated results
Evaluate Formula	—	Steps through a formula one calculation at a time — ideal for debugging
Error Checking	—	Auto-flags #VALUE!, #REF!, #DIV/0!, #NAME?, #N/A errors

Watch Window	—	Monitor cell values from anywhere in the workbook without scrolling

2.19 Financial Functions

Excel includes a comprehensive set of financial functions used for investment appraisal, loan analysis, and cash flow modelling — essential for financial data analysis.

Key Financial Functions

Function	Syntax	Description	Example
PMT	=PMT(rate,nper,pv)	Calculates periodic loan payment	=PMT(5%/12,60,-10000) → monthly payment
PV	=PV(rate,nper,pmt)	Present value of future cash flows	=PV(8%,10,-1000) → value today
FV	=FV(rate,nper,pmt,pv)	Future value of an investment	=FV(6%/12,24,-200,0)
NPV	=NPV(rate,value1,value2,...)	Net present value of cash flows	=NPV(10%,C2:C6)
IRR	=IRR(values)	Internal rate of return	=IRR(C1:C6)
RATE	=RATE(nper,pmt,pv)	Interest rate per period	=RATE(48,-250,10000)*12 → annual rate
NPER	=NPER(rate,pmt,pv)	Number of periods for a loan/investment	=NPER(5%/12,-500,20000)
XNPV	=XNPV(rate,values,dates)	NPV with irregular cash flow dates	More accurate than NPV for real projects
XIRR	=XIRR(values,dates)	IRR with irregular cash flow dates	=XIRR(C2:C10,D2:D10)

> *In financial functions, payments made (outflows) are typically negative, and money received (inflows) are positive. The rate argument is always the rate per period — divide annual rates by 12 for monthly calculations.*

2.20 Excel Best Practices for Data Analysts

Following these best practices will make your Excel work more reliable, reproducible, and professional.

Data Organisation

- Store raw data in a separate sheet — never overwrite or modify source data directly
- One row per record, one column per variable — tidy data structure
- Keep column headers in the first row, with no blank rows or columns within the data

- Avoid merging cells in data ranges — they break sorting, filtering, PivotTables, and Power Query
- Convert all data ranges to Tables (Ctrl+T) for structured references and auto-expansion

Formula Best Practices

- Use named ranges for frequently referenced values — makes formulas readable and maintainable
- Add IFERROR wrappers around lookup formulas to handle missing matches gracefully
- Use absolute references ($A$1) for fixed values like tax rates or conversion factors
- Avoid hardcoding values inside formulas — reference them from input cells instead
- Break complex nested formulas across helper columns for easier auditing and debugging

Collaboration and Documentation

- Add a "Notes" or "Assumptions" sheet documenting what the workbook does, data sources, and key assumptions
- Use cell comments (Shift+F2) or Notes to explain non-obvious calculations
- Use consistent, descriptive sheet names (not "Sheet1", "Sheet2")
- Protect formula cells from accidental editing: Home → Format → Lock Cells → Protect Sheet
- Save in .xlsx format (not legacy .xls) and keep a dated backup before major changes

Performance

- Use Power Query for data import instead of formulas that reference large external files
- Convert VLOOKUP to XLOOKUP or INDEX-MATCH on large datasets — faster recalculation
- Avoid volatile functions (NOW, TODAY, RAND, OFFSET, INDIRECT) in large workbooks — they recalculate constantly
- Use Excel Tables — Excel optimises calculation for structured table data
- If the workbook is slow, check for circular references: Formulas → Error Checking → Circular References

nasrin-a-mahin

UNIT III – SQL for Data Analysis

3.1 Introduction to SQL

SQL (Structured Query Language) is the standard language for communicating with relational database management systems (RDBMS). Originally developed at IBM in the 1970s and standardised by ANSI/ISO, SQL has become the lingua franca of data work — used everywhere from small business databases to petabyte-scale cloud data warehouses.

For data analysts, SQL is indispensable. It lets you retrieve precisely the records you need, transform raw data into meaningful summaries, and combine information spread across multiple tables — all without writing a single line of application code. Industries such as finance, healthcare, marketing, and retail rely on SQL daily to extract insights from transactional systems.

Why SQL Matters for Data Analysis

Almost every data analysis workflow begins with extraction. Whether the source is a transactional database (orders, customers, products), a financial system (ledgers, invoices), or a marketing platform (campaigns, clicks, conversions), SQL is the tool that pulls the right data into the right shape. Its declarative nature — you describe what you want, not how to get it — makes it both readable and powerful.

Anatomy of a Relational Database

A relational database stores data in tables (also called relations). Each table consists of columns (attributes) and rows (records). Tables are linked to one another through keys:

- Primary Key — a column (or combination of columns) that uniquely identifies every row in a table.
- Foreign Key — a column in one table that references the primary key of another, enforcing referential integrity between related data.
- Index — a data structure that speeds up lookups on frequently queried columns.

Note: All SQL examples in this unit use standard ANSI SQL syntax. Minor differences may exist between database engines (MySQL, PostgreSQL, SQL Server, SQLite); these are noted where relevant.

nasrin-a-mahin

3.2 SQL Command Categories

SQL commands are grouped into five categories, each serving a distinct purpose in database management and analysis.

Category	Full Name	Core Purpose
DDL	Data Definition Language	Defines and modifies database structures: tables, indexes, schemas.
DQL	Data Query Language	Retrieves data using the SELECT statement.
DML	Data Manipulation Language	Inserts, updates, and deletes rows within tables.
DCL	Data Control Language	Controls user access rights and permissions.
TCL	Transaction Control Language	Manages groups of operations as atomic transactions.

DDL — Data Definition Language

DDL commands define the blueprint of your database. They deal with the structure (schema) of database objects, not the data stored within them. DDL operations are auto-committed in most RDBMS engines and cannot be rolled back.

Command	Action	Example Syntax
CREATE	Creates a new database object	CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50));
ALTER	Modifies an existing object	ALTER TABLE employees ADD COLUMN department VARCHAR(50);

DROP	Permanently deletes an object and all its data	DROP TABLE employees;
TRUNCATE	Removes all rows but keeps the table structure	TRUNCATE TABLE employees;
RENAME	Renames an existing object	RENAME TABLE employees TO staff;

Example — Creating a Database and Table

```
-- Step 1: Create the database
CREATE DATABASE company_db;


-- Step 2: Switch to the new database
USE company_db;


-- Step 3: Create the employees table
CREATE TABLE employees (
    employee_id   INT          PRIMARY KEY,
    first_name    VARCHAR(50)  NOT NULL,
    last_name     VARCHAR(50)  NOT NULL,
    department    VARCHAR(50),
    hire_date     DATE,
    salary        DECIMAL(10, 2)
);
```

DQL — Data Query Language

DQL has exactly one command: SELECT. However, SELECT is the most powerful command in SQL. It can be combined with numerous clauses to filter, sort, group, and aggregate data.

Clause	Purpose
SELECT	Specifies which columns to retrieve.

FROM	Identifies the table(s) to query.
WHERE	Filters rows before any grouping is applied.
GROUP BY	Groups rows sharing the same value in specified columns.
HAVING	Filters groups after aggregation (used after GROUP BY).
ORDER BY	Sorts the result set — ASC (default) or DESC.
DISTINCT	Removes duplicate rows from the result set.
LIMIT / TOP	Restricts the number of rows returned.

Important: FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT, and LIMIT are clauses of SELECT, not independent commands. DQL technically contains only SELECT.

DML — Data Manipulation Language

DML commands operate on the data stored within tables. Unlike DDL, DML operations can be rolled back within a transaction, giving you a safety net when modifying data.

Command	Action	Example Syntax
INSERT	Adds new rows to a table	INSERT INTO employees (first_name, last_name) VALUES ('Ana', 'Lima');
UPDATE	Modifies existing rows	UPDATE employees SET salary = 72000 WHERE employee_id = 7;
DELETE	Removes specific rows from a table	DELETE FROM employees WHERE department = 'Temp';

DCL — Data Control Language

DCL manages who can access what within the database. Database administrators use DCL commands to enforce the principle of least privilege — users should have only the permissions they genuinely need.

Command	Purpose and Example
GRANT	Gives a user specific permissions. GRANT SELECT, INSERT ON employees TO 'analyst_user';
REVOKE	Removes previously granted permissions. REVOKE INSERT ON employees FROM 'analyst_user';

TCL — Transaction Control Language

A transaction is a logical unit of work comprising one or more SQL statements that must all succeed or all fail together — the "all-or-nothing" property. TCL commands manage this behaviour, which is critical for maintaining data consistency in multi-step operations.

Command	Purpose
BEGIN TRANSACTION	Starts a new transaction block.
COMMIT	Permanently saves all changes made in the current transaction.
ROLLBACK	Undoes all changes made since the last COMMIT or BEGIN.
SAVEPOINT	Creates a named restore point within a transaction.
ROLLBACK TO SAVEPOINT	Reverts only to the specified savepoint, not the very beginning.

Example — Safe Payroll Update Using TCL

```
BEGIN TRANSACTION;

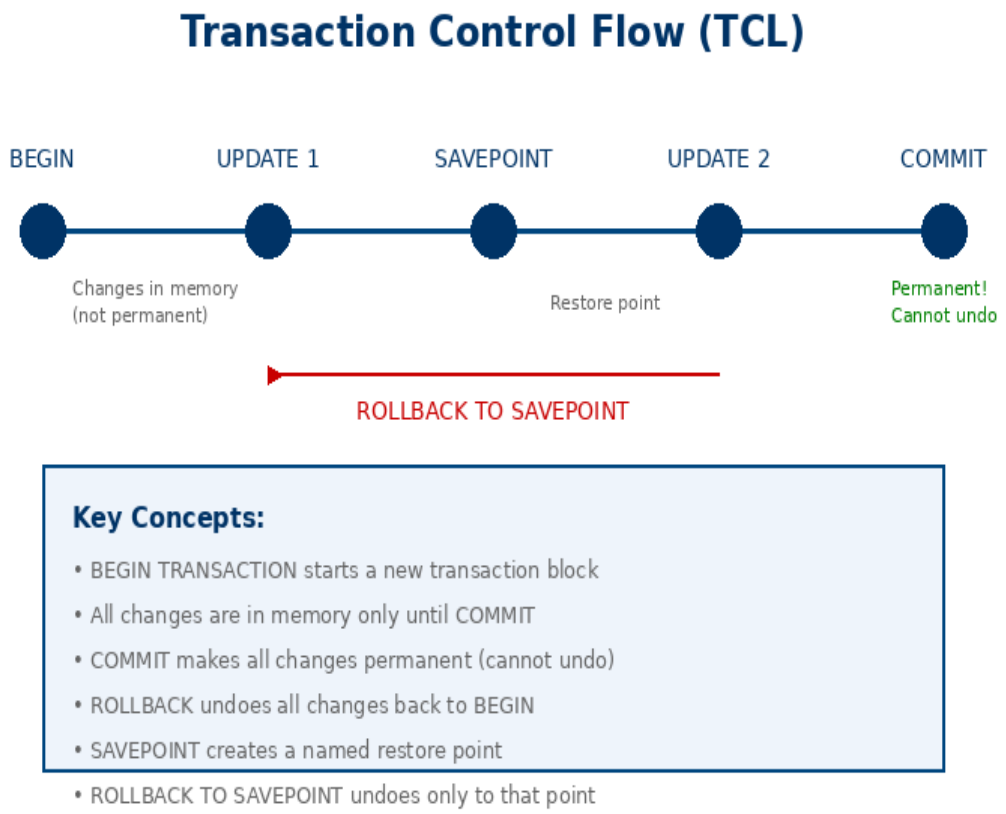
UPDATE employees
SET salary = salary * 1.05
WHERE department = 'Analytics';

SAVEPOINT after_raise;
```

```
-- If something goes wrong later, roll back only to this point
ROLLBACK TO SAVEPOINT after_raise;



COMMIT;
```



3.3 SQL Data Types

Every column in a SQL table must be assigned a data type. Choosing the correct type is fundamental to data integrity, storage efficiency, and query performance. The main categories are numeric, character/string, date & time, binary, and Boolean.

1. Numeric Data Types

Numeric types store numbers on which arithmetic operations can be performed. They divide into exact types (integers and fixed-point decimals) and approximate types (floating-point numbers).

Exact Numeric Data Types

Data Type	Range	Typical Use
TINYINT	0 to 255	Status flags, small enumerations.
SMALLINT	−32,768 to 32,767	Age, year, small counts.
INT	−2,147,483,648 to 2,147,483,647	General-purpose IDs, quantities.
BIGINT	−9.2 quintillion to 9.2 quintillion	Transaction IDs, large counters.
DECIMAL(p,s)	−10^38+1 to 10^38−1	Financial values needing exact precision.
NUMERIC(p,s)	−10^38+1 to 10^38−1	Similar to DECIMAL; used interchangeably.

Approximate Numeric Data Types

Data Type	Range	Use
FLOAT	−1.79E+308 to 1.79E+308	Scientific measurements. Avoid for monetary values.
REAL	−3.40E+38 to 3.40E+38	Less precise than FLOAT. Scientific data only.

Best Practice: Always use DECIMAL(p, s) — never FLOAT or REAL — for monetary amounts. Floating-point arithmetic introduces rounding errors that accumulate and cause discrepancies in financial reports.

2. Character and String Data Types

String types store text. The choice between fixed-length and variable-length types affects both storage and performance.

Data Type	Max Length	Use Case
CHAR(n)	8,000 characters (fixed)	Fixed-length strings: country codes (CHAR(2)), phone prefixes.

VARCHAR(n)	8,000 characters (variable)	Variable-length text: names, email addresses, descriptions.
TEXT	2.1 billion characters	Long-form content: notes, articles, log entries.
NCHAR(n)	4,000 Unicode characters (fixed)	Fixed-length multi-language text.
NVARCHAR(n)	4,000 Unicode characters (variable)	Variable-length multi-language text.

3. Date and Time Data Types

Data Type	What It Stores	Storage	Example
DATE	Year, month, day only	3 bytes	2025-06-15
TIME	Hour, minute, second	3 bytes	14:30:00
DATETIME	Date + time combined	8 bytes	2025-06-15 14:30:00

4. Other Important Data Types

Data Type	Description
BOOLEAN / BIT	Logical TRUE/FALSE. In SQLite, stored as INTEGER (1 = TRUE, 0 = FALSE).
BINARY / VARBINARY	Fixed or variable-length binary data: images, file content.
XML	Stores and queries XML-structured documents.
IMAGE	Stores binary data as images (up to 2.1 billion bytes).

3.4 Core DML Operations: SELECT, INSERT, UPDATE, DELETE

The SELECT Statement

SELECT is the workhorse of SQL for data analysis. It retrieves data from one or more tables and presents it in a structured result set. The full SQL execution order is: FROM → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT.

Basic SELECT Patterns

```
-- Retrieve specific columns
SELECT first_name, last_name, department
FROM employees;

-- Retrieve all columns
SELECT * FROM employees;

-- Filter rows with WHERE
SELECT first_name, salary
FROM employees
WHERE department = 'Analytics';

-- Sort results
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary DESC;

-- Remove duplicates
SELECT DISTINCT department FROM employees;

-- Group and filter with HAVING
SELECT country, COUNT(*) AS customer_count
FROM customers
GROUP BY country
HAVING COUNT(*) >= 3
```

```
ORDER BY customer_count DESC;
```

Key Concept: DISTINCT applies to the entire column list selected, not just one column. SELECT DISTINCT first_name, department returns unique combinations of both columns.

The INSERT Statement

INSERT adds new rows to a table. You can insert a single row, multiple rows in one statement, or data drawn from another table using a subquery.

```
-- Insert a single row (always specify column names explicitly)
INSERT INTO employees (first_name, last_name, department, hire_date, salary)
VALUES ('Sofia', 'Mendes', 'Analytics', '2025-03-01', 68000.00);


-- Insert multiple rows in one statement (more efficient)
INSERT INTO employees (first_name, last_name, department, hire_date, salary)
VALUES
    ('Liam',   'Chen',   'Engineering', '2025-03-15', 82000.00),
    ('Amara',  'Nwosu',  'Marketing',   '2025-04-01', 61000.00),
    ('Hiroshi','Tanaka', 'Analytics',   '2025-04-10', 71000.00);


-- Copy rows from another table (with condition)
INSERT INTO current_employees (first_name, last_name, department)
SELECT first_name, last_name, department
FROM archived_employees
WHERE rehire_date IS NOT NULL;
```

Best Practice: Always list column names explicitly in INSERT statements. Relying on column order breaks silently if the table schema later changes.

The UPDATE Statement

UPDATE modifies values in one or more columns for rows that match a condition. Omitting the WHERE clause updates every row in the table — always double-check before running UPDATE on production data.

```
-- Update a single column for one employee
UPDATE employees
SET salary = 75000.00
WHERE employee_id = 12;


-- Update multiple columns simultaneously
UPDATE employees
SET department = 'Data Science',
    salary     = salary * 1.10
WHERE department = 'Analytics' AND hire_date < '2023-01-01';


-- Safe pattern: preview before committing
BEGIN TRANSACTION;
UPDATE employees SET salary = 90000 WHERE department = 'Engineering';
SELECT * FROM employees WHERE department = 'Engineering';  -- verify
COMMIT;
```

The DELETE Statement

DELETE removes rows from a table based on a condition. Unlike TRUNCATE, DELETE is a DML operation that can be rolled back within a transaction. It retains the table structure, constraints, and indexes.

```
-- Delete a specific row
DELETE FROM employees WHERE employee_id = 5;


-- Delete multiple rows matching a condition
DELETE FROM employees
WHERE department = 'Temp' AND hire_date < '2020-01-01';
```

```
-- Safe deletion: wrap in transaction
BEGIN TRANSACTION;
DELETE FROM employees WHERE department = 'Discontinued';
SELECT * FROM employees WHERE department = 'Discontinued';  -- should return 0
COMMIT;
```

Command	Key Difference
DELETE	Removes specific rows using WHERE. DML — can be rolled back. Fires DELETE triggers. Slower on large tables.
TRUNCATE	Removes all rows instantly. DDL — cannot be rolled back in most RDBMS. Does not fire triggers. Much faster on large tables.

3.5 Schema Changes: ALTER, DROP, TRUNCATE

The ALTER TABLE Statement

ALTER TABLE modifies the structure of an existing table without losing the data already stored in it (unless you drop a column). Common use cases include adding new columns as business requirements evolve, changing data types, and renaming objects.

Operation	Syntax Example
Add a column	ALTER TABLE employees ADD COLUMN email VARCHAR(100);
Modify column size	ALTER TABLE employees MODIFY COLUMN email VARCHAR(150);
Rename a column	ALTER TABLE employees RENAME COLUMN email TO work_email;
Remove a column	ALTER TABLE employees DROP COLUMN work_email;
Rename the table	ALTER TABLE employees RENAME TO staff;

Example — Evolving a Student Table

```
-- Add AGE and COURSE columns
```

```
ALTER TABLE Student ADD (AGE NUMBER(3), COURSE VARCHAR(40));


-- Reduce COURSE column size
ALTER TABLE Student MODIFY COURSE VARCHAR(20);


-- Remove the COURSE column
ALTER TABLE Student DROP COLUMN COURSE;
```

DROP Statement

DROP permanently removes a database object and all its data. There is no undo once committed — always ensure you have a backup before dropping objects in a production environment.

```
-- Drop a table
DROP TABLE old_archive;


-- Safety check (MySQL / PostgreSQL)
DROP TABLE IF EXISTS old_archive;


-- Drop an entire database
DROP DATABASE legacy_system;
```

TRUNCATE Statement

TRUNCATE empties a table completely but preserves the table structure, column definitions, and constraints. It resets auto-increment counters and is significantly faster than DELETE on large tables because it does not log individual row deletions.

Feature	DROP	TRUNCATE
Data removed	Yes — completely	Yes — all rows
Table structure	Deleted permanently	Preserved for reuse

Can be rolled back	No (in most RDBMS)	Depends on RDBMS
Fires triggers	No	No (DELETE triggers not fired)
Speed on large tables	Slower	Much faster
When to use	Permanent removal of table or database	Quick wipe of all data, keeping the schema

3.6 SQL Clauses: WHERE, HAVING, ORDER BY, GROUP BY

WHERE Clause

WHERE filters individual rows before any grouping or aggregation takes place. It can use comparison operators, logical operators (AND, OR, NOT), and special operators (BETWEEN, IN, LIKE, IS NULL).

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
BETWEEN	Within an inclusive range
LIKE	Matches a text pattern
IN	Matches any value in a list
IS NULL	Tests for missing values

a-mahin

```
-- Single condition
SELECT * FROM employees WHERE department = 'Analytics';


-- Multiple conditions
SELECT * FROM employees
WHERE department = 'Analytics' AND salary > 65000;


SELECT * FROM employees
WHERE department = 'Analytics' OR department = 'Data Science';


-- Range, pattern, list, null
SELECT * FROM employees WHERE salary BETWEEN 60000 AND 80000;
SELECT * FROM employees WHERE last_name LIKE 'S%';
SELECT * FROM employees WHERE department IN ('Analytics', 'Engineering');
SELECT * FROM employees WHERE email IS NULL;
```

GROUP BY Clause

GROUP BY collapses multiple rows that share the same value in one or more columns into a single summary row. It is always used with an aggregate function (COUNT, SUM, AVG, MIN, MAX) to produce meaningful summaries.

```
-- Count employees per department
SELECT department, COUNT(*) AS headcount
FROM employees
GROUP BY department;


-- Average salary per department
SELECT department, ROUND(AVG(salary), 2) AS avg_salary
FROM employees
GROUP BY department
ORDER BY avg_salary DESC;
```

```
-- Group by multiple columns
SELECT department, YEAR(hire_date) AS hire_year, COUNT(*) AS new_hires
FROM employees
GROUP BY department, YEAR(hire_date)
ORDER BY department, hire_year;
```

HAVING Clause

HAVING filters groups after GROUP BY has been applied. It is the post-aggregation equivalent of WHERE. Think of it this way: WHERE asks "which rows?" while HAVING asks "which groups?"

```
-- Departments with more than 5 employees
SELECT department, COUNT(*) AS headcount
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;


-- Departments where average salary exceeds 70,000
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 70000
ORDER BY avg_salary DESC;


-- Combine WHERE (row filter) with HAVING (group filter)
SELECT department, COUNT(*) AS senior_count, AVG(salary) AS avg_sal
FROM employees
WHERE hire_date < '2021-01-01'        -- filter rows first
GROUP BY department
HAVING COUNT(*) >= 3                  -- then filter groups
ORDER BY senior_count DESC;
```

Clause	Key Difference
WHERE	Filters rows before grouping. Cannot reference aggregate functions. Used with SELECT, UPDATE, DELETE.
HAVING	Filters groups after aggregation. Can reference aggregate functions. Used only with SELECT (after GROUP BY).

ORDER BY Clause

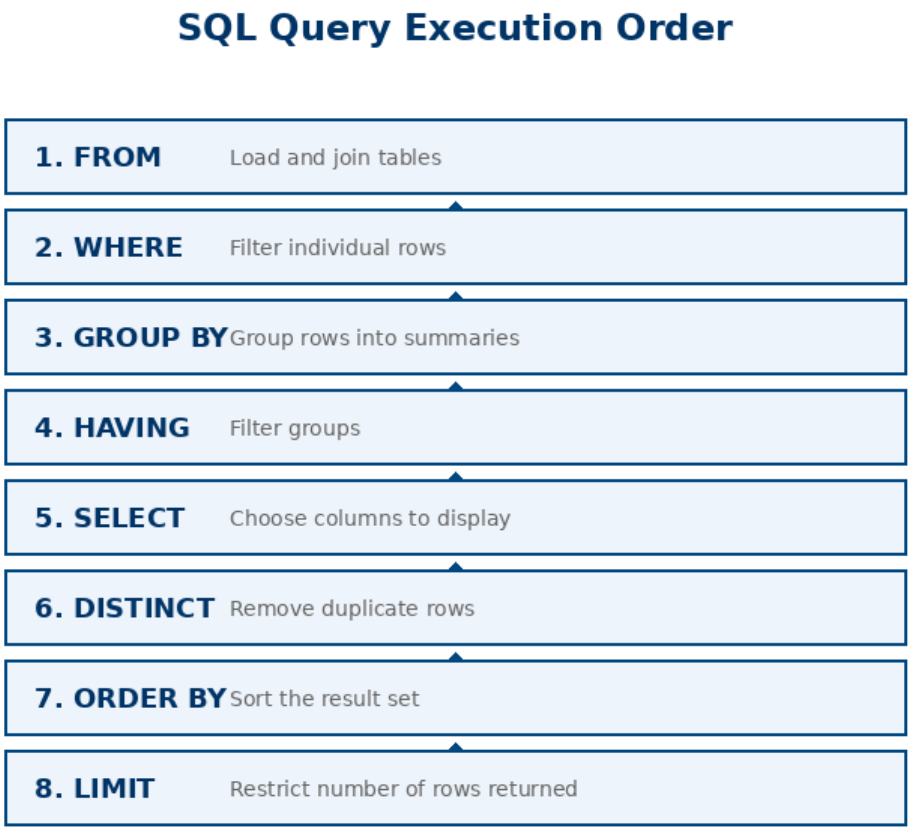
ORDER BY sorts the final result set. You can sort by one or more columns, mix ascending (ASC, the default) and descending (DESC) directions, and even sort by column position number.

```
-- Sort by salary descending
SELECT * FROM employees ORDER BY salary DESC;


-- Sort by multiple columns
SELECT * FROM employees ORDER BY department ASC, salary DESC;


-- Sort by column position (3rd column in SELECT list)
SELECT first_name, last_name, hire_date FROM employees ORDER BY 3;
```

SQL Execution Order: The database engine executes clauses in this specific order: FROM → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT. Understanding this order is critical for writing efficient queries and debugging errors.



nasrin-a

3.7 SQL Operators: BETWEEN, IN, LIKE, IS NULL

BETWEEN Operator

BETWEEN tests whether a value falls within a range, inclusive of both boundary values. It works with numbers, dates, and text.

Feature	BETWEEN	IN
Purpose	Tests if a value falls within a range (inclusive).	Tests if a value matches any item in a list.
Syntax	column BETWEEN value1 AND value2	column IN (value1, value2, ...)

Data types	Numbers, dates, text	Numbers, dates, text, subquery results
Exclusion	NOT BETWEEN value1 AND value2	NOT IN (value1, value2, ...)
Best for	Continuous ranges	Non-sequential discrete values

```
-- Numeric range
SELECT name, salary FROM employees
WHERE salary BETWEEN 60000 AND 80000;


-- Date range
SELECT order_id, order_date FROM orders
WHERE order_date BETWEEN '2025-01-01' AND '2025-03-31';


-- NOT BETWEEN
SELECT name FROM employees
WHERE salary NOT BETWEEN 60000 AND 80000;
```

IN Operator

IN checks whether a value matches any item in a list. It is a concise alternative to multiple OR conditions and can also accept a subquery as its list.

```
-- Equivalent to multiple OR conditions
SELECT * FROM employees
WHERE department IN ('Analytics', 'Engineering', 'Data Science');


-- NOT IN excludes listed values
SELECT * FROM employees
WHERE country NOT IN ('UK', 'US', 'CA');


-- IN with a subquery (dynamic list)
```

```
SELECT * FROM employees
WHERE department_id IN (
    SELECT department_id FROM departments WHERE location = 'London'
);
```

LIKE Operator

LIKE performs pattern matching on text columns using wildcard characters.

- % (percent) — matches zero or more characters of any kind.
- _ (underscore) — matches exactly one character.

Pattern	Matches	Example Match
'A%'	Starts with A	Ahmed, Anna, Analytics
'%son'	Ends with son	Johnson, Thompson
'%data%'	Contains "data" anywhere	Big data, data analyst
'_ohn'	Any single char then "ohn"	John, Bohn
'A_a%'	Starts A, any char, then a	Ana, Ala, Ayasha

```
SELECT * FROM employees WHERE last_name LIKE 'M%';
SELECT * FROM customers WHERE email LIKE '%@company.com';
SELECT * FROM products WHERE product_code LIKE '_____'; -- exactly 5 chars
```

IS NULL / IS NOT NULL

NULL represents missing, unknown, or inapplicable data. It is not the same as zero or an empty string. Because SQL uses three-valued logic (TRUE, FALSE, UNKNOWN), you must use IS NULL — not = NULL — to check for missing values.

```
-- Find rows with missing email
SELECT first_name, last_name FROM employees WHERE email IS NULL;
```

```
-- Find rows that have an email
SELECT first_name, last_name FROM employees WHERE email IS NOT NULL;


-- Count missing vs present values
SELECT
    COUNT(*)              AS total_rows,
    COUNT(email)          AS has_email,
    COUNT(*) - COUNT(email) AS missing_email
FROM employees;


-- Fill in missing values
UPDATE employees
SET email = 'unknown@company.com'
WHERE email IS NULL;
```

3.8 SQL Aggregate Functions

Aggregate functions compute a single result from a set of input values. They are the primary tools for summarising data — essential for reporting, KPI monitoring, and exploratory analysis. All aggregate functions (except COUNT(*)) ignore NULL values automatically.

Function	What It Returns
COUNT(*)	Total number of rows, including rows with NULL values.
COUNT(column)	Number of non-NULL values in the specified column.
COUNT(DISTINCT column)	Number of unique non-NULL values.
SUM(column)	Total of all non-NULL numeric values in the column.
AVG(column)	Arithmetic mean of non-NULL numeric values.
MIN(column)	Smallest non-NULL value (works on numbers, dates, and text).
MAX(column)	Largest non-NULL value (works on numbers, dates, and text).

COUNT()

COUNT is used to determine row counts and cardinality. The three forms behave differently when NULLs are present.

```
SELECT
    COUNT(*)                AS all_rows,          -- counts everything including NULLs
    COUNT(salary)           AS non_null_salaries,  -- skips NULL salary rows
    COUNT(DISTINCT department) AS unique_departments -- unique non-NULL values only
FROM employees;
```

SUM() and AVG()

```
SELECT
    department,
    SUM(salary)   AS total_salary,
    AVG(salary)   AS avg_salary
FROM employees
GROUP BY department;
```

MIN() and MAX()

```
SELECT
    department,
    MIN(hire_date) AS first_hire,
    MAX(hire_date) AS latest_hire,
    MAX(salary)    AS top_earner_salary,
    MIN(salary)    AS lowest_earner_salary
FROM employees
GROUP BY department;
```

Combining Multiple Aggregate Functions

```
-- Department salary dashboard in a single query
SELECT
    department,
    COUNT(*)                    AS headcount,
    ROUND(SUM(salary), 2)       AS total_payroll,
    ROUND(AVG(salary), 2)       AS avg_salary,
    MIN(salary)                 AS min_salary,
    MAX(salary)                 AS max_salary,
    MAX(salary) - MIN(salary)   AS salary_spread
FROM employees
GROUP BY department
HAVING COUNT(*) >= 2
ORDER BY total_payroll DESC;
```

Using HAVING with Aggregate Functions

The HAVING clause filters aggregated results. It works after GROUP BY, whereas WHERE works before grouping.

```
-- Products whose total cost exceeds 3.5
SELECT product_name, product_cost
FROM products
GROUP BY product_name, product_cost
HAVING SUM(product_cost) > 3.5
ORDER BY product_cost;


-- Departments with average salary above 55,000
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 55000;
```

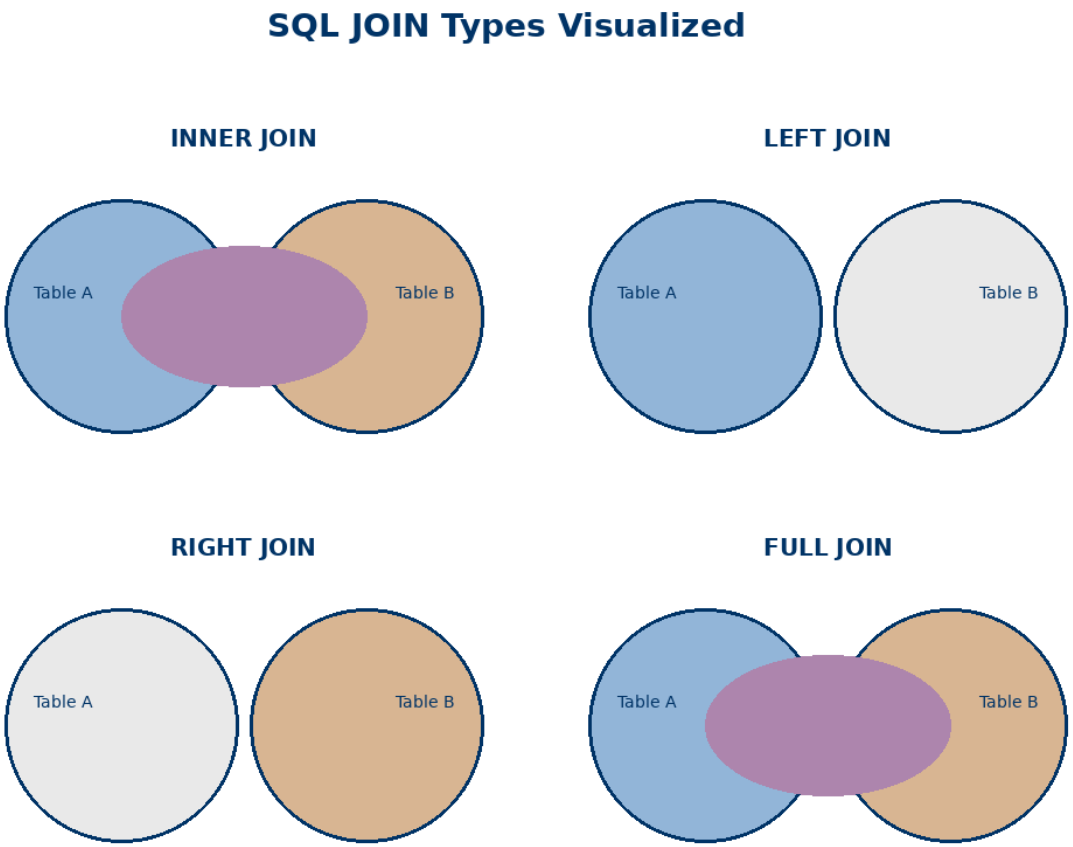
```
-- Customer IDs that appear at least 2 times in the login table
SELECT customer_id
FROM login
GROUP BY customer_id
HAVING COUNT(customer_id) >= 2;
```

3.9 SQL Joins

SQL Joins combine rows from two or more tables based on a related column — typically a primary key/foreign key relationship. Mastering joins is essential for almost every non-trivial data analysis task.

Join Type	Rows Returned
INNER JOIN	Only rows with matching values in both tables.
LEFT JOIN	All rows from the left table; NULLs for unmatched right-side rows.
RIGHT JOIN	All rows from the right table; NULLs for unmatched left-side rows.
FULL JOIN	All rows from both tables; NULLs where no match exists on either side.
NATURAL JOIN	Rows matched automatically on all identically named columns (use with caution).

The following diagram visualizes how each JOIN type determines which rows appear in the result set. Think of each circle as representing all the rows in one table. The shaded regions show which rows are included in the result.



Sample Tables for Join Examples

The examples below use a Student table and a StudentCourse table, linked by the ROLL_NO column.

1. INNER JOIN

INNER JOIN returns only the rows where a matching value exists in both tables. Non-matching rows from either table are excluded.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE

FROM Student

INNER JOIN StudentCourse

ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Note: JOIN and INNER JOIN are identical. You may use either keyword. INNER JOIN is more explicit and preferred in professional code.

2. LEFT JOIN (LEFT OUTER JOIN)

LEFT JOIN returns all rows from the left table, along with matching rows from the right table. Where there is no match, NULL values appear for the right-table columns.

```
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
-- Students with no enrolled course will appear with NULL in COURSE_ID
```

3. RIGHT JOIN (RIGHT OUTER JOIN)

RIGHT JOIN returns all rows from the right table, and matching rows from the left table. Where there is no match, NULL values appear for the left-table columns.

```
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
-- Courses with no enrolled student will appear with NULL in NAME
```

4. FULL JOIN (FULL OUTER JOIN)

FULL JOIN returns every row from both tables. Where there is no match on either side, the columns from the other table contain NULL. It is the union of LEFT and RIGHT joins.

```
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

```
-- MySQL does not support FULL JOIN; simulate with UNION:
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student LEFT JOIN StudentCourse ON Student.ROLL_NO = StudentCourse.ROLL_NO
UNION
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student RIGHT JOIN StudentCourse ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

5. NATURAL JOIN

A NATURAL JOIN automatically joins on all columns with the same name and compatible data types in both tables. While convenient, it can produce unexpected results if tables share column names that are not intended as join keys.

```
SELECT Emp_name, Dept_name
FROM Employee
NATURAL JOIN Department;
-- Joins automatically on all identically named columns
```

3.10 SQL Subqueries

A subquery (also called a nested query) is a SELECT statement embedded within another SQL statement. Subqueries allow you to break complex problems into composable parts — the inner query produces a result that the outer query then uses to filter, aggregate, or transform data.

Subquery Type	Description	Operators Used
Single-Row	Returns exactly one value.	=, >, <, >=, <=, <>
Multi-Row	Returns multiple values.	IN, NOT IN, ANY, ALL
Correlated	References the outer query row by row.	EXISTS, >, =, etc.
Derived Table	Appears in FROM clause as a temporary table.	Must be aliased

Types of Subqueries

Single-Row Subquery

Returns exactly 1 value

Use with: =, >, <, >=, <=

Example:

WHERE salary = (SELECT MAX...)

Multi-Row Subquery

Returns multiple values

Use with: IN, ANY, ALL

Example:

WHERE dept_id IN (SELECT...)

Correlated Subquery

References outer query

Executes once per row

Example:

WHERE salary > (SELECT AVG...
  WHERE dept = e.dept)

Derived Table

Subquery in FROM clause

Must be aliased

Example:

FROM (SELECT ...) AS alias

1. Single-Row Subquery

Returns exactly one value, used with single-value comparison operators.

```
-- Find employees earning the highest salary
SELECT first_name, last_name, salary
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);


-- Find employees earning above the overall average
SELECT first_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

2. Multi-Row Subquery

Returns multiple rows and must be used with set-based operators: IN, NOT IN, ANY, or ALL.

```
-- Find employees in departments located in London or Berlin
SELECT first_name, department_id
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE location IN ('London', 'Berlin')
);


-- Find employees earning more than ALL salaries in the Temp department
SELECT first_name, salary
FROM employees
WHERE salary > ALL (
    SELECT salary FROM employees WHERE department = 'Temp'
);
```

3. Correlated Subquery

A correlated subquery references a column from the outer query. It is executed once for each row the outer query processes — powerful but potentially slower on large datasets.

```
-- Find employees earning above their own department's average
SELECT e.first_name, e.department_id, e.salary
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id   -- references outer query
```

```
);
```

4. Subquery in the FROM Clause (Derived Table)

When a subquery appears in the FROM clause, it acts as a temporary table (derived table). It must be given an alias.

```
-- Find departments above average headcount
SELECT dept_summary.department, dept_summary.headcount
FROM (
    SELECT department, COUNT(*) AS headcount
    FROM employees
    GROUP BY department
) AS dept_summary
WHERE dept_summary.headcount > (
    SELECT AVG(cnt)
    FROM (SELECT COUNT(*) AS cnt FROM employees GROUP BY department) AS counts
);
```

5. Subqueries with DML Statements

Subqueries are equally powerful inside INSERT, UPDATE, and DELETE statements.

```
-- DELETE: remove employees from discontinued departments
DELETE FROM employees
WHERE department_id IN (
    SELECT department_id FROM departments WHERE status = 'Discontinued'
);


-- UPDATE: give a raise to employees in top-rated departments
UPDATE employees
SET salary = salary * 1.08
WHERE department_id IN (
```

```
    SELECT department_id FROM department_performance WHERE rating = 'Excellent'
);



-- INSERT: archive inactive employees
INSERT INTO employee_archive (first_name, last_name, department_id, salary)
SELECT first_name, last_name, department_id, salary
FROM employees
WHERE last_login < DATE_SUB(CURDATE(), INTERVAL 1 YEAR);
```

Situation	Prefer
Need columns from multiple tables in the result	JOIN
Filtering based on existence or non-existence in another table	Subquery with IN / NOT IN
Filter depends on an aggregate from the same or another table	Subquery (WHERE or HAVING)
Need a temporary summarised view inside a larger query	Derived table (subquery in FROM)
Performance is critical on very large tables	JOIN (usually optimised better by query planner)

3.11 CASE Statements

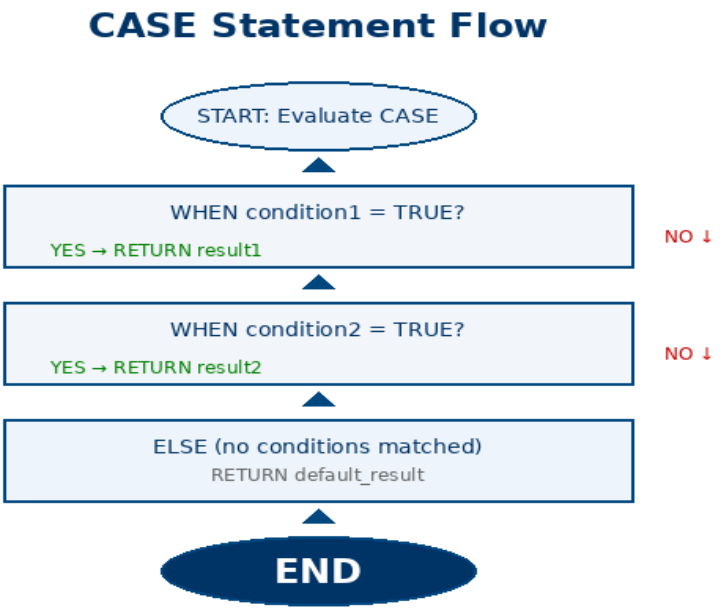
The CASE statement adds conditional logic inside a SQL query — similar to IF/ELSE in programming languages. It is used to create new calculated columns, bucket data into categories, pivot rows into columns, and apply different calculations based on conditions. CASE can appear in SELECT, WHERE, ORDER BY, and inside aggregate functions.

Syntax

```
-- Searched CASE (most common form)
CASE
```

```
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ELSE default_result
END


-- Simple CASE (compares one expression to multiple values)
CASE expression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    ELSE default_result
END
```



1. Categorising / Bucketing Data

One of the most common uses is to group numeric values into labelled bands — salary grades, age groups, score tiers, etc.

```
SELECT
```

```
    first_name, salary,
    CASE
        WHEN salary < 40000                    THEN 'Entry Level'
        WHEN salary BETWEEN 40000 AND 70000     THEN 'Mid Level'
        WHEN salary BETWEEN 70001 AND 100000    THEN 'Senior Level'
        ELSE 'Executive'
    END AS salary_grade
FROM employees;
```

2. CASE Inside Aggregate Functions

Combining CASE with COUNT or SUM lets you build conditional counts and pivot summaries in a single query — a pattern used constantly in reporting.

```
-- Count employees per grade without multiple queries
SELECT
    COUNT(CASE WHEN salary < 40000 THEN 1 END)              AS entry_level,
    COUNT(CASE WHEN salary BETWEEN 40000 AND 70000 THEN 1 END) AS mid_level,
    COUNT(CASE WHEN salary > 70000 THEN 1 END)              AS senior_level
FROM employees;


-- Sum revenue by region (pivot-style)
SELECT
    SUM(CASE WHEN region = 'North' THEN revenue ELSE 0 END) AS north_revenue,
    SUM(CASE WHEN region = 'South' THEN revenue ELSE 0 END) AS south_revenue,
    SUM(CASE WHEN region = 'East'  THEN revenue ELSE 0 END) AS east_revenue
FROM sales;
```

3. CASE in ORDER BY

```
-- Custom sort order: Active first, then Pending, then Closed
SELECT order_id, status FROM orders
ORDER BY
    CASE status
```

```
    WHEN 'Active'  THEN 1
    WHEN 'Pending' THEN 2
    WHEN 'Closed'  THEN 3
    ELSE 4
  END;
```

4. CASE in WHERE Clause

```
-- Different salary threshold per department
SELECT first_name, department, salary FROM employees
WHERE salary > CASE
  WHEN department = 'Engineering' THEN 80000
  WHEN department = 'Analytics'   THEN 60000
  ELSE 50000
END;
```

3.12 NULL Handling Functions: COALESCE, IFNULL, NULLIF

NULL values appear in almost every real-world dataset. SQL provides dedicated functions to handle them gracefully — substituting defaults, comparing values safely, and avoiding NULL propagation in calculations.

Function	What It Does	Database Support
COALESCE(v1, v2, ...)	Returns the first non-NULL value from the list.	All major RDBMS (ANSI standard)
IFNULL(v1, v2)	Returns v2 if v1 is NULL; otherwise returns v1.	MySQL, MariaDB
ISNULL(v1, v2)	Returns v2 if v1 is NULL; otherwise returns v1.	SQL Server
NVL(v1, v2)	Returns v2 if v1 is NULL; otherwise returns v1.	Oracle
NULLIF(v1, v2)	Returns NULL if v1 = v2; otherwise returns v1.	All major RDBMS (ANSI standard)

COALESCE — Substitute a Default Value

```
-- Replace NULL email with a placeholder
SELECT first_name,
    COALESCE(email, 'No email on file') AS contact_email
FROM employees;


-- Use the first available phone number
SELECT customer_name,
    COALESCE(mobile_phone, home_phone, work_phone, 'No phone') AS best_contact
FROM customers;


-- Replace NULL salary with 0 in sum calculations
SELECT department,
    SUM(COALESCE(salary, 0)) AS total_payroll
FROM employees
GROUP BY department;
```

NULLIF — Safe Division and Avoiding Zero Errors

NULLIF(a, b) returns NULL when a equals b. Its most important use is preventing division-by-zero errors: wrap the denominator in NULLIF(denominator, 0) so the result is NULL instead of a runtime error.

