

Data Engineering

→ **Data Engineering**: Preparing, processing, and moving data so that it is ready for analysis.

→ Focus: **Data quality, reliability, scalability, and automation.**

→ Enables **Data Scientists and Analysts** to work with structured, clean data.

ETL vs. ELT

- **ETL (Extract → Transform → Load)**
 - Traditional process.
 - Data is transformed **before** loading into the target system.
 - Good for structured warehouses (e.g., financial reporting).
- **ELT (Extract → Load → Transform)**
 - Modern approach (cloud warehouses like Snowflake, BigQuery).
 - Data is loaded **first** (raw) → transformed inside the warehouse.
 - Good for big data & analytics.

Data Pipeline

• A **data pipeline** is a set of processes that move data from **source systems** → **processing steps** → **storage** → **consumption**.

- Automates repetitive tasks of data movement & preparation.
- Ensures data is **fresh, reliable, and ready to use**.

Data Pipeline Components

◆ 1. Ingestion

- Collect raw data from **APIs, files, databases, streams**.
- Tools: requests, kafka-python, boto3.

◆ 2. Processing

- Clean, validate, and transform data.
- Tools: numpy, pandas, pyspark, dask.

◆ 3. Storage

- Store processed data for analysis.
- Options:
 - **Databases** → PostgreSQL, MySQL

- **Data Lakes** → S3, HDFS
- **Warehouses** → Snowflake, BigQuery, Redshift

◆ 4. Output / Consumption

- Data made available to **dashboards, ML models, or reports**.
- Tools: Tableau, Power BI, Jupyter, APIs.

Why Data Pipeline Automation is