```
-- Avoid division by zero
SELECT product_name, total_revenue, units_sold,
    total_revenue / NULLIF(units_sold, 0) AS revenue_per_unit
FROM product_sales;


-- Treat empty strings the same as NULL, then apply a default
SELECT COALESCE(NULLIF(TRIM(notes), ''), 'No notes provided') AS clean_notes
FROM tickets;
```

3.13 String Functions

String functions are indispensable for data cleaning and preparation. Real-world data arrives with inconsistent casing, extra whitespace, embedded codes that need extracting, and fields that need combining. SQL string functions handle all of these tasks directly in the query.

Function	Description	Example Result
UPPER(str)	Converts to uppercase	UPPER('hello') = 'HELLO'
LOWER(str)	Converts to lowercase	LOWER('HELLO') = 'hello'
TRIM(str)	Removes leading and trailing spaces	TRIM(' hi ') = 'hi'
LTRIM(str)	Removes leading spaces only	LTRIM(' hi') = 'hi'
RTRIM(str)	Removes trailing spaces only	RTRIM('hi ') = 'hi'
LENGTH(str)	Returns the number of characters	LENGTH('hello') = 5
CONCAT(a,b,...)	Joins strings together	CONCAT('Ana',' ','Lima') = 'Ana Lima'
SUBSTRING(str,pos,len)	Extracts part of a string	SUBSTRING('Analytics',1,5) = 'Analy'
LEFT(str, n)	Returns the first n characters	LEFT('Analytics', 4) = 'Anal'
RIGHT(str, n)	Returns the last n characters	RIGHT('Analytics', 4) = 'tics'
REPLACE(str,old,new)	Substitutes all occurrences of a substring	REPLACE('AA-100','AA','BB') = 'BB-100'
INSTR(str, sub)	Returns position of first occurrence	INSTR('hello world','world') = 7
LPAD(str, n, pad)	Left-pads string to length n	LPAD('5', 4, '0') = '0005'

RPAD(str, n, pad)	Right-pads string to length n	RPAD('5', 4, '0') = '5000'

Data Cleaning Examples

```
-- Standardise casing and remove extra spaces
SELECT CONCAT(
    UPPER(LEFT(TRIM(first_name), 1)),
    LOWER(SUBSTRING(TRIM(first_name), 2))
) AS clean_first_name
FROM customers;


-- Extract email domain
SELECT email,
       SUBSTRING(email, INSTR(email, '@') + 1) AS domain
FROM customers;


-- Remove dashes and spaces from phone numbers
SELECT REPLACE(REPLACE(phone_number, '-', ''), ' ', '') AS clean_phone
FROM customers;


-- Pad product codes to exactly 8 characters
SELECT LPAD(product_code, 8, '0') AS padded_code FROM products;


-- Case-insensitive search in WHERE
SELECT * FROM customers
WHERE LOWER(email) LIKE '%@gmail.com';
```

3.14 Date and Time Functions

Time-series analysis is one of the most common tasks in data analytics — tracking trends, comparing periods, calculating durations, and grouping data by day, week, month, or year. SQL date functions make these operations straightforward.

Function	Description	Example / Result
NOW()	Current date and time	2025-06-15 14:30:00
CURDATE()	Current date only	2025-06-15
YEAR(date)	Extracts the year	YEAR('2025-06-15') = 2025
MONTH(date)	Extracts the month number	MONTH('2025-06-15') = 6
DAY(date)	Extracts the day of month	DAY('2025-06-15') = 15
QUARTER(date)	Returns quarter (1-4)	QUARTER('2025-06-15') = 2
DAYNAME(date)	Returns day name	DAYNAME('2025-06-15') = Sunday
MONTHNAME(date)	Returns month name	MONTHNAME('2025-06-15') = June
DATEDIFF(d1, d2)	Days between two dates	DATEDIFF('2025-06-15','2025-01-01') = 165
DATE_ADD(date, INTERVAL n unit)	Adds time to a date	DATE_ADD('2025-01-01', INTERVAL 3 MONTH)
DATE_SUB(date, INTERVAL n unit)	Subtracts time from a date	DATE_SUB(NOW(), INTERVAL 1 YEAR)
DATE_FORMAT(date, fmt)	Formats a date as string	DATE_FORMAT(NOW(), '%Y-%m') = '2025-06'
TIMESTAMPDIFF(unit, d1, d2)	Difference in specified unit	TIMESTAMPDIFF(MONTH, hire_date, NOW())
EXTRACT(unit FROM date)	ANSI standard extraction	EXTRACT(YEAR FROM hire_date)

Common Time-Series Patterns

```
-- Monthly revenue trend
SELECT
    YEAR(order_date)          AS year,
    MONTH(order_date)         AS month,
    MONTHNAME(order_date)     AS month_name,
    COUNT(*)                  AS total_orders,
    ROUND(SUM(order_amount), 2) AS total_revenue
FROM orders
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY year, month;


-- Quarterly sales summary
SELECT YEAR(order_date) AS year, QUARTER(order_date) AS quarter,
       SUM(order_amount) AS quarterly_revenue
FROM orders
GROUP BY YEAR(order_date), QUARTER(order_date);


-- Last 30 days filter
SELECT * FROM orders
WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);


-- Employee tenure in years
SELECT first_name, hire_date,
       TIMESTAMPDIFF(YEAR,  hire_date, CURDATE()) AS years_employed,
       TIMESTAMPDIFF(MONTH, hire_date, CURDATE()) AS months_employed
FROM employees;


-- Days since last purchase per customer
SELECT customer_id,
       MAX(order_date)                AS last_order_date,
       DATEDIFF(CURDATE(), MAX(order_date))   AS days_since_purchase
```

```
FROM orders
GROUP BY customer_id
ORDER BY days_since_purchase DESC;
```

3.15 UNION, UNION ALL, INTERSECT, EXCEPT

Set operators combine the results of two or more SELECT statements into a single result set. They are used to merge data from different tables or query conditions, find overlaps between datasets, or identify differences between them.

Rules for All Set Operators

- Both SELECT statements must return the same number of columns.
- Corresponding columns must have compatible data types.
- Column names in the result are taken from the first SELECT.
- ORDER BY can only appear once, at the very end of the combined query.

Operator	What It Returns	Duplicates
UNION	All rows from both queries combined.	Removed (like DISTINCT)
UNION ALL	All rows from both queries combined.	Kept — faster than UNION
INTERSECT	Only rows that appear in BOTH queries.	Removed
EXCEPT / MINUS	Rows in the first query but NOT in the second.	Removed

UNION and UNION ALL

```
-- Combine current and archived customers (removes duplicates)
SELECT customer_id, name, email FROM current_customers
UNION
SELECT customer_id, name, email FROM archived_customers;


-- UNION ALL: faster — no deduplication step
SELECT product_id, product_name FROM products_2024
UNION ALL
```

```
SELECT product_id, product_name FROM products_2025;
```

INTERSECT — Rows Present in Both Results

```
-- Customers who ordered in BOTH January AND February
SELECT customer_id FROM orders WHERE MONTH(order_date) = 1
INTERSECT
SELECT customer_id FROM orders WHERE MONTH(order_date) = 2;
```

EXCEPT — Rows in First but Not Second

```
-- Customers who ordered in January but NOT February
SELECT customer_id FROM orders WHERE MONTH(order_date) = 1
EXCEPT
SELECT customer_id FROM orders WHERE MONTH(order_date) = 2;


-- MySQL equivalent (does not support EXCEPT natively):
SELECT DISTINCT customer_id FROM orders WHERE MONTH(order_date) = 1
AND customer_id NOT IN (
    SELECT customer_id FROM orders WHERE MONTH(order_date) = 2
);
```

Tip: Use UNION ALL instead of UNION whenever you know duplicates cannot exist or are acceptable — it avoids the extra deduplication pass and is significantly faster on large datasets.

3.16 Common Table Expressions (CTEs)

A Common Table Expression (CTE) is a named temporary result set defined with the WITH keyword at the top of a query. CTEs make complex queries more readable and maintainable by breaking them into logical, named steps — like creating a series of intermediate views within a single statement.

Syntax

```
-- Single CTE
WITH cte_name AS (
    SELECT ... FROM ... WHERE ...
)
SELECT * FROM cte_name;


-- Multiple CTEs chained with commas
WITH
    cte1 AS (SELECT ... FROM ...),
    cte2 AS (SELECT ... FROM cte1 WHERE ...)
SELECT * FROM cte2;
```

CTEs	Subqueries
Named — readable and referenceable multiple times in the same query.	Anonymous — must be repeated every time it is used.
Defined at the top — logic flows top to bottom.	Nested inside — harder to read in complex cases.
Can reference earlier CTEs in the same WITH block.	Cannot reference sibling subqueries.
Support RECURSIVE queries for hierarchies and trees.	Cannot be recursive.

Common Table Expression (CTE) Structure

```
WITH
  high_value_customers AS (
    SELECT customer_id, SUM(amount) AS total
    FROM orders
    GROUP BY customer_id
    HAVING SUM(amount) > 1000
  ),
  recent_orders AS (
    SELECT customer_id, MAX(order_date) AS last_order
    FROM orders
    GROUP BY customer_id
  )
SELECT c.name, hv.total, ro.last_order
FROM customers c
JOIN high_value_customers hv ON c.id = hv.customer_id
JOIN recent_orders ro ON c.id = ro.customer_id;
```

CTE #1: Named query
Can reference in main query

CTE #2: Can reference
other CTEs defined above

Main Query: Uses both
CTEs like regular tables

Benefits vs Subqueries:
- ✓ More readable — named steps flow top to bottom
- ✓ Reusable — reference the same CTE multiple times
- ✓ Chainable — each CTE can build on previous ones

1. Simplifying a Multi-Step Query

```
-- Step 1: Identify high-value customers (spend > 1000)
-- Step 2: Find their most recent order date
-- Step 3: Return combined results


WITH
high_value AS (
   SELECT customer_id, SUM(order_amount) AS total_spend
   FROM orders
   GROUP BY customer_id
   HAVING SUM(order_amount) > 1000
),
last_order AS (
```

```
    SELECT customer_id, MAX(order_date) AS last_order_date
    FROM orders
    GROUP BY customer_id
)
SELECT c.customer_name, hv.total_spend, lo.last_order_date
FROM customers c
JOIN high_value hv ON c.customer_id = hv.customer_id
JOIN last_order  lo ON c.customer_id = lo.customer_id
ORDER BY hv.total_spend DESC;
```

2. Reusing the Same Calculation

```
-- Calculate department averages once, reference twice
WITH dept_avg AS (
    SELECT department, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department
)
SELECT e.first_name, e.department, e.salary,
       da.avg_salary                      AS dept_avg,
       e.salary - da.avg_salary           AS variance,
       ROUND(e.salary / da.avg_salary * 100, 1) AS pct_of_avg
FROM employees e
JOIN dept_avg da ON e.department = da.department
ORDER BY variance DESC;
```

3. Recursive CTE — Traversing Hierarchies

Recursive CTEs traverse hierarchical data such as org charts, category trees, and bill-of-materials structures. They consist of an anchor member (the starting point) unioned with a recursive member (the self-referencing step).

```
-- Traverse an org chart downward from the CEO
WITH RECURSIVE org_chart AS (
    -- Anchor: top-level (no manager)
```

```
    SELECT employee_id, first_name, manager_id, 0 AS level
    FROM employees WHERE manager_id IS NULL

    UNION ALL

    -- Recursive: find direct reports at each level
    SELECT e.employee_id, e.first_name, e.manager_id, oc.level + 1
    FROM employees e
    JOIN org_chart oc ON e.manager_id = oc.employee_id
)
SELECT level, first_name, employee_id
FROM org_chart
ORDER BY level, first_name;
```

3.17 Window Functions

Window functions perform calculations across a set of rows related to the current row — without collapsing those rows into a single result the way GROUP BY does. They are among the most powerful tools in SQL for data analysis, enabling running totals, rankings, period-over-period comparisons, and moving averages in a single query.

Core Syntax

```
function_name() OVER (
    [PARTITION BY column1, column2, ...]   -- divide into independent groups
    [ORDER BY column3 ASC|DESC]            -- ordering within each group
    [ROWS|RANGE BETWEEN ... AND ...]       -- define the window frame
)
```

- PARTITION BY — splits rows into independent groups (like GROUP BY but rows are kept). Omit to treat the entire result set as one window.
- ORDER BY — sets row order within each partition. Required for ranking and running-total functions.
- Window Frame — defines which rows are included (e.g., all preceding rows, last 3 rows).

Window Functions: PARTITION BY Concept

Sample Data: Employees Table

Name	Dept	Salary	Rank
Ana	Sales	75000	**1**
Bob	Sales	68000	**2**
Carl	Sales	62000	**3**
Diana	Tech	95000	**1**
Eve	Tech	88000	**2**
Frank	Tech	81000	**3**

PARTITION BY Dept:

- Creates independent ranking within each department
- Rank resets to 1 when department changes

Without PARTITION BY:

- Diana would be rank 1, Ana rank 2, Eve rank 3, etc.
- No independent grouping — one continuous ranking across all rows

Category	Function	Description
Ranking	ROW_NUMBER()	Unique sequential number per partition. No ties.
Ranking	RANK()	Same rank for ties; skips numbers after ties (1,1,3).
Ranking	DENSE_RANK()	Same rank for ties; no gaps (1,1,2).
Ranking	NTILE(n)	Divides rows into n equal buckets (quartiles, deciles).
Ranking	PERCENT_RANK()	Relative rank as a percentage from 0 to 1.
Value	LAG(col, n)	Value from n rows before the current row.
Value	LEAD(col, n)	Value from n rows after the current row.
Value	FIRST_VALUE(col)	First value in the window frame.
Value	LAST_VALUE(col)	Last value in the window frame.

Aggregate	SUM() OVER()	Running or partitioned total.
Aggregate	AVG() OVER()	Running or partitioned average.
Aggregate	COUNT() OVER()	Running or partitioned row count.
Aggregate	MIN() / MAX() OVER()	Running or partitioned min/max value.

1. Ranking Functions

```
-- Rank employees by salary within each department
SELECT first_name, department, salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS row_num,
    RANK()       OVER (PARTITION BY department ORDER BY salary DESC) AS rank_pos,
    DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS dense_rank
FROM employees;


-- Top earner per department (using CTE + ROW_NUMBER)
WITH ranked AS (
    SELECT first_name, department, salary,
           ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rn
    FROM employees
)
SELECT first_name, department, salary FROM ranked WHERE rn = 1;
```

2. Running Totals and Cumulative Aggregates

```
SELECT order_date, daily_revenue,
    SUM(daily_revenue) OVER (ORDER BY order_date) AS running_total,
    AVG(daily_revenue) OVER (ORDER BY order_date) AS running_avg,
    COUNT(*)           OVER (ORDER BY order_date) AS running_count
FROM daily_sales;


-- Running total reset per month
SELECT order_date, MONTH(order_date) AS month, daily_revenue,
```

```
    SUM(daily_revenue) OVER (
        PARTITION BY MONTH(order_date)
        ORDER BY order_date
    ) AS monthly_running_total
FROM daily_sales;
```

3. LAG and LEAD — Period-over-Period Comparisons

```
-- Month-over-month revenue change and percentage growth
SELECT order_month, total_revenue,
    LAG(total_revenue, 1) OVER (ORDER BY order_month) AS prev_month,
    total_revenue - LAG(total_revenue, 1) OVER (ORDER BY order_month) AS mom_change,
    ROUND(
        (total_revenue - LAG(total_revenue, 1) OVER (ORDER BY order_month))
        / NULLIF(LAG(total_revenue, 1) OVER (ORDER BY order_month), 0) * 100, 2
    ) AS mom_pct_change
FROM monthly_revenue
ORDER BY order_month;
```

4. NTILE — Quartiles and Percentile Tiers

```
-- Divide customers into spend quartiles
SELECT customer_name, total_spend,
    CASE NTILE(4) OVER (ORDER BY total_spend DESC)
        WHEN 1 THEN 'Top 25%'
        WHEN 2 THEN 'Upper Middle'
        WHEN 3 THEN 'Lower Middle'
        WHEN 4 THEN 'Bottom 25%'
    END AS spend_tier
FROM customer_totals;
```

5. Moving Averages — Window Frames

The ROWS BETWEEN clause defines exactly which rows are included in the calculation, enabling rolling window aggregates such as 7-day moving averages.

```
-- 7-day moving average of daily sales
SELECT sale_date, daily_amount,
    ROUND(
        AVG(daily_amount) OVER (
            ORDER BY sale_date
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
        ), 2
    ) AS moving_avg_7_day
FROM daily_sales;


-- Common frame definitions:
-- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW  -> cumulative from start
-- ROWS BETWEEN 6 PRECEDING AND CURRENT ROW          -> rolling 7-row window
-- ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING          -> centred 3-row window
```

3.18 Views, Indexes, and Query Performance

SQL Views

A view is a saved SELECT query stored in the database as a named object. When queried, the database executes the underlying SELECT and returns the result. Views do not store data themselves — they are virtual tables that simplify complex logic, enforce security, and ensure consistent business definitions across reports.

```
-- Create a view for the analytics team (hides salary column)
CREATE VIEW employee_directory AS
SELECT employee_id, first_name, last_name, department, hire_date
FROM employees;


-- Query a view exactly like a table
```

```
SELECT * FROM employee_directory WHERE department = 'Analytics';


-- View encapsulating a complex join
CREATE VIEW sales_summary AS
SELECT o.order_id, c.customer_name, p.product_name,
    o.quantity, o.quantity * p.price AS line_total, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN products  p ON o.product_id  = p.product_id;


-- Update or replace a view
CREATE OR REPLACE VIEW employee_directory AS
SELECT employee_id, first_name, last_name, department, hire_date, email
FROM employees;


-- Remove a view
DROP VIEW IF EXISTS employee_directory;
```

Indexes

An index is a data structure that allows the database engine to find rows much faster — similar to an index at the back of a book. Without an index, the engine must scan every row (a full table scan). With an index on the queried column, it jumps directly to the relevant rows.

Index Type	Description
Single-column	Index on one column — useful when that column appears frequently in WHERE or JOIN conditions.
Composite (Multi-column)	Index on two or more columns — useful for queries filtering or sorting on multiple columns together.
Unique	Enforces uniqueness — automatically created with PRIMARY KEY and UNIQUE constraints.

Full-Text	Optimised for text search using MATCH...AGAINST on TEXT and VARCHAR columns.

```
-- Create an index on a frequently queried column
CREATE INDEX idx_department ON employees(department);

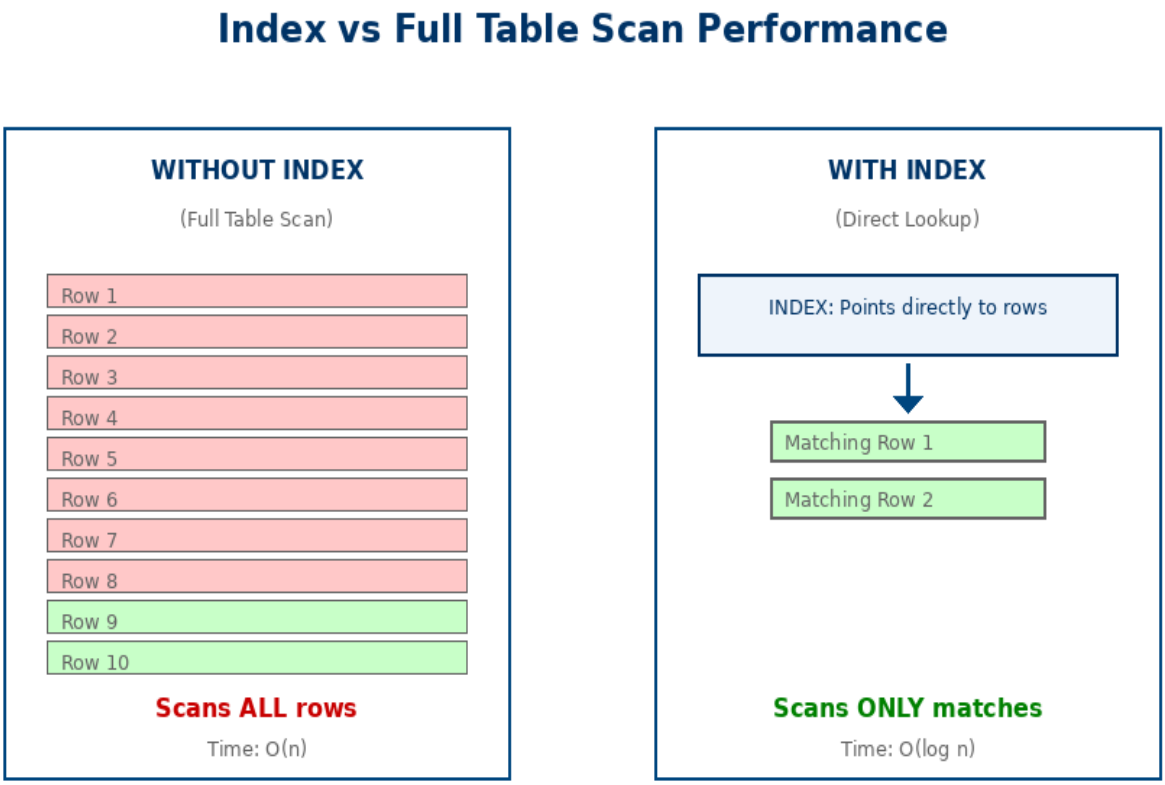

-- Composite index for queries filtering on both columns
CREATE INDEX idx_dept_salary ON employees(department, salary);


-- Unique index
CREATE UNIQUE INDEX idx_email ON employees(email);


-- View existing indexes (MySQL)
SHOW INDEX FROM employees;


-- Drop an index
DROP INDEX idx_department ON employees;
```

When to Index: Create indexes on columns used in WHERE, JOIN, and ORDER BY. Avoid over-indexing — each index speeds up reads but slows INSERT, UPDATE, and DELETE because indexes must also be updated.



Query Performance with EXPLAIN

EXPLAIN shows the query execution plan — how the database intends to retrieve the data. Use it to identify slow full table scans, missing indexes, and inefficient join strategies.

```
-- See the execution plan (MySQL)
EXPLAIN SELECT * FROM employees WHERE department = 'Analytics';


-- Key EXPLAIN output columns:
-- type:      ALL = full scan (slow) | ref/eq_ref = index used (fast)
-- key:       Index actually used (NULL means no index)
-- rows:      Estimated rows scanned
-- Extra:     'Using index', 'Using where', 'Using filesort', etc.


-- PostgreSQL: with actual timing
```

```
EXPLAIN ANALYZE SELECT * FROM employees WHERE department = 'Analytics';
```

Query Optimisation Best Practices

Practice	Why It Matters
Avoid SELECT * in production	Fetches unnecessary columns; increases network and memory usage.
Filter early with WHERE, not HAVING	WHERE reduces rows before aggregation; cheaper than filtering after.
Index JOIN and WHERE columns	Prevents full table scans on large tables.
Avoid functions on indexed columns in WHERE	WHERE YEAR(hire_date)=2025 prevents index use. Use a date range instead.
Use EXISTS for existence checks	EXISTS (SELECT 1 FROM ...) is faster than COUNT(*) > 0.
Prefer JOINs over correlated subqueries	Correlated subqueries run once per outer row; JOINs are typically better optimised.
Use UNION ALL over UNION when possible	UNION adds a deduplication pass — UNION ALL skips it and is faster.
Use LIMIT when exploring data	Prevents accidentally returning millions of rows during analysis.

Practice Exercises

Complete the following exercises using any SQL environment (MySQL Workbench, PostgreSQL, SQLite, or an online sandbox such as db-fiddle.com):

1. Create a retail_db database with customers, products, and orders tables. Insert at least 10 customers, 8 products, and 20 orders.
2. Write a query returning each customer's total spend (quantity x price) and label them as High, Medium, or Low value using CASE.
3. Find customers who ordered in January but not February using EXCEPT (or NOT IN for MySQL).
4. Use a CTE to find the top-spending customer in each product category.
5. Write a window function query showing monthly revenue with a running total and month-over-month percentage change using LAG.
6. Use NTILE(4) to divide customers into spend quartiles and count how many fall into each tier.
7. Create a view called active_customers_summary showing each customer's name, total orders, total spend, and days since last order.
8. Use string functions to clean a customer table: trim whitespace, standardise email to lowercase, and extract the email domain into a separate column.
9. Write a query using COALESCE to replace NULL phone numbers with "Not Provided" and NULL emails with "Unknown".
10. Run EXPLAIN on your most complex query and identify whether indexes are being used. Create an appropriate index and compare the plan.

nasrin-a-mahin

UNIT IV – Statistics for Data Analysis

4.1 Introduction to Statistics

Statistics is the science of collecting, analyzing, interpreting, and presenting data to uncover patterns, test hypotheses, and make informed decisions. It provides the mathematical foundation for data analysis, allowing analysts to summarise large datasets, quantify uncertainty, identify relationships between variables, and draw reliable conclusions from sample data about entire populations.

In data analytics, statistics transforms raw data into actionable insights. Every business report, A/B test, forecasting model, and machine learning algorithm relies on statistical principles. Understanding statistics is not optional for a data analyst — it is the core discipline that separates rigorous analysis from guesswork.

Why Statistics Matters in Data Analysis

- Summarisation — Condense millions of data points into meaningful metrics: mean revenue per customer, median order value, variance in delivery times.
- Inference — Make predictions about entire populations based on representative samples, avoiding the need to survey or measure every individual.
- Hypothesis Testing — Rigorously test claims ("Does this marketing campaign increase sales?") rather than relying on intuition.
- Uncertainty Quantification — Express confidence in results with p-values, confidence intervals, and standard errors.
- Predictive Modelling — Build regression models, classification algorithms, and time-series forecasts grounded in probability theory.

Descriptive vs Inferential Statistics

Statistics is divided into two broad categories, each serving a distinct purpose in the analysis workflow.

Type	Purpose	Examples
Descriptive Statistics	Summarize and describe data you already have. No generalization beyond the dataset.	Mean salary, median age, variance in test scores, frequency tables, bar charts
Inferential Statistics	Draw conclusions about a population based on a sample. Involves uncertainty and probability.	Hypothesis tests, confidence intervals, regression analysis, A/B testing, forecasting

yasrin-a-mahin

Key Concept: Descriptive statistics tell you what happened. Inferential statistics tell you what is likely to be true beyond your sample, and with what confidence.

4.2 Types of Data

Before applying any statistical method, you must understand what type of data you are working with. The data type determines which summary statistics, visualizations, and tests are appropriate.

1. Qualitative (Categorical) Data

Qualitative data describes categories or groups. It cannot be measured numerically in a meaningful way — only counted and categorized.

Sub-Type	Definition	Examples
Nominal	Categories with no inherent order.	Gender, country, product category, color
Ordinal	Categories with a meaningful order, but intervals between values are not uniform or measurable.	Customer satisfaction (Low/Medium/High), education level (High School/Bachelor/Master), survey ratings (1-5 stars)

2. Quantitative (Numerical) Data

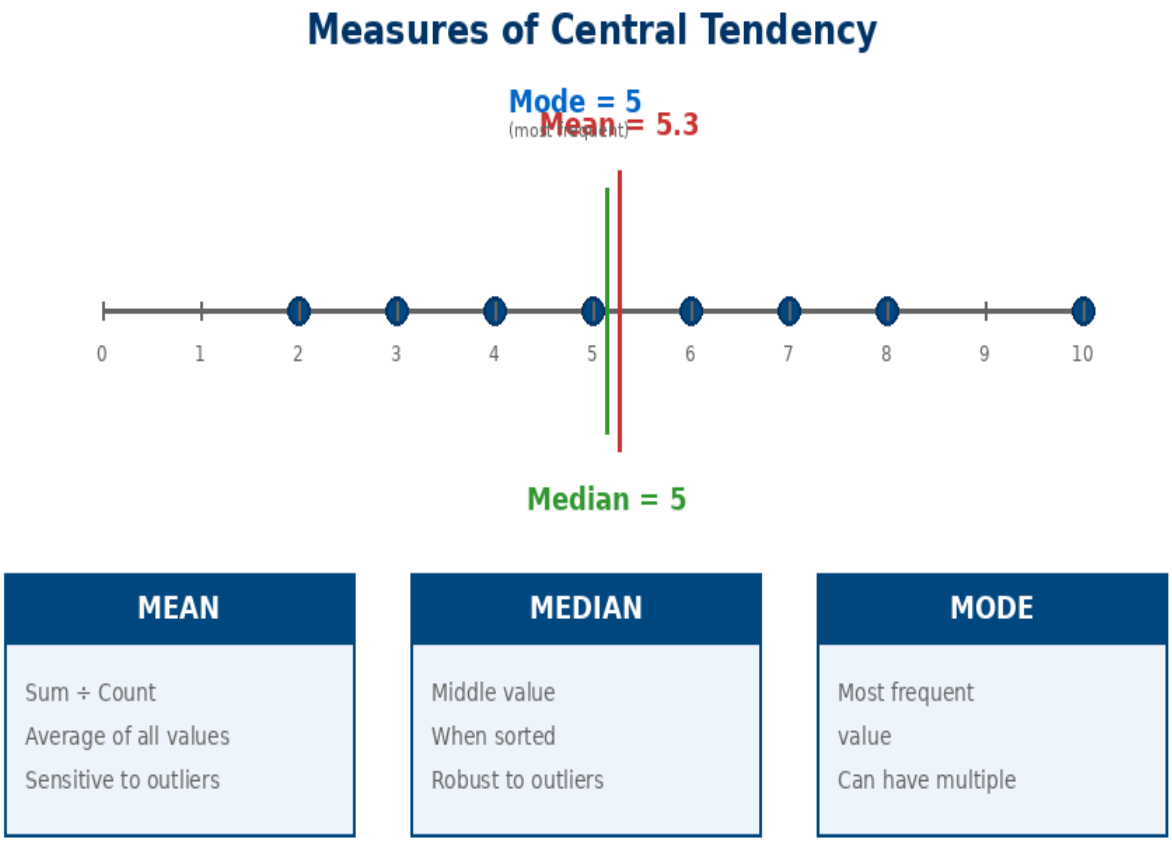
Quantitative data consists of numerical measurements. Arithmetic operations (mean, sum, difference) are meaningful.

Sub-Type	Definition	Examples
Discrete	Countable values, typically integers. Cannot take every value in a range.	Number of customers, units sold, page views, transactions per day
Continuous	Can take any value within a range. Measured, not counted.	Height, weight, temperature, time, revenue, distance

Choosing the Right Analysis: Categorical data: use frequency tables, mode, chi-square tests. Numerical data: use mean, median, standard deviation, t-tests, regression. Mixing up data types leads to nonsensical results (e.g., calculating the mean of "Red, Blue, Green").

4.3 Measures of Central Tendency

Measures of central tendency identify the center or "typical" value in a dataset. They answer the question: What is the most representative value? The three primary measures are mean, median, and mode.



1. Mean (Average)

The mean is the arithmetic average — the sum of all values divided by the count. It is the most commonly reported measure of central tendency but is sensitive to outliers.

```
Mean (μ or x̄) = Σx / n



Example:

Data: [10, 15, 20, 25, 30]

Mean = (10 + 15 + 20 + 25 + 30) / 5 = 100 / 5 = 20
```

Use when: Data is approximately symmetric and has no extreme outliers. Appropriate for continuous numerical data like height, temperature, revenue.

2. Median

The median is the middle value when data is sorted. For an even number of data points, it is the average of the two middle values. The median is robust to outliers — extreme values do not distort it.

```
Example (odd count):

Data: [10, 15, 20, 25, 30]

Median = 20 (middle value)



Example (even count):

Data: [10, 15, 20, 25, 30, 100]

Median = (20 + 25) / 2 = 22.5
```

Use when: Data is skewed or contains outliers. Median is preferred for income, house prices, and any distribution where a few extreme values could distort the mean.

3. Mode

The mode is the most frequently occurring value in the dataset. A dataset can have no mode, one mode (unimodal), two modes (bimodal), or multiple modes (multimodal).

```
Example:

Data: [10, 15, 15, 20, 25, 30]

Mode = 15 (appears twice)



Example (bimodal):
```

nasrin mahin

```
Data: [10, 15, 15, 20, 20, 25]

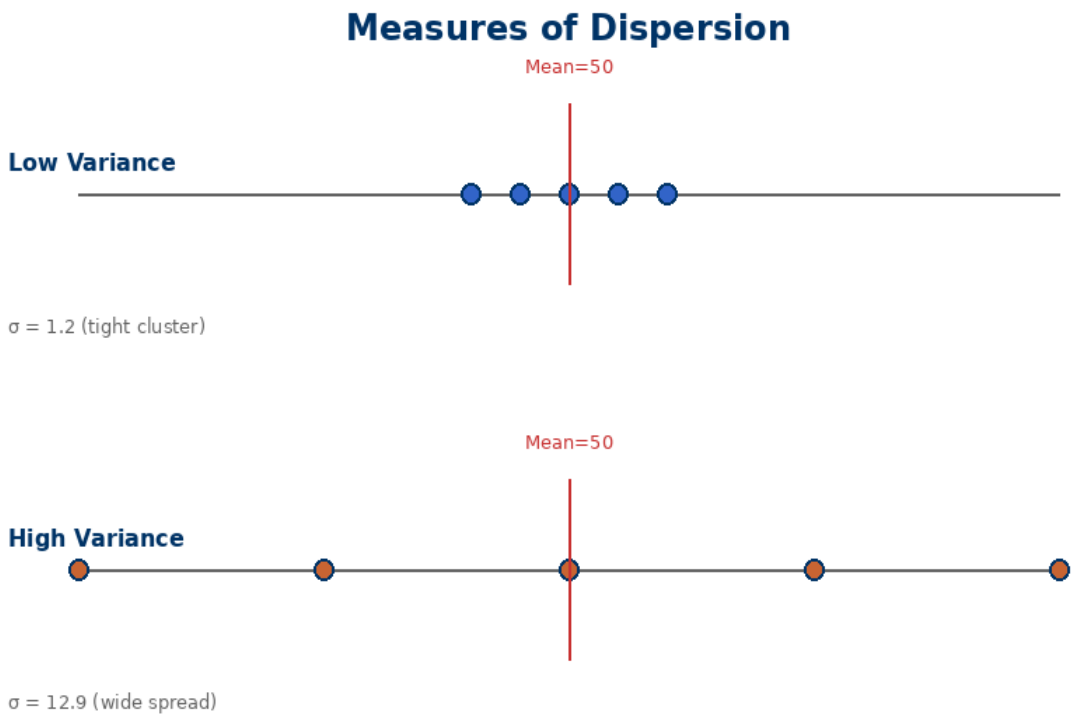
Modes = 15 and 20
```

Use when: Working with categorical data (most common product category, most frequent complaint type) or identifying peaks in distributions.

Measure	When Mean ≠ Median	Interpretation
Mean > Median	Positive (right) skew	A few very high values pull the mean upward (e.g., income data)
Mean < Median	Negative (left) skew	A few very low values pull the mean downward (e.g., age at retirement)
Mean = Median	Symmetric distribution	Data is evenly distributed around the center (e.g., heights, test scores)

4.4 Measures of Dispersion

Dispersion measures quantify how spread out or variable the data is. Two datasets can have the same mean but wildly different spreads. Understanding dispersion is critical for assessing risk, reliability, and variability.



Key Dispersion Measures:

- Range: Max - Min (simple but affected by outliers)
- Variance (σ^2): Average squared deviation from mean
- Standard Deviation (σ): Square root of variance (same units as data)
- IQR (Interquartile Range): Q3 - Q1 (robust to outliers)

1. Range

The range is the difference between the maximum and minimum values. It is the simplest measure of dispersion but is highly sensitive to outliers.

```
Range = Max - Min


Example:

Data: [10, 15, 20, 25, 100]

Range = 100 - 10 = 90
```

2. Variance (σ^2 or s^2)

Variance measures the average squared deviation from the mean. It quantifies how much individual data points differ from the center. Because values are squared, variance is in squared units (e.g., dollars²), which is not intuitive.

Population Variance: $\sigma^2 = \Sigma(x - \mu)^2 / N$

Sample Variance: $s^2 = \Sigma(x - \bar{x})^2 / (n - 1)$

Example:

Data: [10, 15, 20, 25, 30], Mean = 20

$s^2 = [(10-20)^2 + (15-20)^2 + (20-20)^2 + (25-20)^2 + (30-20)^2] / (5-1)$

$= [100 + 25 + 0 + 25 + 100] / 4 = 250 / 4 = 62.5$

Why n-1? Sample variance uses n-1 (Bessel's correction) instead of n because we are estimating the population variance from a sample. This correction makes the estimate unbiased.

3. Standard Deviation (σ or s)

Standard deviation is the square root of variance. It returns dispersion to the original units of the data, making it much more interpretable. A low standard deviation means data points cluster tightly around the mean; a high standard deviation indicates wide spread.

Standard Deviation: $\sigma = \sqrt{(\sigma^2)}$ or $s = \sqrt{(s^2)}$

Example (continuing from above):

$s = \sqrt{62.5} \approx 7.91$

Interpretation: On average, data points deviate from the mean by about 7.91 units.

4. Interquartile Range (IQR)

IQR measures the spread of the middle 50% of the data. It is robust to outliers and is the foundation of the box plot visualization.

```
IQR = Q3 - Q1


Where:
  Q1 (25th percentile) = median of the lower half
  Q3 (75th percentile) = median of the upper half


Example:
Data: [10, 15, 20, 25, 30, 35, 40, 45, 50]
Q1 = 17.5,  Q3 = 42.5
IQR = 42.5 - 17.5 = 25
```

Outlier Detection Rule: Values below Q1 - 1.5×IQR or above Q3 + 1.5×IQR are considered potential outliers.

5. Coefficient of Variation (CV)

CV expresses standard deviation as a percentage of the mean. It is useful for comparing variability between datasets with different units or scales.

```
CV = (σ / μ) × 100%

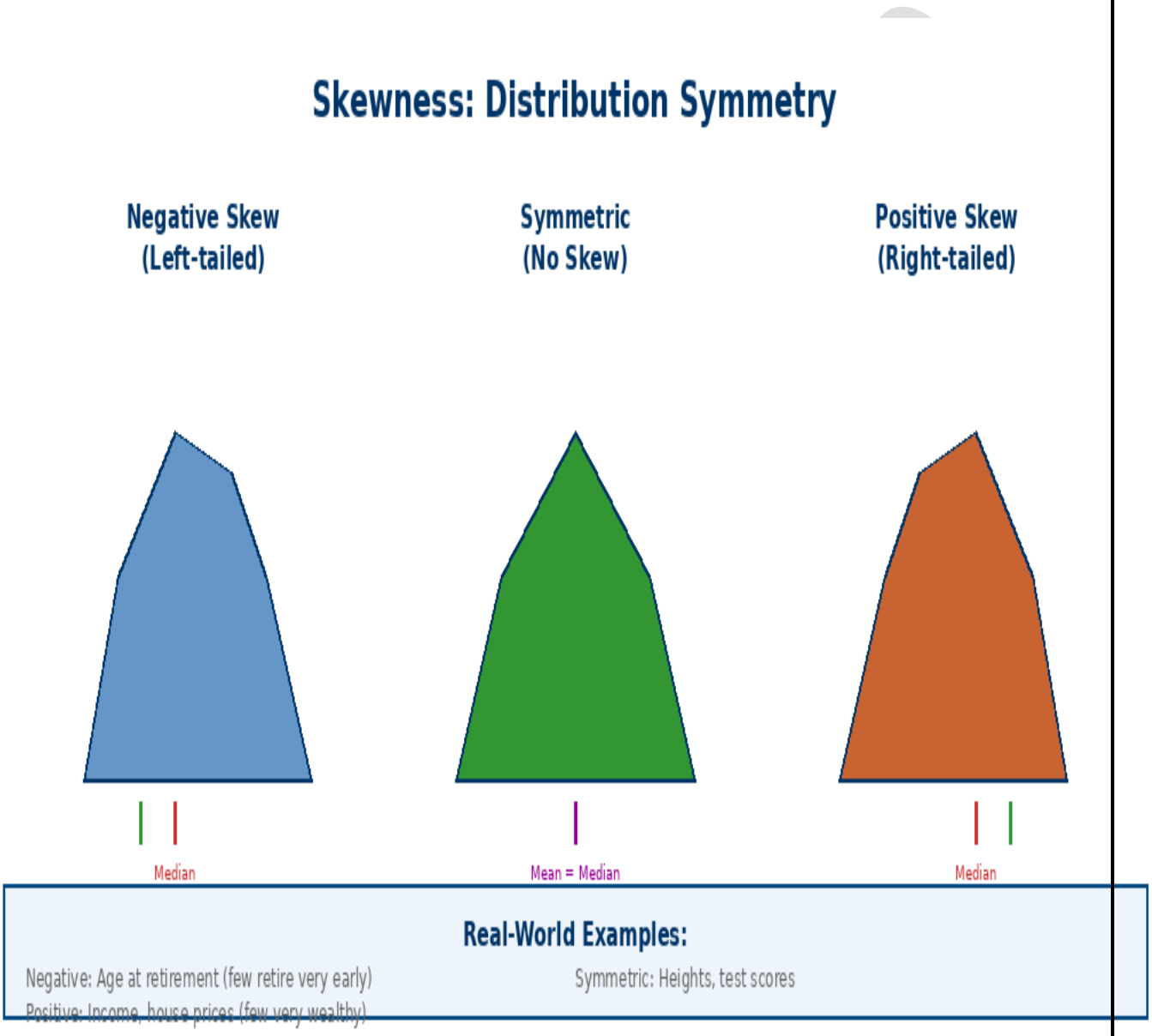

Example:
Dataset A: Mean = 50, SD = 5  →  CV = (5/50) × 100 = 10%
Dataset B: Mean = 500, SD = 50  →  CV = (50/500) × 100 = 10%


Interpretation: Both datasets have the same relative variability despite different absolute values.
```

4.5 Measures of Shape: Skewness and Kurtosis

Skewness

Skewness measures the asymmetry of a distribution. A symmetric distribution has a skewness near zero. Positive skew means the tail extends to the right; negative skew means the tail extends to the left.



Skewness Value	Interpretation	Relationship
Skew = 0	Perfectly symmetric distribution	Mean = Median = Mode

Skew > 0	Positive (right) skew — long tail on the right	Mean > Median
Skew < 0	Negative (left) skew — long tail on the left	Mean < Median

Real-World Examples: Income (positive skew — most earn moderate amounts, few earn millions), Age at retirement (negative skew — most retire around 65, few retire very young), Test scores (often symmetric if well-designed).

Kurtosis

Kurtosis measures the "tailedness" of a distribution — how much probability mass is in the tails versus the center. High kurtosis means more extreme values (heavy tails); low kurtosis means fewer extremes (light tails).

Kurtosis Type	Kurtosis Value	Interpretation
Mesokurtic	≈ 3 (or 0 excess)	Normal distribution — moderate tails
Leptokurtic	> 3 (or > 0 excess)	Heavy tails — more outliers, sharper peak
Platykurtic	< 3 (or < 0 excess)	Light tails — fewer outliers, flatter distribution

Kurtosis is important in finance (measuring tail risk, likelihood of extreme market moves) and quality control (detecting production defects).

nasir-mahin

4.6 Measures of Relationship: Covariance and Correlation

When analyzing two variables, we often want to know if they move together. Do sales increase when advertising spend increases? Does temperature affect ice cream sales? Covariance and correlation quantify these relationships.

Covariance

Covariance measures the direction of the linear relationship between two variables. Positive covariance means they tend to increase together; negative covariance means one increases as the other decreases.

```
Cov(X,Y) = Σ(Xi - X̄)(Yi - Ȳ) / n

Interpretation:
  Cov > 0: Variables move together (positive relationship)
  Cov < 0: Variables move in opposite directions (negative relationship)
  Cov ≈ 0: No linear relationship
```

Limitation: Covariance's magnitude is not standardized — a covariance of 50 could be strong or weak depending on the scales of X and Y. This makes it hard to interpret or compare across datasets.

Correlation (Pearson's r)

Correlation is standardized covariance, always ranging from -1 to +1. It measures both the strength and direction of a linear relationship.

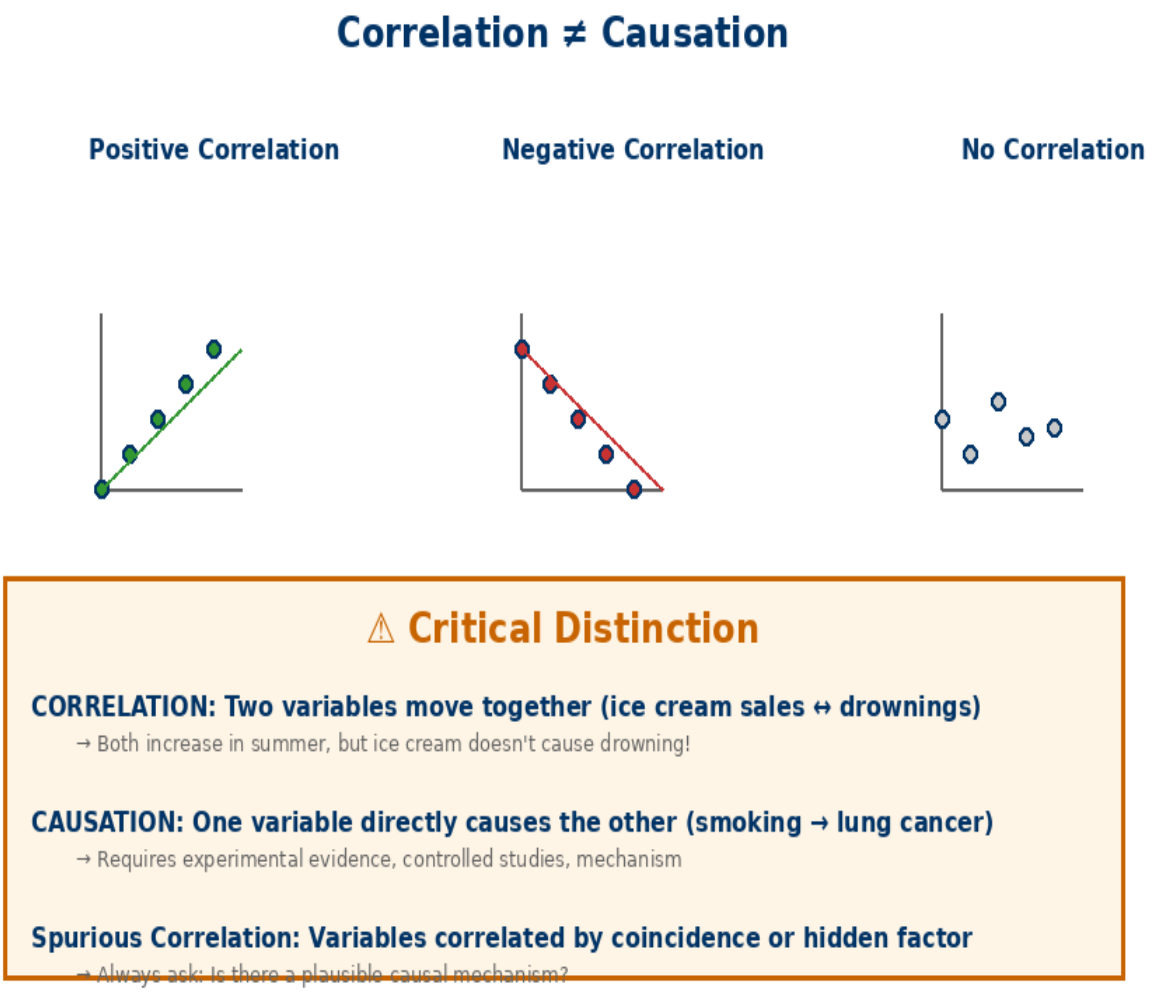
```
Correlation:  r = Cov(X,Y) / (σX × σY)

Range: -1 ≤ r ≤ +1

Interpretation:
  r = +1:  Perfect positive linear relationship
  r = -1:  Perfect negative linear relationship
  r = 0:   No linear relationship
  |r| > 0.7:  Strong correlation
```

$0.3 < |r| < 0.7$: Moderate correlation

$|r| < 0.3$: Weak correlation



Critical Warning: Correlation does NOT imply causation. Two variables can be correlated due to coincidence, a hidden third variable (confounding), or reverse causality. Always investigate plausible causal mechanisms before concluding that one variable causes another.

Examples of Spurious Correlation: Ice cream sales and drownings both increase in summer — but ice cream does not cause drowning. The hidden variable is warm weather. Number of Nicolas Cage films and swimming pool drownings are correlated over time — pure coincidence with no causal link.

nasrin-a-mahin

4.7 Probability Theory Fundamentals

Probability is the mathematical language of uncertainty. It quantifies the likelihood of events and is the foundation of statistical inference, hypothesis testing, and machine learning.

Basic Probability Concepts

Term	Definition	Formula
Sample Space	Set of all possible outcomes	Ω = {H, T} for a coin flip
Event	Subset of the sample space	A = {H}
Probability	Likelihood of an event (0 to 1)	P(A) = Favorable outcomes / Total outcomes
Joint Probability	Probability of A AND B occurring	P(A ∩ B) = P(A) × P(B) if independent
Conditional Probability	Probability of A given B has occurred	P(A|B) = P(A ∩ B) / P(B)

Bayes' Theorem

Bayes' Theorem updates probabilities based on new evidence. It is fundamental to machine learning, spam filters, medical diagnostics, and A/B testing.

$$P(A|B) = [P(B|A) \times P(A)] / P(B)$$

Where:

P(A|B) = Posterior probability (updated belief after seeing evidence B)

P(B|A) = Likelihood (probability of evidence given hypothesis)

P(A) = Prior probability (initial belief before evidence)

P(B) = Marginal probability (total probability of evidence)

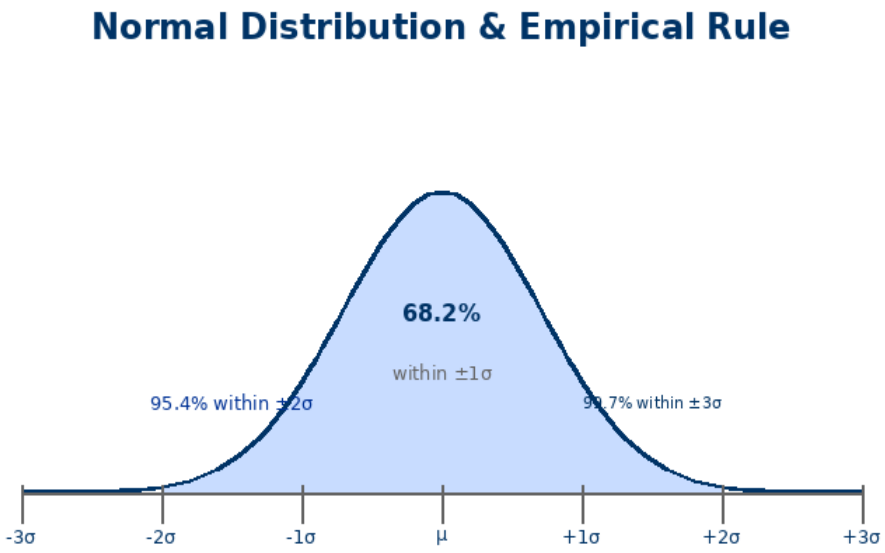
Example: Medical Test. Disease prevalence P(Disease) = 0.01 (1%). Test sensitivity P(Positive|Disease) = 0.95. Test specificity P(Negative|No Disease) = 0.90. If you test positive, what is P(Disease|Positive)? Using Bayes' Theorem, P(Disease|Positive) ≈ 0.087 or 8.7%, much lower than 95% because the disease is rare.

4.8 Probability Distributions

A probability distribution describes how probabilities are distributed over possible values of a random variable. Distributions are the building blocks of statistical modeling and machine learning.

Normal Distribution (Gaussian)

The normal distribution is the most important distribution in statistics. It is symmetric, bell-shaped, and characterized by mean (μ) and standard deviation (σ).



Empirical Rule (68-95-99.7):

- 68% of data falls within 1 standard deviation of the mean
- 95% of data falls within 2 standard deviations of the mean
- 99.7% of data falls within 3 standard deviations of the mean

Use: Identifying outliers, quality control, setting confidence intervals

Empirical Rule (68-95-99.7): ~68% of data falls within ±1σ of the mean, ~95% within ±2σ, ~99.7% within ±3σ. This rule is used for outlier detection and quality control.

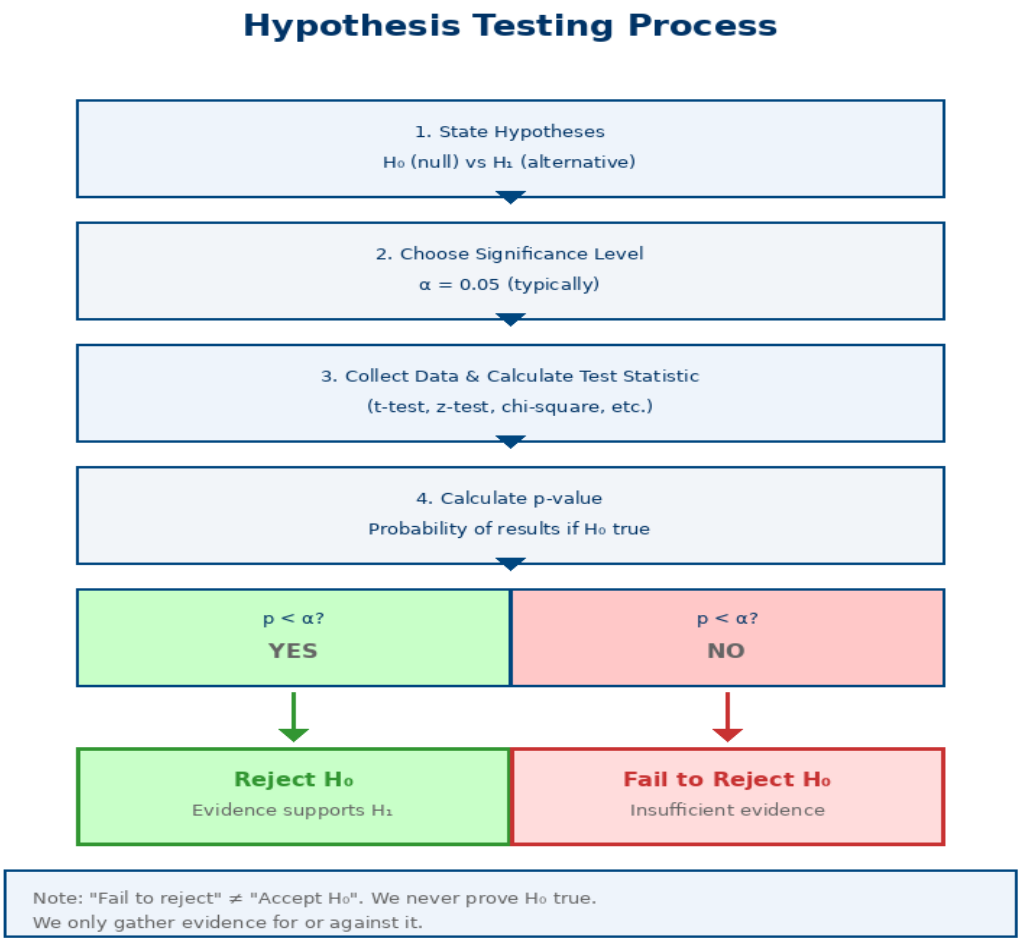
Central Limit Theorem: Regardless of the original population distribution, the sampling distribution of the sample mean approaches a normal distribution as sample size increases. This is why the normal distribution is so fundamental.

Other Important Distributions

Distribution	Use Case	Parameters
Binomial	Count of successes in n trials (coin flips, yes/no surveys)	n (trials), p (success probability)
Poisson	Count of rare events in fixed interval (calls per hour, defects per batch)	λ (average rate)
t-distribution	Hypothesis testing with small samples (n < 30)	degrees of freedom (df)
Chi-square	Goodness-of-fit tests, independence tests	degrees of freedom (df)
Uniform	Equal probability across a range (random number generation)	min, max

4.10 Hypothesis Testing

Hypothesis testing is a formal procedure for using sample data to decide whether a claim about a population is likely to be true. It is the statistical workhorse behind A/B testing, clinical trials, quality control, and scientific research.

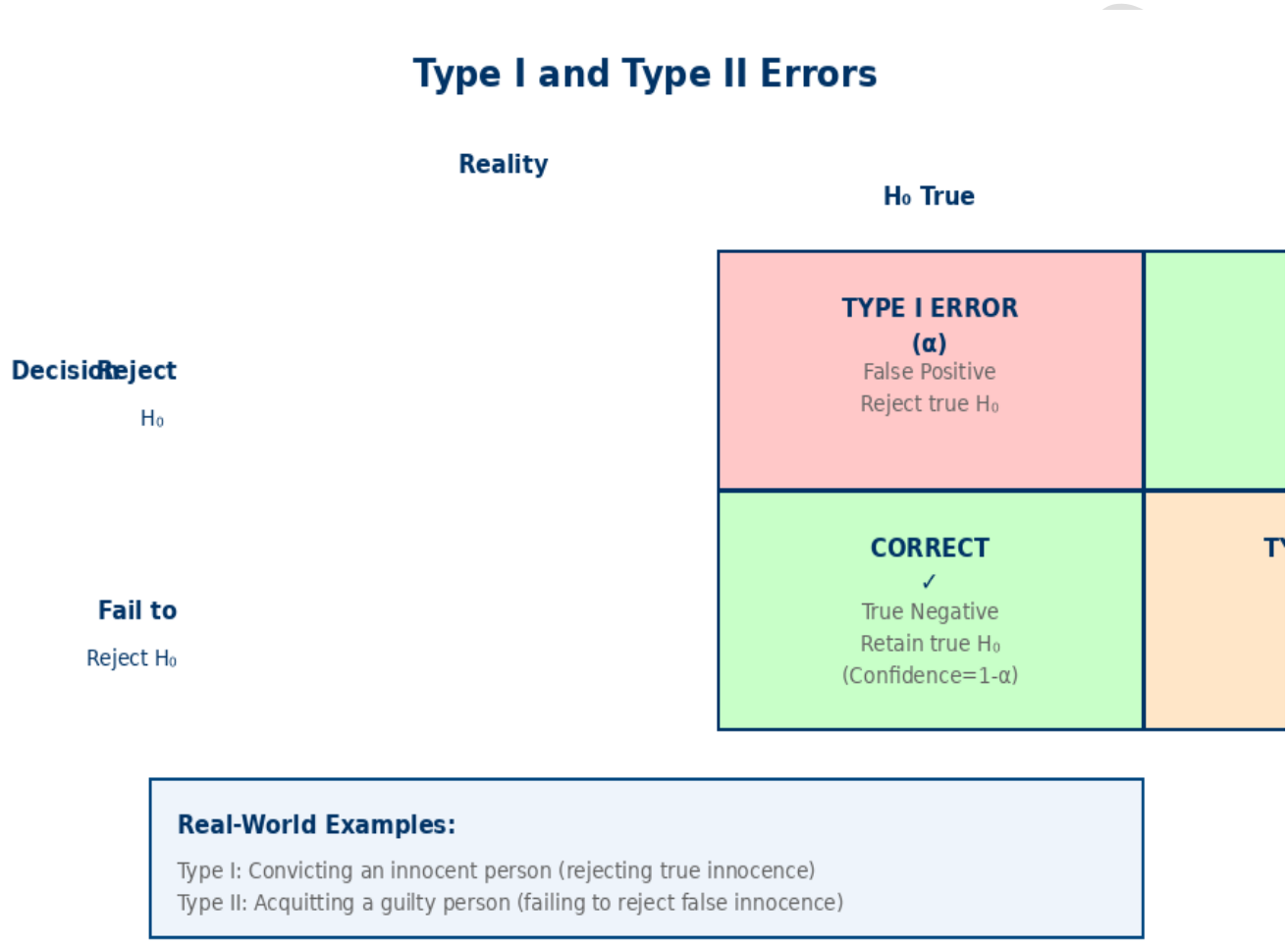


The Hypothesis Testing Process

1. State Hypotheses: H_0 (null hypothesis — no effect) vs H_1 (alternative hypothesis — there is an effect).
2. Choose Significance Level: $\alpha = 0.05$ (typically). This is the probability of making a Type I error.
3. Collect Data and Calculate Test Statistic: Use t-test, z-test, chi-square, etc., depending on data type and sample size.
4. Calculate p-value: The probability of observing results at least as extreme as yours, assuming H_0 is true.
5. Make a Decision: If $p < \alpha$, reject H_0. If $p \geq \alpha$, fail to reject H_0.

Interpretation: Rejecting H_0 does NOT prove H_1 is true — it means the data provides sufficient evidence against H_0. Failing to reject H_0 does NOT prove H_0 is true — it means insufficient evidence to conclude otherwise.

Type I and Type II Errors

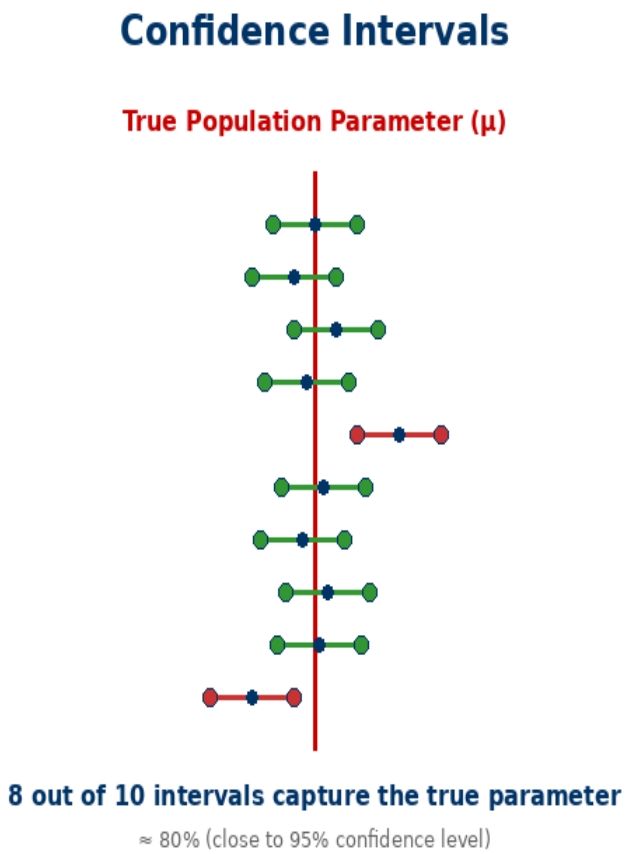


Error Type	Definition	Consequence
Type I (α)	Reject true H_0 (false positive)	Conclude an effect exists when it does not (e.g., approve ineffective drug)
Type II (β)	Fail to reject false H_0 (false negative)	Miss a real effect (e.g., reject effective drug)

Power = 1 - β. Statistical power is the probability of correctly rejecting a false H_0. High power (≥0.80) is desirable.

4.11 Confidence Intervals

A confidence interval provides a range of plausible values for a population parameter, along with a level of confidence (typically 95%) that the true parameter lies within that range.



95% CI Interpretation: If we repeat sampling many times, 95% of intervals will contain the true parameter. This does NOT mean 95% probability it's in one interval!

95% Confidence Interval for Mean:

$CI = \bar{x} \pm (t \times SE)$

Where:

$\bar{x}$ = sample mean

t = critical value from t-distribution (depends on confidence level and df)

```
SE = standard error = s / √n
```

Interpretation: If we repeated the study many times and calculated a 95% CI each time, approximately 95% of those intervals would contain the true population mean. This does NOT mean there is a 95% probability that this particular interval contains the true mean — the interval either does or does not contain it.

Common Mistake: Confidence intervals quantify estimation uncertainty, not the variability of individual data points. A 95% CI for the mean does NOT mean 95% of individual observations fall in that range.

4.12 Common Statistical Tests

Test	When to Use	Example
One-sample t-test	Compare sample mean to a known value	Is average customer satisfaction = 7?
Two-sample t-test	Compare means of two independent groups	Do men and women have different salaries?
Paired t-test	Compare means of paired observations	Before/after treatment comparison
ANOVA	Compare means of 3+ groups	Do sales differ across 4 regions?
Chi-square test	Test independence of categorical variables	Is product preference related to gender?
Correlation test	Test if correlation is significantly different from 0	Is age correlated with income?

Practice Exercises

1. Calculate mean, median, mode, range, variance, and standard deviation for: [12, 15, 18, 18, 20, 22, 25, 30, 35, 100]
2. A sample has mean = 50 and SD = 10. What percentage of data falls between 30 and 70 if normally distributed?
3. Dataset A: [10, 12, 14] has mean 12. Dataset B: [2, 12, 22] also has mean 12. Which has higher variance? Calculate both.

4. Two variables have correlation r = 0.85. Does this prove one causes the other? Explain.
5. A company tests a new website design. Old site: 100 visitors, 15% conversion. New site: 100 visitors, 22% conversion. Perform a two-proportion z-test at α=0.05.
6. Calculate a 95% confidence interval for mean income given: sample mean = $55,000, sample SD = $12,000, n = 50.

nasim-a-mahin

UNIT V: Data Analysis with Python

5.1 Introduction to Python for Data Analysis

Python has become the dominant language for data analysis, surpassing R, SAS, and MATLAB in industry adoption. Its rise is driven by a powerful ecosystem of libraries, readability, versatility, and strong community support.

> *Python is the Swiss Army knife of data science — equally capable of data wrangling, statistical analysis, machine learning, web scraping, automation, and production deployment.*

Why Python for Data Analysis?

- Rich ecosystem: NumPy, pandas, matplotlib, seaborn, scikit-learn, statsmodels — everything you need is pre-built
- Readable syntax: code reads like pseudocode, making it easy to learn and maintain
- Versatile: same language for data analysis, ML, web apps, automation, and deployment
- Industry standard: used by Google, Netflix, NASA, finance, healthcare, and academia

- Free and open source: no licensing costs, massive community support
- Jupyter Notebooks: interactive environment perfect for exploratory data analysis
- Integrates with everything: SQL databases, Excel, APIs, Spark, cloud platforms

The Data Analysis Stack in Python

Library	Purpose	Key Features
NumPy	Numerical computing	Fast arrays, mathematical operations, linear algebra
pandas	Data manipulation	DataFrames, Series, cleaning, grouping, merging
matplotlib	Visualization	Line, bar, scatter, histogram, customizable plots
seaborn	Statistical viz	Beautiful default styles, built-in statistical plots
scikit-learn	Machine learning	Regression, classification, clustering, pipelines
statsmodels	Statistics	Hypothesis tests, regression, time series analysis
Jupyter	Interactive notebooks	Code, output, and markdown in one document

Installing Python and Essential Libraries

```
# Install Anaconda distribution (recommended) - includes everything
# Download from: anaconda.com


# OR install libraries individually with pip:
pip install numpy pandas matplotlib seaborn scikit-learn jupyter


# Verify installation:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
print("NumPy version:", np.__version__)
print("Pandas version:", pd.__version__)
```

5.2 Python Basics – Data Types, Variables, Operators

Variables and Assignment

```
# Python is dynamically typed - no need to declare types
```

```
x = 10            # Integer
price = 19.99      # Float
name = "Alice"      # String
is_active = True    # Boolean
```

Core Data Types

Type	Example	Description
int	42	Whole numbers
float	3.14	Decimal numbers
str	"Hello"	Text enclosed in quotes
bool	True / False	Logical values
list	[1, 2, 3]	Ordered, mutable collection
tuple	(1, 2, 3)	Ordered, immutable collection
dict	{"key": "value"}	Key-value pairs
set	{1, 2, 3}	Unordered, unique elements

Lists – Ordered Collections

```
# Creating lists
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True]


# Accessing elements (zero-indexed)
print(numbers[0])   # First element: 1
print(numbers[-1])  # Last element: 5
print(numbers[1:4]) # Slice: [2, 3, 4]


# Modifying lists
numbers.append(6)        # Add to end
numbers.insert(0, 0)      # Insert at index 0
numbers.remove(3)        # Remove first occurrence of 3
numbers.sort()           # Sort in place
```

Dictionaries – Key-Value Pairs

```
# Creating dictionaries
person = {
    "name": "Alice",
```

```
    "age": 25,
    "city": "NYC"
}


# Accessing values
print(person["name"])          # Alice
print(person.get("age"))       # 25
print(person.get("country", "Unknown"))  # Default value


# Adding/modifying
person["email"] = "alice@example.com"
person["age"] = 26
```

Operators

Category	Operators	Example
Arithmetic	+, -, *, /, //, %, **	10 / 3 = 3.333
Comparison	==, !=, >, <, >=, <=	5 > 3 → True
Logical	and, or, not	(x>0) and (y<10)
Membership	in, not in	"a" in "apple" → True

```
# Floor division and modulus
print(10 // 3)    # 3 (integer division)
print(10 % 3)     # 1 (remainder)
print(2 ** 3)     # 8 (exponentiation)
```

Control Flow

```
# If-elif-else
score = 85
if score >= 90:
    grade = "A"
elif score >= 75:
    grade = "B"
else:
    grade = "C"
```

```
# For loop
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4


for item in ["apple", "banana", "cherry"]:
    print(item)


# While loop
count = 0
while count < 5:
    print(count)
    count += 1
```

5.3 NumPy – Arrays and Numerical Computing

NumPy (Numerical Python) is the foundation of scientific computing in Python. It provides fast, efficient multidimensional arrays and a vast library of mathematical functions.

> *NumPy arrays are 10-100x faster than Python lists for numerical operations because they use contiguous memory and are implemented in C.*

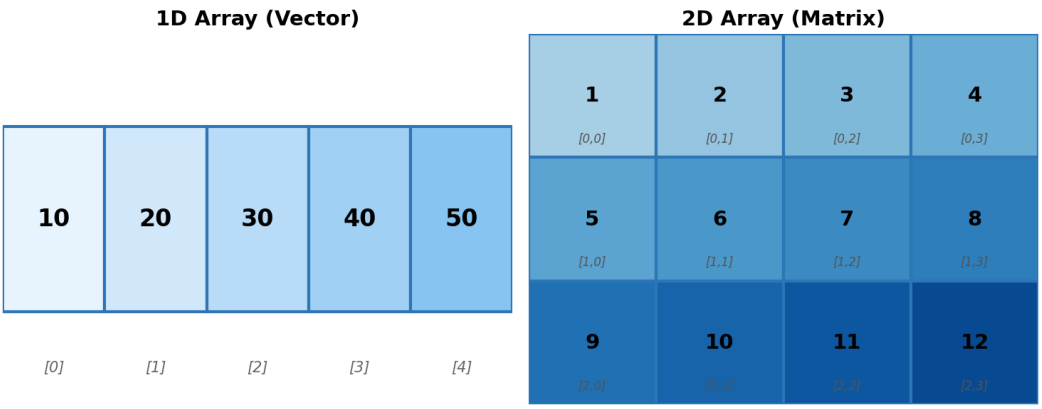


Figure 5.1 — NumPy Array Structures: 1D Vector and 2D Matrix

Creating NumPy Arrays

```
import numpy as np


# From Python list
```

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)            # [1 2 3 4 5]
print(type(arr))      # <class "numpy.ndarray">


# 2D array (matrix)
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])


# Array creation functions
zeros = np.zeros((3, 4))          # 3x4 array of zeros
ones = np.ones((2, 5))            # 2x5 array of ones
empty = np.empty((2, 3))          # Uninitialized array
range_arr = np.arange(0, 10, 2)   # [0, 2, 4, 6, 8]
linspace = np.linspace(0, 1, 5)   # 5 evenly spaced numbers from 0 to 1
eye = np.eye(3)                   # 3x3 identity matrix
```

Array Attributes

```
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8]])


print(arr.shape)     # (2, 4) - dimensions
print(arr.ndim)      # 2 - number of dimensions
print(arr.size)      # 8 - total number of elements
print(arr.dtype)     # dtype("int64") - data type
print(arr.itemsize)  # 8 - bytes per element
```

5.4 NumPy – Array Operations and Indexing

Array Indexing and Slicing

```
arr = np.array([10, 20, 30, 40, 50])


# Indexing
```

```
print(arr[0])       # 10 (first element)
print(arr[-1])      # 50 (last element)


# Slicing
print(arr[1:4])     # [20 30 40] (elements 1, 2, 3)
print(arr[:3])      # [10 20 30] (first 3 elements)
print(arr[2:])      # [30 40 50] (from index 2 to end)
print(arr[::2])     # [10 30 50] (every 2nd element)


# 2D array indexing
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])


print(matrix[0, 0])   # 1 (row 0, column 0)
print(matrix[1, 2])   # 6 (row 1, column 2)
print(matrix[0, :])   # [1 2 3] (entire first row)
print(matrix[:, 1])   # [2 5 8] (entire second column)
print(matrix[0:2, 1:3])# [[2 3], [5 6]] (submatrix)
```

Boolean Indexing

```
arr = np.array([10, 25, 30, 15, 40])


# Create boolean mask
mask = arr > 20
print(mask)         # [False True True False True]


# Use mask to filter
print(arr[mask])    # [25 30 40]
print(arr[arr > 20]) # [25 30 40] (shorthand)
```

Array Arithmetic (Element-wise)

```
a = np.array([1, 2, 3, 4])
```

```
b = np.array([10, 20, 30, 40])


print(a + b)    # [11 22 33 44]
print(a * b)    # [10 40 90 160]
print(a ** 2)   # [1 4 9 16]
print(b / 10)   # [1. 2. 3. 4.]


# Broadcasting - scalar operations
print(a + 10)   # [11 12 13 14]
print(a * 2)    # [2 4 6 8]
```

5.5 NumPy – Mathematical and Statistical Functions

Aggregate Functions

```
arr = np.array([10, 20, 30, 40, 50])


print(np.sum(arr))       # 150
print(np.mean(arr))      # 30.0
print(np.median(arr))    # 30.0
print(np.std(arr))       # 14.14 (standard deviation)
print(np.var(arr))       # 200.0 (variance)
print(np.min(arr))       # 10
print(np.max(arr))       # 50
print(np.argmin(arr))    # 0 (index of min value)
print(np.argmax(arr))    # 4 (index of max value)
```

Mathematical Functions

```
arr = np.array([1, 4, 9, 16, 25])


print(np.sqrt(arr))      # [1. 2. 3. 4. 5.]
print(np.exp(arr))       # Exponential
print(np.log(arr))       # Natural logarithm
print(np.sin(arr))       # Sine
```

```
print(np.cos(arr))       # Cosine
print(np.abs(arr))       # Absolute value
print(np.round(arr, 2))  # Round to 2 decimal places
```

Matrix Operations

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Matrix multiplication (dot product)
print(np.dot(A, B))
# OR using @ operator
print(A @ B)

# Transpose
print(A.T)

# Determinant and inverse (requires linear algebra)
from numpy.linalg import det, inv
print(det(A))       # Determinant
print(inv(A))       # Matrix inverse
```

5.6 Pandas – Series and DataFrames

Pandas is built on top of NumPy and provides two primary data structures: Series (1D) and DataFrame (2D). These are the workhorses of data analysis in Python.

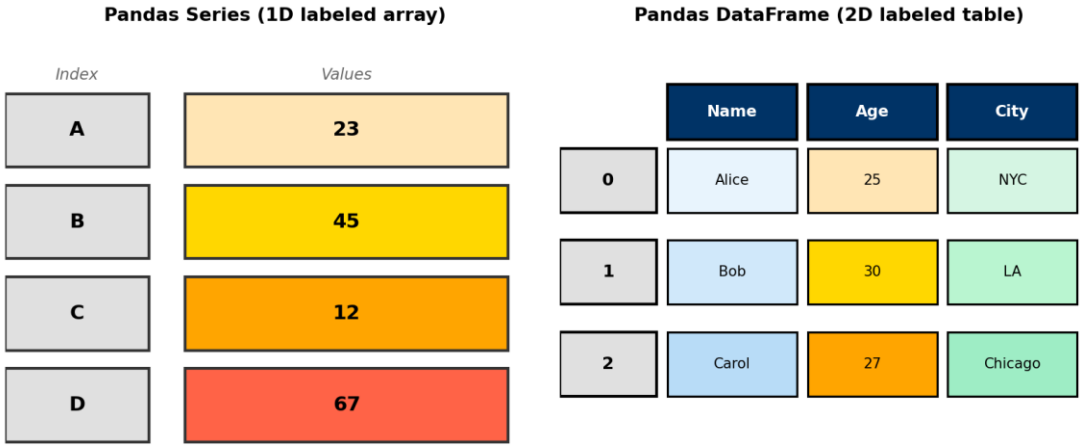


Figure 5.2 — Pandas Data Structures: Series and DataFrame

Pandas Series

```
import pandas as pd


# Create a Series from a list
s = pd.Series([10, 20, 30, 40, 50])
print(s)


# Series with custom index
s = pd.Series([10, 20, 30], index=["A", "B", "C"])
print(s)
print(s["B"])        # Access by label: 20


# Series from dictionary
data = {"A": 10, "B": 20, "C": 30}
s = pd.Series(data)
print(s)
```

Pandas DataFrame

```
# Create DataFrame from dictionary
data = {
    "Name": ["Alice", "Bob", "Carol", "David"],
    "Age": [25, 30, 27, 35],
```

```
    "City": ["NYC", "LA", "Chicago", "Houston"],
    "Salary": [70000, 80000, 75000, 90000]
}


df = pd.DataFrame(data)
print(df)


# DataFrame from list of lists
data = [[25, "NYC"], [30, "LA"], [27, "Chicago"]]
df = pd.DataFrame(data, columns=["Age", "City"])
print(df)
```

DataFrame Attributes

```
print(df.shape)       # (4, 4) - rows x columns
print(df.columns)     # Column names
print(df.index)       # Row index
print(df.dtypes)      # Data type of each column
print(df.info())      # Summary info
print(df.describe())  # Statistical summary
```

5.7 Pandas – Reading and Writing Data

Reading Data from Files

```
# Read CSV
df = pd.read_csv("data.csv")


# Read Excel
df = pd.read_excel("data.xlsx", sheet_name="Sheet1")


# Read from SQL database
import sqlite3
conn = sqlite3.connect("database.db")
df = pd.read_sql_query("SELECT * FROM customers", conn)
```

```
# Read JSON
df = pd.read_json("data.json")


# Read from URL
url = "https://example.com/data.csv"
df = pd.read_csv(url)
```

Writing Data to Files

```
# Write to CSV
df.to_csv("output.csv", index=False)


# Write to Excel
df.to_excel("output.xlsx", index=False, sheet_name="Data")


# Write to SQL
df.to_sql("table_name", conn, if_exists="replace", index=False)


# Write to JSON
df.to_json("output.json", orient="records")
```

5.8 Pandas – Data Selection and Indexing

Selecting Columns

```
# Single column - returns Series
print(df["Name"])
print(df.Name)          # Dot notation (if column name is valid Python identifier)


# Multiple columns - returns DataFrame
print(df[["Name", "Age", "City"]])
```

Selecting Rows

```
# By position with iloc (integer-location)
```

```
print(df.iloc[0])        # First row
print(df.iloc[-1])       # Last row
print(df.iloc[0:3])      # First 3 rows
print(df.iloc[[0, 2, 4]]) # Specific rows by position


# By label with loc (label-location)
df.index = ["A", "B", "C", "D"]  # Set custom index
print(df.loc["A"])       # Row with index "A"
print(df.loc[["A", "C"]]) # Multiple rows by label


# Boolean indexing
print(df[df["Age"] > 27])
print(df[df["City"] == "NYC"])
print(df[(df["Age"] > 25) & (df["Salary"] > 75000)])
```

Selecting Rows and Columns Together

```
# iloc[rows, columns]
print(df.iloc[0:3, 0:2])  # First 3 rows, first 2 columns


# loc[rows, columns]
print(df.loc[df["Age"] > 25, ["Name", "Salary"]])
```

5.9 Pandas – Data Cleaning and Handling Missing Values

Detecting Missing Values

```
# Check for NaN values
print(df.isnull())       # Boolean DataFrame
print(df.isnull().sum())  # Count NaN per column
print(df.notnull())      # Inverse of isnull()
```

Handling Missing Values

```
# Drop rows with any NaN
df_clean = df.dropna()
```

```
# Drop columns with any NaN
df_clean = df.dropna(axis=1)

# Drop rows where ALL values are NaN
df_clean = df.dropna(how="all")

# Fill missing values with a constant
df_filled = df.fillna(0)

# Fill with column mean
df["Age"].fillna(df["Age"].mean(), inplace=True)

# Forward fill (use previous value)
df.fillna(method="ffill", inplace=True)

# Backward fill (use next value)
df.fillna(method="bfill", inplace=True)
```

Removing Duplicates

```
# Check for duplicates
print(df.duplicated())

# Remove duplicates
df_unique = df.drop_duplicates()

# Remove duplicates based on specific columns
df_unique = df.drop_duplicates(subset=["Name", "Email"])
```

Data Type Conversion

```
# Convert column to numeric
df["Age"] = pd.to_numeric(df["Age"], errors="coerce")
```

```
# Convert to datetime
df["Date"] = pd.to_datetime(df["Date"])


# Convert to string
df["ID"] = df["ID"].astype(str)


# Convert to categorical
df["City"] = df["City"].astype("category")
```

5.10 Pandas – Data Transformation

Creating New Columns

```
# From existing columns
df["Tax"] = df["Salary"] * 0.2
df["Net Salary"] = df["Salary"] - df["Tax"]


# Using apply() with a function
def categorize_age(age):
    if age < 25:
        return "Young"
    elif age < 35:
        return "Middle"
    else:
        return "Senior"


df["Age_Group"] = df["Age"].apply(categorize_age)


# Lambda function
df["Bonus"] = df["Salary"].apply(lambda x: x * 0.1 if x > 80000 else 0)
```

String Operations

```
# Convert to uppercase
df["City_Upper"] = df["City"].str.upper()
```

```
# Check if contains
df["Has_NY"] = df["City"].str.contains("NY")


# Replace text
df["City"] = df["City"].str.replace("NYC", "New York")


# Split strings
df[["First", "Last"]] = df["Name"].str.split(" ", expand=True)
```

Sorting

```
# Sort by single column
df_sorted = df.sort_values("Age")
df_sorted = df.sort_values("Age", ascending=False)


# Sort by multiple columns
df_sorted = df.sort_values(["City", "Age"], ascending=[True, False])
```

5.11 Pandas – GroupBy and Aggregation

Basic GroupBy

```
# Group by single column
grouped = df.groupby("City")


# Aggregate with single function
print(grouped["Salary"].mean())
print(grouped["Age"].sum())
print(grouped.size())  # Count rows per group


# Multiple aggregations
print(grouped["Salary"].agg(["mean", "min", "max", "count"]))
```

GroupBy with Multiple Columns

```
# Group by multiple columns
grouped = df.groupby(["City", "Age_Group"])
print(grouped["Salary"].mean())
```

Custom Aggregations

```
# Different aggregation per column
agg_dict = {
    "Salary": ["mean", "sum"],
    "Age": ["min", "max"],
    "Name": "count"
}


result = df.groupby("City").agg(agg_dict)
print(result)
```

5.12 Pandas – Merge, Join, and Concatenate

Merge (Join)

```
# Inner join (keep only matching rows)
merged = pd.merge(df1, df2, on="ID", how="inner")


# Left join (keep all from left table)
merged = pd.merge(df1, df2, on="ID", how="left")


# Right join (keep all from right table)
merged = pd.merge(df1, df2, on="ID", how="right")


# Outer join (keep all from both tables)
merged = pd.merge(df1, df2, on="ID", how="outer")


# Join on multiple columns
```

```
merged = pd.merge(df1, df2, on=["ID", "Name"])



# Join when column names differ

merged = pd.merge(df1, df2, left_on="CustomerID", right_on="ID")
```

Concatenate

```
# Stack DataFrames vertically

combined = pd.concat([df1, df2], axis=0, ignore_index=True)



# Combine DataFrames horizontally

combined = pd.concat([df1, df2], axis=1)
```

5.13 Pandas – Pivot Tables and Reshaping

Pivot Tables

```
# Create pivot table

pivot = df.pivot_table(

    values="Salary",

    index="City",

    columns="Age_Group",

    aggfunc="mean"

)

print(pivot)
```

Melt (Unpivot)

```
# Convert wide to long format

df_long = pd.melt(

    df,

    id_vars=["Name"],

    value_vars=["Q1_Sales", "Q2_Sales", "Q3_Sales"],

    var_name="Quarter",

    value_name="Sales"

)
```

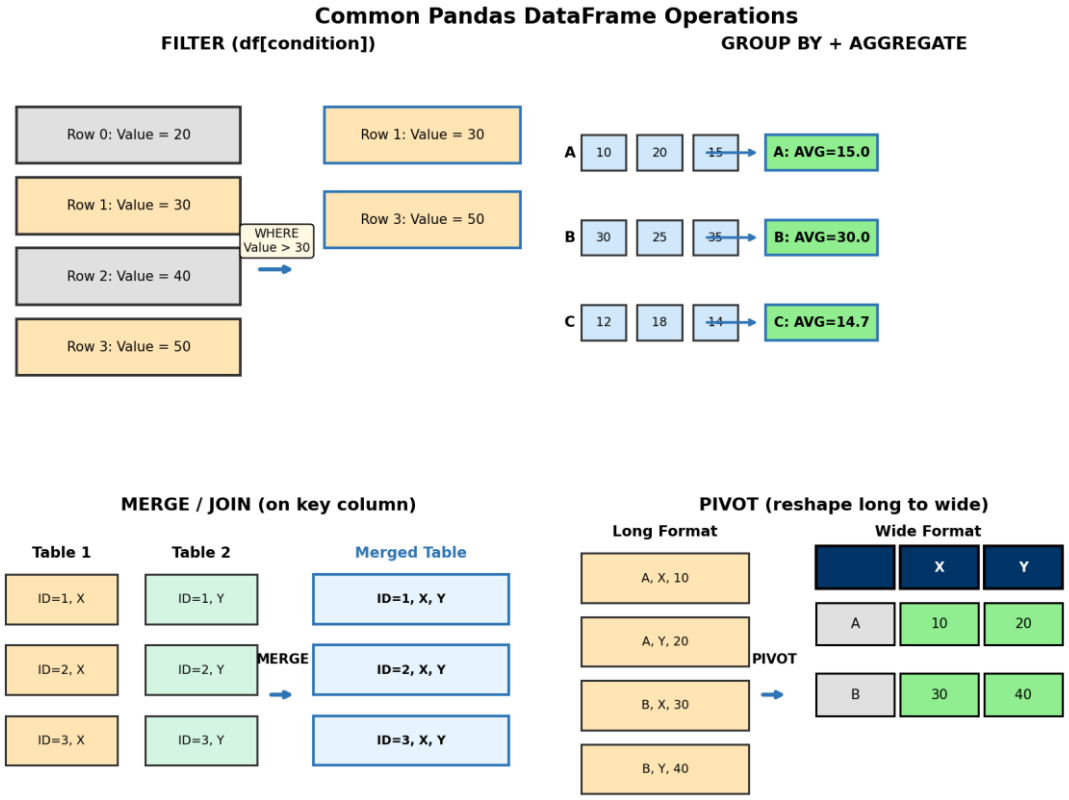


Figure 5.3 — Common DataFrame Operations: Filter, GroupBy, Merge, and Pivot

5.14 Data Visualization with Matplotlib

Basic Line Plot

```
import matplotlib.pyplot as plt

import numpy as np


x = np.linspace(0, 10, 100)

y = np.sin(x)


plt.figure(figsize=(10, 6))

plt.plot(x, y, label="sin(x)", linewidth=2)

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.title("Sine Wave")
```

```
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Bar Chart

```
categories = ["A", "B", "C", "D", "E"]
values = [23, 45, 56, 31, 42]


plt.figure(figsize=(8, 6))
plt.bar(categories, values, color="skyblue", edgecolor="black")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Chart Example")
plt.show()
```

Scatter Plot

```
x = np.random.randn(100)
y = 2*x + np.random.randn(100)*0.5


plt.figure(figsize=(8, 6))
plt.scatter(x, y, c=y, cmap="viridis", alpha=0.6, s=100)
plt.colorbar(label="Y value")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Scatter Plot with Color Mapping")
plt.show()
```

Histogram

```
data = np.random.randn(1000)


plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, color="steelblue", alpha=0.7, edgecolor="black")
plt.xlabel("Value")
plt.ylabel("Frequency")
```

```
plt.title("Histogram")
plt.show()
```

Subplots

```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))


# Line plot
axes[0, 0].plot(x, np.sin(x))
axes[0, 0].set_title("Sine")


# Bar chart
axes[0, 1].bar(categories, values)
axes[0, 1].set_title("Bar Chart")


# Scatter
axes[1, 0].scatter(x, y)
axes[1, 0].set_title("Scatter")


# Histogram
axes[1, 1].hist(data, bins=30)
axes[1, 1].set_title("Histogram")


plt.tight_layout()
plt.show()
```

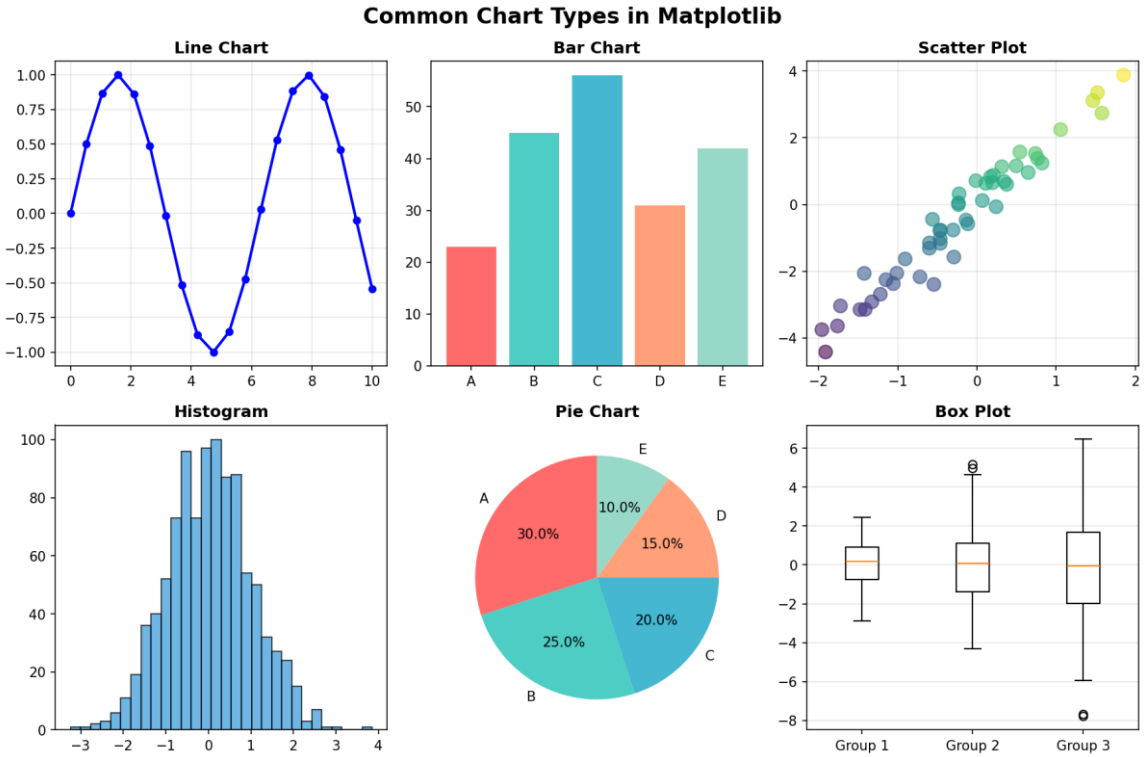


Figure 5.4 — Common Chart Types in Matplotlib

5.15 Advanced Visualization with Seaborn

Seaborn is built on matplotlib and provides beautiful default styles and high-level statistical visualizations.

Setting Seaborn Style

```
import seaborn as sns


# Set style
sns.set_style("whitegrid")   # whitegrid, darkgrid, white, dark, ticks
sns.set_palette("husl")      # Color palette
```

Distribution Plots

```
# Histogram with KDE
sns.histplot(data=df, x="Age", kde=True, bins=20)
plt.show()


# Box plot
sns.boxplot(data=df, x="City", y="Salary")
```

```
plt.show()



# Violin plot
sns.violinplot(data=df, x="Age_Group", y="Salary")
plt.show()
```

Relationship Plots

```
# Scatter plot with regression line
sns.regplot(data=df, x="Age", y="Salary")
plt.show()



# Pair plot (all variable combinations)
sns.pairplot(df[["Age", "Salary", "City"]], hue="City")
plt.show()
```

Heatmaps

```
# Correlation heatmap
correlation = df[["Age", "Salary"]].corr()
sns.heatmap(correlation, annot=True, cmap="coolwarm", fmt=".2f")
plt.show()
```

Count Plots

```
sns.countplot(data=df, x="City")
plt.show()
```

5.16 Python Best Practices for Data Analysis

Code Organization

- Import all libraries at the top of the script or notebook
- Use meaningful variable names: df_sales, not df1
- Keep one concept per notebook cell for clarity
- Write docstrings for custom functions
- Use comments to explain WHY, not WHAT

Pandas Best Practices

- Use vectorized operations instead of loops — 10-100x faster
- Chain methods for readability: df.dropna().groupby("City").mean()

- Use inplace=False (default) to avoid unintended side effects
- Always use .copy() when creating a subset to avoid SettingWithCopyWarning
- Prefer pd.to_datetime() for date columns instead of manual parsing
- Use categorical dtype for columns with few unique values to save memory

Performance Tips

- Filter early: reduce DataFrame size before expensive operations
- Use query() for complex boolean filters — more readable and sometimes faster
- Prefer .loc and .iloc over chained indexing
- Use .itertuples() instead of .iterrows() if you must loop
- Consider Dask or Polars for datasets > 1GB that don't fit in RAM

Common Pitfalls to Avoid

> *NEVER modify a DataFrame while iterating over it. NEVER use df[df.Age > 30]["Salary"] = 0 (chained indexing). Always use df.loc[df.Age > 30, "Salary"] = 0 instead.*

- Forgetting to reset_index() after groupby — causes multi-level index confusion
- Not checking data types with df.dtypes before analysis
- Assuming NaN and None are the same (they are not)
- Using == to compare NaN values (use pd.isna() instead)
- Not handling duplicates before merging tables

Reproducibility Checklist

- Set random seeds: np.random.seed(42), random.seed(42)
- Document library versions: pandas.__version__
- Save intermediate results to avoid re-running expensive computations
- Use version control (Git) for scripts and notebooks
- Export conda environment or requirements.txt for package versions

UNIT VI: Data Visualization

6.1 Introduction to Data Visualization

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization provides an accessible way to see and understand trends, outliers, and patterns in data.

> *The human brain processes visual information 60,000 times faster than text. A well-designed visualization can reveal insights in seconds that would take hours to find in a spreadsheet.*

Why Data Visualization Matters

- Pattern Recognition: Humans are wired to spot visual patterns — charts reveal trends invisible in raw numbers
- Faster Decision-Making: Executives can grasp key insights at a glance without reading lengthy reports
- Storytelling: Visualizations make data memorable and persuasive in presentations
- Accessibility: Non-technical stakeholders can understand complex data through visuals
- Anomaly Detection: Outliers and unusual data points stand out visually
- Communication: A single chart can replace pages of text and tables

The History of Data Visualization

Data visualization has a rich history dating back centuries:

- 1786: William Playfair invented the bar chart, line chart, and pie chart
- 1854: John Snow's cholera map — used visualization to trace disease outbreak to a water pump
- 1869: Charles Minard's Napoleon's March — considered the best statistical graphic ever drawn
- 1970s-1980s: Edward Tufte established principles of excellent data visualization
- 2000s-Present: Interactive dashboards, web-based visualizations, and real-time data streaming

The Data Visualization Process

56. Define the objective: What question are you trying to answer? Who is the audience?
57. Understand the data: Explore the dataset, check data types, distributions, and relationships
58. Choose the right chart type: Match the visualization to the data and the message
59. Design the visualization: Apply design principles — simplicity, clarity, accuracy
60. Add context: Titles, labels, annotations, legends, source citations
61. Iterate: Get feedback, refine, and improve based on how users interpret it

6.2 Principles of Effective Data Visualization

1. Clarity Over Complexity

A visualization should communicate its message immediately. If the viewer needs more than 5 seconds to understand what the chart shows, it is too complex.

- Use simple, familiar chart types unless a complex chart is absolutely necessary
- Remove every element that does not contribute to understanding
- Label axes clearly with units of measurement
- Avoid 3D charts unless showing actual 3D data — they distort perception

2. Accuracy and Honesty

Never distort data to make a point. Misleading visualizations damage credibility and can lead to poor decisions.

- Always start bar charts at zero to avoid exaggerating differences
- Use consistent scales when comparing multiple charts

nasrin-a-mahin

- Do not cherry-pick date ranges to show only favorable trends
- Clearly indicate when data is estimated, projected, or incomplete
- Avoid dual Y-axes unless absolutely necessary — they can mislead

> *CRITICAL: Truncated Y-axes (starting above zero) make small differences look dramatic. This is acceptable only for displaying precise values where the baseline is irrelevant (e.g., stock prices), but NEVER for comparing magnitudes.*

3. Simplicity (Occam's Razor for Visualizations)

If a simpler chart type conveys the same message, always choose the simpler one.

Situation	Poor Choice	Better Choice
Showing parts of a whole	3D exploded pie chart	Simple 2D pie or stacked bar
Comparing 5 products	Radar chart	Simple bar chart
Trend over time	Stacked area with 10 categories	Line chart with top 3-5 categories
Single value	Gauge chart taking 1/4 of dashboard	Large number card (KPI)

6.3 Basic Chart Types

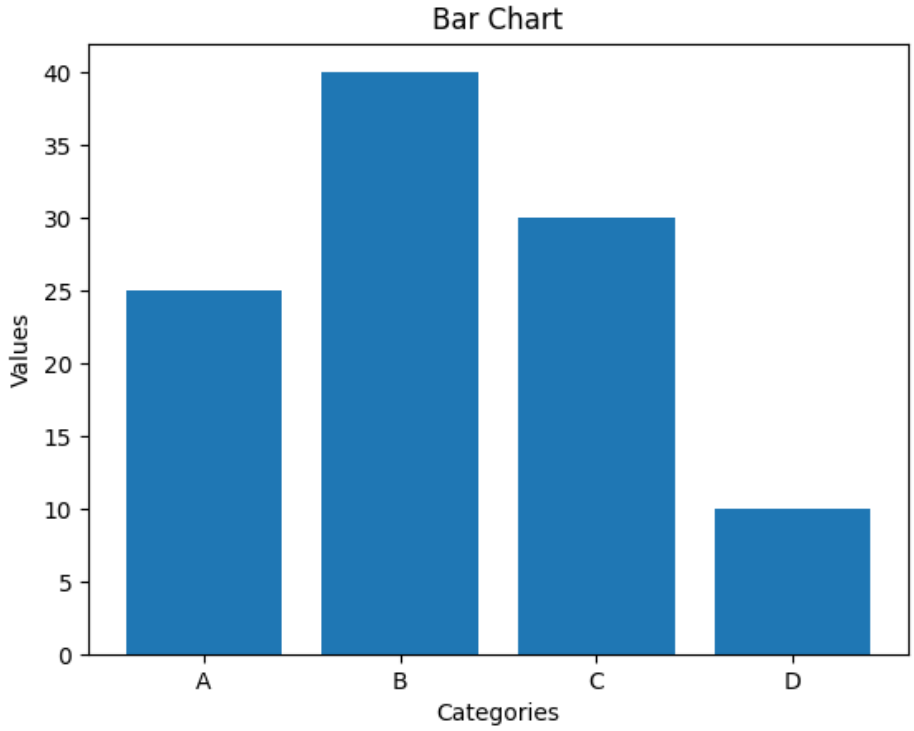
These five chart types solve 80% of visualization needs. Master them before exploring advanced charts.

1. Bar Chart — Comparing Categories

Bar charts compare values across discrete categories using horizontal or vertical bars.

- Best for: Comparing sales by region, products by revenue, departments by headcount
- Orientation: Horizontal bars for long category names; vertical (column) for time periods
- Sorting: Always sort bars by value (descending) unless there is a natural order (months, ratings)
- Spacing: Leave small gaps between bars for visual separation

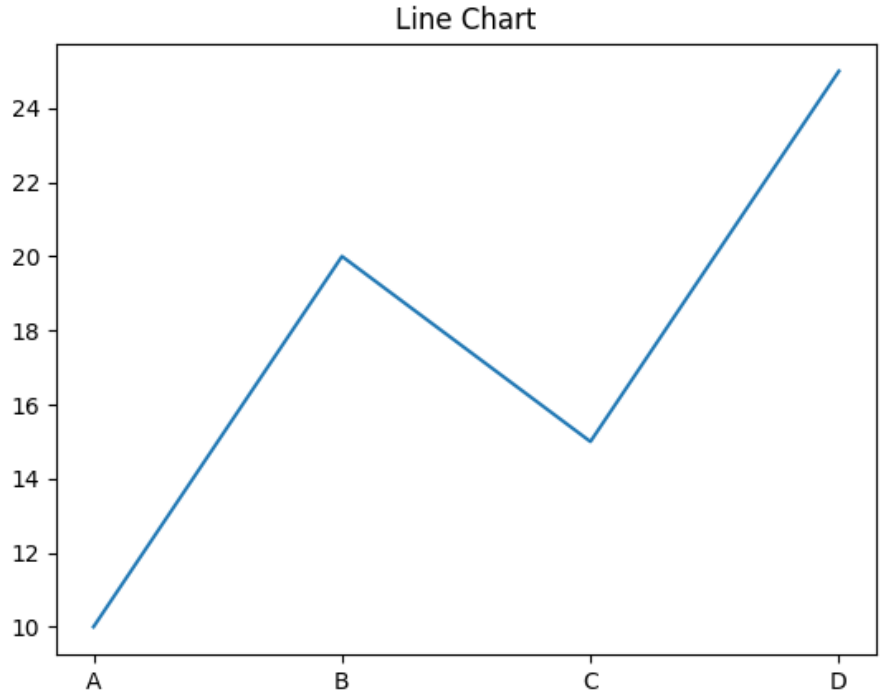
nasrin-a-nasrin



2. Line Chart — Trends Over Time

Line charts show how values change over time by connecting data points with lines.

- Best for: Stock prices, website traffic, temperature over days, sales trends
- Multiple lines: Use up to 5 lines maximum — more becomes unreadable
- Markers: Add markers to data points only if you have few points (<20)
- Smoothing: Use sparingly — it can hide important variations

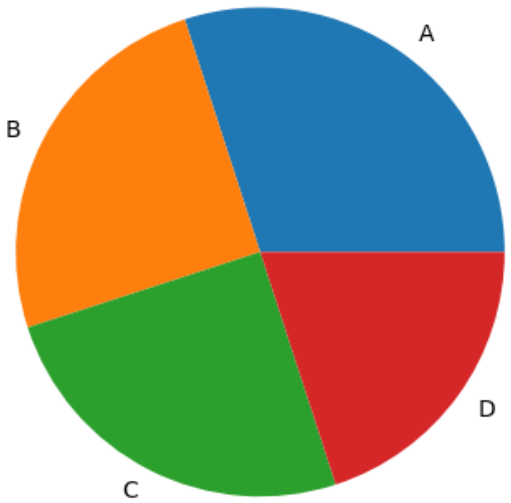


3. Pie Chart — Parts of a Whole

Pie charts show proportions of a total as slices of a circle.

> *WARNING: Pie charts are often overused and misused. Use them ONLY when you have 2-5 categories that sum to 100%. For 6+ categories, use a bar chart instead.*

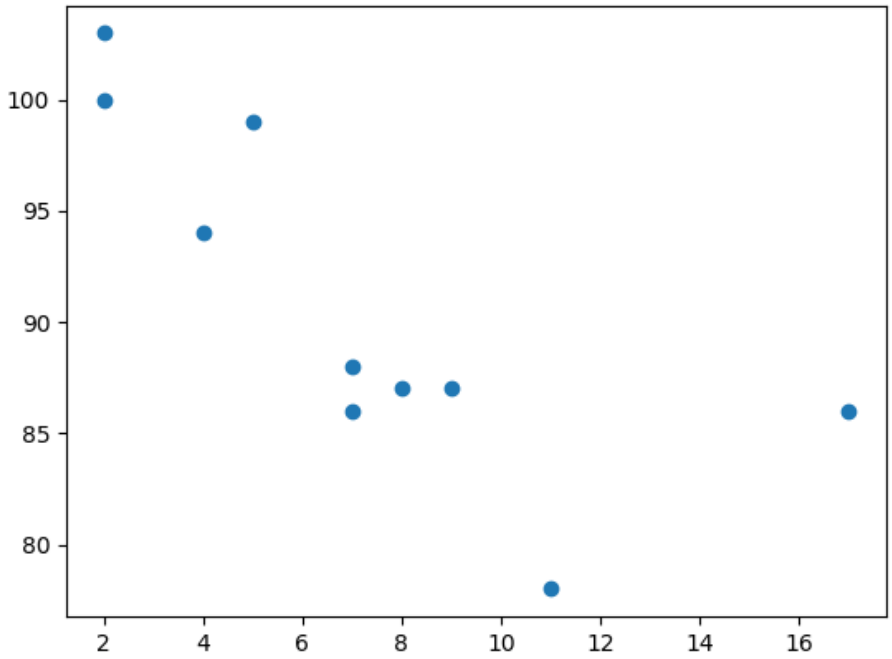
- Best for: Market share (3 competitors), budget allocation (5 departments)
- Limit to 2-5 slices — humans struggle to compare angles beyond this
- Start at 12 o'clock and arrange slices from largest to smallest clockwise
- Avoid 3D pie charts — they distort perception of slice sizes
- Consider donut charts if you need to display a central value



4. Scatter Plot — Relationships Between Variables

Scatter plots show the relationship (correlation) between two numeric variables.

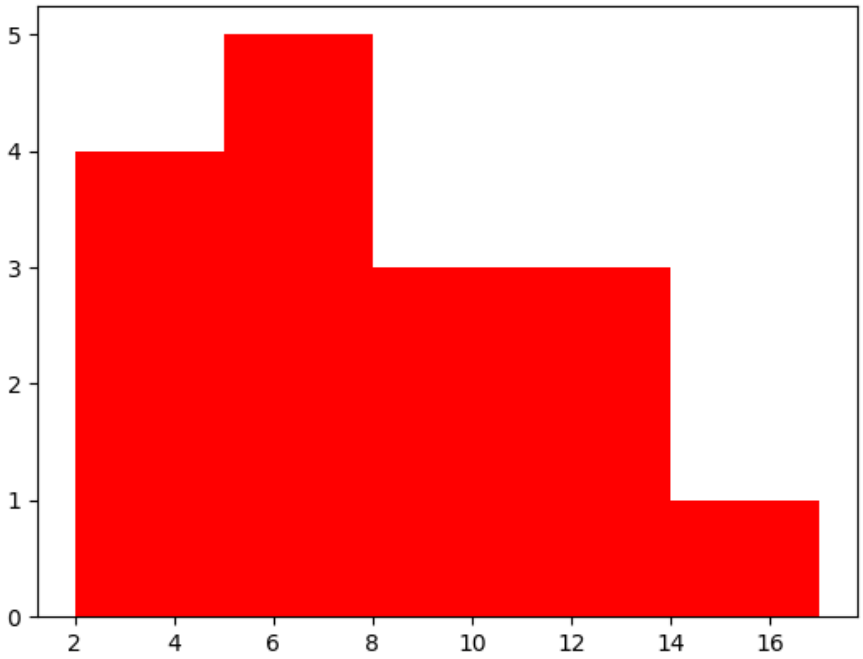
- Best for: Height vs. weight, advertising spend vs. revenue, test scores vs. study hours
- Add trend line: Include a regression line to show the overall relationship
- Size/Color: Use as a third dimension — bubble charts add a third variable via circle size
- Outliers: Scatter plots make outliers immediately visible



5. Histogram — Distribution of a Single Variable

Histograms show the frequency distribution of a numeric variable by grouping values into bins.

- Best for: Age distribution, salary ranges, response times, test scores
- Bins: Start with 10-20 bins and adjust based on data spread
- Not a bar chart: Bars touch in histograms (continuous data); gaps in bar charts (categories)
- Add mean/median lines to show central tendency

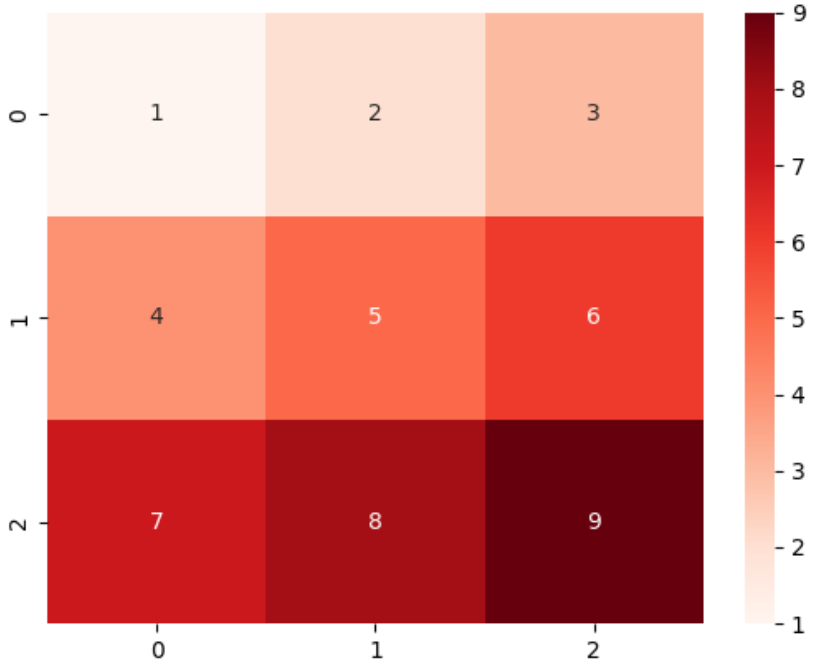


6.4 Advanced Chart Types

1. Heatmap — Matrix of Values by Color Intensity

Heatmaps use color intensity to show values across two dimensions in a matrix format.

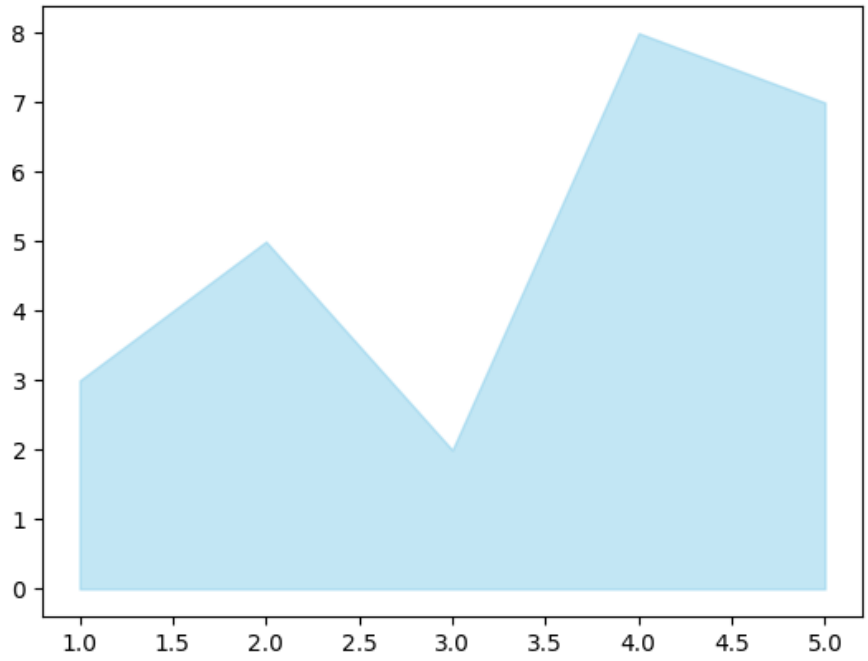
- Best for: Correlation matrices, time-based patterns (hour × day), geographic intensity
- Color scale: Sequential for magnitude, diverging for positive/negative
- Annotations: Add numbers in cells if precise values matter
- Example: Website traffic heatmap showing visits by hour of day and day of week



2. Area Chart — Cumulative Totals Over Time

Area charts are like line charts with the area below the line filled in.

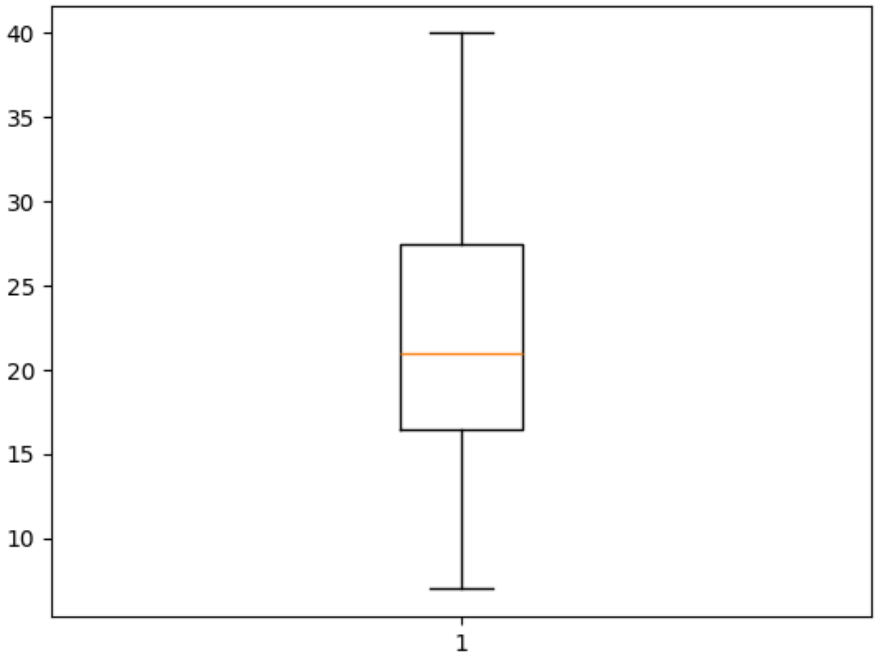
- Best for: Total market size over time, cumulative revenue, resource allocation
- Stacked area: Shows composition + total, but hard to read individual series
- Use sparingly: Line charts are often clearer for trends
- 100% stacked area: Shows changing proportions over time



3. Box Plot (Box-and-Whisker Plot)

Box plots show the distribution of data through quartiles and highlight outliers.

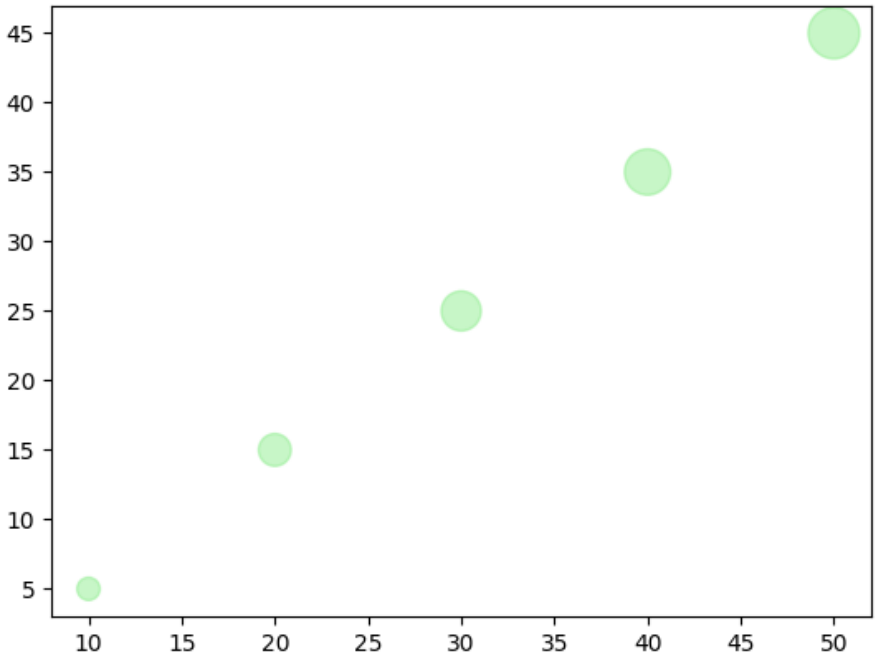
- Best for: Comparing distributions across groups, spotting outliers
- Components: Box shows IQR (25th-75th percentile); line in box is median; whiskers show range; dots are outliers
- Multiple groups: Compare salary distributions across departments or test scores across schools
- Not intuitive: Requires explanation for non-technical audiences



4. Bubble Chart — Scatter Plot with a Third Variable

Bubble charts extend scatter plots by encoding a third variable as circle size.

- Best for: Comparing companies by revenue (x), profit margin (y), and market cap (bubble size)
- Limit bubbles: Too many overlapping bubbles become unreadable
- Add labels: Label key bubbles to guide interpretation
- Consider alternatives: Sometimes two separate scatter plots are clearer



5. Waterfall Chart — Sequential Impact

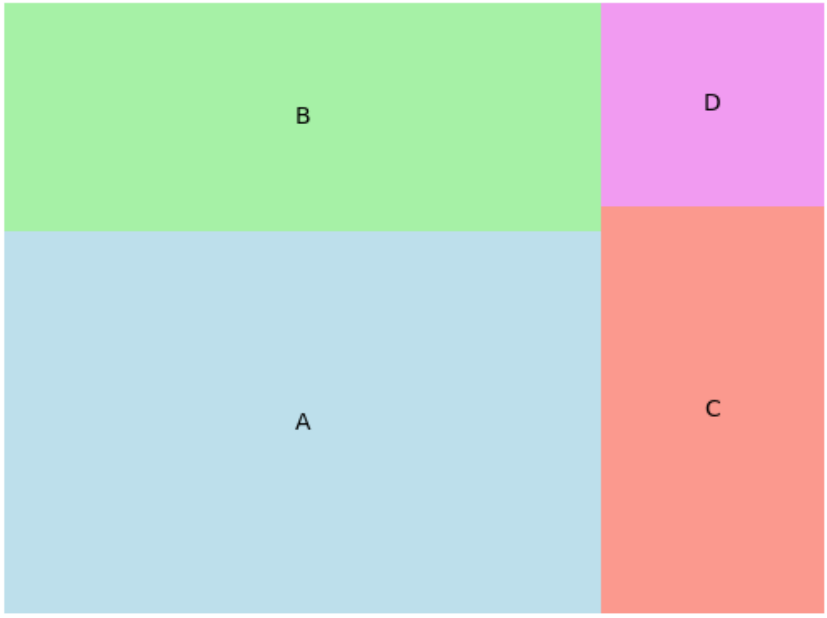
Waterfall charts show how a starting value changes through a series of positive and negative adjustments.

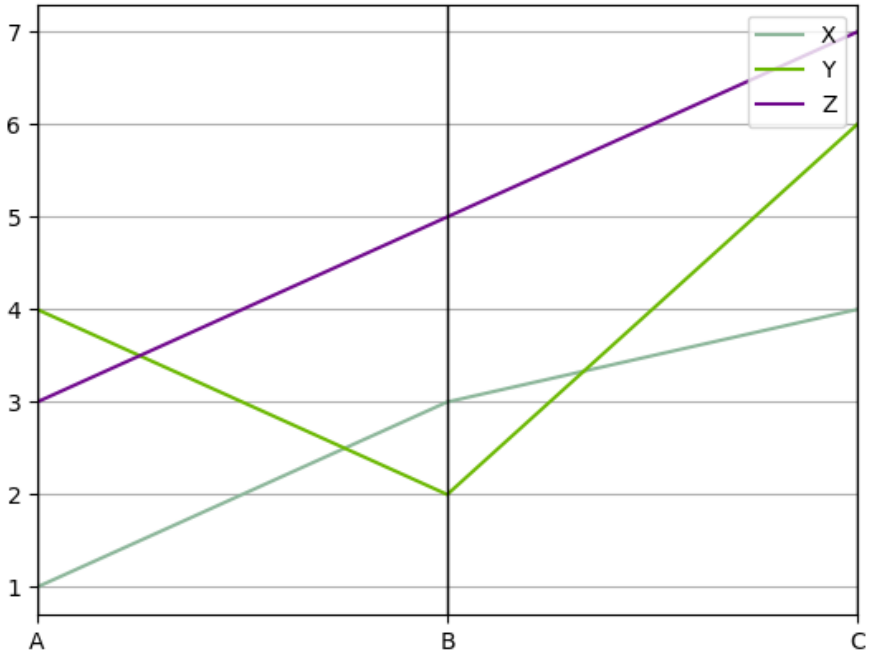
- Best for: P&L statements (revenue - costs = profit), bridge charts (last year to this year)
- Color coding: Green for increases, red for decreases, blue/gray for totals
- Connecting lines: Floating bars connect to show cumulative impact
- Example: Start with Q1 revenue, subtract costs, add Q2 revenue, subtract costs → Net Profit

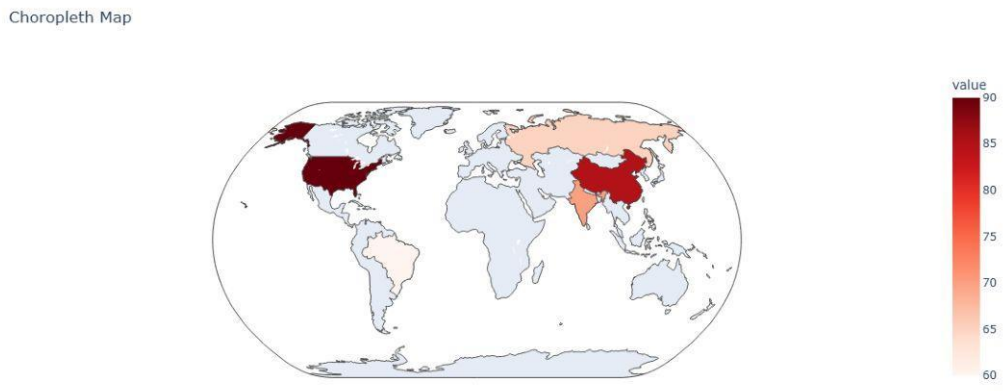
6. Treemap — Hierarchical Rectangles

Treemaps show hierarchical data as nested rectangles sized by value.

- Best for: Disk space usage, budget allocation across departments and teams, market share by category and brand
- Color: Use to show a second metric (e.g., growth rate) or just to distinguish categories
- Labels: Only label large enough rectangles — avoid clutter
- Alternative: Sunburst charts (circular treemap)


B
D
A
C


7
6
5
4
3
2
1
A
B
C
X
Y
Z



6.5 Specialized and Domain-Specific Charts

1. Gantt Chart — Project Timelines

Gantt charts show tasks, their durations, and dependencies in project management.

- Best for: Project schedules, resource planning, task dependencies
- Horizontal bars: Each bar represents a task across a timeline
- Dependencies: Arrows show which tasks depend on others
- Tools: Microsoft Project, Asana, Monday.com, or Excel with conditional formatting

2. Sankey Diagram — Flow and Movement

Sankey diagrams show flow quantities through a system with varying widths.

- Best for: Energy flow, budget allocation flow, customer journey paths
- Width = magnitude: Thicker flows represent larger quantities
- Node structure: Flows connect between nodes representing states or categories
- Example: Website traffic flow from source → landing page → signup → purchase

3. Funnel Chart — Conversion Rates

Funnel charts show progressive reduction through stages of a process.

- Best for: Sales pipeline, marketing conversion funnel, recruitment process
- Narrowing stages: Each stage shows fewer items than the previous
- Percentage labels: Show conversion rate between stages
- Example: 1000 visitors → 100 trials → 20 purchases (2% conversion)

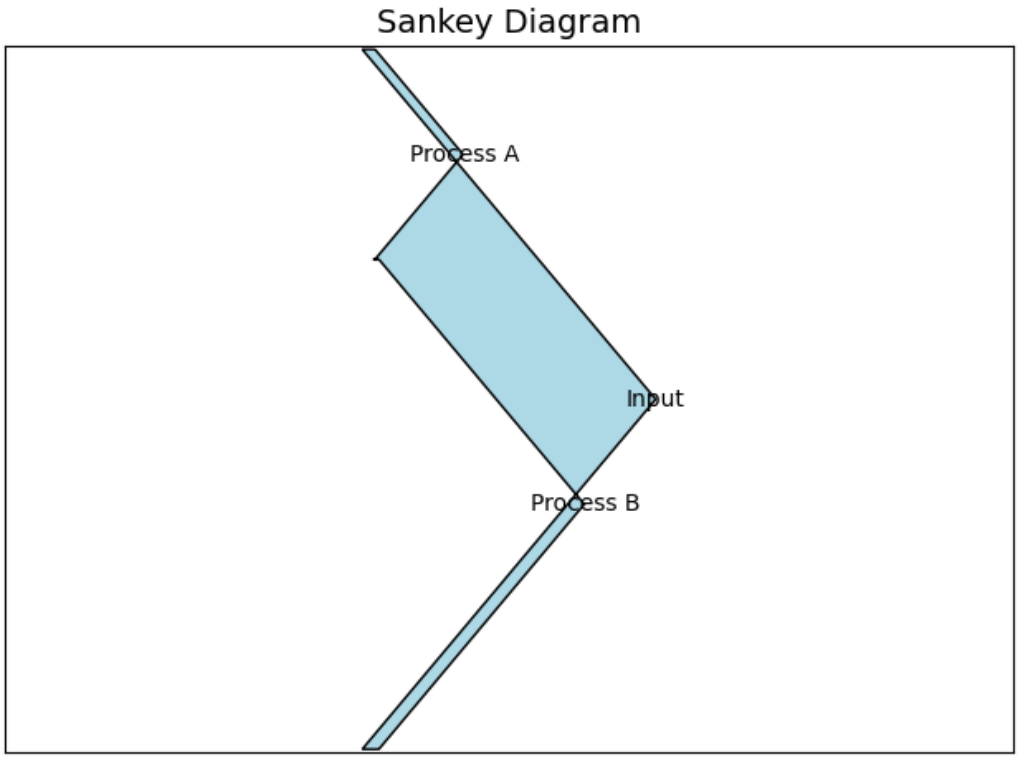
4. Radar Chart (Spider Chart) — Multivariate Comparison

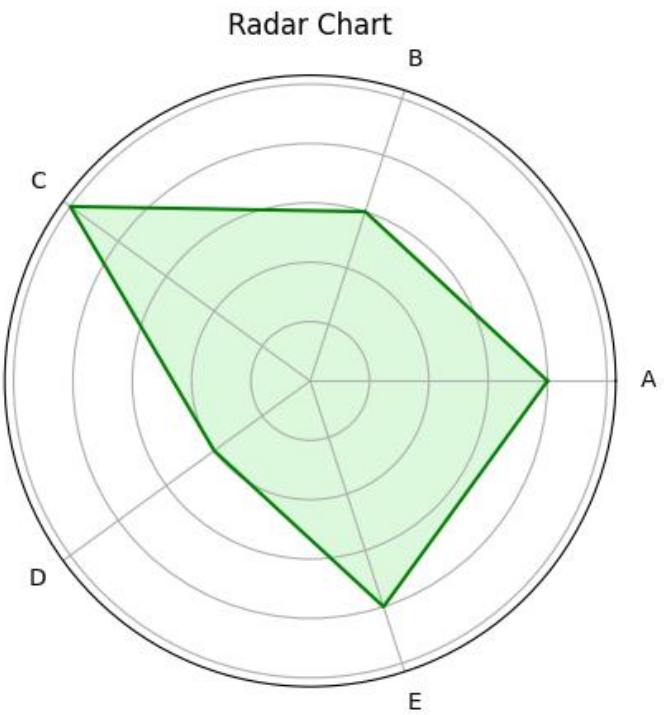
Radar charts display multiple variables on axes radiating from a central point.

- Best for: Skill assessments, product feature comparisons, performance scorecards
- Symmetric: Works best when all variables are on similar scales (0-100)

nasrin-a-mahi

- Overlapping shapes: Compare profiles of 2-3 entities
- Limitation: Hard to read precisely, not good for exact value comparison

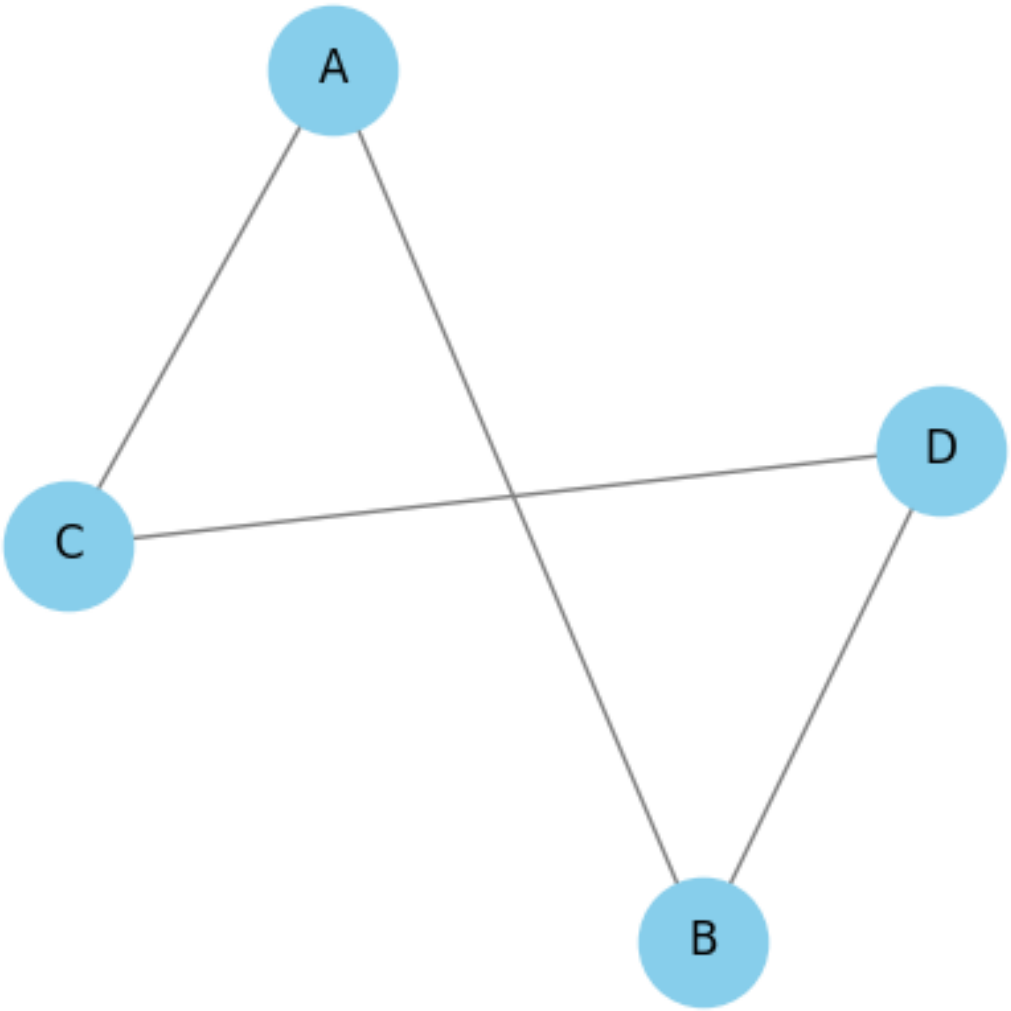




5. Choropleth Map — Geographic Heat Map

Choropleth maps use color shading to show values across geographic regions.

- Best for: Population density, election results, disease prevalence by country/state
- Equal-area projection: Avoid Mercator — it distorts size
- Color scale: Use sequential (low to high) or diverging (below/above average)
- Normalization: Show per-capita or percentage, not absolute counts (avoids large-area bias)



6. Network Graph — Relationships and Connections

Network graphs show entities (nodes) and their relationships (edges).

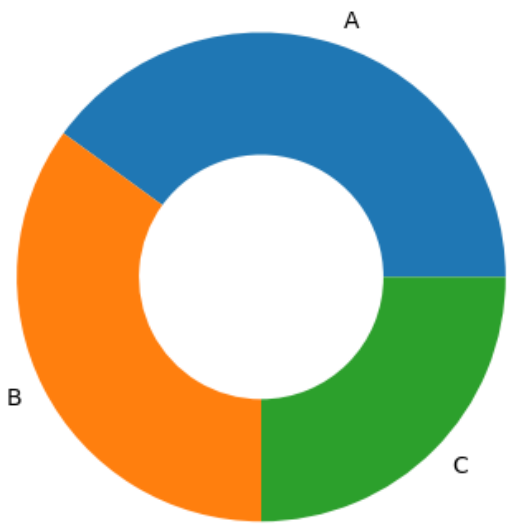
- Best for: Social networks, organizational structures, knowledge graphs
- Node size: Encode importance (e.g., number of connections)
- Edge thickness: Show strength of relationship
- Layout algorithms: Force-directed layouts naturally cluster related nodes

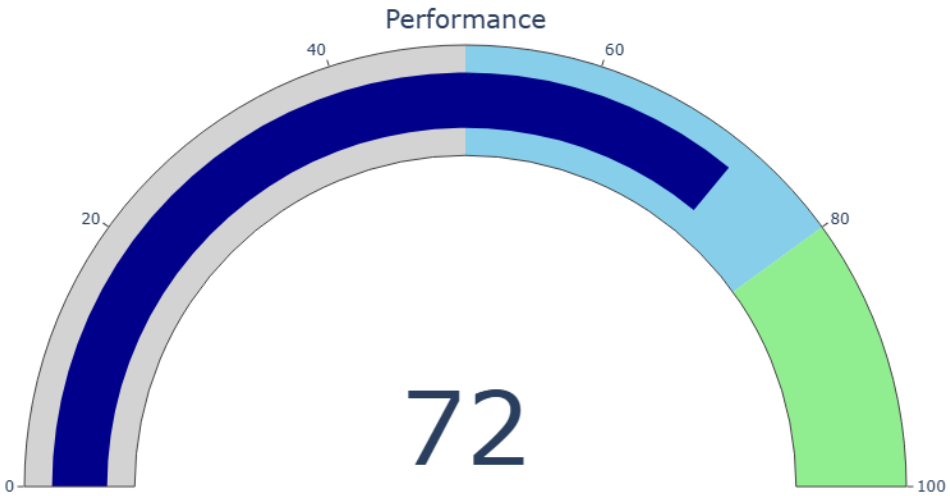
7. Bullet Graph — KPI Progress

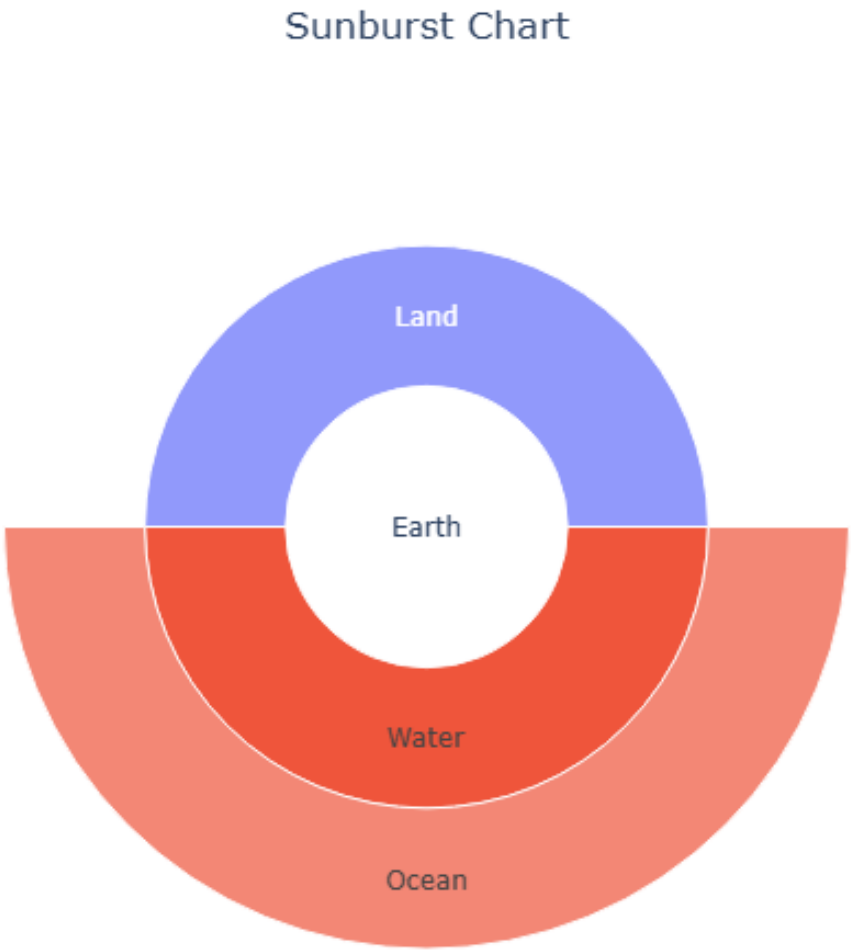
Bullet graphs show progress toward a goal in a compact format — a modern alternative to gauge charts.

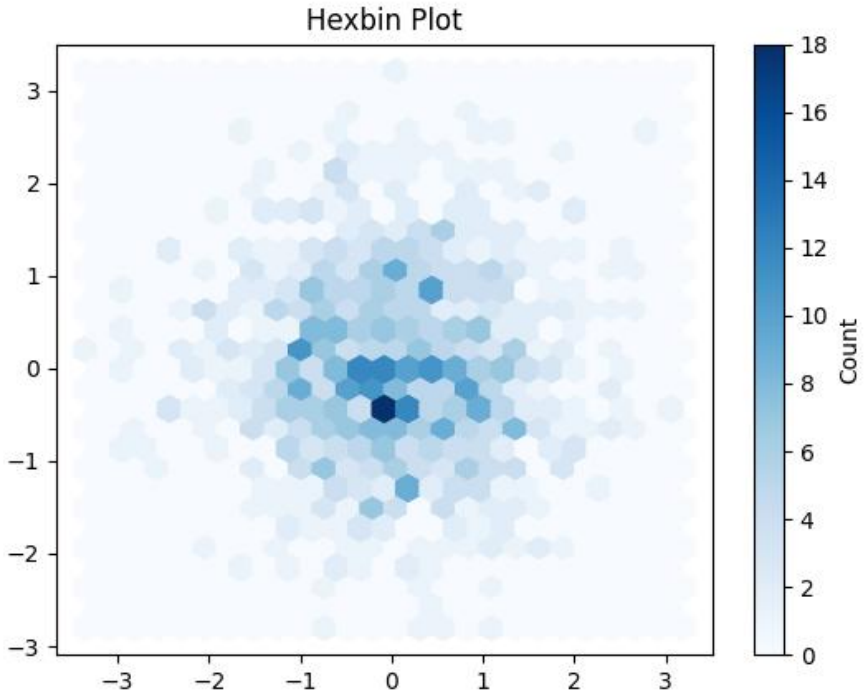
- Best for: Dashboards showing KPIs vs. targets

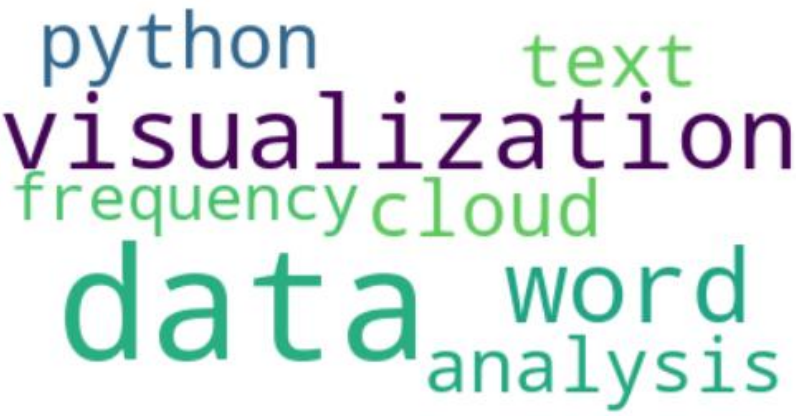
- Components: Actual value (bar), target (marker), ranges (background shading for poor/fair/good)
- Space-efficient: Shows more information in less space than a gauge
- Example: Sales: $45K (actual) vs. $50K target, with poor (<$30K), fair ($30-45K), good (>$45K) zones






Sunburst Chart
Land
Earth
Water
Ocean


Hexbin Plot
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
18
16
14
12
10
8
6
4
2
0
Count



6.6 Choosing the Right Chart Type

The choice of chart type depends on three factors: the data structure, the message you want to convey, and the audience.

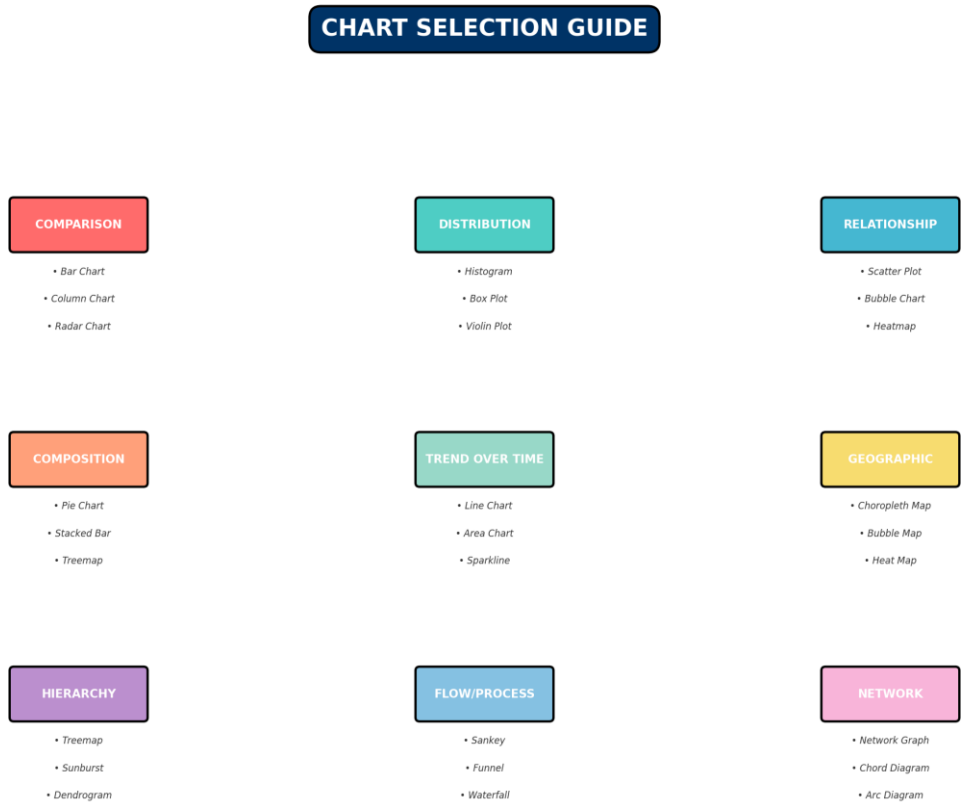


Figure 6.1 — Chart Selection Guide by Purpose

Decision Framework

Question	If Yes, Consider
Comparing values across categories?	Bar chart, column chart, radar chart
Showing change over time?	Line chart, area chart, sparkline
Displaying parts of a whole?	Pie chart, donut chart, stacked bar, treemap
Showing relationship between 2 variables?	Scatter plot, bubble chart, heatmap
Displaying distribution of a variable?	Histogram, box plot, violin plot
Showing geographic patterns?	Choropleth map, bubble map, heat map
Illustrating hierarchy?	Treemap, sunburst, dendrogram
Visualizing flow or process?	Sankey diagram, funnel chart, waterfall
Displaying network relationships?	Network graph, chord diagram

6.7 Color Theory in Data Visualization

Color is one of the most powerful tools in visualization — and one of the most misused.

Types of Color Palettes

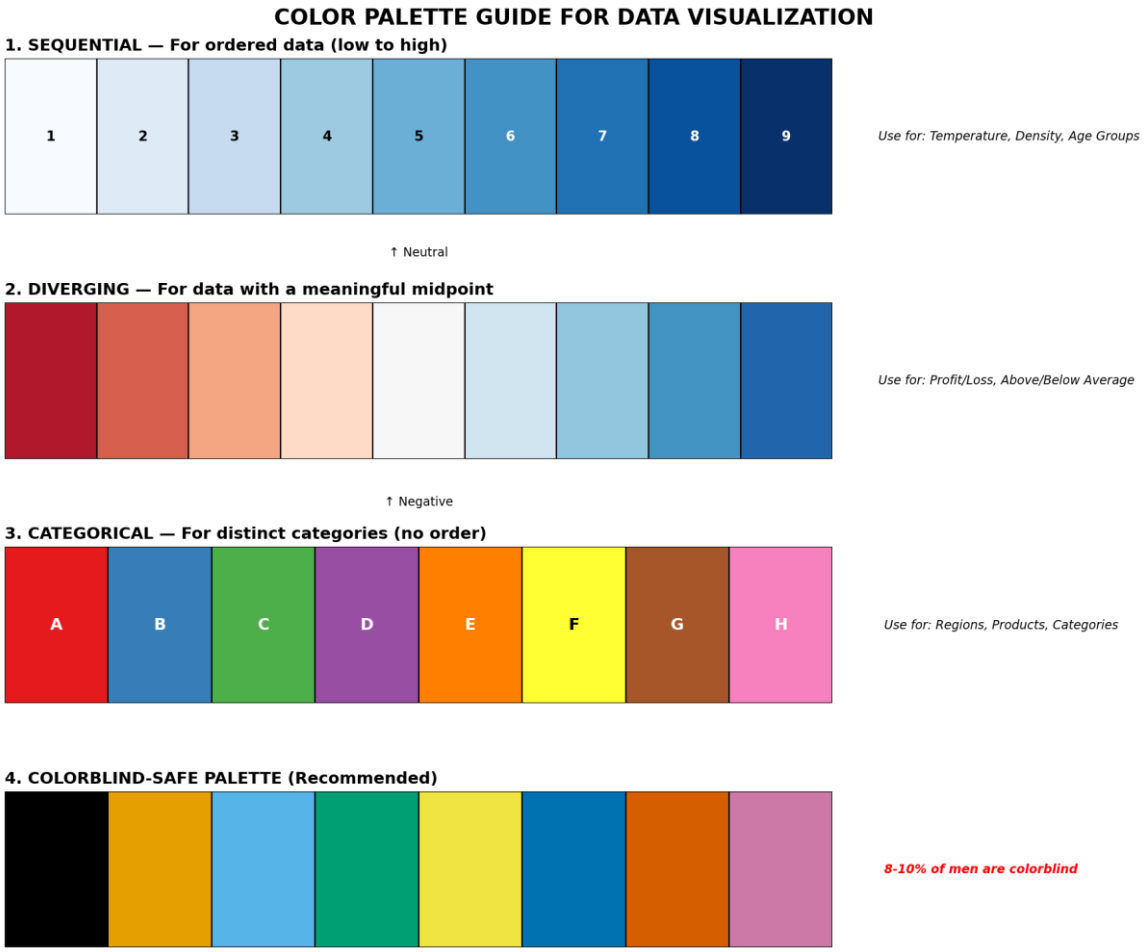


Figure 6.2 — Color Palette Types and When to Use Them

1. Sequential Palettes — For Ordered Data

Sequential palettes use a single hue that transitions from light to dark.

- Use for: Temperature (cold to hot), age (young to old), density (sparse to dense)
- Examples: Light blue → Dark blue, White → Red, Yellow → Green → Blue
- Rule: Darker = higher value in most contexts

2. Diverging Palettes — For Data with a Meaningful Midpoint

Diverging palettes use two contrasting hues with a neutral midpoint.

- Use for: Profit/loss, above/below average, positive/negative sentiment
- Examples: Red ← White → Blue, Orange ← Gray → Purple
- Midpoint: Should represent zero, neutral, or average depending on context

3. Categorical Palettes — For Distinct Categories

Categorical palettes use maximally distinct colors with no inherent order.

- Use for: Product categories, regions, departments — any unordered groups
- Limit: Use 8-10 colors maximum — more becomes indistinguishable
- Avoid: Rainbow scales — they imply order where none exists
- Colorblind-safe: Use palettes tested for deuteranopia (red-green colorblindness)

Color Accessibility — Designing for Colorblindness

> *8-10% of men (1 in 12) and 0.5% of women have some form of color vision deficiency. Always test visualizations for colorblind accessibility.*

- Use colorblind-safe palettes (e.g., Viridis, Okabe-Ito palette)
- Never rely on color alone — use patterns, labels, or shapes as well
- Avoid red-green combinations — use blue-orange or purple-yellow instead
- Test with simulators: Coblis, Color Oracle, or browser extensions
- Provide alternative views: Allow users to switch to grayscale or high-contrast mode

6.8 Design Principles – Data-Ink Ratio and Chart Junk

Edward Tufte's Data-Ink Ratio

> *Data-Ink Ratio = (Ink used to display data) / (Total ink used in the graphic). Maximize this ratio by removing non-essential elements.*

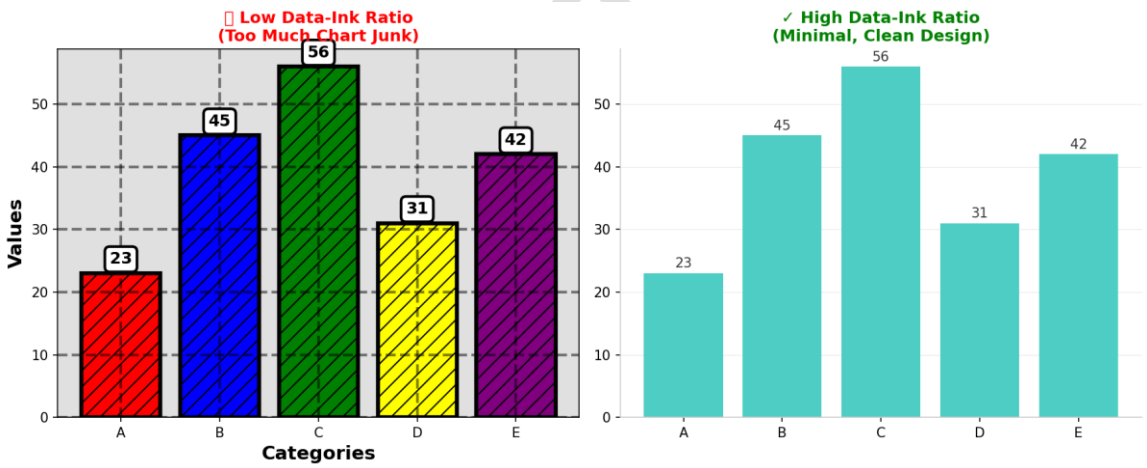


Figure 6.3 — Data-Ink Ratio: Bad vs. Good Examples

What is Chart Junk?

Chart junk refers to unnecessary or distracting visual elements that do not enhance understanding:

- 3D effects on 2D charts (shadows, depth, rotation)
- Excessive gridlines — use light gray, minimal gridlines only
- Decorative images or icons that add no information
- Unnecessary borders, boxes, and shading
- Too many colors in a single chart

- Ornate fonts or oversized legends

Simplification Checklist

62. Remove background color or set to white/light gray
63. Eliminate or minimize gridlines — only horizontal lines for bar/column charts
64. Remove borders around the chart area
65. De-emphasize axes — use light gray instead of black
66. Remove redundant labels — if data labels are shown, axis labels may not be needed
67. Use direct labeling instead of legends when possible
68. Remove chart title if the surrounding text already explains the chart

6.9 Dashboard Design Best Practices

Dashboards consolidate multiple visualizations into a single view for monitoring and decision-making.

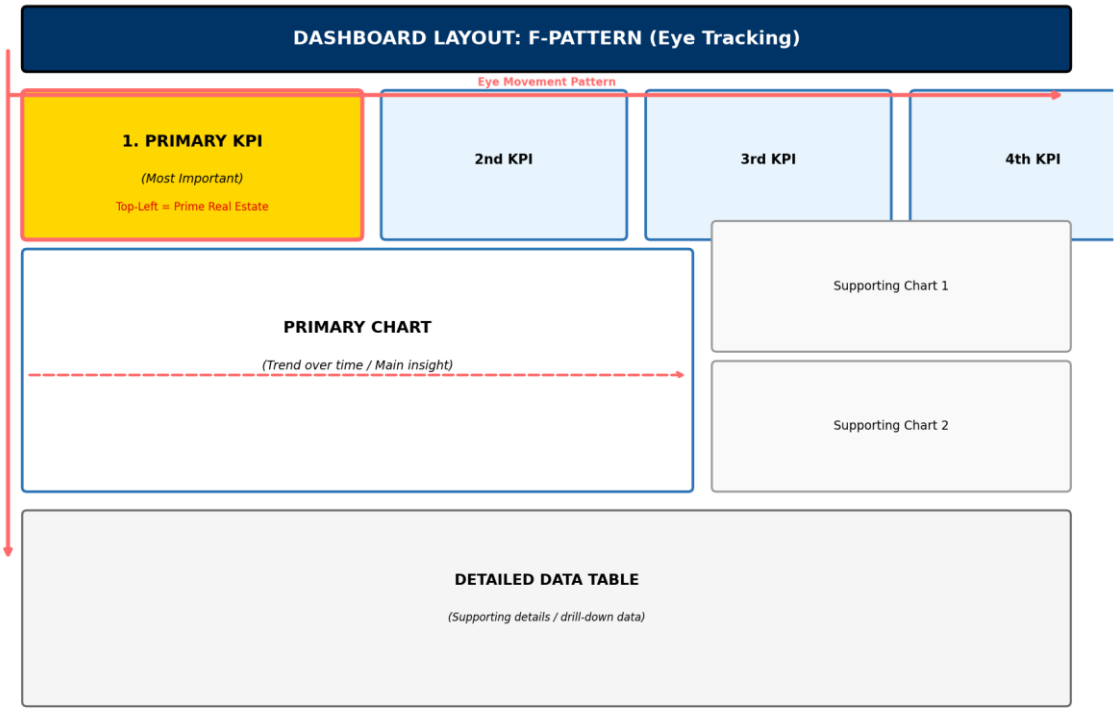


Figure 6.4 — Dashboard Layout: F-Pattern and Visual Hierarchy

The F-Pattern (Eye-Tracking Studies)

Eye-tracking research shows users scan dashboards in an F-shaped pattern: left to right across the top, then down the left side, then right again.

- Top-left: Place the MOST important metric (primary KPI) in the top-left corner
- Top row: Secondary KPIs across the top
- Left column: Next most important information flows down the left
- Center: Main chart or primary visualization occupies center real estate

- Bottom/Right: Supporting details, tables, and secondary charts

Dashboard Design Rules

- One dashboard = One purpose: Don't try to serve multiple audiences with one dashboard
- Above the fold: Key metrics visible without scrolling
- 5-second rule: Users should grasp the main insight within 5 seconds
- Consistent layout: Use a grid system for alignment
- Limit chart types: Stick to 3-4 chart types per dashboard
- Interactive filters: Allow users to slice data by date, region, category
- Context: Always include date range, data source, and last updated timestamp

KPI Cards vs. Charts

Use KPI Cards When	Use Charts When
Single value is sufficient	Trend or comparison is important
% change vs. prior period	Historical context matters
Simple yes/no status	Distribution or outliers need to be visible
Executive summary dashboard	Analyst/operational dashboard

6.10 Interactive Visualizations

Interactive visualizations allow users to explore data dynamically — filtering, drilling down, and discovering insights on their own.

Key Interactive Features

- Tooltips: Hovering over data points reveals detailed values
- Filtering: Dropdowns or slicers to filter by date, category, region
- Drill-down: Click on a high-level summary to see underlying details
- Zoom and pan: For time series or maps with dense data
- Cross-highlighting: Selecting data in one chart filters related charts
- Export: Allow users to download data or images for reports

Tools for Interactive Dashboards

Tool	Type	Strengths
Tableau	Desktop + Cloud	Drag-and-drop, beautiful default styles, fast
Power BI	Desktop + Cloud	Microsoft integration, DAX language, low cost
Plotly Dash	Python framework	Full customization, embed in web apps
D3.js	JavaScript library	Complete control, web-based, steep learning curve
Google Data Studio	Cloud	Free, integrates with Google Analytics/Sheets
Looker	Cloud BI	SQL-based, enterprise-grade, Google Cloud integration

nasin-a-mobin

6.11 Data Storytelling

Data storytelling combines data, visualization, and narrative to communicate insights persuasively and memorably.

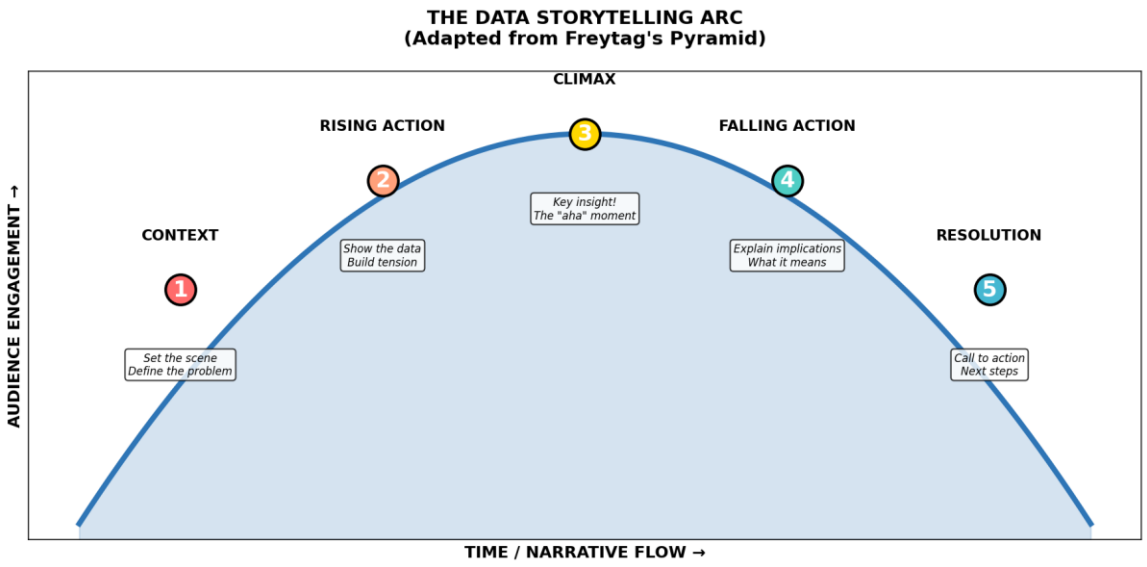


Figure 6.5 — The Data Storytelling Arc

The Three Elements of Data Storytelling

- Data: The quantitative foundation — accurate, relevant, and reliable
- Visualization: The visual representation — clear, honest, and well-designed
- Narrative: The story that ties it together — context, conflict, resolution

The Storytelling Arc (Freytag's Pyramid Adapted)

69. Context: Set the scene. What is the business problem? Why does it matter? Who is affected?
70. Rising Action: Show the data. Build tension by revealing unexpected patterns or concerning trends.
71. Climax: The key insight. The "aha" moment that answers the question or reveals the problem.
72. Falling Action: Explain the implications. What does this insight mean for the business or audience?
73. Resolution: Call to action. What should the audience do with this information? Next steps.

Storytelling Techniques

- Annotations: Add text callouts to highlight key data points or inflection points
- Comparison: Show before/after, actual vs. target, us vs. competitors
- Progression: Guide the audience through a logical sequence of charts
- Emotion: Use real examples or case studies to make data human
- Simplicity: One insight per slide/chart — don't overwhelm

> *The goal of data storytelling is not just to inform — it is to persuade and inspire action. Data alone rarely changes minds; stories do.*

6.12 Common Visualization Mistakes to Avoid

1. Dual Y-Axes with Different Scales

Dual Y-axes allow you to show two series with different units on the same chart — but they are easily manipulated to exaggerate or hide correlation.

- Problem: By adjusting the scales independently, you can make any two variables appear correlated
- When acceptable: Only if both scales start at zero and the relationship is genuinely meaningful
- Better alternative: Use two separate charts or normalize both variables to 0-100 scale

2. Truncated Y-Axis (Not Starting at Zero)

Starting the Y-axis above zero exaggerates small differences.

- When acceptable: For showing precise changes in stable metrics (stock prices, temperatures)
- When misleading: When comparing magnitudes or implying large differences
- Fix: Always start at zero for bar charts; clearly label if you don't

3. Using Pie Charts for Too Many Categories

Pie charts become unreadable beyond 5 slices because humans cannot accurately compare angles.

- Problem: 10-slice pie chart forces tiny, indistinguishable wedges
- Fix: Use a sorted bar chart instead — easier to compare and label

4. 3D Charts That Distort Perception

3D effects make charts look impressive but distort data by adding perspective.

- Problem: Front slices of 3D pie charts appear larger; 3D bars are hard to read accurately
- Fix: Use 2D charts unless showing actual 3D data (x, y, z coordinates)

5. Inconsistent Scales Across Panels

When comparing multiple charts side-by-side, inconsistent scales mislead.

- Problem: Chart A ranges 0-100, Chart B ranges 0-10 — visually appears similar but isn't
- Fix: Use identical scales for all panels when comparing the same metric

6. Cherry-Picking Date Ranges

Selecting a specific time period to show only favorable trends while hiding unfavorable periods.

- Problem: "Sales grew 50% from Q2 to Q3!" — but omits that Q1 to Q2 dropped 60%
- Fix: Show the full time series or clearly state why a specific period was chosen

nasrin-a-mahin

7. Overuse of Color

Using too many colors creates visual chaos and makes the chart hard to interpret.

- Problem: 15 different colors in a single chart — impossible to match legend to chart
- Fix: Limit to 5-7 colors maximum; use shades of one color for related categories

6.13 Visualization Tools and Software

Business Intelligence (BI) Tools

Tool	Pricing	Best For	Learning Curve
Tableau	$70/user/month	Beautiful visualizations, fast prototyping	Medium
Power BI	$10/user/month	Microsoft integration, enterprise	Medium
Looker	Enterprise only	SQL-based, scalable, Google Cloud	High
Qlik Sense	$30/user/month	Associative data model, fast filtering	Medium
Google Data Studio	Free	Simple dashboards, Google integration	Low

Programming Libraries

Library	Language	Best For
Matplotlib	Python	Static charts, publication-quality plots
Seaborn	Python	Statistical visualizations, beautiful defaults
Plotly	Python/R/JS	Interactive web-based charts
ggplot2	R	Grammar of graphics, publication-quality
D3.js	JavaScript	Custom web visualizations, full control
Chart.js	JavaScript	Simple web charts, easy to implement

Specialized Tools

- Gephi: Network graph visualization and analysis
- Flourish: Animated and interactive storytelling visuals
- RAWGraphs: Quick exploratory charts from CSVs
- Datawrapper: Publication-ready charts for journalists
- Mermaid: Diagrams and flowcharts from text (Markdown-based)

6.14 Accessibility in Data Visualization

Accessible visualizations ensure that everyone, including people with disabilities, can understand and interact with data.

Visual Accessibility

- Colorblind-safe palettes: Test all visualizations with colorblind simulators
- Text alternatives: Provide alt text descriptions for screen readers
- High contrast: Ensure sufficient contrast between foreground and background (WCAG AA: 4.5:1)
- Patterns and textures: Use in addition to color (hatching, dotted lines) to distinguish categories
- Large text: Use readable font sizes (minimum 12pt for body text, 18pt for headings)

Interactive Accessibility

- Keyboard navigation: All interactive elements accessible via Tab/Enter/Arrow keys
- Screen reader support: Label all interactive elements with ARIA attributes
- Focus indicators: Visible outlines when elements are focused
- Sufficient click targets: Minimum 44×44 pixels for touch interfaces

Cognitive Accessibility

- Simple language: Avoid jargon; define technical terms
- Clear labels: Every axis, legend, and data series clearly labeled
- Consistent design: Use the same chart types and color schemes across a dashboard
- Progressive disclosure: Start simple, allow users to drill down for detail

> *Accessibility is not just the right thing to do — it expands your audience and often improves usability for everyone.*

nasrin-a mahin

UNIT VII – R Programming for Data Analysis

7.1 Introduction to R

R is a powerful, open-source programming language and environment designed specifically for statistical computing and data analysis. Created in 1993 by Ross Ihaka and Robert Gentleman at the University of Auckland, R has grown into the de facto standard for statistical analysis in academia, research, and increasingly in industry.

Unlike general-purpose languages like Python, R was built from the ground up with data analysis in mind. Its syntax is optimized for vectorized operations, statistical modeling, and data visualization. R's greatest strength lies in its extensive ecosystem of over 19,000 packages on CRAN (Comprehensive R Archive Network), covering every imaginable statistical method and domain-specific application.

Why R for Data Analysis?

- Statistical Focus — R provides built-in functions for virtually every statistical test and model: regression, ANOVA, time series, survival analysis, Bayesian methods.
- Visualization Excellence — ggplot2, the grammar of graphics, produces publication-quality plots with minimal code. Unmatched for exploratory data analysis.
- Reproducibility — R Markdown integrates code, results, and narrative into a single document, ensuring analyses are transparent and replicable.
- Community and Packages — CRAN hosts specialized packages for every domain: genomics (Bioconductor), finance (quantmod), text mining (tm), machine learning (caret, mlr).
- Free and Open Source — No licensing fees. Complete transparency into how every function works.

RStudio IDE

RStudio is the most popular integrated development environment for R. It provides a four-pane interface: Source editor (top-left), Console (bottom-left), Environment/History (top-right), Files/Plots/Packages (bottom-right). RStudio includes code completion, syntax highlighting, integrated help, version control, and project management — essential tools for professional data analysis work.

Installation: Download R from r-project.org, then install RStudio from posit.co/download/rstudio-desktop. Both are free. RStudio requires R to be installed first.

nasrin-a-mahin

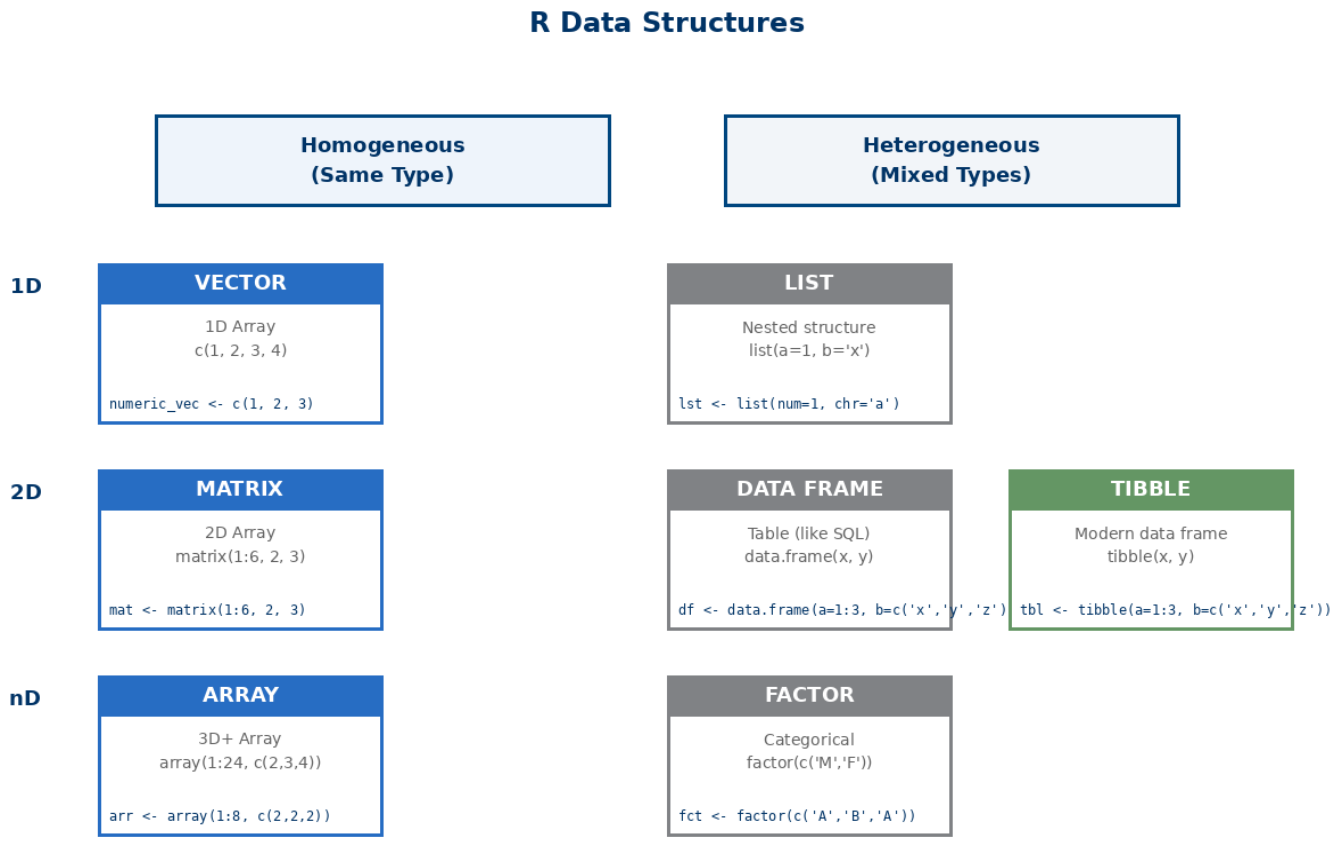
R vs Python for Data Analysis

Aspect	R	Python
Primary Use	Statistical analysis, academia, research	General programming, web, automation, ML
Statistical Functions	Built-in, extensive	Via libraries (NumPy, SciPy, statsmodels)
Visualization	ggplot2 — superior for statistical plots	matplotlib, seaborn — general-purpose
Syntax	Optimized for data frames and vectors	General-purpose, object-oriented
Package Ecosystem	19,000+ on CRAN (stats-focused)	PyPI (broader, general-purpose)
Learning Curve	Steeper for programmers, natural for statisticians	Gentle, Pythonic principles
Industry Adoption	Finance, pharma, academia	Tech companies, startups, web

Verdict: R excels at statistical modeling and academic research. Python dominates in production ML pipelines and software integration. Many data scientists use both.

7.2 R Data Types and Structures

R organizes data into structures optimized for statistical operations. Understanding these structures is fundamental — every analysis workflow depends on choosing the right container for your data.



Atomic Data Types

R has six atomic types (the building blocks), but data analysts primarily use four:

Type	Description	Example	Check Function
numeric	Real numbers (double precision)	3.14, -5.2, 1000	is.numeric()
integer	Whole numbers (add L suffix)	5L, -3L, 100L	is.integer()
character	Text strings	"Hello", 'R', "2025"	is.character()
logical	Boolean values	TRUE, FALSE, T, F	is.logical()

1. Vectors — The Foundation

Vectors are 1-dimensional arrays containing elements of the same type. They are the most fundamental data structure in R — even a single number is a vector of length 1.

```
# Creating vectors
numeric_vec <- c(1, 2, 3, 4, 5)
char_vec <- c("apple", "banana", "cherry")
logical_vec <- c(TRUE, FALSE, TRUE, TRUE)


# Sequences
seq_vec <- 1:10            # 1 2 3 ... 10
seq_vec2 <- seq(0, 1, 0.1)   # 0.0 0.1 0.2 ... 1.0
rep_vec <- rep(5, times=3)   # 5 5 5


# Vectorized operations (element-wise)
x <- c(1, 2, 3, 4)
y <- c(10, 20, 30, 40)
x + y                      # 11 22 33 44
x * 2                      # 2 4 6 8
sqrt(x)                    # 1.000 1.414 1.732 2.000


# Indexing (1-based, unlike Python's 0-based)
numeric_vec[1]             # First element: 1
numeric_vec[c(1,3,5)]      # Elements 1, 3, 5
numeric_vec[-2]            # All except 2nd element
numeric_vec[numeric_vec > 2] # Logical subsetting: 3 4 5
```

2. Matrices — 2D Homogeneous Arrays

```
# Create matrix
mat <- matrix(1:6, nrow=2, ncol=3)
#      [,1] [,2] [,3]
# [1,]    1    3    5
```

```
# [2,]    2    4    6


# By row instead of by column
mat2 <- matrix(1:6, nrow=2, byrow=TRUE)


# Matrix operations
dim(mat)          # Dimensions: 2 3
nrow(mat)         # 2
ncol(mat)         # 3
t(mat)            # Transpose


# Indexing
mat[1, 2]         # Row 1, Column 2
mat[1, ]          # Entire first row
mat[, 3]          # Entire third column


# Matrix algebra
A <- matrix(c(1,2,3,4), nrow=2)
B <- matrix(c(5,6,7,8), nrow=2)
A + B             # Element-wise addition
A %*% B           # Matrix multiplication
solve(A)          # Matrix inverse
```

3. Data Frames — The Workhorse

Data frames are 2-dimensional tables where each column can be a different type (like a SQL table or Excel sheet). This is the structure you will use 90% of the time in data analysis.

```
# Create data frame
df <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 35),
  salary = c(50000, 60000, 70000),
```

```
  employed = c(TRUE, TRUE, FALSE)
)


# Inspect
head(df)          # First 6 rows
tail(df, 2)       # Last 2 rows
str(df)           # Structure
summary(df)       # Summary statistics
names(df)         # Column names
nrow(df); ncol(df)


# Access columns
df$name                  # By name (returns vector)
df[, "age"]              # By name (alternative)
df[[2]]                  # By position


# Access rows
df[1, ]                  # First row
df[df$age > 28, ]        # Rows where age > 28


# Add column
df$bonus <- df$salary * 0.1


# Read CSV into data frame
sales <- read.csv("sales_data.csv")
```

4. Lists — Heterogeneous Containers

Lists can hold elements of different types and lengths. Used for storing complex, nested data structures and returning multiple values from functions.

```
# Create list
my_list <- list(
```

```
  numbers = c(1, 2, 3),
  text = "Hello",
  matrix = matrix(1:4, nrow=2),
  dataframe = data.frame(x=1:3, y=c("a","b","c"))
)

# Access elements
my_list$numbers        # By name
my_list[[1]]           # By position (double bracket)
my_list[["text"]]      # By name (alternative)

# Nested access
my_list$dataframe$x    # Column x from dataframe element
```

5. Factors — Categorical Data

Factors store categorical variables with predefined levels. Essential for statistical modeling and visualization.

```
# Create factor
gender <- factor(c("M", "F", "M", "F", "M"))
levels(gender)                  # "F" "M"
table(gender)                   # Frequency count

# Ordered factors
education <- factor(
  c("High School", "Bachelor", "Master", "Bachelor"),
  levels = c("High School", "Bachelor", "Master", "PhD"),
  ordered = TRUE
)

# Convert character to factor
df$category <- as.factor(df$category)
```

tibble vs data.frame: tibble (from tidyverse) is a modern data frame: never converts strings to factors, better printing, stricter subsetting. Use tibble() instead of data.frame() in modern R code.

nasrin-a-mahin

7.3 Control Structures and Functions

Control structures direct the flow of program execution. Functions encapsulate reusable logic. Mastering both is essential for automating analyses and building custom tools.

R Control Structures

IF - ELSE

```
if (condition) {
  # execute if TRUE
} else {
  # execute if FALSE
}
```

FOR LOOP

```
for (i in 1:10) {
  print(i)
}
```

WHILE LOOP

```
while (condition) {
  # repeat
  # update condition
}
```

REPEAT (infinite)

```
repeat {
  # execute
  if (condition) break
}
```

FUNCTIONS

```
# Define function
my_function <- function(x, y = 10) {
  result <- x + y
  return(result)
}

# Call function
output <- my_function(5, 3)   # output = 8
```

⚡ Vectorization: R's Superpower

```
# SLOW: Loop approach
result <- numeric(1000)
for (i in 1:1000) {
  result[i] <- x[i] * 2
}

# FAST: Vectorized approach
result <- x * 2    # Operates on entire vector at once!

# Vectorized functions: sum(), mean(), sqrt(), log(), etc
total <- sum(x)          # Much faster than loop
```

if / else

```
x <- 75

if (x > 80) {
  grade <- "A"
} else if (x > 60) {
  grade <- "B"
} else {
  grade <- "C"
```

```
}
print(grade)  # "B"


# Vectorized alternative: ifelse()
scores <- c(85, 72, 91, 58)
grades <- ifelse(scores >= 60, "Pass", "Fail")
# "Pass" "Pass" "Pass" "Fail"
```

for Loops

```
# Loop through vector
for (i in 1:5) {
  print(i^2)
}


# Loop through names
fruits <- c("apple", "banana", "cherry")
for (fruit in fruits) {
  print(paste("I like", fruit))
}


# Loop through data frame rows (NOT recommended — use apply instead)
for (i in 1:nrow(df)) {
  print(df[i, "name"])
}
```

while and repeat Loops

```
# while loop
count <- 1
while (count <= 5) {
  print(count)
  count <- count + 1
```

```
}


# repeat loop (infinite until break)
x <- 1
repeat {
  print(x)
  x <- x + 1
  if (x > 5) break
}
```

Functions

Functions are the backbone of R programming. Write functions to avoid repetition and make code modular and testable.

```
# Define function
calculate_bmi <- function(weight_kg, height_m) {
  bmi <- weight_kg / (height_m^2)
  return(bmi)
}


# Call function
calculate_bmi(70, 1.75)  # 22.86


# Default arguments
greet <- function(name, greeting = "Hello") {
  message <- paste(greeting, name)
  return(message)
}


greet("Alice")            # "Hello Alice"
greet("Bob", "Hi")        # "Hi Bob"
```

```
# Multiple return values (use list)
stats_summary <- function(x) {
  list(
    mean = mean(x),
    median = median(x),
    sd = sd(x),
    min = min(x),
    max = max(x)
  )
}

result <- stats_summary(c(10, 20, 30, 40, 50))
result$mean    # 30
```

Vectorization — R's Superpower

Vectorization means operations apply to entire vectors at once, without explicit loops. Vectorized code is faster, cleaner, and more idiomatic.

```
# SLOW: Loop approach
x <- 1:1000
result <- numeric(1000)
for (i in 1:length(x)) {
  result[i] <- x[i]^2 + 2*x[i] + 1
}

# FAST: Vectorized approach
result <- x^2 + 2*x + 1

# Vectorized functions
sum(x)          # Sum all elements
mean(x)         # Mean
max(x); min(x)   # Max and min
sqrt(x)         # Square root of each element
```

```
log(x)          # Natural log of each element


# apply family for row/column operations
mat <- matrix(1:12, nrow=3)
apply(mat, 1, sum)     # Row sums
apply(mat, 2, mean)    # Column means

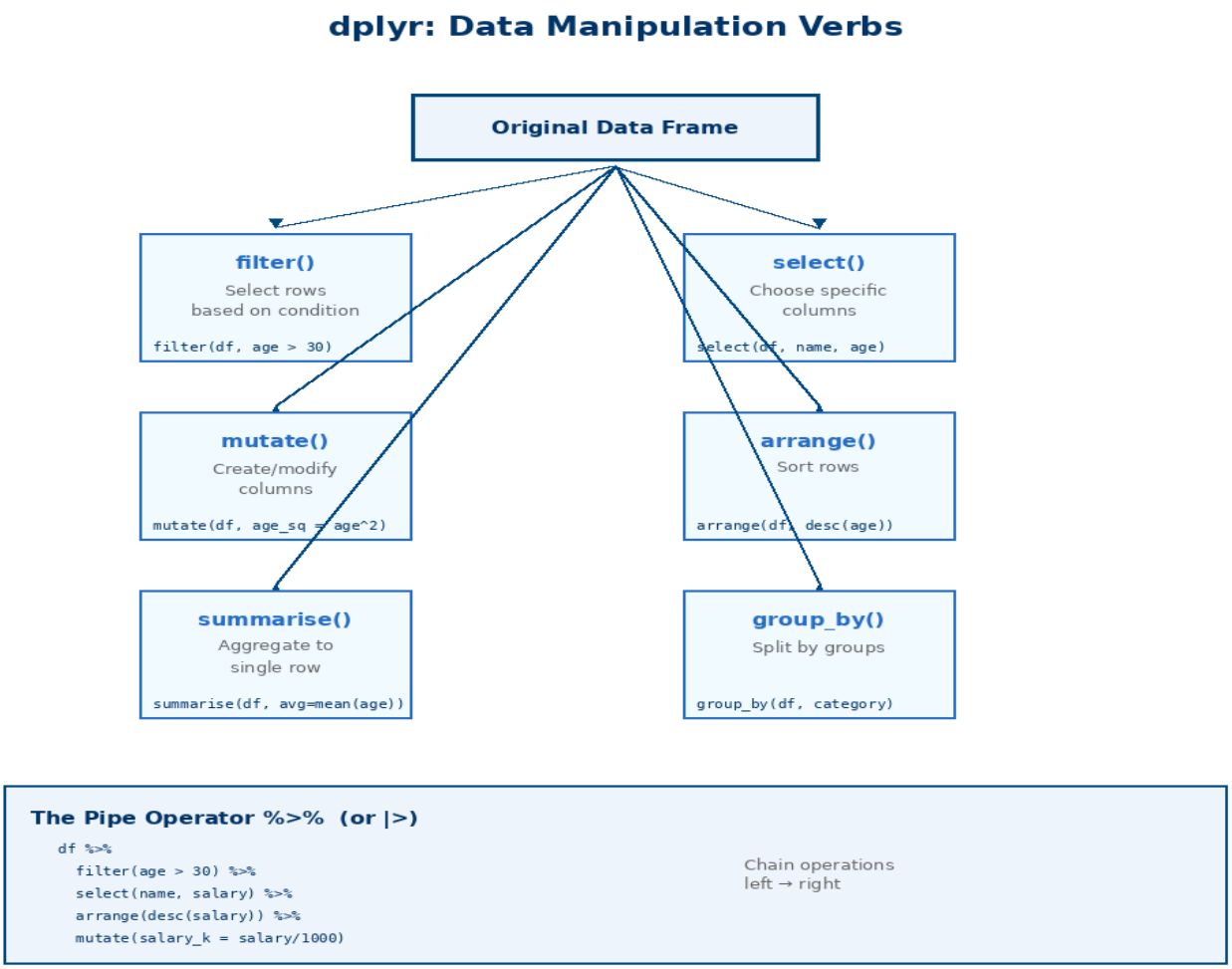

# lapply: apply function to list/vector, return list
lapply(1:5, sqrt)


# sapply: simplified version (returns vector)
sapply(1:5, sqrt)      # 1.000 1.414 1.732 2.000 2.236
```

Performance Tip: Always prefer vectorized operations over loops. For complex row-wise operations on data frames, use dplyr::rowwise() or purrr::pmap() instead of for loops.

7.4 Data Manipulation with dplyr

dplyr is the most important package for data manipulation in R. It provides a consistent, readable grammar for transforming data frames. Every analyst must master its six core verbs.



dplyr is part of the tidyverse — a collection of packages designed for data science. Install and load it:

```
install.packages("tidyverse")

library(dplyr)
```

The Six Core Verbs

Verb	What It Does	Example
filter()	Select rows based on condition	filter(df, age > 30)
select()	Choose specific columns	select(df, name, salary)
mutate()	Create or modify columns	mutate(df, age_sq = age^2)
arrange()	Sort rows	arrange(df, desc(salary))

summarise()	Aggregate to summary statistics	summarise(df, avg = mean(salary))
group_by()	Split data into groups	group_by(df, department)

The Pipe Operator: %>% or |>

The pipe passes the left-hand side as the first argument to the right-hand function. It makes code read like a narrative.

```
# Without pipe (nested, hard to read)
result <- arrange(
  filter(
    select(df, name, age, salary),
    age > 30
  ),
  desc(salary)
)

# With pipe (clear, left-to-right flow)
result <- df %>%
  select(name, age, salary) %>%
  filter(age > 30) %>%
  arrange(desc(salary))

# Modern R (4.1+) uses native pipe: |>
result <- df |>
  filter(age > 30) |>
  select(name, salary)
```

filter() — Row Selection

```
library(dplyr)

# Single condition
```

```
filter(mtcars, mpg > 25)


# Multiple conditions (AND)
filter(mtcars, mpg > 20, cyl == 4)


# OR condition
filter(mtcars, mpg > 25 | hp > 200)


# Between
filter(mtcars, between(mpg, 20, 30))


# NOT
filter(mtcars, !is.na(mpg))


# String matching
filter(mtcars, grepl("Merc", rownames(mtcars)))
```

select() — Column Selection

```
# Select specific columns
select(mtcars, mpg, cyl, hp)


# Select range
select(mtcars, mpg:hp)


# Exclude columns
select(mtcars, -cyl, -disp)


# Select by pattern
select(mtcars, starts_with("m"))  # mpg
select(mtcars, ends_with("p"))    # hp, disp
```

```
select(mtcars, contains("ar"))    # gear, carb


# Rename while selecting
select(mtcars, miles_per_gallon = mpg)
```

mutate() — Create/Modify Columns

```
# Add new column
mtcars %>%
  mutate(
    kpl = mpg * 0.425,            # Convert mpg to km/l
    hp_per_cyl = hp / cyl,
    is_efficient = mpg > 25
  )


# Overwrite existing
mtcars %>% mutate(mpg = mpg * 1.6)  # Convert to km/l


# Conditional mutations with case_when()
mtcars %>%
  mutate(
    efficiency = case_when(
      mpg > 30 ~ "Excellent",
      mpg > 20 ~ "Good",
      mpg > 15 ~ "Average",
      TRUE ~ "Poor"
    )
  )
```

arrange() — Sorting

```
# Sort ascending
arrange(mtcars, mpg)
```

```
# Sort descending
arrange(mtcars, desc(mpg))

# Multiple columns
arrange(mtcars, cyl, desc(mpg))
```

group_by() + summarise() — Aggregation

The most powerful combination. Split data into groups, compute summaries for each group.

```
# Group by cylinder count, compute stats
mtcars %>%
  group_by(cyl) %>%
  summarise(
    count = n(),
    avg_mpg = mean(mpg),
    median_hp = median(hp),
    max_wt = max(wt)
  )

# Multiple grouping variables
mtcars %>%
  group_by(cyl, gear) %>%
  summarise(avg_mpg = mean(mpg))

# Count occurrences
mtcars %>% count(cyl)          # Shortcut for group_by + summarise(n())

# Grouped mutations (within-group calculations)
mtcars %>%
  group_by(cyl) %>%
  mutate(
```

```
    mpg_vs_group_avg = mpg - mean(mpg),
    rank = rank(desc(mpg))
  )
```

Complete dplyr Workflow Example

```
library(dplyr)


# Sales analysis pipeline
sales_summary <- sales_data %>%
  filter(date >= "2024-01-01", !is.na(revenue)) %>%
  select(region, product, revenue, quantity) %>%
  mutate(
    revenue_k = revenue / 1000,
    price = revenue / quantity
  ) %>%
  group_by(region, product) %>%
  summarise(
    total_revenue = sum(revenue),
    avg_price = mean(price),
    units_sold = sum(quantity),
    transactions = n()
  ) %>%
  arrange(desc(total_revenue))


# Export
write.csv(sales_summary, "sales_summary.csv")
```

7.5 Data Visualization with ggplot2

ggplot2 implements the "grammar of graphics" — a principled framework for building any visualization by combining independent components. It produces publication-quality plots with minimal code and is the gold standard for statistical graphics.

ggplot2: Grammar of Graphics

1. DATA
`ggplot(data = df)`

2. AESTHETICS
`aes(x = var1, y = var2)`

3. GEOMETRIES
`geom_point(), geom_line(), geom_bar()`

4. FACETS
`facet_wrap(~category)`

5. STATISTICS
`stat_smooth(), stat_summary()`

6. COORDINATES
`coord_flip(), coord_polar()`

7. THEME
`theme_minimal(), theme_classic()`

Complete Example

```
ggplot(data = mtcars,
       aes(x = wt,
           y = mpg,
           color = cyl)) +
  geom_point(size = 3) +
  geom_smooth(method = 'lm') +
  facet_wrap(~gear) +
  labs(title = 'MPG vs Weight',
       x = 'Weight (1000 lbs)',
       y = 'Miles Per Gallon') +
  theme_bw()
```

Each layer added with +

Build plots incrementally

The Grammar of Graphics

Every ggplot is built from layers. The three essential layers are: (1) Data, (2) Aesthetics (mappings from data to visual properties), (3) Geometries (the type of plot).

```
library(ggplot2)



# Basic template
ggplot(data = <DATA>,
      mapping = aes(x = <VAR1>, y = <VAR2>)) +
 geom_<TYPE>()
```

Essential Geometries

geom	Plot Type	Example Use
geom_point()	Scatter plot	Correlation, relationships

geom_line()	Line chart	Time series, trends
geom_bar()	Bar chart (count)	Categorical frequencies
geom_col()	Bar chart (values)	Pre-aggregated data
geom_histogram()	Histogram	Distributions
geom_boxplot()	Box plot	Comparing distributions
geom_smooth()	Trend line	Regression lines, smoothing

Scatter Plot

```
# Basic scatter
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()


# With color by category
ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 3) +
  labs(
    title = "Fuel Efficiency vs Weight",
    x = "Weight (1000 lbs)",
    y = "Miles Per Gallon",
    color = "Cylinders"
  ) +
  theme_minimal()


# Add regression line
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE, color = "blue")
```

Bar Charts

```
# Counts (geom_bar)
ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar() +
  labs(x = "Cylinders", y = "Count")


# Pre-aggregated values (geom_col)
summary_df <- mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg))


ggplot(summary_df, aes(x = factor(cyl), y = avg_mpg)) +
  geom_col(fill = "steelblue") +
  labs(x = "Cylinders", y = "Average MPG")
```

Histograms and Density Plots

```
# Histogram
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram(bins = 10, fill = "skyblue", color = "black") +
  labs(title = "Distribution of MPG")


# Density plot (smoothed histogram)
ggplot(mtcars, aes(x = mpg)) +
  geom_density(fill = "lightblue", alpha = 0.5)
```

Box Plots

```
# Compare distributions across groups
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot(fill = "lightgreen") +
  labs(x = "Cylinders", y = "MPG")
```

Line Charts (Time Series)

```
# Sample time series data
dates <- seq(as.Date("2024-01-01"), by = "day", length.out = 30)
revenue <- cumsum(rnorm(30, mean = 1000, sd = 200))
ts_data <- data.frame(date = dates, revenue = revenue)


ggplot(ts_data, aes(x = date, y = revenue)) +
  geom_line(color = "darkblue", size = 1) +
  geom_point(color = "red", size = 2) +
  labs(title = "Daily Revenue Trend", x = "Date", y = "Revenue ($)") +
  theme_classic()
```

Faceting — Small Multiples

Faceting creates separate plots for each level of a categorical variable.

```
# Separate plot for each cylinder count
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  facet_wrap(~cyl) +
  labs(title = "MPG vs Weight by Cylinder Count")


# Grid layout with two variables
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  facet_grid(gear ~ cyl)
```

Themes and Customization

```
# Built-in themes
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
```

```
  theme_minimal()      # Clean, minimal grid


# Others: theme_bw(), theme_classic(), theme_dark()


# Custom modifications
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.text = element_text(size = 12),
    legend.position = "bottom"
  )
```

Saving Plots: Use ggsave() to save the last plot: ggsave("my_plot.png", width = 8, height = 6, dpi = 300)

7.6 Statistical Analysis in R

R was built for statistics. It provides built-in functions for every common statistical test, plus thousands of specialized packages for advanced methods.

Statistical Analysis in R

Descriptive Statistics

```
mean(x)
median(x)
sd(x)
quantile(x)
summary(df)
```

Correlation Analysis

```
cor(x, y)
cor.test(x, y)
cor(df)
```

Regression

```
lm(y ~ x)
summary(model)
predict(model)
```

t-tests

```
t.test(x)
t.test(x, y)
t.test(x, mu=50)
```

ANOVA

```
aov(y ~ group)
anova(model)
TukeyHSD()
```

Chi-square

```
chisq.test(x, y)
table(x, y)
```

Example: Linear Regression Analysis

```
# Fit model
model <- lm(mpg ~ wt + hp, data = mtcars)

# View results
summary(model)    # R², coefficients, p-values

# Diagnostics
plot(model)       # Residual plots

# Predictions
new_data <- data.frame(wt = 3.0, hp = 120)
```

Descriptive Statistics

```
# Basic stats
mean(mtcars$mpg)
median(mtcars$mpg)
sd(mtcars$mpg)        # Standard deviation
var(mtcars$mpg)       # Variance
min(mtcars$mpg); max(mtcars$mpg)
range(mtcars$mpg)     # Min and max
IQR(mtcars$mpg)       # Interquartile range


# Quantiles
quantile(mtcars$mpg, probs = c(0.25, 0.5, 0.75))
```

```
# Summary of entire data frame
summary(mtcars)


# More detailed summary
library(psych)
describe(mtcars)
```

Correlation and Covariance

```
# Correlation between two variables
cor(mtcars$mpg, mtcars$wt)          # -0.868 (strong negative)


# Test significance
cor.test(mtcars$mpg, mtcars$wt)
# p-value < 0.001 → significant


# Correlation matrix
cor(mtcars[, c("mpg", "wt", "hp", "disp")])


# Visualize with corrplot
library(corrplot)
corrplot(cor(mtcars), method = "circle")
```

t-tests

t-tests compare means between groups or against a known value.

```
# One-sample t-test (compare mean to known value)
t.test(mtcars$mpg, mu = 20)
# H0: mean = 20. If p < 0.05, reject H0


# Two-sample t-test (independent groups)
auto <- mtcars$mpg[mtcars$am == 0]      # Automatic transmission
```

```
manual <- mtcars$mpg[mtcars$am == 1]    # Manual transmission
t.test(auto, manual)
# p-value = 0.001374 → manual cars have higher MPG


# Paired t-test (before/after measurements)
before <- c(120, 135, 128, 140, 132)
after <- c(115, 130, 125, 135, 128)
t.test(before, after, paired = TRUE)
```

ANOVA — Comparing 3+ Groups

```
# One-way ANOVA
model <- aov(mpg ~ factor(cyl), data = mtcars)
summary(model)
# p < 0.001 → cylinder count affects MPG


# Post-hoc test (which groups differ?)
TukeyHSD(model)


# Two-way ANOVA
model2 <- aov(mpg ~ factor(cyl) + factor(gear), data = mtcars)
summary(model2)
```

Linear Regression

Regression models the relationship between a dependent variable and one or more predictors.

```
# Simple linear regression
model <- lm(mpg ~ wt, data = mtcars)
summary(model)
# R² = 0.75 (weight explains 75% of MPG variance)
# Coefficient: -5.34 (each 1000 lbs reduces MPG by 5.34)
```

```
# Multiple regression
model2 <- lm(mpg ~ wt + hp + cyl, data = mtcars)
summary(model2)


# Predictions
new_data <- data.frame(wt = 3.0, hp = 120, cyl = 4)
predict(model2, new_data, interval = "confidence")


# Diagnostics
plot(model)          # 4 diagnostic plots
# 1. Residuals vs Fitted
# 2. Normal Q-Q
# 3. Scale-Location
# 4. Residuals vs Leverage
```

Chi-Square Test

Test independence of categorical variables.

```
# Create contingency table
table_data <- table(mtcars$cyl, mtcars$gear)


# Chi-square test
chisq.test(table_data)
# p < 0.05 → cylinder and gear are related
```

Model Selection: Compare models with AIC (Akaike Information Criterion). Lower AIC = better model: AIC(model1); AIC(model2)

7.7 R Markdown and Reporting

R Markdown integrates code, results, and narrative into a single document. It is the gold standard for reproducible research — ensuring analyses are transparent, replicable, and shareable.

R Markdown: Reproducible Reports

1. YAML Header

```
---
title: 'Sales Analysis Report'
author: 'Data Analyst'
date: '2025-02-18'
output: html_document
---
```

2. Markdown Text

```
# Main Heading

## Sub-heading

Regular text with **bold** and *italic*.

- Bullet point 1
- Bullet point 2
```

3. R Code Chunks

```
```{r setup, include=FALSE}
library(tidyverse)
```

```{r analysis, echo=TRUE, fig.width=8}
# Load data
df <- read_csv('data.csv')

# Create plot
ggplot(df, aes(x=date, y=sales)) +
  geom_line() +
  theme_minimal()
```
```

Output Formats:
HTML, PDF, Word, PowerPoint, Shiny Dashboard, Websites, Books

Why R Markdown?

- Reproducibility — Code and narrative in one file. Re-run the entire analysis with one click.
- Multiple Output Formats — HTML, PDF, Word, PowerPoint, dashboards, websites, books from the same source.
- Dynamic Reports — Update numbers/plots automatically when data changes.
- Professional Output — Publication-quality documents without manual formatting.

Document Structure

An R Markdown document has three parts: YAML header (metadata), Markdown text (narrative), R code chunks (analysis).

1. YAML Header

```
---

title: "Sales Analysis Report"
```

```
author: "Data Analytics Team"

date: "2025-02-18"

output: html_document

---
```

2. Markdown Text

```
# Main Heading



## Sub-heading



Regular text with **bold** and *italic*.



- Bullet point 1

- Bullet point 2



[Link text](https://example.com)



![Image caption](path/to/image.png)
```

3. R Code Chunks

```
```{r setup, include=FALSE}

library(tidyverse)

knitr::opts_chunk$set(echo = TRUE, warning = FALSE)

```



```{r analysis, fig.width=8, fig.height=5}

# Load data

df <- read_csv("sales_data.csv")



# Plot
```
```

```
ggplot(df, aes(x=date, y=revenue)) +
  geom_line() +
  theme_minimal()
```

The average revenue was `r mean(df$revenue)`.

Chunk Options

Option	What It Does	Example
echo	Show code in output?	echo=FALSE hides code
eval	Run the code?	eval=FALSE shows but doesn't run
include	Include in output?	include=FALSE runs but hides everything
warning	Show warnings?	warning=FALSE suppresses warnings
message	Show messages?	message=FALSE suppresses messages
fig.width, fig.height	Plot dimensions (inches)	fig.width=8, fig.height=5
cache	Cache results?	cache=TRUE (speeds up re-knitting)

Output Formats

```
---
output:
  html_document:
    toc: true             # Table of contents
    toc_float: true       # Floating TOC
    theme: cerulean       # Visual theme
    code_folding: hide    # Hide code by default
---
```

```
---
```

```
output: pdf_document      # Requires LaTeX
---



---
output: word_document      # Microsoft Word
---



---
output: powerpoint_presentation
---



---
output:
  flexdashboard::flex_dashboard:
    orientation: rows
---
```

Knitting the Document

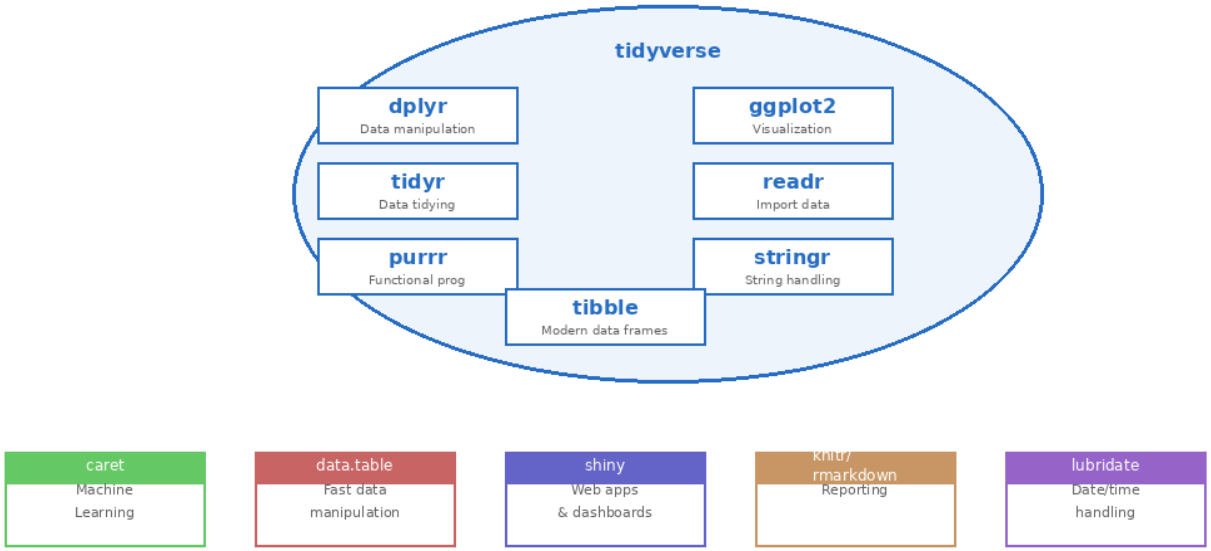
Click "Knit" button in RStudio or run: rmarkdown::render("report.Rmd")

Tip: Use inline R code `r` to embed results directly in text. Example: "The dataset contains `r nrow(df)` rows." This auto-updates when data changes.

Practice Exercises

1. Create a vector of 20 random numbers. Calculate mean, median, SD, and identify values > 1 SD from mean.
2. Load mtcars. Use dplyr to filter cars with mpg > 20, select mpg, wt, hp, and arrange by descending mpg.
3. Create a ggplot scatter plot of mtcars: wt vs mpg, colored by cyl, with a linear regression line.
4. Perform a t-test comparing mpg between automatic (am=0) and manual (am=1) cars. Interpret the p-value.
5. Build a linear regression model: mpg ~ wt + hp. Report R^2 and interpret coefficients.
6. Create an R Markdown document with: (1) YAML header, (2) Summary stats of iris dataset, (3) Boxplot of Sepal.Length by Species.

R Package Ecosystem for Data Analysis



Installing Packages:

```
install.packages('tidyverse')    # Install tidyverse suite
install.packages('caret')        # Install single package
library(dplyr)                   # Load package into session
```

UNIT VIII – Power BI for Business Intelligence

8.1 Introduction to Power BI

Power BI is Microsoft's flagship business intelligence and data visualization platform. Launched in 2015 (building on Excel Power Pivot), Power BI has rapidly become one of the world's most adopted BI tools, serving over 15 million users across 200+ countries. It combines powerful data modeling, sophisticated analytics, and intuitive visual design in a unified ecosystem tightly integrated with Microsoft's cloud and productivity suite.

Power BI transforms raw data from disparate sources into interactive dashboards and reports that reveal insights instantly. Its low-code, drag-and-drop interface makes advanced analytics accessible to business users while providing enterprise-grade scalability, security, and governance. Organizations use Power BI to monitor KPIs, track performance, identify trends, and make data-driven decisions at every level — from executives to front-line teams.

Why Power BI?

- Microsoft Ecosystem Integration — Seamlessly connects with Excel, SQL Server, Azure, Dynamics 365, Microsoft 365, SharePoint, and Teams. Natural choice for Microsoft-centric organizations.
- Cost-Effective — Power BI Desktop is free. Pro licenses at $10/user/month are significantly cheaper than competitors. Premium offers enterprise features at predictable pricing.
- Ease of Use — Intuitive interface designed for business analysts, not IT specialists. Build reports in hours, not weeks.
- Scalability — From personal reports to enterprise deployments handling billions of rows. Cloud-native architecture scales automatically.
- AI-Powered Insights — Built-in machine learning, natural language Q&A, smart narratives, and anomaly detection without coding.
- Active Community — Extensive documentation, training, forums, and thousands of custom visuals from Microsoft and partners.

Power BI Market Position

Power BI competes primarily with Tableau (Salesforce), Qlik, and Looker (Google Cloud). It consistently ranks as a Leader in Gartner's Magic Quadrant for Analytics and Business Intelligence Platforms.

Tool	Strengths	Best For
Power BI	Microsoft integration, cost, DAX modeling, AI features	Microsoft 365 organizations, enterprise BI, cost-conscious buyers

nasim-a-mahin

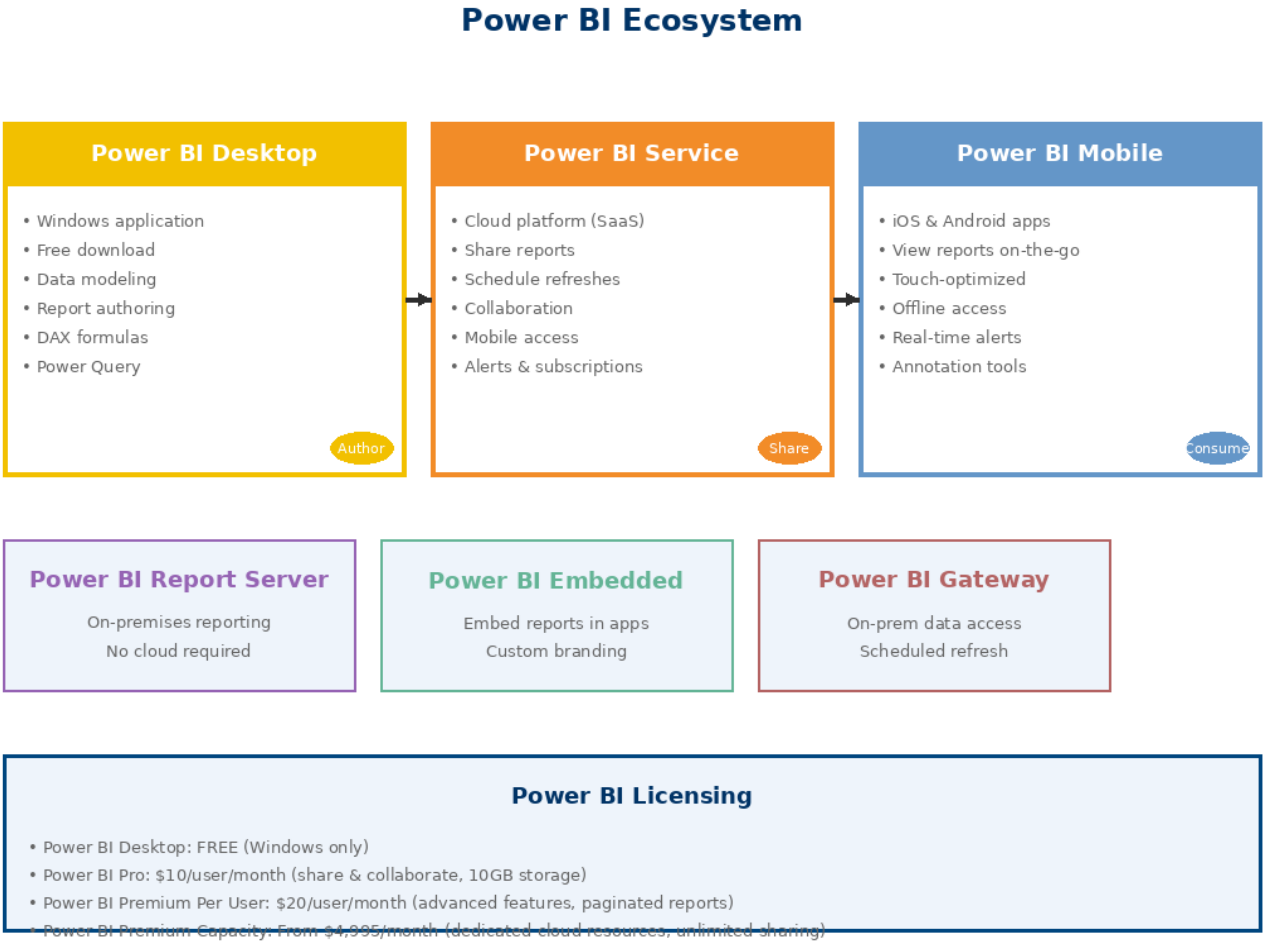
Tableau	Visual analytics sophistication, exploration, community	Data viz professionals, exploratory analysis, design-first culture
Qlik	Associative engine, governed data models	Complex data relationships, governed enterprise BI
Looker	LookML (code-first), cloud-native, data modeling	Technical teams, cloud-first architecture, Google Cloud users

Microsoft Acquisition History: Power BI evolved from Excel Power Pivot (2010) and Power View (2012). Microsoft made a strategic bet on self-service BI in 2015, competing directly with Tableau. By 2020, Power BI had surpassed Tableau in user count.

nasrin-a-mamin

8.2 Power BI Components

Power BI is an ecosystem of interconnected tools, each serving a specific role in the analytics workflow. Understanding the components and how they work together is essential for effective deployment.



Power BI Desktop

The primary authoring tool for creating reports and data models. A free Windows application (Mac users can use virtualization or Parallels).

Key Capabilities

- Connect to 100+ data sources (files, databases, cloud services, APIs)
- Transform and clean data with Power Query (M language)
- Build data models with relationships, calculated columns, and measures
- Create DAX formulas for advanced calculations

- Design interactive reports with 30+ built-in visualizations
- Publish to Power BI Service for sharing

System Requirements

- Windows 10 or 11 (64-bit)
- 4GB RAM minimum (8GB+ recommended)
- .NET Framework 4.7.2+
- 1280×768 display minimum

Power BI Service

Cloud-based SaaS platform (app.powerbi.com) for sharing, collaborating, and consuming reports. Think of it as "Power BI Online" or "Power BI Cloud".

Key Features

- Publish reports from Power BI Desktop
- Share dashboards and reports with colleagues
- Schedule automatic data refreshes
- Create apps for end-user distribution
- Set up data-driven alerts and subscriptions
- Collaborate with comments and annotations
- Mobile-optimized viewing
- Row-level security (RLS) for data governance

Power BI Mobile

iOS and Android apps for consuming reports on phones and tablets. Touch-optimized with offline access and push notifications.

Mobile Capabilities

- View reports and dashboards
- Filter and drill through data
- Offline access (cached reports)
- Scan QR codes to open specific reports
- Annotate reports with ink
- Receive real-time data alerts

nasrin-a-mahin

Power BI Report Server

On-premises reporting server for organizations that cannot use cloud services due to regulatory, security, or connectivity constraints. Similar to SQL Server Reporting Services (SSRS) but with Power BI capabilities.

- Host reports behind firewall
- No internet required
- Integration with existing Microsoft infrastructure
- Full control over data location
- Requires Software Assurance or Power BI Premium license

Power BI Embedded

Embeds Power BI reports into custom applications and websites. Used by ISVs (Independent Software Vendors) and SaaS providers to white-label analytics.

- Embed reports in your app (React, Angular, .NET, etc.)
- Custom branding and styling
- Row-level security per user
- Programmatic control via REST APIs
- Pay-per-usage pricing (Azure service)

Power BI Gateway

Software agent installed on-premises to enable secure data transfer between on-prem data sources and Power BI Service. Required for scheduled refresh of on-prem data.

- Personal Mode — Single user, lightweight
- Enterprise Mode — Shared across organization, centrally managed
- Supports DirectQuery and Import modes
- Encrypted data transfer

Licensing Overview

License	Cost	Key Features	Target User
Power BI Desktop	FREE	Authoring, modeling, local reports	Analysts, report authors

nasrin-a-mahin

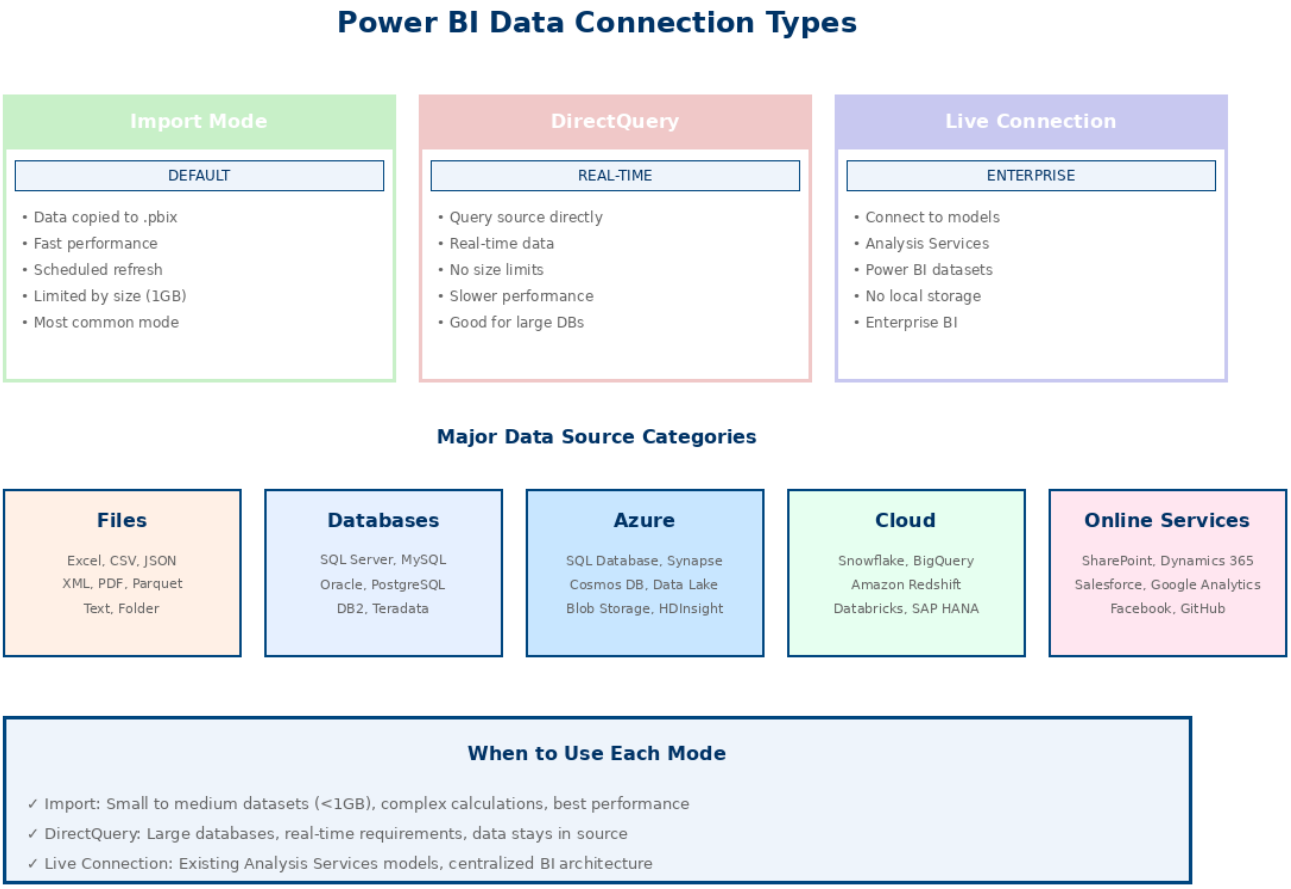
Power BI Pro	$10/user/month	Publish, share, collaborate, 10GB storage	Business users, small teams
Power BI Premium Per User	$20/user/month	Advanced features, paginated reports, AI	Power users, analysts
Power BI Premium Capacity	From $4,995/month	Dedicated cloud capacity, unlimited sharing	Enterprises, large deployments

Pro vs Premium: Power BI Pro is per-user licensing (both creator AND viewer need Pro to share). Premium Per User removes viewer license requirement. Premium Capacity provides dedicated cloud resources and enterprise features.

nasrin-a-mamin

8.3 Connecting and Importing Data

Power BI connects to virtually any data source — from Excel spreadsheets to enterprise data warehouses. Understanding connection modes and supported sources is fundamental to building effective solutions.



Three Connection Modes

1. Import Mode (Default)

Data is copied into Power BI and stored in a compressed columnar format (.pbix file or Power BI Service). This is the most common and recommended mode.

Advantages:

- Fast query performance (data is in-memory)
- Supports all DAX functions and visuals

- Works offline
- Compression reduces data size (5-10x typically)

Limitations:

- 1GB file size limit in Power BI Service (10GB with Premium)
- Data refresh required for updates
- Refresh limits (8/day for Pro, 48/day for Premium)

2. DirectQuery Mode

Queries are sent directly to the source database in real-time. No data is imported into Power BI.

Advantages:

- Always shows current data (no refresh needed)
- No size limits
- Suitable for very large datasets

Limitations:

- Slower performance (depends on database speed)
- Limited to certain data sources (SQL Server, Azure, Snowflake, etc.)
- Some DAX functions unavailable
- Requires constant database connection

3. Live Connection

Connect to existing Analysis Services (Tabular or Multidimensional) or Power BI datasets. Similar to DirectQuery but connects to pre-built models, not raw databases.

- Leverage existing enterprise data models
- Single source of truth
- No data duplication
- Cannot add new data sources or modify model

Feature	Import	DirectQuery	Live Connection
Data in Power BI	Yes (compressed)	No	No
Performance	Fast	Depends on source	Depends on source

nasrin-a-mahin

Data Freshness	Scheduled refresh	Real-time	Real-time
Size Limit	1GB (10GB Premium)	No limit	No limit
Offline Access	Yes	No	No
Best For	Most scenarios	Large databases, real-time	Enterprise models

Supported Data Sources (100+)

Files

- Excel Workbooks (.xlsx, .xls)
- CSV, Text, TSV
- JSON, XML
- PDF Tables
- Parquet, Apache ORC
- Folder (combine multiple files)

Databases

- Microsoft SQL Server (on-prem and Azure)
- MySQL, PostgreSQL, Oracle, IBM DB2, Teradata
- Amazon Redshift, Google BigQuery, Snowflake
- MongoDB, Cassandra, Spark, Databricks
- SAP HANA, SAP Business Warehouse

Cloud & Services

- Azure: SQL Database, Synapse Analytics, Cosmos DB, Data Lake, Blob Storage
- SharePoint Online Lists, SharePoint Folder
- Dynamics 365 (Sales, Customer Service, Finance)
- Salesforce (Reports and Objects)
- Google Analytics, Adobe Analytics
- Web APIs (REST, OData)

Microsoft Ecosystem

- Microsoft Dataverse (Power Platform)
- Azure DevOps, GitHub

nasim-a-mahim

- Teams, Exchange Online
- Microsoft Graph (Office 365 data)

Connector Support: Most connectors support Import mode. DirectQuery is limited to enterprise databases (SQL Server, Oracle, SAP HANA, Snowflake, BigQuery, etc.). Always check connector documentation for capabilities.

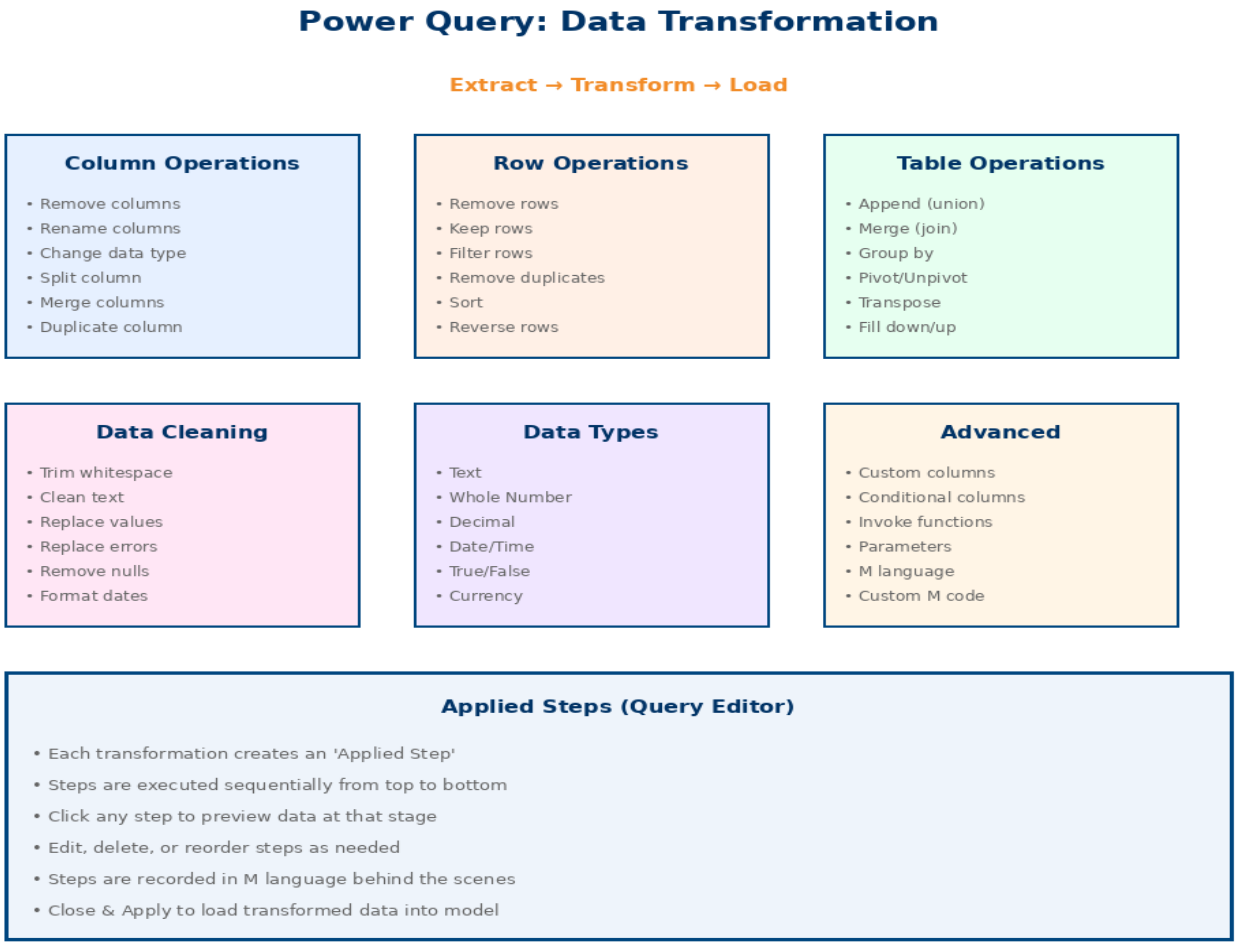
Connecting to Data — Step by Step

1. Open Power BI Desktop → Home → Get Data
2. Search or browse for your data source
3. Enter connection details (server, database, credentials)
4. Choose Import or DirectQuery (if available)
5. Navigator window: Select tables/views to load
6. Transform Data → Opens Power Query Editor (next section)
7. Close & Apply → Loads data into model

nasrin-a-mahin

8.4 Data Transformation with Power Query

Power Query is Power BI's ETL (Extract, Transform, Load) engine. It provides a visual interface for cleaning, shaping, and combining data before loading it into the data model. Every transformation is recorded as a step in M language (Power Query formula language).



Power Query Editor Interface

Access: Home → Transform Data. Opens a separate window with:

- Queries Pane (left) — List of all queries/tables
- Preview Grid (center) — Shows data transformations
- Query Settings (right) — Applied Steps, Properties
- Ribbon — Transform operations grouped by category

Common Transformations

Column Operations

- Remove Columns — Delete unnecessary columns
- Rename Columns — Use business-friendly names
- Change Data Type — Text, Number, Date, Boolean
- Split Column — By delimiter, position, or pattern
- Merge Columns — Combine first name + last name
- Duplicate Column — Create a copy for transformation

Row Operations

- Remove Rows — Top N, bottom N, errors, blanks, duplicates
- Keep Rows — Opposite of remove (keep top N, keep errors)
- Filter Rows — By value, condition, or pattern
- Sort Ascending/Descending

Table Operations

- Append Queries — Stack tables vertically (UNION)
- Merge Queries — Join tables horizontally (LEFT/RIGHT/INNER/OUTER)
- Group By — Aggregate data (SUM, COUNT, AVG, etc.)
- Pivot/Unpivot — Reshape data (wide ↔ tall)
- Transpose — Flip rows and columns

Data Cleaning

- Trim — Remove leading/trailing spaces
- Clean — Remove non-printable characters
- Replace Values — Find and replace
- Replace Errors — Substitute error values
- Fill Down/Up — Copy values to empty cells

Applied Steps

Every transformation creates an Applied Step in the Query Settings pane. Steps execute sequentially top to bottom.

- Click any step to preview data at that stage
- Delete steps by clicking the X
- Reorder steps by dragging (careful — dependencies matter)

nasrin-a-mahin

- Edit steps by clicking the gear icon
- View M code: Home → Advanced Editor

Example: Cleaning Sales Data

8. Remove unnecessary columns (internal IDs, debug fields)
9. Change OrderDate from Text to Date
10. Split CustomerName into FirstName and LastName
11. Remove rows where SalesAmount is null or negative
12. Replace errors in Quantity column with 1
13. Trim whitespace from ProductName
14. Rename columns to business-friendly names
15. Close & Apply

Combining Data Sources

Append Queries (Union)

Stack multiple tables with the same structure. Example: Combine sales from 12 monthly files into one table.

16. Home → Append Queries
17. Select tables to combine
18. Power Query stacks rows

Merge Queries (Join)

Join tables on a key column. Example: Add customer details to orders table.

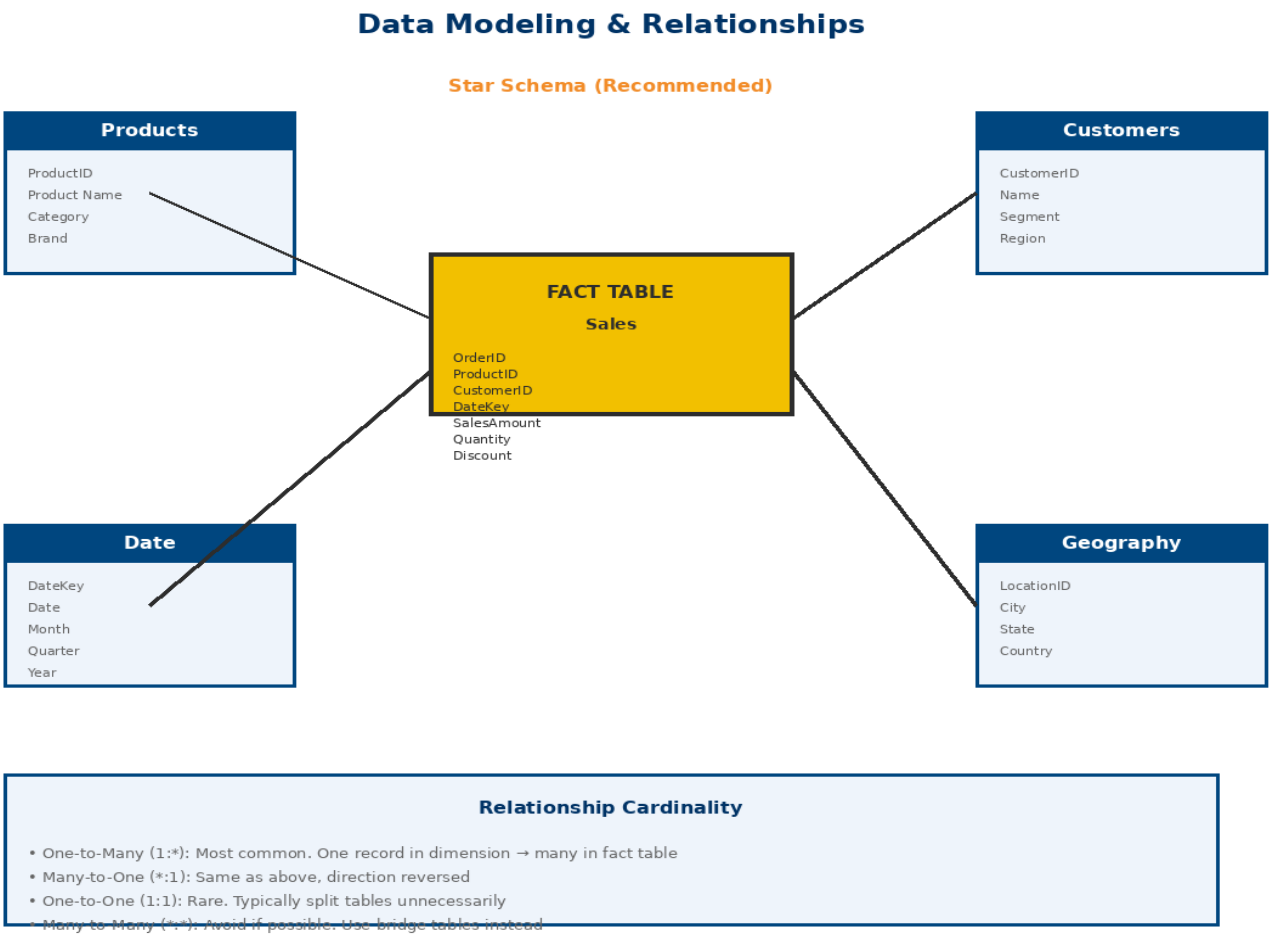
19. Home → Merge Queries
20. Select tables and join column
21. Choose join type (Left, Right, Inner, Outer)
22. Expand columns to include in result

Query Folding: When Power Query pushes transformations back to the data source (database) instead of doing them locally. Much faster. Use native database operations (filters, aggregations) to enable folding. Avoid adding custom columns early in the pipeline.

nasrin-a-mahin

8.5 Data Modelling and Relationships

Data modeling is the foundation of effective Power BI solutions. A well-designed model enables fast, intuitive analysis. A poor model leads to slow performance, confusing reports, and incorrect calculations.



Star Schema — The Gold Standard

Star schema is the recommended approach for Power BI models. It consists of fact tables (measures) surrounded by dimension tables (attributes) — visually resembling a star.

Fact Tables (Center)

- Contain quantitative data: sales amounts, quantities, costs, durations
- High row count (transactions, events, observations)
- Include foreign keys to dimension tables
- Examples: Sales, Orders, Inventory, Web Traffic

Dimension Tables (Points)

- Contain descriptive attributes: product names, customer segments, dates
- Lower row count (reference data)
- Include a primary key (unique identifier)
- Examples: Products, Customers, Dates, Geography

Creating Relationships

Relationships connect tables so measures aggregate correctly. Access: Model View (left sidebar icon).

Creating a Relationship

23. Drag a column from one table to another
24. Or: Manage Relationships → New
25. Select columns to join (typically IDs)
26. Set cardinality: One-to-Many (most common)
27. Set cross-filter direction: Single (default) or Both

Relationship Cardinality

Type	Notation	When to Use	Example
One-to-Many	1:*	One dimension record → many fact records	Customer → Orders
Many-to-One	*:1	Same as above, direction reversed	Orders → Customer
One-to-One	1:1	Rarely used (split table unnecessarily)	Employee → EmployeeDetails
Many-to-Many	*:*	AVOID. Use bridge table instead.	Students ↔ Classes (use Enrollment bridge)

Cross-Filter Direction

- Single (default) — Filter flows dimension → fact only
- Both — Filter flows in both directions (use sparingly — can cause ambiguity)

Bi-Directional Warning: Both direction filters can cause circular dependencies and unpredictable results. Use only when necessary (e.g., many-to-many scenarios with bridge tables). Single direction is safer.

Best Practices for Data Modeling

- Star schema over snowflake — Flatten hierarchies when possible
- Integer keys — Use integer surrogate keys, not text IDs (faster)
- Hide columns — Hide foreign keys and internal columns from report view
- Name tables clearly — Use singular names (Customer, not Customers)
- Mark date tables — Table → Mark as Date Table (enables time intelligence)
- Minimize relationships — Too many create performance issues
- Avoid circular dependencies — Relationships must form directed acyclic graph (DAG)

Date Tables

A dedicated Date table is essential for time intelligence (YTD, QTD, year-over-year). Must contain continuous dates with no gaps.

Creating a Date Table

```
DateTable = CALENDAR(DATE(2020,1,1), DATE(2025,12,31))
```

Or use built-in:

```
DateTable = CALENDARAUTO()
```

Add Calculated Columns

- Year = YEAR([Date])
- Quarter = QUARTER([Date])
- Month = FORMAT([Date], "MMMM")
- MonthNumber = MONTH([Date])
- Day = DAY([Date])
- Weekday = FORMAT([Date], "dddd")

nasim-a-mahin

8.6 DAX – Data Analysis Expressions

DAX (Data Analysis Expressions) is the formula language of Power BI. It is used to create calculated columns, measures, and calculated tables. DAX is similar to Excel formulas but designed for relational data models with millions of rows.

DAX: Data Analysis Expressions

Aggregation

SUM, AVERAGE, MIN, MAX
COUNT, DISTINCTCOUNT
COUNTA, COUNTBLANK

Filter Context

CALCULATE, FILTER
ALL, ALLEXCEPT
REMOVEFILTERS
VALUES, DISTINCT

Time Intelligence

DATESYTD, TOTALYTD
SAMEPERIODLASTYEAR
DATEADD, DATESBETWEEN
CALENDARAUTO

Logical

IF, SWITCH, AND, OR
ISBLANK, IFERROR
NOT, TRUE, FALSE

Text

CONCATENATE, FORMAT
LEFT, RIGHT, MID
UPPER, LOWER, TRIM
FIND, SUBSTITUTE

Relationship

RELATED, RELATEDTABLE
USERELATIONSHIP
CROSSFILTER

Table Functions

SUMMARIZE, ADDCOLUMNS
GROUPBY, SELECTCOLUMNS
GENERATE, CROSSJOIN

Statistical

RANK, RANKX
PERCENTILE.INC
MEDIAN, STDEV.P
VAR.P, GEOMEAN

Iterator

SUMX, AVERAGEX
MINX, MAXX
COUNTX, COUNTROWS
FILTER

Common DAX Patterns

```
Measure:  Total Sales = SUM(Sales[Amount])
Calculate:  Sales YTD = CALCULATE([Total Sales], DATESYTD('Date'[Date]))
Time Intelligence:  Sales LY = CALCULATE([Total Sales], SAMEPERIODLASTYEAR('Date'[Date]))
Filter:  High Value = CALCULATE([Total Sales], Sales[Amount] > 1000)
```

Measures vs Calculated Columns

	Measures	Calculated Columns
When Evaluated	At query time (dynamic)	At refresh time (static)
Storage	Not stored (calculated on-demand)	Stored in model (uses memory)
Icon	Calculator icon	fx icon

Use Case	Aggregations (SUM, COUNT, AVERAGE)	Row-by-row calculations
Performance	Fast (compressed)	Slower (stored)
Example	Total Sales = SUM(Sales[Amount])	Full Name = [FirstName] & " " & [LastName]

Rule of Thumb: Use measures for aggregations. Use calculated columns only when you need row-level values for filtering or grouping. Measures are almost always better for performance.

Creating Measures

28. Right-click table in Fields pane → New Measure
29. Enter DAX formula in formula bar
30. Press Enter

Example Measures:

```
Total Sales = SUM(Sales[Amount])
Average Order Value = DIVIDE([Total Sales], COUNT(Sales[OrderID]))
Profit Margin = DIVIDE([Total Profit], [Total Sales])
```

Essential DAX Functions

Aggregation Functions

- SUM, AVERAGE, MIN, MAX, COUNT, DISTINCTCOUNT
- Example: Total Quantity = SUM(Sales[Quantity])

CALCULATE — The Most Important Function

Modifies filter context. Essential for percentage of total, time intelligence, and conditional aggregations.

```
Sales Last Year = CALCULATE([Total Sales], SAMEPERIODLASTYEAR('Date'[Date]))
High Value Sales = CALCULATE([Total Sales], Sales[Amount] > 1000)
```

Time Intelligence

- DATESYTD, TOTALYTD — Year-to-date
- SAMEPERIODLASTYEAR — Prior year comparison
- DATEADD — Shift dates by period

```
YTD Sales = TOTALYTD([Total Sales], 'Date'[Date])
```

Logical Functions

- IF, SWITCH, AND, OR, NOT

```
Sales Category = IF([Total Sales] > 10000, "High", "Low")
```

Common DAX Patterns

Percent of Total:

```
% of Total Sales = DIVIDE([Total Sales], CALCULATE([Total Sales], ALL(Sales)))
```

Running Total:

```
Running Total = CALCULATE([Total Sales], FILTER(ALL('Date'), 'Date'[Date] <= MAX('Date'[Date])))
```

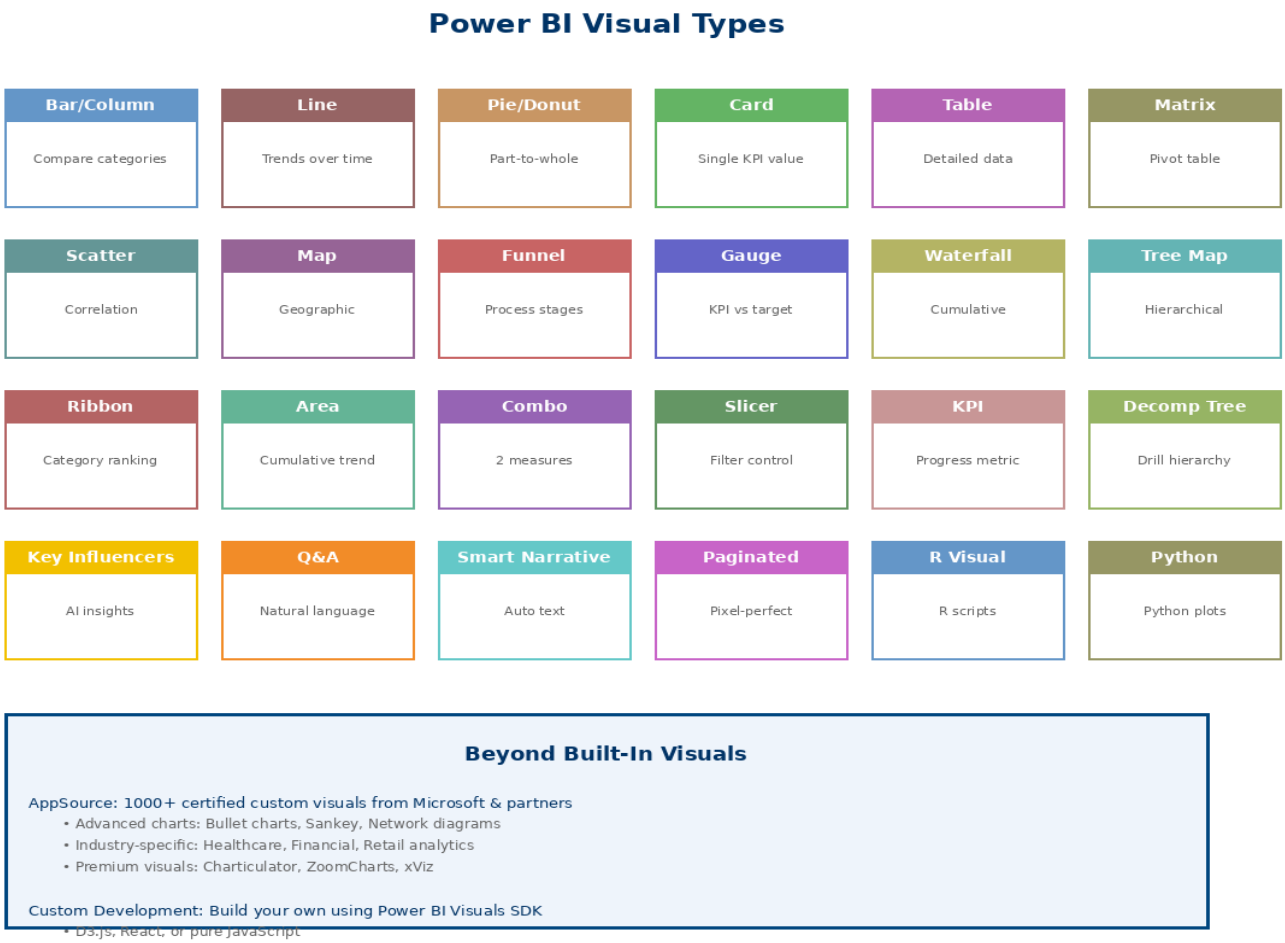
Year-over-Year Growth:

```
YoY Growth = DIVIDE([Total Sales] - [Sales Last Year], [Sales Last Year])
```

nasir-a-mahin

8.7 Building Reports and Dashboards

Power BI reports are interactive, multi-page documents containing visualizations. Dashboards are single-page aggregations of key visuals pinned from multiple reports.



Beyond Built-In Visuals

AppSource: 1000+ certified custom visuals from Microsoft & partners

- Advanced charts: Bullet charts, Sankey, Network diagrams
- Industry-specific: Healthcare, Financial, Retail analytics
- Premium visuals: Charticulator, ZoomCharts, xViz

Custom Development: Build your own using Power BI Visuals SDK

- D3.js, React, or pure JavaScript

Building a Visual

31. Select visual type from Visualizations pane
32. Drag fields to: Values, Axis, Legend, Tooltips
33. Format visual: Paint roller icon
34. Add interactions: Format → Edit Interactions

Essential Visuals

- Bar/Column Chart — Compare categories
- Line Chart — Trends over time
- Card — Single KPI number

- Table/Matrix — Detailed data
- Map — Geographic analysis
- Slicer — Filter control
- Gauge — KPI vs target

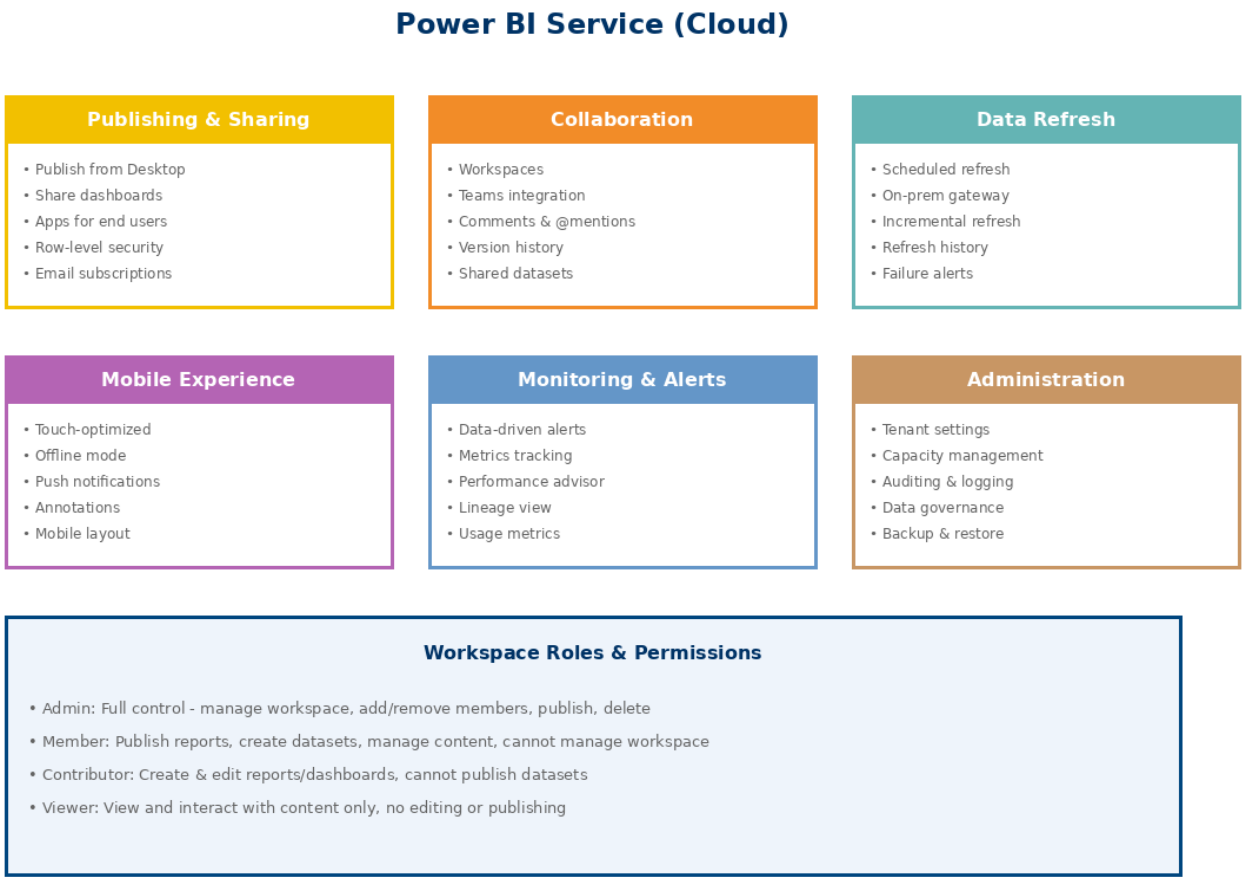
Report Best Practices

- One clear message per page
- 5-7 visuals maximum per page
- Consistent color scheme
- Clear, descriptive titles
- Strategic use of white space
- Meaningful tooltips

nasrin-a-mahin

8.8 Power BI Service and Sharing

Power BI Service (app.powerbi.com) is where reports are published, shared, and consumed. It provides collaboration, refresh scheduling, and enterprise features.



Publishing from Desktop

35. File → Publish → Publish to Power BI
36. Select workspace
37. Report uploads to Power BI Service
38. Open in browser to view

Workspaces

Containers for related reports, dashboards, and datasets. Types:

- My Workspace — Personal, private
- Shared Workspaces — Team collaboration

- Apps — Curated collections for end users

Scheduled Refresh

39. Dataset → Settings → Scheduled Refresh
40. Configure gateway (if on-prem data)
41. Set schedule (daily, weekly)
42. Enter credentials

8.9 Best Practices and Performance Optimization

Data Modeling

- Star schema over snowflake

- Use integer keys, not text
- Hide unused columns
- Minimize relationships
- Avoid bi-directional filters

DAX Performance

- Use measures, not calculated columns
- Use variables (VAR) to avoid recalculation
- Filter before calculating
- Avoid FILTER when CALCULATE suffices

Query Performance

- Enable query folding in Power Query
- Remove unused columns early
- Use incremental refresh for large datasets
- Disable Auto Date/Time (creates hidden tables)

Report Design

- Limit to 5-7 visuals per page
- Consistent color palette (6-8 colors max)
- Clear visual hierarchy
- Mobile layout consideration

Practice Exercises

43. Download Power BI Desktop. Connect to Sample - Superstore dataset. Create bar chart of Sales by Category.
44. In Power Query: Remove unused columns, change data types, filter out returns. Close & Apply.
45. Create relationships: Orders → Customers (CustomerID), Orders → Products (ProductID).
46. Create measures: Total Sales = SUM(Sales[Amount]), Profit Margin = DIVIDE([Total Profit], [Total Sales]).
47. Build dashboard with 5 visuals: Sales by Region (map), Sales Trend (line), Top Products (bar), KPI card, Matrix.
48. Add slicers for Date and Category. Configure cross-filtering.
49. Publish to Power BI Service. Share with colleague. Schedule daily refresh.

nasrin-a-mahin

UNIT IX – Tableau for Data Visualization

9.1 Introduction to Tableau

Tableau is the industry-leading data visualization and business intelligence platform. Founded in 2003 at Stanford University and acquired by Salesforce in 2019 for $15.7 billion, Tableau revolutionized analytics by making it accessible to business users without requiring programming or SQL knowledge. Its drag-and-drop interface, combined with powerful analytical capabilities, has made it the preferred tool for data storytelling in organizations worldwide.

Tableau transforms raw data into interactive, visually compelling dashboards that reveal insights at a glance. Unlike traditional BI tools that require IT involvement for every report, Tableau empowers analysts and business users to explore data independently, ask new questions on the fly, and share discoveries instantly. This self-service approach has democratized data analytics across industries — from retail and finance to healthcare and government.

Why Tableau?

- Intuitive Interface — Drag-and-drop simplicity. No coding required. Analysts can focus on insights rather than syntax.
- Speed — Connect to data and create visualizations in minutes. Real-time analysis without waiting for IT.
- Interactive Dashboards — Users can filter, drill down, and explore data dynamically. Not static reports.
- Visual Best Practices — Built-in design principles guide users toward effective visualizations. "Show Me" panel suggests appropriate chart types.
- Scalability — From personal analysis to enterprise-wide deployment. Handles millions of rows with optimized data engines.
- Community and Resources — Tableau Public gallery, forums, training, and certifications. Extensive learning ecosystem.

Tableau's Position in the BI Market

Tableau consistently ranks in the Gartner Magic Quadrant Leaders category for Analytics and Business Intelligence Platforms. Its primary competitors include Microsoft Power BI (tight Excel/Office integration), Qlik (associative analytics), and Looker (cloud-native SQL-based). Tableau's strengths lie in visual analytics sophistication, ease of use, and community-driven innovation.

Tool	Strength	Best For

nasim-a-mahin

Tableau	Visual analytics, exploration, ease of use	Analysts, data visualization professionals
Power BI	Microsoft ecosystem integration, cost	Organizations using Office 365, Excel users
Qlik	Associative engine, data relationships	Complex data models, governed BI
Looker	LookML (SQL modeling), cloud-native	Technical teams, data engineering

nasrin-a-mahin

9.2 Tableau Products

Tableau offers a family of products tailored to different roles and deployment scenarios. Understanding which product fits your needs is critical for licensing, workflow design, and architecture decisions.

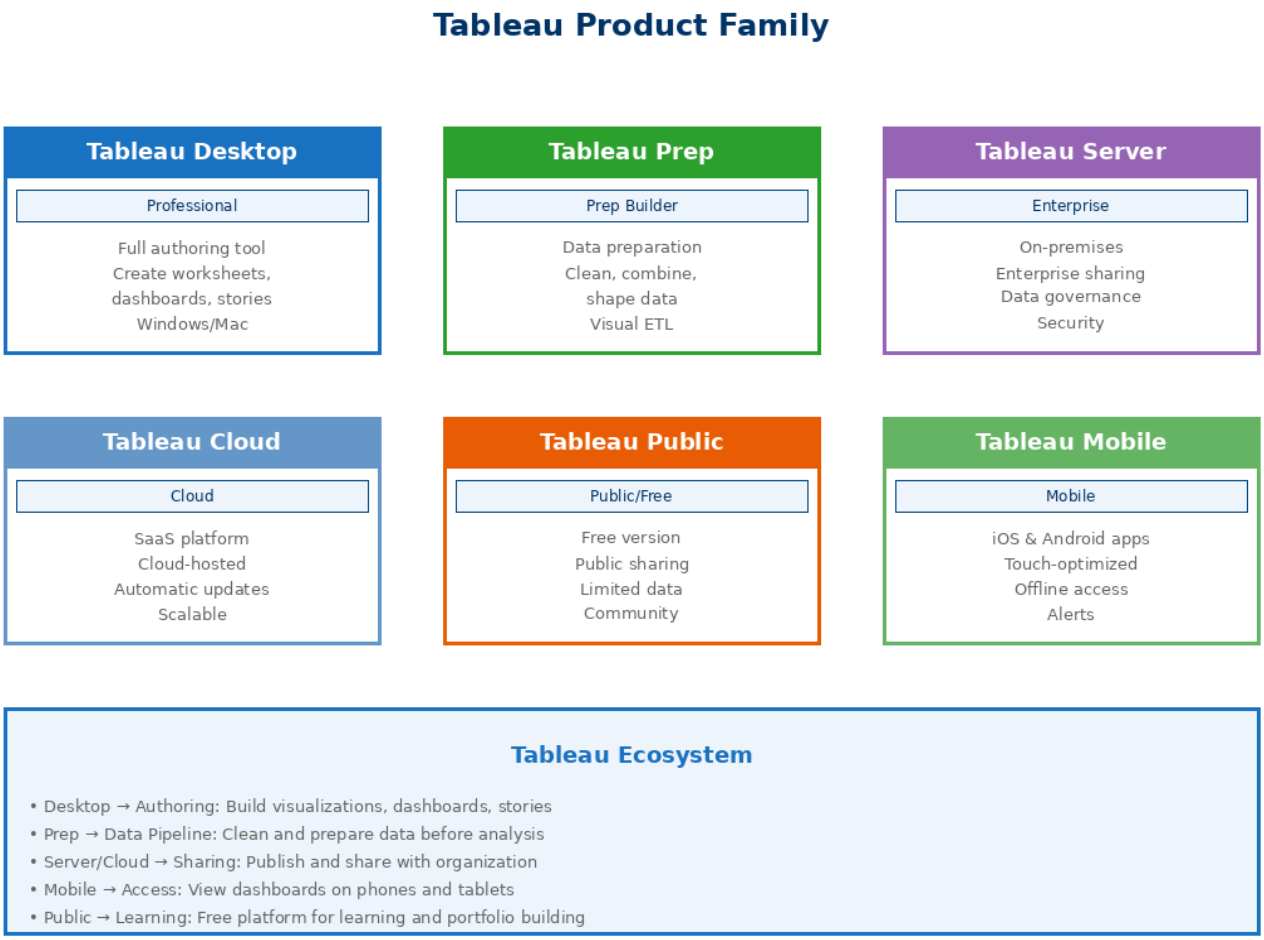


Tableau Desktop

The full-featured authoring tool for creating visualizations, dashboards, and stories. Available in two editions:

Tableau Desktop Professional

- Connect to all data sources (databases, cloud, files, APIs)
- Publish to Tableau Server or Tableau Cloud
- Create and edit data sources
- Full advanced analytics (statistics, forecasting, clustering)
- License: ~$840/year per user

Tableau Desktop Personal (discontinued in 2021)

Previously available for individuals, now replaced by Tableau Creator license bundles.

Tableau Prep Builder

Visual data preparation tool for cleaning, shaping, and combining data before analysis. Think of it as an ETL (Extract, Transform, Load) tool with a visual interface.

- Clean data (remove nulls, fix data types, rename fields)
- Combine datasets (joins, unions, pivots)
- Create reusable flows
- Output cleaned data for Tableau Desktop or databases
- Included with Tableau Creator license

Tableau Server

On-premises enterprise platform for sharing and governing Tableau content. Installed on your own infrastructure (Windows or Linux).

- Centralized repository for dashboards and data sources
- Role-based access control and permissions
- Scheduled data refreshes
- Custom branding and embedding
- Full control over data location and security
- License: Per-user or per-core pricing

Tableau Cloud (formerly Tableau Online)

SaaS version of Tableau Server. Fully managed by Tableau/Salesforce — no infrastructure to maintain.

- Same features as Tableau Server
- Automatic updates and maintenance
- Scalable cloud infrastructure
- 99.9% uptime SLA
- License: Per-user subscription

Tableau Public

Free version of Tableau Desktop for creating visualizations that are publicly shared. Ideal for students, journalists, and portfolio building.

nasrin-a-mahin

- Completely free
- Publish to public Tableau Public gallery
- Cannot save work locally (must publish online)
- 10GB storage limit
- Data is publicly visible (no private dashboards)

Tableau Mobile

iOS and Android apps for viewing dashboards on phones and tablets. Touch-optimized interface with offline access.

- View published dashboards
- Filter and interact
- Subscribe to dashboards for updates
- Data alerts and notifications

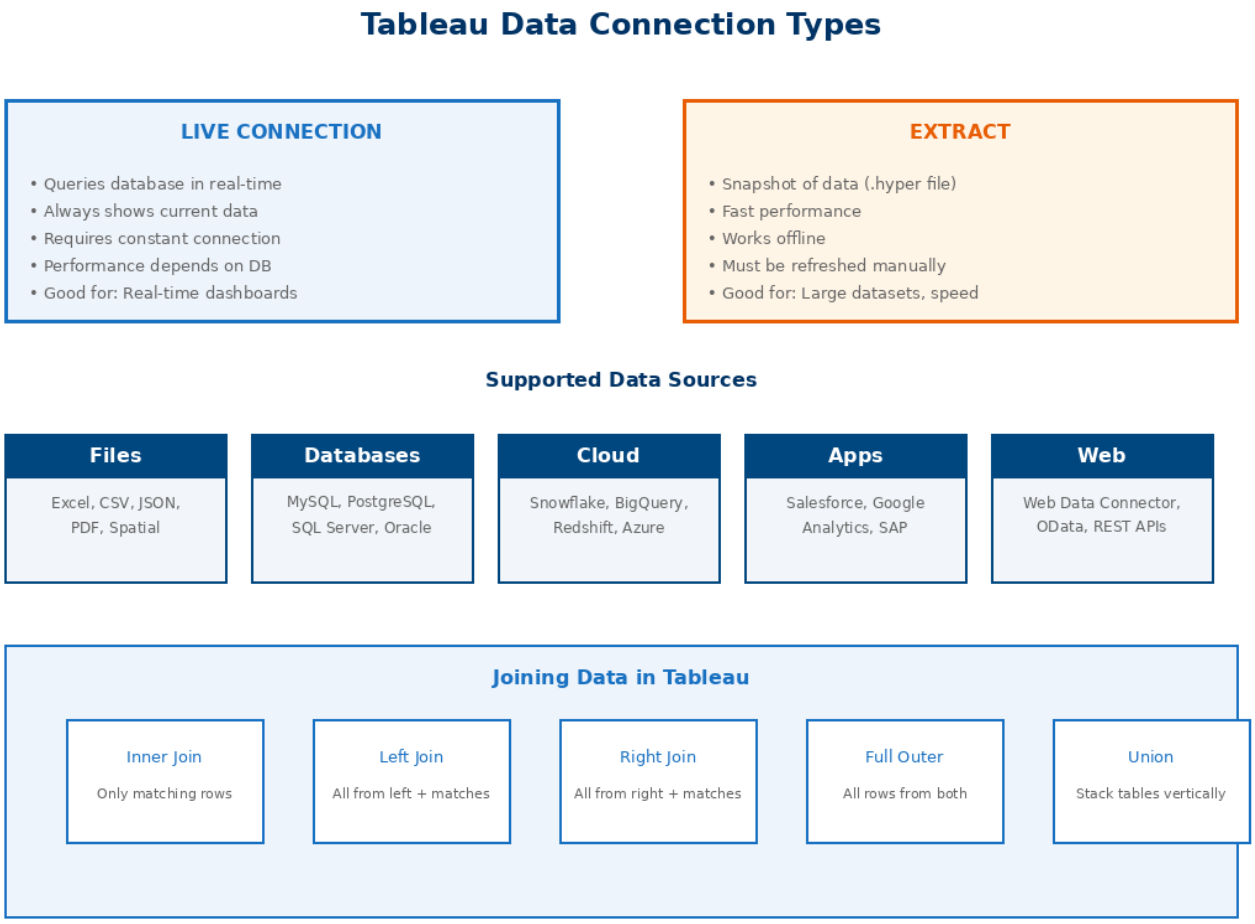
License Roles

Role	Capabilities	Typical User
Creator	Desktop + Prep + Server/Cloud (full access)	Data analysts, BI developers
Explorer	Edit & publish on web, limited authoring	Business analysts, managers
Viewer	View and interact with dashboards only	Executives, end users

nasrin mahin

9.3 Connecting to Data Sources

Tableau connects to virtually any data source — from Excel files to cloud data warehouses. Understanding connection types and join strategies is fundamental to building efficient, scalable analytics.



Live Connection vs Extract

The first major decision when connecting to data is whether to use a live connection or create an extract.

	Live Connection	Extract
Data Freshness	Real-time (always current)	Snapshot at refresh time
Performance	Depends on database speed	Fast (local .hyper file)

Offline Access	Requires connection	Works offline
Data Volume	Full database access	Subset (filtered)
Use Case	Real-time dashboards, small queries	Large datasets, complex calculations, mobile

Best Practice: Use extracts for large datasets (>1M rows), slow databases, or when performance matters. Use live connections for real-time monitoring or when data changes constantly. You can schedule extract refreshes (hourly, daily, weekly).

Supported Data Sources

Tableau connects to 100+ data sources out of the box:

Files

- Excel (.xlsx, .xls) — Most common for analysts
- CSV, TSV — Text files
- JSON — Nested data structures
- PDF — Extract tables from PDFs (Tableau 2020.2+)
- Spatial Files — Shapefiles, KML, GeoJSON for mapping

Databases

- Microsoft SQL Server, MySQL, PostgreSQL, Oracle
- Amazon Redshift, Google BigQuery, Snowflake
- Teradata, SAP HANA, IBM DB2
- MongoDB, Cassandra (NoSQL)

Cloud & SaaS Applications

- Salesforce, Google Analytics, Adobe Analytics
- Amazon Athena, Azure SQL
- Dropbox, OneDrive

Other

- Web Data Connector — Custom APIs and web services
- OData — Open data protocol

- ODBC — Generic database connector

Joining Data

Tableau supports multiple methods for combining datasets:

Joins (Horizontal)

- Inner Join — Only matching rows from both tables
- Left Join — All rows from left table + matches from right
- Right Join — All rows from right table + matches from left
- Full Outer Join — All rows from both tables

Union (Vertical)

Stack tables with the same structure on top of each other. Useful for combining monthly files or multiple years of data.

Blending

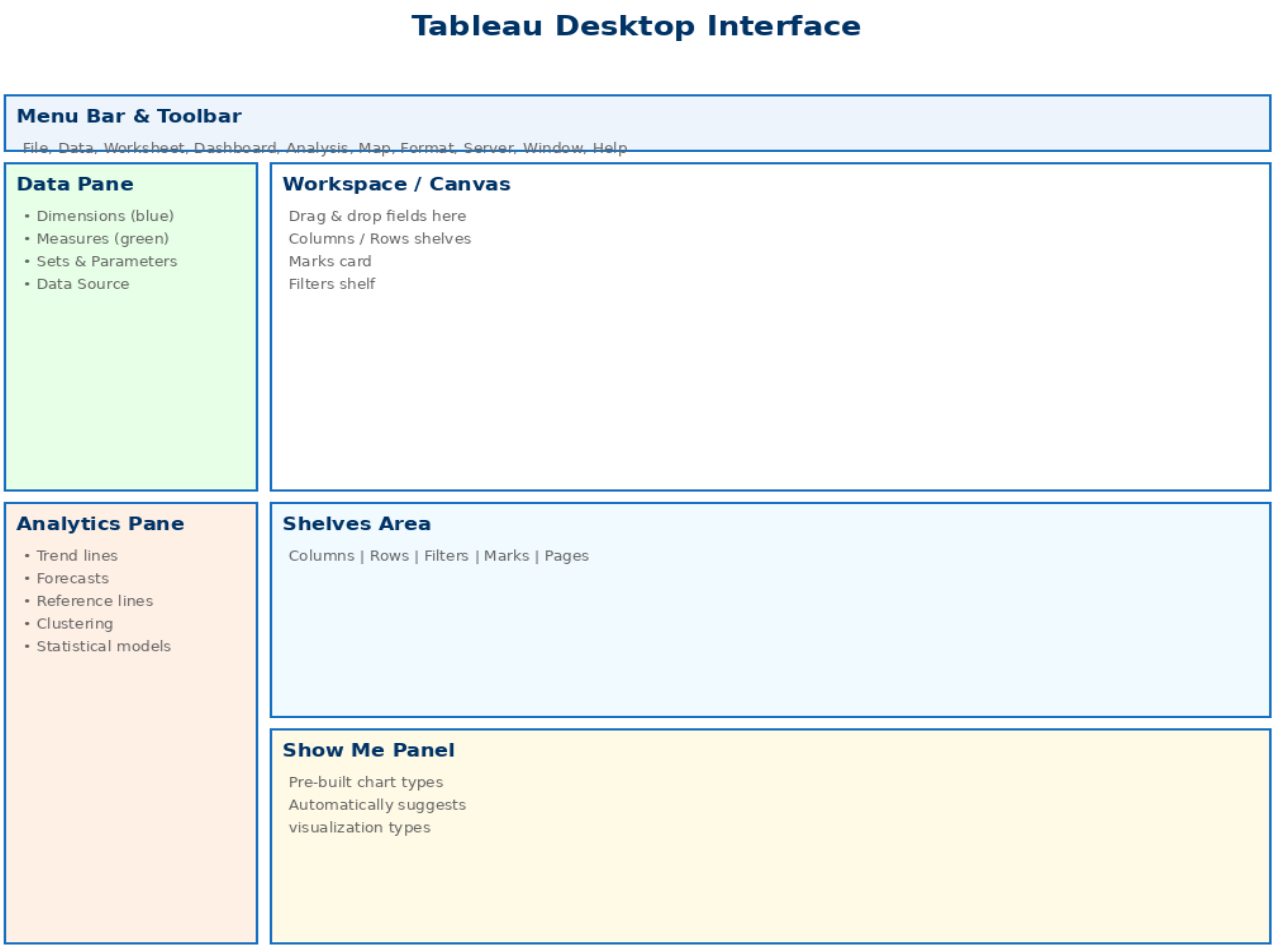
Combine data from different sources in a single visualization. Unlike joins (which happen at data source level), blending occurs at the worksheet level. Useful when joining isn't possible (different databases, different granularity).

Performance Tip: Joins and unions happen before Tableau loads data (efficient). Blending happens during visualization (slower). Use joins when possible. For very large datasets, pre-aggregate in the database before connecting to Tableau.

nasin-a-mahin

9.4 Tableau Workspace and Interface

Tableau Desktop's interface is organized around drag-and-drop simplicity. Understanding the key components — Data Pane, Canvas, Shelves, and Marks Card — is essential for efficient workflow.



Key Interface Components

1. Data Pane (Left Side)

Lists all available fields from your data source, divided into Dimensions (blue) and Measures (green).

Dimensions vs Measures

DIMENSIONS

(Qualitative / Categorical)

Color: Blue pill

Data Types:
- Text (Customer Name)
- Dates (Order Date)
- Boolean (True/False)
- Geographic (City, State)

Used For:
- Grouping data
- Creating categories
- Filtering
- Row/column headers

Examples:
- Product Category
- Region
- Customer Segment
- Order Date (year, quarter)

MEASURES

(Quantitative / Numerical)

Color: Green pill

Data Types:
- Numbers (integers, floats)
- Counts
- Aggregations

Used For:
- Calculations
- Aggregations (SUM, AVG)
- Quantitative analysis
- Y-axis values

Default Aggregation:
- SUM (totals)

Examples:
- Sales Amount
- Profit
- Quantity
- Discount %

Example: "Total Sales by Region" → Region (Dimension) | Sales (Measure)

- Dimensions (Blue Pills) — Categorical fields that slice data: Product Category, Region, Customer Name, Order Date (date parts)
- Measures (Green Pills) — Numerical fields that aggregate: Sales, Profit, Quantity, Discount
- Sets — Custom groups of dimension members (e.g., "Top 10 Customers")
- Parameters — User-controlled inputs (e.g., sales target)

2. Workspace / Canvas (Center)

The main area where visualizations appear. Surrounded by shelves where you drag fields.

3. Shelves (Top and Sides)

- Columns Shelf — Fields here create horizontal structure (x-axis)
- Rows Shelf — Fields here create vertical structure (y-axis)
- Filters Shelf — Limit data shown in the view
- Marks Card — Control visual properties (color, size, label, detail, tooltip)
- Pages Shelf — Create animations or step through data (rarely used)

4. Show Me Panel (Top Right)

Suggests appropriate chart types based on selected fields. Click a chart type to automatically build it.

5. Analytics Pane (Tab next to Data Pane)

Add statistical elements: trend lines, reference lines, forecasts, clusters, and more.

Building Your First Visualization

Basic workflow:

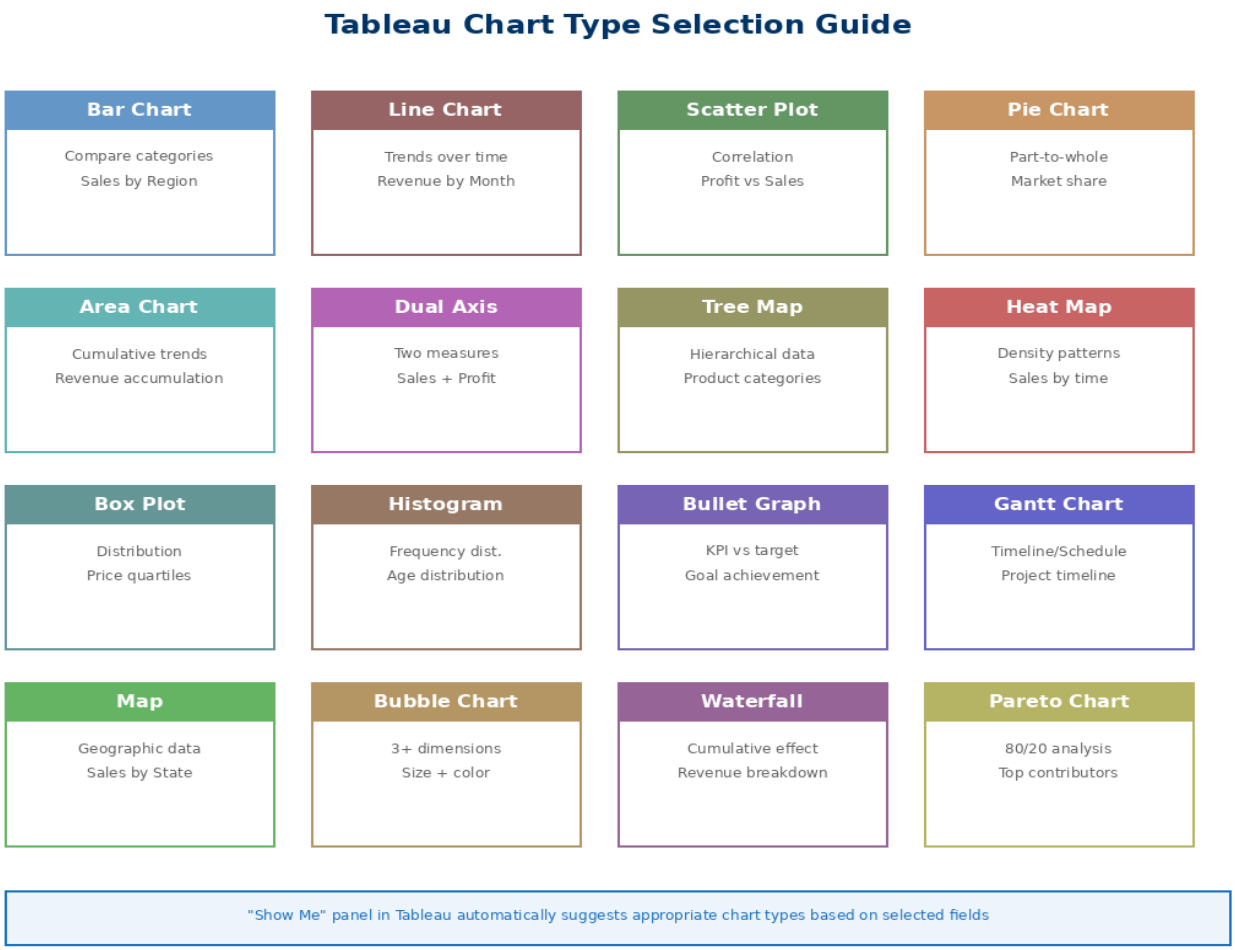
1. Connect to data source
2. Drag a Dimension to Columns (e.g., Region)
3. Drag a Measure to Rows (e.g., Sales)
4. Tableau automatically aggregates (SUM of Sales) and creates a bar chart
5. Customize using Marks Card (color, labels, tooltips)
6. Add filters to focus on specific data

Pills and Shelves: Fields become "pills" when dragged onto shelves. Blue pills = dimensions (discrete). Green pills = measures (continuous). Right-click any pill to change aggregation, format, or convert between dimension/measure.

nasrin mahin

9.5 Building Basic Visualizations

Tableau supports dozens of chart types. Mastering the most common visualizations — bar charts, line charts, scatter plots, maps, and heat maps — covers 90% of analytical needs.



Bar Chart — Comparing Categories

Most common chart type. Compare values across categories.

Steps:

7. Drag Dimension to Rows (e.g., Product Category)
8. Drag Measure to Columns (e.g., Sales)
9. Tableau creates horizontal bars
10. Swap Rows/Columns for vertical bars
11. Sort descending: Click sort icon on toolbar or right-click axis
12. Add color: Drag another dimension to Color on Marks Card

When to Use:

- Comparing sales by region
- Ranking top products
- Showing survey responses by category

Line Chart — Trends Over Time

Show how values change over time. Essential for time-series analysis.

Steps:

13. Drag Date field to Columns
14. Drag Measure to Rows (e.g., Revenue)
15. Right-click date pill → select granularity (Year, Quarter, Month, Day)
16. Add multiple measures to show comparisons (dual axis or separate lines)

When to Use:

- Revenue trends by month
- Website traffic over time
- Stock prices, temperature changes

Scatter Plot — Correlation and Outliers

Show relationship between two measures. Each mark represents one data point.

Steps:

17. Drag first Measure to Columns (e.g., Sales)
18. Drag second Measure to Rows (e.g., Profit)
19. Change mark type to Circle (on Marks Card)
20. Drag Dimension to Detail to separate marks (e.g., Customer)
21. Add Color and Size for additional dimensions

When to Use:

- Sales vs Profit correlation
- Finding outliers (high sales, low profit)
- Marketing: Reach vs Engagement

nasrin-a-mahin

Maps — Geographic Analysis

Tableau has built-in geocoding for countries, states, cities, postal codes.

Steps:

22. Drag geographic field (State, Country, City) to the canvas
23. Tableau automatically creates a map
24. Drag Measure to Color or Size (e.g., Sales)
25. Adjust map style: Map menu → Background Maps
26. Add filters to focus on specific regions

Map Types:

- Symbol Map — Circles sized by measure
- Filled Map — Regions colored by measure (choropleth)
- Density Map — Heat map showing concentration

Heat Map — Patterns in Tables

Color-coded table showing relationships between two dimensions.

Steps:

27. Drag first Dimension to Columns (e.g., Month)
28. Drag second Dimension to Rows (e.g., Product Category)
29. Drag Measure to Color (e.g., Sales)
30. Change mark type to Square
31. Adjust color palette: Click Color → Edit Colors

When to Use:

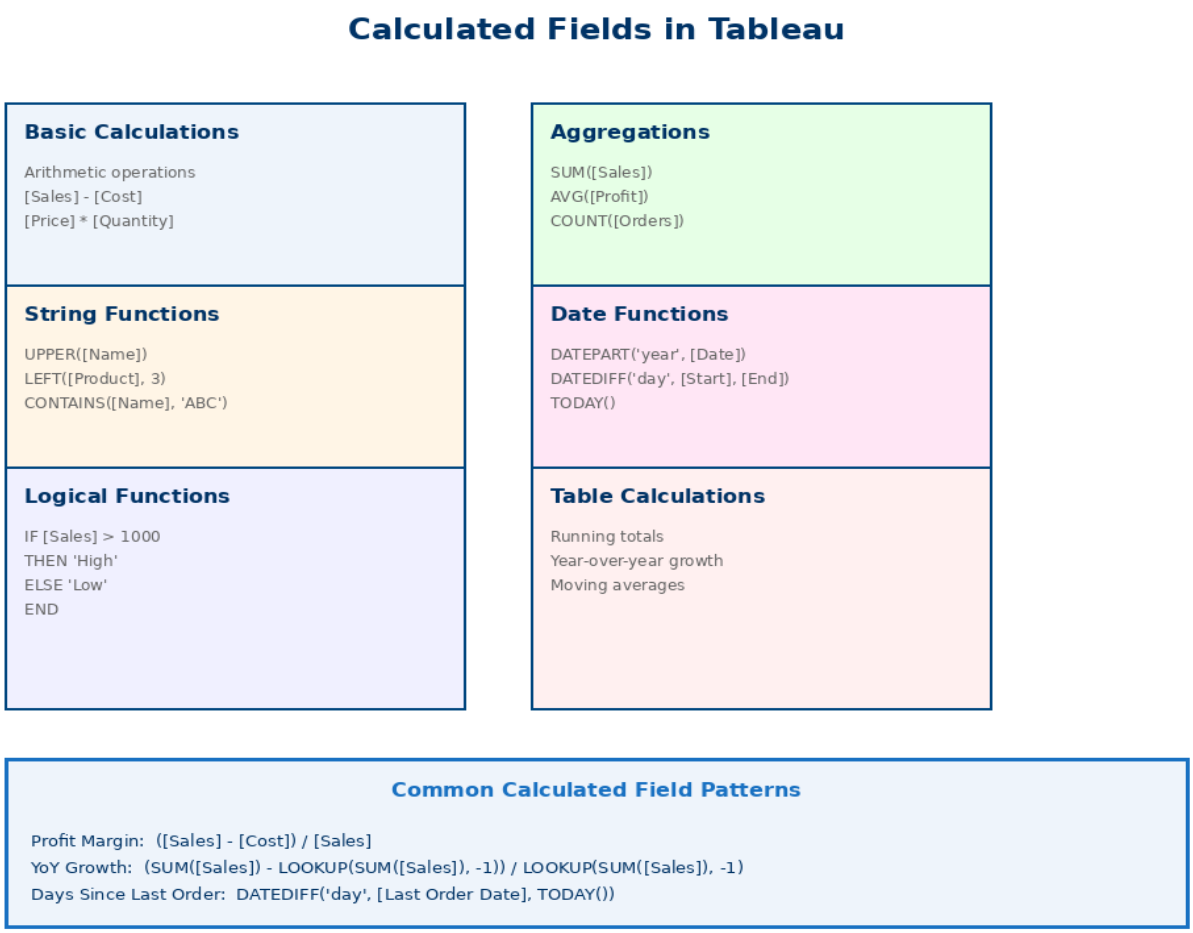
- Sales by month and category (identify seasonal patterns)
- Website traffic by day of week and hour
- Any time you need a colored pivot table

Show Me Panel: Select fields in Data Pane, then click Show Me. Tableau highlights appropriate chart types and builds them automatically. Great for exploring options quickly.

nasrin-a-mahin

9.6 Calculated Fields and Parameters

Calculated fields extend your data by creating new fields based on existing ones. Parameters add user-controlled inputs for interactive dashboards. Together, they unlock advanced analytics.



Calculated Fields

Create new fields using formulas. Similar to Excel formulas but more powerful.

Creating a Calculated Field

32. Right-click in Data Pane → Create Calculated Field
33. Name the field
34. Write formula using Tableau functions
35. Click OK
36. New field appears in Data Pane (can be Dimension or Measure)

Common Calculated Field Examples

Profit Margin:

[Profit] / [Sales]

Returns decimal (format as percentage: right-click field → Default Properties → Number Format → Percentage)

Discount Amount:

[Sales] * [Discount]

Full Name:

[First Name] + " " + [Last Name]

Sales Category (IF statement):

IF [Sales] > 10000 THEN "High"

ELSEIF [Sales] > 5000 THEN "Medium"

ELSE "Low"

END

Year-over-Year Growth:

(SUM([Sales]) - LOOKUP(SUM([Sales]), -1)) / LOOKUP(SUM([Sales]), -1)

(Table calculation — shows growth from previous year)

Function Categories

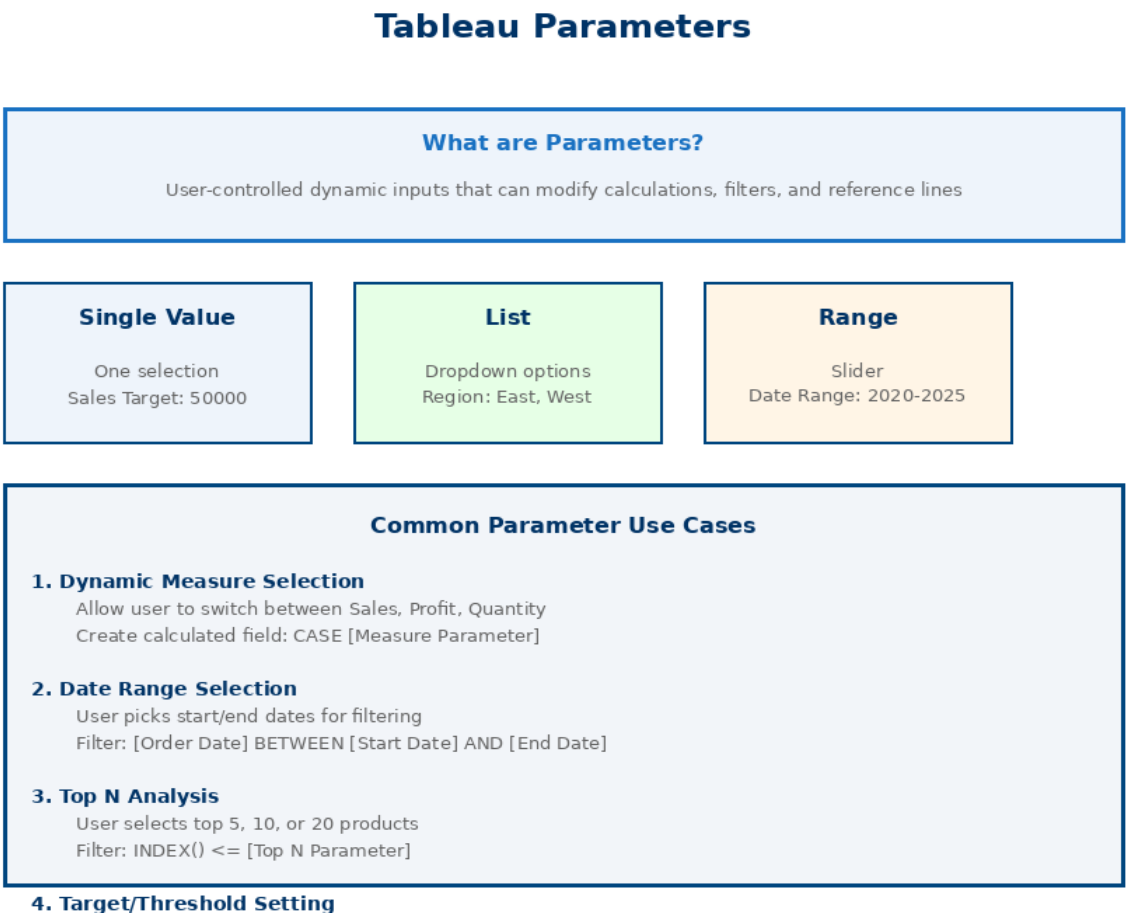
Category	Examples	Use Case
Number	ROUND, ABS, CEILING, FLOOR	Mathematical operations
String	UPPER, LOWER, LEFT, RIGHT, CONTAINS, SPLIT	Text manipulation

nasrin-a-mahin

Date	DATEPART, DATEDIFF, DATEADD, TODAY, NOW	Date calculations
Logical	IF, CASE, IIF, IFNULL, ISNULL	Conditional logic
Aggregation	SUM, AVG, MIN, MAX, COUNT, COUNTD	Summarize data
Table Calc	RUNNING_SUM, WINDOW_AVG, RANK, INDEX	Across-row calculations

Parameters

User-controlled inputs that make dashboards interactive. Unlike filters (which filter data), parameters are variables used in calculations.



Creating a Parameter

37. Right-click in Data Pane → Create Parameter
38. Set Name, Data Type, and Values (single value, list, or range)
39. Click OK
40. Right-click parameter → Show Parameter Control (adds slider/dropdown to dashboard)

Example: Dynamic Measure Selector

Allow user to switch between Sales, Profit, and Quantity in a single chart.

Step 1: Create Parameter

- Name: "Select Measure"
- Data Type: String
- List: Sales, Profit, Quantity

Step 2: Create Calculated Field using Parameter

- Name: "Dynamic Measure"
- Formula:

```
CASE [Select Measure]
  WHEN "Sales" THEN SUM([Sales])
  WHEN "Profit" THEN SUM([Profit])
  WHEN "Quantity" THEN SUM([Quantity])
END
```

Step 3: Use Dynamic Measure in visualization

- Drag "Dynamic Measure" to Rows
- Show parameter control
- Users can now switch measures with dropdown

Other Parameter Use Cases

- Top N filter (show top 5, 10, or 20 products)
- Date range selection
- Target/threshold lines
- What-if scenarios (change assumptions in financial models)

nasrin-a-mahin

9.7 Dashboards and Stories

Dashboards combine multiple visualizations into a single interactive view. Stories sequence dashboards to tell a narrative. Both are essential for communicating insights to stakeholders.

Creating a Dashboard

41. Create multiple worksheets (individual visualizations)
42. Click Dashboard → New Dashboard
43. Drag worksheets from left panel onto canvas
44. Arrange using containers (Horizontal, Vertical)
45. Add interactivity (filters, actions)
46. Format and publish

Dashboard Objects

- Worksheets — Your visualizations
- Horizontal/Vertical Containers — Layout structure
- Text — Titles, instructions, annotations
- Images — Logos, icons
- Web Page — Embed external content
- Blank — Spacers for layout

Dashboard Sizing

Dashboard menu → Size → Choose:

- Fixed Size — Specific dimensions (1920×1080 for presentations)
- Automatic — Adjusts to window size
- Range — Min/max dimensions
- Device Layouts — Separate designs for desktop/tablet/phone

Dashboard Actions

Make dashboards interactive. Actions connect visualizations so clicking in one affects others.

Filter Action

Clicking a mark in one sheet filters other sheets.

47. Dashboard → Actions → Add Action → Filter
48. Source: Chart where user clicks
49. Target: Charts that get filtered

nasrin-a-mahin

50. Select "Selected fields" to control which dimension filters

Example: Click a region on a map → Bar chart shows only that region's data.

Highlight Action

Hovering or clicking highlights related marks across sheets.

URL Action

Click a mark to open a URL (e.g., drill to external report, Google search).

Stories

Stories are sequences of dashboards and worksheets presented like slides. Use for presentations and guided analytics.

51. Story → New Story
52. Drag dashboards/worksheets onto story points
53. Add captions for each story point
54. Navigate forward/backward with arrows

Example Story Flow:

- Story Point 1: Executive summary dashboard (high-level KPIs)
- Story Point 2: Regional breakdown
- Story Point 3: Product performance deep dive
- Story Point 4: Recommendations

nasrin-a-mahin

Dashboard Design Best Practices

Layout Principles

- ✓ F-pattern reading flow
- ✓ Most important viz top-left
- ✓ Consistent spacing & alignment
- ✓ White space for clarity
- ✓ Limit to 5-7 visualizations
- ✓ Mobile-responsive design

Color Principles

- ✓ Consistent color palette
- ✓ Max 6-8 colors
- ✓ Use color purposefully
- ✓ Colorblind-safe palettes
- ✓ Highlight key insights
- ✓ Avoid rainbow colors

Interactivity

- ✓ Filters for exploration
- ✓ Actions (filter, highlight, URL)
- ✓ Parameters for flexibility
- ✓ Tooltips with details
- ✓ Drill-down capabilities
- ✓ Clear instructions

Performance

- ✓ Use extracts for speed
- ✓ Limit data to what's needed
- ✓ Aggregate before visualizing
- ✓ Minimize complex calculations
- ✓ Reduce mark count
- ✓ Test with real data volume

Pre-Publication Checklist

- ☐ Title clearly states dashboard purpose
- ☐ All visualizations have descriptive titles
- ☐ Axis labels are clear and include units
- ☐ Legends are necessary and well-positioned
- ☐ Filters are intuitive and labeled
- ☐ Color scheme is consistent and accessible
- ☐ No misleading visualizations (truncated axes, inappropriate chart types)
- ☐ Data sources and last refresh date visible
- ☐ Performance is acceptable (<3 seconds load time)
- ☐ Tested on target devices (desktop/mobile)
- ☐ Instructions provided for interactive elements
- ☐ Published to appropriate workspace with correct permissions

nasrin-

9.8 Advanced Features

Tableau's advanced features enable sophisticated analytics beyond basic charts and dashboards.

1. Level of Detail (LOD) Expressions

Control the granularity of calculations independently of the visualization. Three types: FIXED, INCLUDE, EXCLUDE.

Example: Sales per Customer (regardless of view granularity)

{ FIXED [Customer Name] : SUM([Sales]) }

Example: Average sales per region (include region even if not in view)

{ INCLUDE [Region] : AVG([Sales]) }

Use Case: Calculate customer lifetime value, cohort analysis, or per-entity metrics.

2. Table Calculations

Calculations that operate on the results of a query (after aggregation). Enable running totals, percent of total, moving averages, rankings.

- Running Total: Add a table calc to SUM(Sales) → Running Total
- Percent of Total: Table Calc → Percent of Total
- Rank: RANK() function
- Moving Average: WINDOW_AVG(SUM([Sales]), -2, 0) for 3-period moving average

3. Forecasting

Built-in time-series forecasting using exponential smoothing.

55. Create a time-series line chart (Date on Columns, Measure on Rows)
56. Analytics pane → Drag "Forecast" to canvas
57. Tableau automatically extends the line with 95% confidence interval
58. Edit forecast → Adjust model, length, and seasonality

nasrin-a-mahin

4. Clustering

K-means clustering to group similar data points. Useful for customer segmentation.

59. Create scatter plot (2+ measures)
60. Analytics pane → Drag "Cluster" to canvas
61. Tableau finds optimal number of clusters
62. Each point is color-coded by cluster

5. Reference Lines and Bands

Add context to charts with average lines, constant lines, or confidence bands.

- Right-click axis → Add Reference Line
- Options: Constant, Average, Median, Percentile, Table calc
- Use for targets, thresholds, benchmarks

6. Sets

Dynamic groups of dimension members. Create "Top 10 Customers" or "High-Performing Products" that update automatically.

63. Right-click dimension → Create → Set
64. Select members manually OR use condition (e.g., SUM(Sales) > 10000)
65. Use set as dimension (In/Out) or in filters

7. Data Blending

Combine data from different sources in a single view when joins aren't possible. Common when integrating SaaS data with internal databases.

66. Connect to primary data source
67. Add a second data source
68. Tableau automatically blends on common dimensions (shown with link icon)
69. Edit blend relationships if needed

nasrin-a-mahin

LOD vs Table Calcs: LOD expressions aggregate at a fixed level (before visualization). Table calculations operate on the viz results (after aggregation). Use LOD for complex, multi-level aggregations. Use table calcs for row-by-row operations like running totals and rankings.

9.9 Best Practices

Following best practices ensures dashboards are clear, performant, and maintainable. These guidelines reflect lessons from thousands of Tableau implementations.

Data Preparation Best Practices

- Clean data before connecting — Fix nulls, inconsistent naming, and data types upstream
- Use extracts for large datasets — Orders of magnitude faster than live connections for >1M rows
- Aggregate before Tableau — Pre-aggregate in SQL or Tableau Prep for very large datasets
- Document data sources — Add descriptions, field comments, and data lineage notes
- Hide unused fields — Simplify Data Pane by hiding irrelevant fields

Visualization Design Best Practices

- Choose the right chart type — Bar for comparison, line for trends, scatter for correlation
- Avoid pie charts with >5 slices — Use bar charts instead for readability
- Start axes at zero — Unless showing deltas or variance (avoid misleading viewers)
- Use color purposefully — Not decoration. Limit to 6-8 colors. Use colorblind-safe palettes.
- Label directly on chart — Avoid forcing users to reference legends
- Remove chart junk — No 3D effects, unnecessary gridlines, or decorative elements
- Provide context — Add reference lines, benchmarks, or targets

Dashboard Design Best Practices

- One dashboard, one message — Clear purpose stated in title
- F-pattern layout — Most important content top-left, details bottom-right
- Limit to 5-7 visualizations — Too many overwhelms users
- Consistent layout — Align edges, use containers, maintain spacing
- Mobile-friendly — Test Device Layouts for phones/tablets
- Provide instructions — Tell users how to interact (filters, actions)
- Add filters strategically — Too many filters create decision paralysis

nasim-a-mahin

Performance Best Practices

- Use extracts — Fastest option for analysis
- Filter data early — Apply filters at data source level, not visualization level
- Minimize marks — More marks = slower rendering. Aggregate or filter.
- Simplify calculations — Complex nested LODs slow performance
- Use ATTR() not MIN() — ATTR is faster when you know there's only one value
- Avoid cross-database joins — Bring data together in Tableau Prep instead
- Test with real data volumes — Performance degrades nonlinearly with data size

Governance and Collaboration

- Use Server/Cloud Projects — Organize by department, topic, or audience
- Set permissions carefully — Row-level security for sensitive data
- Create certified data sources — Single source of truth for company metrics
- Version control — Save workbook versions before major changes
- Subscribe to dashboards — Email snapshots to stakeholders automatically
- Monitor usage analytics — Identify popular content and dead dashboards

Tableau Public Portfolio: Build your portfolio on Tableau Public. Top visualizations get featured. Employers look for Tableau Public profiles when hiring analysts. Include diverse viz types and storytelling.

Practice Exercises

70. Connect to Sample - Superstore dataset. Create bar chart showing Sales by Category. Sort descending.
71. Build line chart of Sales over time (by Month). Add trend line from Analytics pane.
72. Create scatter plot: Sales vs Profit. Color by Category. Add Size for Quantity. Identify outliers.
73. Build map showing Sales by State. Use filled map (choropleth). Add tooltip showing Profit Margin.
74. Create calculated field for Profit Margin: [Profit] / [Sales]. Format as percentage.
75. Build dashboard with 4 sheets: (1) Sales by Category bar, (2) Monthly trend line, (3) Regional map, (4) Top 10 Products. Add filter action from map.
76. Create parameter for Top N. Allow user to select top 5, 10, or 20 products. Use in filter.
77. Publish to Tableau Public. Add to portfolio with descriptive title and insights.

nasrin mahin

UNIT X: Machine Learning for Data Analytics

10.1 Introduction to Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence that enables computers to learn from data without being explicitly programmed. Instead of writing rules, we provide data and let algorithms discover patterns automatically.

> *Machine Learning is the science of getting computers to act without being explicitly programmed.*
> *— Andrew Ng, Stanford University*

What is Machine Learning?

Traditional programming involves writing explicit rules: if X happens, do Y. Machine Learning reverses this — we provide examples of X and Y, and the algorithm learns the rules.

Traditional Programming	Machine Learning
Input: Data + Rules	Input: Data + Answers (Labels)
Output: Answers	Output: Rules (Model)
Example: Calculate tax based on rules	Example: Learn spam patterns from examples
Static – rules must be updated manually	Dynamic – improves with more data

AI, Machine Learning, and Deep Learning

These terms are often confused. Here is the hierarchy:

- Artificial Intelligence (AI): Broadest concept — any technique that enables computers to mimic human intelligence
- Machine Learning (ML): Subset of AI — algorithms that learn from data
- Deep Learning (DL): Subset of ML — neural networks with multiple layers (mimics human brain)

Why Machine Learning Matters

- Handles complexity: Learns patterns too complex for manual rules (e.g., image recognition)
- Scales automatically: Performance improves with more data
- Adapts to change: Retrains on new data without rewriting code
- Discovers hidden patterns: Finds insights humans might miss
- Automates decisions: Makes predictions in real-time at scale

Real-World ML Success Stories

- Netflix: 80% of watched content comes from ML recommendations (saves $1B annually in retention)
- Gmail: Spam filter blocks 99.9% of spam using ML
- Tesla: Self-driving cars learn from billions of miles of driving data

- Amazon: 35% of revenue from ML-powered product recommendations
- Google Translate: Neural ML reduced translation errors by 60%

When to Use Machine Learning

Use ML when:

- The problem has patterns that repeat (e.g., fraud detection, customer churn)
- You have sufficient data (typically 1000+ examples minimum)
- The problem is too complex for explicit rules (e.g., facial recognition)
- The problem changes over time (e.g., stock prices, spam patterns)
- You need to scale (millions of decisions per day)

Do NOT use ML when:

- Simple rules suffice (e.g., calculate discount: if age > 65, discount = 10%)
- Insufficient data (< 100 examples)
- Explainability is critical and model must be transparent (ML models can be black boxes)
- The cost of errors is catastrophic (medical diagnosis without human oversight)

10.2 Types of Machine Learning

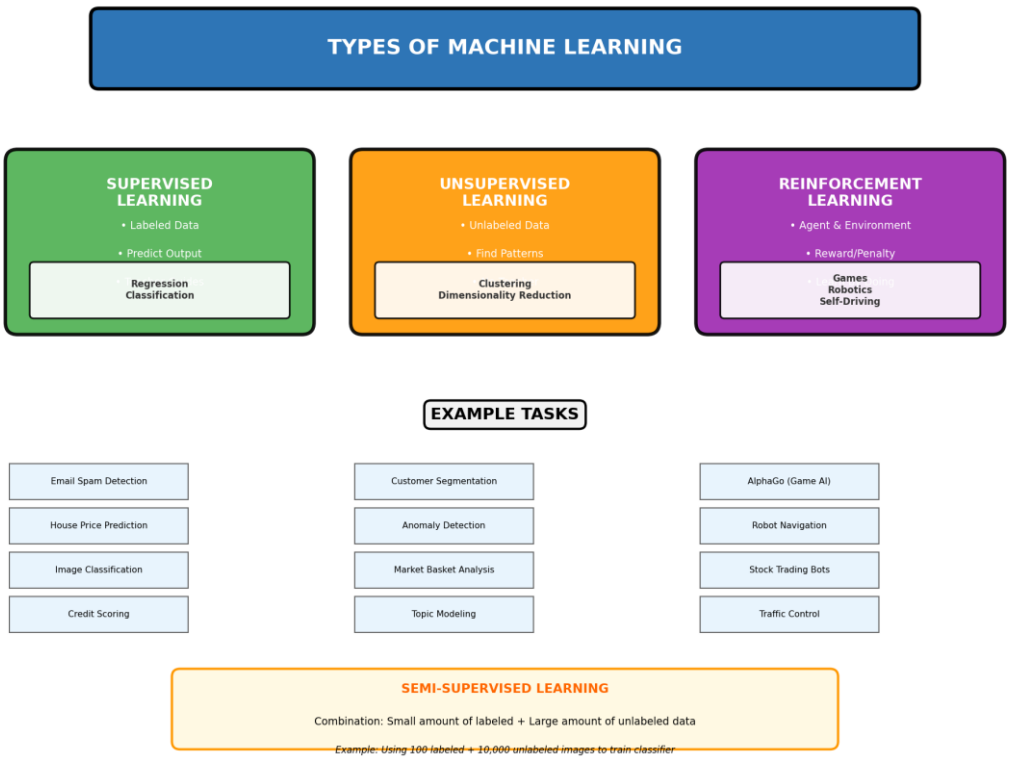


Figure 10.1 — Types of Machine Learning

1. Supervised Learning

Supervised learning uses labeled data — each example has an input and the correct output (label). The algorithm learns to map inputs to outputs.

> *Supervised = Learning with a teacher. You show the algorithm examples with answers, and it learns to predict answers for new examples.*

Two types:

- Regression: Predict continuous values (price, temperature, sales)
- Classification: Predict categories (spam/not spam, cat/dog, fraud/legitimate)

2. Unsupervised Learning

Unsupervised learning uses unlabeled data — no correct answers provided. The algorithm finds patterns, groups, or structure in the data on its own.

- Clustering: Group similar items together (customer segmentation)
- Dimensionality Reduction: Reduce number of features while preserving information (PCA)
- Anomaly Detection: Find unusual patterns (fraud, system failures)

3. Reinforcement Learning

Reinforcement learning trains an agent to make decisions by rewarding good actions and penalizing bad ones. The agent learns through trial and error.

- Agent: The learner (e.g., game AI, robot)
- Environment: The world the agent operates in
- Actions: Choices the agent makes
- Rewards/Penalties: Feedback on actions
- Goal: Maximize cumulative reward over time

Examples: AlphaGo (defeated world champion), Self-driving cars, Game AI, Robot navigation

4. Semi-Supervised Learning

Semi-supervised learning combines a small amount of labeled data with a large amount of unlabeled data. Useful when labeling data is expensive or time-consuming.

Example: Using 100 labeled images + 10,000 unlabeled images to train an image classifier.

Type	Data	Task	Example
Supervised	Labeled	Predict output	Email spam detection
Unsupervised	Unlabeled	Find patterns	Customer segmentation
Reinforcement	Reward signals	Learn actions	Game AI

nasim-a-mahin

Semi-Supervised	Mostly unlabeled	Predict with limited labels	Image classification

10.3 The Machine Learning Workflow

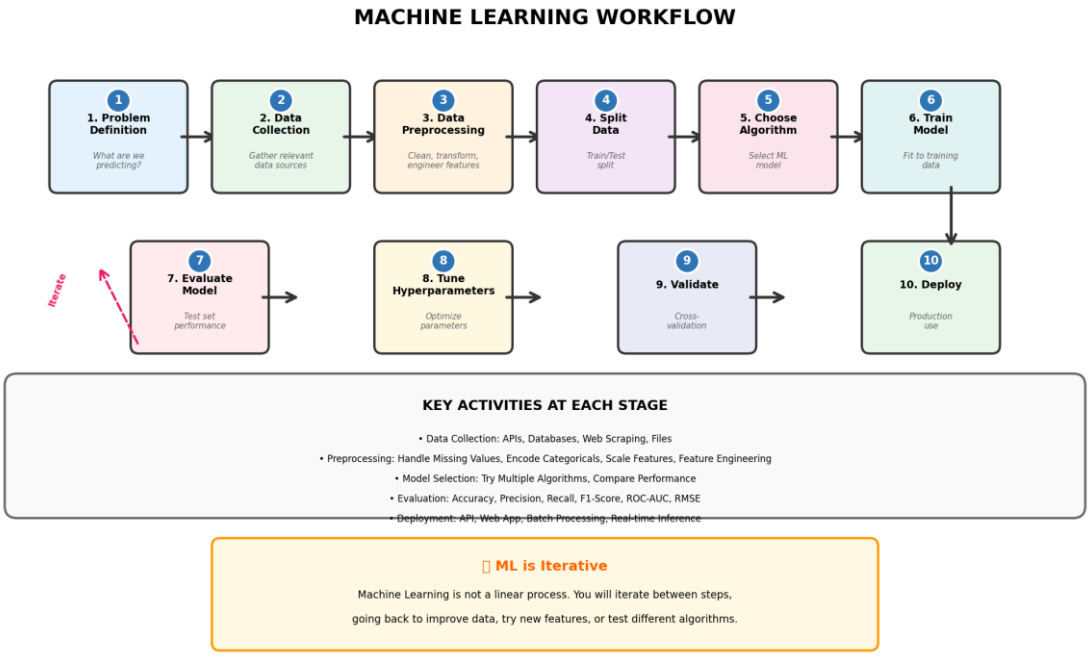


Figure 10.2 — The Machine Learning Workflow Pipeline

Machine Learning is an iterative process. You will cycle through these steps multiple times to improve your model.

74. Define the Problem: What are you predicting? Who will use it? What defines success?
75. Collect Data: Gather relevant data from databases, APIs, files, web scraping, or experiments
76. Explore Data: Understand distributions, missing values, outliers, and relationships (EDA)
77. Preprocess Data: Clean, transform, encode, scale, and engineer features
78. Split Data: Separate into training, validation, and test sets (typically 70/15/15 or 80/20)
79. Choose Algorithm: Select appropriate ML algorithm(s) based on problem type and data
80. Train Model: Fit the algorithm to training data — the model learns patterns
81. Evaluate Model: Test performance on validation/test set using appropriate metrics
82. Tune Hyperparameters: Optimize algorithm settings to improve performance
83. Validate: Use cross-validation to ensure model generalizes well
84. Deploy: Integrate model into production environment (API, web app, batch processing)
85. Monitor: Track performance over time and retrain when performance degrades

> *Machine Learning is 80% data preparation and 20% modeling. Clean, well-prepared data is the foundation of successful ML.*

10.4 Data Preprocessing for ML

Raw data is rarely ready for machine learning. Preprocessing transforms data into a format that algorithms can understand and learn from effectively.

1. Handling Missing Values

Missing data must be addressed before training. Three approaches:

- Drop rows/columns: If < 5% missing, drop rows. If a column is >50% missing, drop the column
- Impute with statistics: Fill with mean (numeric), median (if outliers), or mode (categorical)
- Advanced imputation: Use KNN imputer or model-based imputation (predict missing values)

```
# Python example
df.dropna()  # Drop rows with any missing value
df.fillna(df.mean())  # Fill with column mean
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
X = imputer.fit_transform(X)
```

2. Encoding Categorical Variables

ML algorithms require numeric input. Categorical variables must be converted to numbers.

- Label Encoding: Convert categories to integers (0, 1, 2, ...). Use for ordinal data (low/medium/high)
- One-Hot Encoding: Create binary columns for each category. Use for nominal data (red/blue/green)
- Target Encoding: Replace category with mean of target variable for that category

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["color_encoded"] = le.fit_transform(df["color"])


# One-Hot Encoding
df = pd.get_dummies(df, columns=["color"], drop_first=True)
```

3. Feature Scaling

Algorithms that use distance (KNN, SVM, Neural Networks) require features on similar scales.

Method	Formula	Range	When to Use
Normalization	(x - min) / (max - min)	[0, 1]	When distribution is NOT Gaussian

Standardization	(x - mean) / std	Mean=0, Std=1	When distribution is Gaussian

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler


# Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Normalization
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
```

10.5 Supervised Learning – Regression

Regression predicts continuous numeric values.

Algorithm	Best For	Pros	Cons
Linear Regression	Linear relationships	Simple, interpretable	Assumes linearity
Polynomial Regression	Non-linear curves	Captures curves	Overfits easily
Ridge/Lasso	High-dimensional data	Prevents overfitting	Requires tuning
Random Forest	Complex patterns	Handles non-linearity	Black box

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

10.6 Supervised Learning – Classification

Classification predicts categorical labels.

Algorithm	Best For	Strengths
Logistic Regression	Binary classification	Fast, interpretable
Decision Trees	Non-linear, interpretable	Visual, handles mixed data
Random Forest	High accuracy	Robust, reduces overfitting
SVM	High-dimensional data	Effective in complex spaces

KNN	Small datasets	Simple, no training phase
Naive Bayes	Text classification	Fast, works with little data

10.7 Unsupervised Learning – Clustering

Clustering groups similar items without labels.

- K-Means: Partition data into K clusters
- Hierarchical: Build tree of clusters
- DBSCAN: Density-based, finds arbitrary shapes

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)

clusters = kmeans.fit_predict(X)
```

10.8 Unsupervised Learning – Dimensionality Reduction

Reduce features while preserving information.

- PCA: Projects data onto principal components (linear)
- t-SNE: Visualize high-dimensional data in 2D/3D (non-linear)

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_reduced = pca.fit_transform(X)
```

nasrin-mamun

10.9 Model Evaluation Metrics

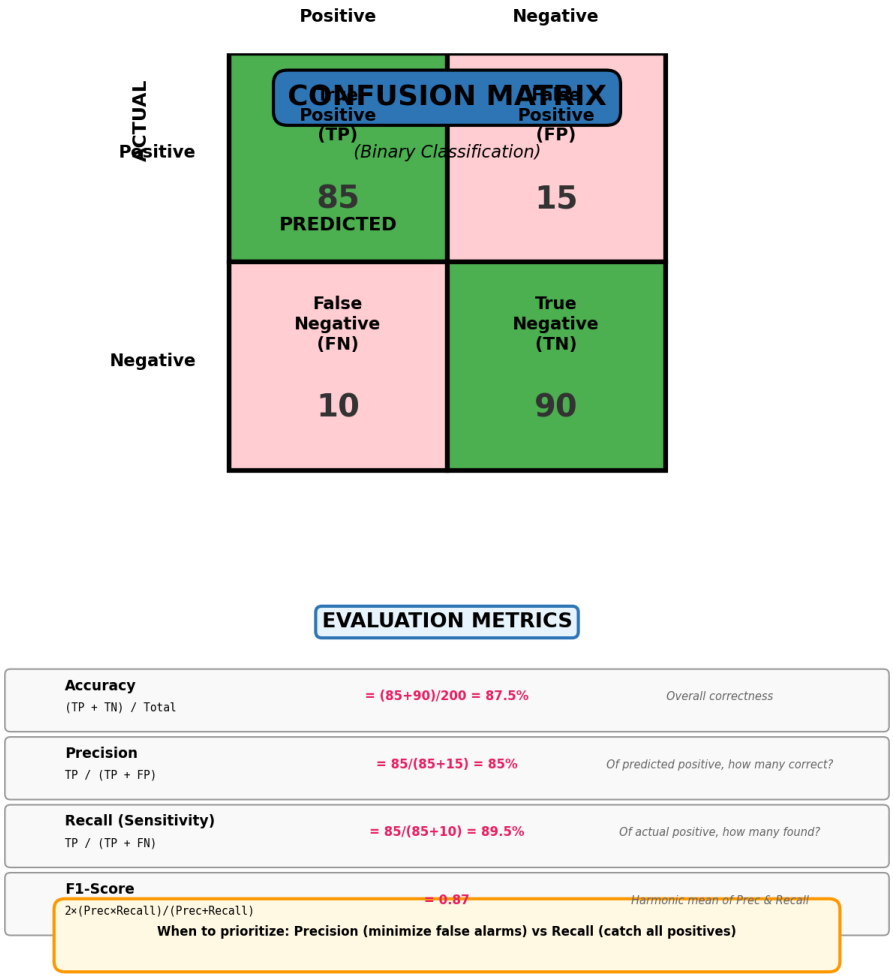


Figure 10.3 — Confusion Matrix and Evaluation Metrics

Regression Metrics

- MSE (Mean Squared Error): Average squared difference — penalizes large errors
- RMSE (Root Mean Squared Error): Square root of MSE — same units as target
- MAE (Mean Absolute Error): Average absolute difference — less sensitive to outliers
- R^2 (R-squared): Proportion of variance explained (0-1, higher is better)

Classification Metrics

- Accuracy: (TP + TN) / Total — use when classes are balanced
- Precision: TP / (TP + FP) — minimize false positives
- Recall: TP / (TP + FN) — minimize false negatives
- F1-Score: Harmonic mean of Precision & Recall — balance both
- ROC-AUC: Area under ROC curve — overall performance measure

10.10 Train-Test Split and Cross-Validation

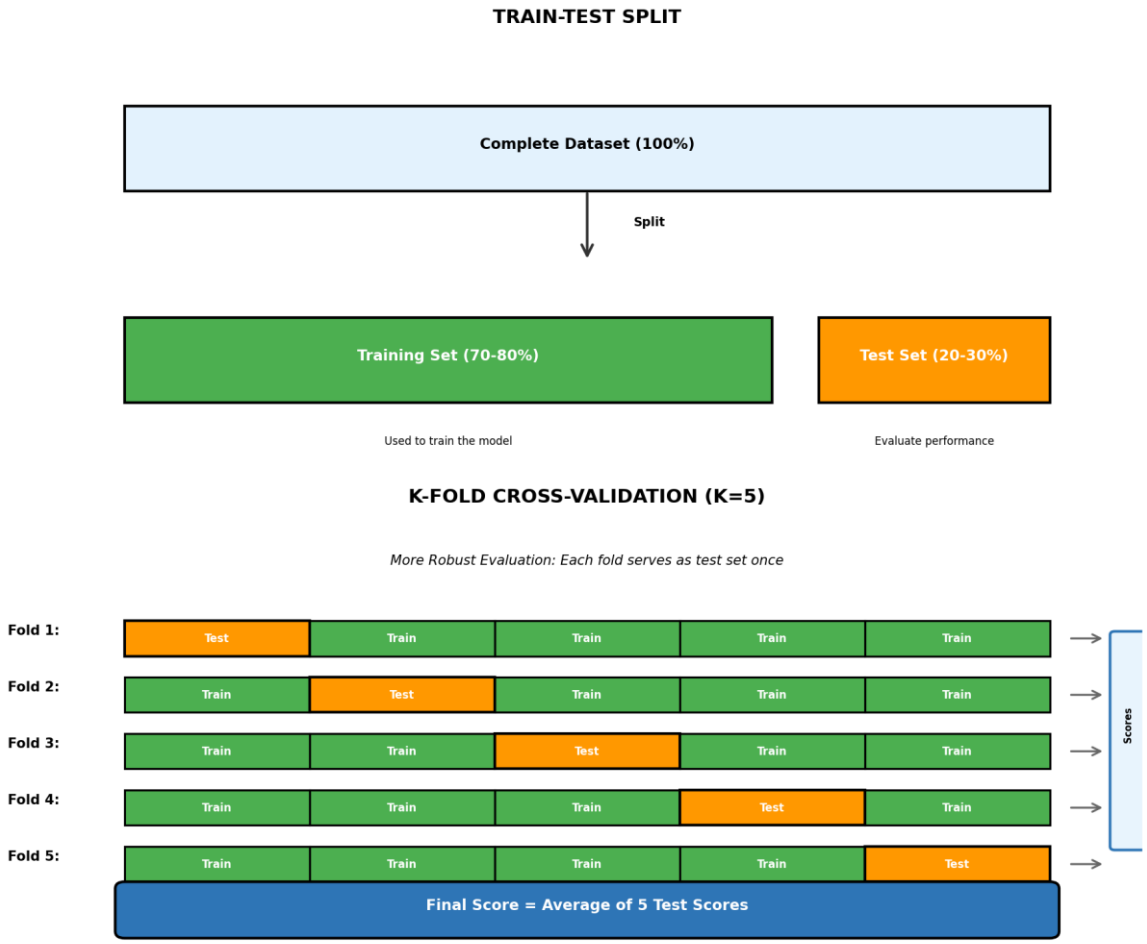


Figure 10.4 — Train-Test Split and K-Fold Cross-Validation

```
from sklearn.model_selection import train_test_split, cross_val_score


# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# K-Fold Cross-Validation
scores = cross_val_score(model, X, y, cv=5)
print(f"Average Score: {scores.mean():.3f} (+/- {scores.std():.3f})")
```

10.11 Overfitting, Underfitting, and the Bias-Variance Tradeoff

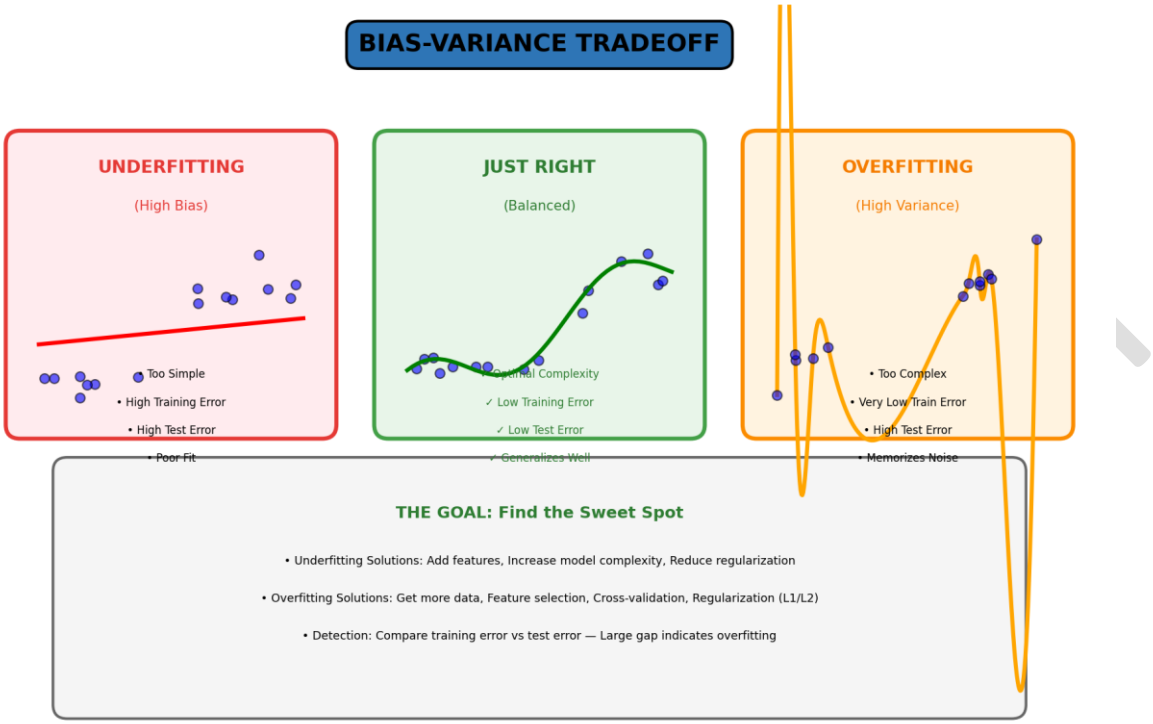


Figure 10.5 — Bias-Variance Tradeoff Visualization

> *The goal of machine learning is to find the sweet spot between underfitting (high bias) and overfitting (high variance) — a model that generalizes well to new, unseen data.*

Problem	Cause	Symptoms	Solutions
Underfitting	Model too simple	High training & test error	Add features, increase complexity
Overfitting	Model too complex	Low training error, high test error	More data, regularization, reduce features
Just Right	Balanced complexity	Low training & test error	Achieved through tuning and validation

10.12 Regularization Techniques

Regularization prevents overfitting by penalizing complex models.

- L1 (Lasso): Adds absolute value of coefficients to loss — performs feature selection
- L2 (Ridge): Adds squared value of coefficients to loss — shrinks all coefficients
- Elastic Net: Combines L1 + L2 — balance of both approaches
- Dropout: Randomly drop neurons during training (neural networks)
- Early Stopping: Stop training when validation error starts increasing

```
from sklearn.linear_model import Ridge, Lasso

ridge = Ridge(alpha=1.0)  # L2

lasso = Lasso(alpha=0.1)  # L1
```

10.13 Feature Engineering

Creating new features from existing ones often improves model performance more than choosing a better algorithm.

- Domain knowledge: Use expertise to create meaningful features
- Polynomial features: x^2 , x^3, $x \times y$
- Binning: Convert continuous to categorical (age → age_group)
- Log transformation: Reduce skewness in distributions
- Interaction terms: Multiply features that interact
- Date features: Extract year, month, day, day_of_week from dates

10.14 Machine Learning with Python – Scikit-Learn

Scikit-learn is the most popular ML library in Python.

```
# Complete ML Pipeline Example

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, accuracy_score



# 1. Load data

iris = load_iris()

X, y = iris.data, iris.target



# 2. Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# 3. Scale features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

```
# 4. Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 5. Predict
y_pred = model.predict(X_test)

# 6. Evaluate
print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
print(classification_report(y_test, y_pred))
```

10.15 Real-World ML Applications

Healthcare: Disease prediction, drug discovery, medical image analysis, patient risk stratification

Finance: Fraud detection, credit scoring, algorithmic trading, risk assessment, customer churn

E-commerce: Recommendation systems, demand forecasting, dynamic pricing, customer segmentation

Manufacturing: Predictive maintenance, quality control, supply chain optimization, defect detection

Marketing: Customer segmentation, churn prediction, sentiment analysis, ad targeting, CLV

Transportation: Self-driving cars, route optimization, demand prediction, traffic forecasting

Entertainment: Content recommendation (Netflix, Spotify), personalization, content generation

Security: Facial recognition, intrusion detection, malware detection, fraud prevention

10.16 ML Best Practices and Common Pitfalls

Best Practices

- Always split data before preprocessing (prevent data leakage)
- Use cross-validation for robust evaluation
- Scale features for distance-based algorithms
- Set random_state for reproducibility
- Try multiple algorithms before choosing
- Tune hyperparameters systematically (grid search/random search)

nasrin-a-ma

- Monitor for data leakage (test data influencing training)
- Document your process and decisions
- Version control your code and models
- Start simple, then add complexity

Common Pitfalls

> *Avoiding these mistakes will save hours of debugging and prevent incorrect results.*

- Training on test data (data leakage) — ensures overfitting
- Not scaling features for algorithms that need it
- Using accuracy on imbalanced datasets (use F1 or ROC-AUC)
- Forgetting to encode categorical variables
- Overfitting to training data (test error >> training error)
- Not handling missing values properly
- Using test set multiple times to tune (use validation set)
- Ignoring class imbalance (99% class A, 1% class B)
- Not checking for multicollinearity in regression
- Failing to understand your data (EDA is critical)

ML Project Checklist

86. Define clear success metrics before starting
87. Perform thorough exploratory data analysis
88. Handle missing values and outliers
89. Encode categorical variables appropriately
90. Scale/normalize features if needed
91. Split data into train/validation/test sets
92. Try baseline model first (simple model)
93. Experiment with multiple algorithms
94. Tune hyperparameters systematically
95. Evaluate on test set only once at the end
96. Document assumptions, decisions, and results
97. Plan for model deployment and monitoring

nasim-a-mahin

Unit-wise Summary & Practice Questions

nasrin-a-mahin

UNIT I — Overview of Data Analytics

This unit introduces the fundamentals of data analytics — what it is, why it matters, and how it is applied across industries. It lays the conceptual foundation for the entire course.

Key Topics Covered

- Introduction to Data Analytics — converting raw data into actionable insights
- Need for Data Analytics — data explosion, competitive pressure, business benefits
- Business Analytics vs Data Analytics — scope, tools, and objectives
- Types of Analytics — Descriptive, Diagnostic, Predictive, Prescriptive
- Data Scientist vs Data Engineer vs Data Analyst — roles and skill sets
- Role of a Data Analyst — responsibilities, tools, soft skills
- The Six-Step Data Analysis Process — Define → Collect → Clean → Analyse → Visualise → Interpret
- Tools & Technologies — Excel, SQL, Python, R, Power BI, Tableau
- Roadmap for Aspiring Data Analysts

Important Concepts to Remember

- Descriptive: 'What happened?' — reports, dashboards, summaries
- Diagnostic: 'Why did it happen?' — root cause, drill-down
- Predictive: 'What will happen?' — ML models, forecasting
- Prescriptive: 'What should we do?' — optimization, recommendations
- Data Analyst focuses on past/present; Data Scientist on predictions; Data Engineer on pipelines

Practice Questions

1. Define Data Analytics and explain its importance in modern business decision-making.
2. Differentiate between Descriptive, Diagnostic, Predictive, and Prescriptive analytics with one real-world example each.
3. Compare the roles of a Data Analyst, Data Scientist, and Data Engineer in terms of responsibilities and required skills.
4. Explain the Six-Step Data Analysis Process with a practical example.
5. List any five tools used in Data Analytics and state one key use of each.
6. What is the difference between Business Analytics and Data Analytics?
7. Describe how data analytics is applied in any two industries (e.g., Healthcare, Finance).

nasrin-a-mahin

UNIT II — Excel for Data Analysis

Microsoft Excel is the most widely used data analysis tool. This unit covers Excel from basic navigation to advanced features like Power Query, PivotTables, and What-If Analysis.

Key Topics Covered

- Excel Interface — Ribbon, Workbook, Sheets, Cell References (relative, absolute, mixed)
- Essential Formulas — SUM, AVERAGE, COUNT, IF, IFERROR, VLOOKUP, HLOOKUP, INDEX/MATCH
- Conditional Aggregation — SUMIF, COUNTIF, AVERAGEIF, SUMIFS, COUNTIFS
- Text Functions — TRIM, CLEAN, LEFT, RIGHT, MID, CONCATENATE, TEXT, SUBSTITUTE
- Date/Time Functions — TODAY, NOW, DATE, DATEDIF, YEAR, MONTH, DAY, WEEKDAY
- Logical Functions — IF, AND, OR, NOT, IFS, SWITCH
- Statistical Functions — STDEV, VAR, CORREL, PERCENTILE, RANK, Data Analysis ToolPak
- PivotTables & PivotCharts — summarise, group, filter, slice large datasets
- Conditional Formatting — highlight rules, data bars, color scales, icon sets
- Data Validation — drop-down lists, number constraints, custom rules
- Power Query — import, transform, merge, append data from multiple sources
- What-If Analysis — Goal Seek, Scenario Manager, Data Tables
- Named Ranges & Formula Auditing — Trace Precedents, Error Checking

Practice Questions

8. Explain absolute vs relative vs mixed cell referencing with examples.
9. What is VLOOKUP? Write its syntax and explain each argument. When would you use INDEX/MATCH instead?
10. Differentiate between SUMIF and SUMIFS with an example.
11. What are PivotTables? How do you create one, and what are slicers used for?
12. Explain Conditional Formatting — list 3 types of rules and their use cases.
13. What is Power Query? List 5 common transformations it can perform.
14. Explain Goal Seek and Scenario Manager with a practical business example.
15. Write the formula to find the average sales for 'North' region from a dataset with Region and Sales columns.

nasrin-armahin

UNIT III — SQL for Data Analysis

SQL (Structured Query Language) is the standard language for querying relational databases. This unit covers everything from basic SELECT queries to advanced window functions and query optimization.

Key Topics Covered

- SQL Categories — DDL (CREATE, ALTER, DROP), DML (SELECT, INSERT, UPDATE, DELETE), DCL, TCL
- SQL Data Types — INT, VARCHAR, DECIMAL, DATE, BOOLEAN, TEXT
- Core SELECT — WHERE, ORDER BY, GROUP BY, HAVING, DISTINCT, LIMIT
- SQL Operators — BETWEEN, IN, LIKE, IS NULL, AND, OR, NOT
- Aggregate Functions — COUNT, SUM, AVG, MIN, MAX
- SQL Joins — INNER, LEFT, RIGHT, FULL OUTER, CROSS, SELF JOIN
- Subqueries — correlated and non-correlated, scalar, table subqueries
- CASE Statements — conditional logic in SELECT
- Window Functions — ROW_NUMBER, RANK, DENSE_RANK, LAG, LEAD, NTILE, running totals
- CTEs (Common Table Expressions) — WITH clause for readable, reusable queries
- Views, Stored Procedures, Indexes — performance and code reuse
- Query Optimization — use indexes, avoid SELECT *, prefer JOINs over subqueries

Practice Questions

16. Write a SQL query to find the top 5 customers by total purchase amount.
17. Explain the difference between WHERE and HAVING clauses.
18. What are SQL Joins? Explain INNER JOIN, LEFT JOIN, and FULL OUTER JOIN with examples.
19. Write a query using a CTE to find the highest-spending customer per product category.
20. What are Window Functions? Write a query showing monthly revenue with a running total using SUM OVER.
21. Explain RANK() vs DENSE_RANK() vs ROW_NUMBER() with examples.
22. Write a CASE statement to categorize customers as High/Medium/Low value based on their total spend.
23. What is a View? Create a view called active_customers_summary with customer name, total orders, and total spend.
24. Write a query using COALESCE to replace NULL phone numbers with 'Not Provided'.
25. List 5 SQL query optimization best practices and explain why each matters.

nasrin-a-mahin

UNIT IV — Statistics for Data Analysis

Statistics is the mathematical backbone of data analytics. This unit covers descriptive statistics, probability, distributions, hypothesis testing, correlation, and regression — all essential for rigorous analysis.

Key Topics Covered

- Descriptive Statistics — Mean, Median, Mode, Range, Variance, Standard Deviation
- Data Distributions — Normal, Skewed, Binomial, Poisson, Uniform
- Probability Fundamentals — events, rules, conditional probability, Bayes' theorem
- Sampling Techniques — random, stratified, systematic, cluster sampling
- Confidence Intervals — constructing and interpreting CI for means and proportions
- Hypothesis Testing — Null/Alternate hypothesis, p-value, alpha, Type I & II errors
- Common Tests — t-test (one-sample, two-sample, paired), Chi-Square, ANOVA, z-test
- Correlation — Pearson's r, Spearman's rank, correlation vs causation
- Linear Regression — simple and multiple regression, R^2, residuals, assumptions
- Outlier Detection — Z-score method, IQR method, box plots

Practice Questions

26. Calculate mean, median, mode, variance, and standard deviation for: [12, 15, 18, 18, 20, 22, 25, 30, 35, 100]
27. A sample has mean = 50 and SD = 10. What percentage of data falls between 30 and 70 if normally distributed? (Empirical Rule)
28. Explain Type I and Type II errors in hypothesis testing with examples.
29. What is a p-value? If $p = 0.03$ and $\alpha = 0.05$, what is your conclusion?
30. A company tests a new website design. Old site: 100 visitors, 15% conversion. New site: 100 visitors, 22% conversion. State the null and alternate hypotheses.
31. Calculate a 95% confidence interval for mean income: sample mean = ₹55,000, SD = ₹12,000, n = 50.
32. Two variables have $r = 0.85$. Does this prove causation? Explain the difference between correlation and causation.
33. Explain the difference between one-tailed and two-tailed hypothesis tests.
34. What is ANOVA? When would you use it instead of a t-test?

nasim-a-mahin

UNIT V — Data Analysis with Python

Python is the dominant language for data analysis. This unit covers the full Python data stack: NumPy for numerical computing, Pandas for data manipulation, Matplotlib/Seaborn for visualization, and SciPy/Statsmodels for statistics.

Key Topics Covered

- Python Basics — variables, data types, loops, functions, list comprehensions
- NumPy — arrays, vectorization, broadcasting, mathematical operations
- Pandas — Series, DataFrame, read_csv, indexing (loc/iloc), filtering
- Data Cleaning — handle missing values (dropna, fillna), duplicates, data types
- Data Transformation — groupby, merge, join, pivot_table, apply, lambda
- Exploratory Data Analysis (EDA) — describe(), info(), value_counts(), correlation matrix
- Matplotlib — line, bar, scatter, histogram, pie charts
- Seaborn — heatmap, boxplot, pairplot, violin plot, regression plot
- Statistical Analysis — scipy.stats for t-tests, chi-square, ANOVA
- Python Best Practices — readable code, vectorization, avoid loops, use method chaining

Practice Questions

35. What is the difference between loc[] and iloc[] in Pandas?
36. Write Python code to load a CSV, check for missing values, and fill numeric columns with their median.
37. Explain groupby() in Pandas with an example — find total sales by product category.
38. What is the difference between merge() and join() in Pandas?
39. Write code to create a correlation heatmap using Seaborn for a given DataFrame.
40. Explain the difference between apply() and map() in Pandas.
41. How do you detect and handle outliers in Python using the IQR method?
42. Write code for a complete EDA: load data, check shape/info/describe, find nulls, plot distributions.

nasrin-a-mahin

UNIT VI — Data Visualization

Data visualization transforms numbers into visual stories. This unit covers the principles of effective visualization, chart selection, storytelling with data, and dashboard design using tools like Matplotlib, Seaborn, and Plotly.

Key Topics Covered

- Why Visualize — human brain processes visuals faster; reveal patterns hidden in tables
- Chart Types — bar, line, scatter, pie, histogram, box plot, heatmap, treemap, funnel
- Choosing the Right Chart — comparisons, trends, distributions, relationships, compositions
- Design Principles — color theory, data-ink ratio, decluttering, preattentive attributes
- Storytelling with Data — Context, Challenge, Resolution; the 'So What?' test
- Interactive Visualization — Plotly, Bokeh, Dash for web-based interactive charts
- Dashboard Design Best Practices — layout, hierarchy, whitespace, KPI cards
- Common Visualization Mistakes — misleading axes, 3D charts, too many colors, chart junk

Practice Questions

43. When would you use a bar chart vs a line chart vs a scatter plot? Give an example for each.
44. What is the 'data-ink ratio' principle by Edward Tufte? How does it improve charts?
45. Explain the concept of 'Storytelling with Data' — what are the 3 key components of a data story?
46. List 5 common data visualization mistakes and how to avoid them.
47. What chart would you use to show: (a) sales trend over 12 months, (b) market share by product, (c) relationship between price and demand?
48. Explain the purpose of a box plot and what information it conveys about a dataset.
49. What are preattentive attributes in visualization? List 4 examples.

nasrin-a-mahin

UNIT VII — R Programming for Data Analysis

R is a powerful statistical computing language widely used in academia and research. This unit covers R basics, data manipulation with dplyr/tidyr, visualization with ggplot2, statistical modeling, and R Markdown for reproducible reporting.

Key Topics Covered

- R Basics — vectors, data frames, lists, factors, matrices, functions
- Data Import — read.csv(), read_excel(), connecting to databases
- dplyr — filter(), select(), mutate(), summarise(), group_by(), arrange()
- tidyr — pivot_longer(), pivot_wider(), separate(), unite() for tidy data
- ggplot2 — Grammar of Graphics, geoms (geom_bar, geom_line, geom_point), aesthetics, themes
- Statistical Functions — t.test(), chisq.test(), aov(), cor(), lm()
- Linear Regression — lm(), summary(), R^2, residual plots, assumptions
- R Markdown — reproducible reports combining code, output, and narrative
- The Pipe Operator (%>%) — readable data transformation chains

Practice Questions

50. Create a vector of 20 random numbers. Calculate mean, median, SD, and identify values > 1 SD from mean.
51. Using the mtcars dataset, filter cars with mpg > 20, select mpg/wt/hp, and arrange by descending mpg (use dplyr).
52. Create a ggplot2 scatter plot of mtcars: wt vs mpg, colored by cyl, with a linear regression line.
53. Perform a t-test comparing mpg between automatic (am=0) and manual (am=1) cars. Interpret the p-value.
54. Build a linear regression model: mpg ~ wt + hp. Report R^2 and interpret coefficients.
55. What is the pipe operator (%>%) in R? Rewrite this without pipes: arrange(select(filter(mtcars, mpg>20), mpg, wt), mpg)
56. Explain tidy data principles. Use pivot_longer() to convert wide to long format.

Yasin-a-mahin

UNIT VIII — Power BI for Business Intelligence

Power BI is Microsoft's leading BI platform for creating interactive dashboards and reports. This unit covers data connection, Power Query transformation, data modeling, DAX measures, and report sharing via Power BI Service.

Key Topics Covered

- Power BI Desktop vs Power BI Service vs Power BI Mobile
- Data Sources — Excel, SQL Server, SharePoint, web, REST APIs
- Power Query Editor — remove columns, change types, filter, merge queries, unpivot
- Data Model — star schema, relationships (one-to-many), cardinality, cross-filter direction
- DAX (Data Analysis Expressions) — calculated columns vs measures
- Key DAX Functions — SUM, CALCULATE, FILTER, ALL, RELATED, DIVIDE, IF, SWITCH, DATEADD
- Visualizations — bar, line, map, matrix, KPI, card, slicer, donut, scatter
- Report Design — themes, bookmarks, drill-through, tooltips, mobile layout
- Power BI Service — publish, share, dashboards, scheduled refresh, Row-Level Security

Practice Questions

57. What is the difference between a calculated column and a measure in Power BI/DAX?
58. Write a DAX measure for Profit Margin using DIVIDE to avoid division by zero.
59. Explain the difference between CALCULATE and FILTER in DAX with an example.
60. What is a Star Schema? Why is it preferred in Power BI data models?
61. How do you set up Row-Level Security (RLS) in Power BI?
62. Connect to the Sample Superstore dataset. Describe the steps to create a Sales by Region map visual.
63. What is the difference between DirectQuery, Import, and Live Connection in Power BI?
64. List 5 Power Query transformations and when you would use each.
65. How do you create a drill-through page in Power BI? What is its purpose?

nasrin-a-mahin

UNIT IX — Tableau for Data Visualization

Tableau is the industry-leading drag-and-drop visualization platform. This unit covers connecting to data, building visualizations, calculated fields, parameters, dashboards, and publishing to Tableau Public/Server.

Key Topics Covered

- Tableau Products — Tableau Desktop, Public, Online, Server, Prep
- Data Connection — live vs extract, file sources (Excel, CSV), databases
- Tableau Interface — Dimensions vs Measures, Shelves (Rows, Columns, Pages, Filters, Marks)
- Chart Types — bar, line, scatter, map (choropleth, point), treemap, bullet, Gantt
- Calculated Fields — basic formulas, LOD expressions (FIXED, INCLUDE, EXCLUDE)
- LOD Expressions — compute at different granularity than the view
- Parameters — dynamic user-controlled inputs for Top N, date ranges, thresholds
- Filters — dimension, measure, context, data source, extract filters
- Dashboard Design — layout containers, device designer, actions (filter, highlight, URL)
- Tableau Prep — visual data preparation and cleaning workflow
- Publishing — Tableau Public for portfolio; Tableau Server/Online for enterprise sharing

Practice Questions

66. Connect to Sample - Superstore. Create a bar chart of Sales by Category, sorted descending.
67. What is the difference between a Dimension and a Measure in Tableau?
68. Explain LOD (Level of Detail) expressions. Write a FIXED LOD to calculate total sales per customer regardless of filters.
69. What are Parameters in Tableau? Create a 'Top N' parameter to dynamically show top 5/10/20 products.
70. Explain the difference between a Tableau Dashboard and a Story.
71. What is a Context Filter? When should you use it?
72. Create a calculated field for Profit Margin: [Profit] / [Sales]. Format as percentage.
73. List the steps to build a dashboard with 4 sheets and add a filter action from the map to other charts.
74. What is Tableau Prep? How does it differ from Tableau Desktop?
75. Build a map showing Sales by State. What type of geographic field is needed? What is a choropleth map?

nasrin-a-mahin

UNIT X — Machine Learning for Data Analytics

Machine Learning enables computers to learn from data automatically. This unit covers the ML workflow, data preprocessing, supervised and unsupervised learning algorithms, model evaluation, and best practices using scikit-learn in Python.

Key Topics Covered

- ML vs Traditional Programming — rules vs learning from data
- Types of ML — Supervised (labeled data), Unsupervised (no labels), Reinforcement, Semi-supervised
- ML Workflow — Define Problem → Collect Data → EDA → Preprocess → Split → Train → Evaluate → Tune → Deploy → Monitor
- Data Preprocessing — handling missing values, encoding (Label/One-Hot), feature scaling (Normalization/Standardization)
- Supervised — Regression (Linear, Ridge, Lasso, Random Forest) & Classification (Logistic, Decision Tree, SVM, KNN, Naive Bayes, Random Forest)
- Unsupervised — Clustering (K-Means, DBSCAN, Hierarchical) & Dimensionality Reduction (PCA, t-SNE)
- Model Evaluation — Regression: MSE, RMSE, MAE, R^2; Classification: Accuracy, Precision, Recall, F1, ROC-AUC
- Train-Test Split & Cross-Validation — K-Fold CV for robust performance estimation
- Overfitting vs Underfitting — Bias-Variance Tradeoff, regularization (L1/L2), dropout
- Feature Engineering — polynomial features, binning, log transforms, interaction terms
- scikit-learn Pipeline — StandardScaler → Model → evaluate

Practice Questions

76. What is the difference between Supervised and Unsupervised Machine Learning? Give two examples of each.
77. Explain the Machine Learning Workflow in 8-10 steps with a real-world example.
78. What is the Bias-Variance Tradeoff? Explain Overfitting and Underfitting with solutions for each.
79. Explain Label Encoding vs One-Hot Encoding. When would you use each?
80. What is the difference between Normalization and Standardization? When does each method apply?
81. Explain K-Fold Cross-Validation. Why is it better than a simple train-test split?
82. What is the confusion matrix? Define Precision, Recall, and F1-Score with formulas.
83. Write a complete scikit-learn Python script to train a Random Forest classifier on the Iris dataset and evaluate it.
84. What is K-Means clustering? Explain how the algorithm works step-by-step.
85. Explain L1 (Lasso) and L2 (Ridge) regularization. How do they prevent overfitting?
86. What is PCA (Principal Component Analysis)? What problem does it solve?

nasim-a-mahin

87. List 5 common ML pitfalls and explain how to avoid each one.

Quick Reference — Key Formulas & Concepts

Statistics Formulas

Mean = $\Sigma x / n$ | Variance = $\Sigma(x-\bar{x})^2 / (n-1)$ | SD = $\sqrt{\text{Variance}}$

Z-Score = $(x - \mu) / \sigma$ | Pearson r = $\Sigma(x-\bar{x})(y-\bar{y}) / [(n-1) \cdot Sx \cdot Sy]$

95% CI = $\bar{x} \pm 1.96 \times (\sigma/\sqrt{n})$ | Margin of Error = $1.96 \times SE$

ML Evaluation Formulas

Accuracy = (TP+TN) / Total | Precision = TP / (TP+FP) | Recall = TP / (TP+FN)

F1-Score = 2 × (Precision × Recall) / (Precision + Recall)

MSE = $\Sigma(y-\hat{y})^2 / n$ | RMSE = $\sqrt{\text{MSE}}$ | MAE = $\Sigma|y-\hat{y}| / n$

SQL Quick Reference

SELECT cols FROM table WHERE condition GROUP BY col HAVING condition ORDER BY col LIMIT n;

INNER JOIN: matching rows only | LEFT JOIN: all left + matching right | FULL JOIN: all rows

Window: ROW_NUMBER() OVER (PARTITION BY col ORDER BY col)

DAX Quick Reference

Total Sales = SUM(Sales[Amount])

Profit Margin = DIVIDE([Total Profit], [Total Sales], 0)

Sales LY = CALCULATE([Total Sales], DATEADD(Dates[Date], -1, YEAR))

— End of Summary Document —

nasrin-a-mahin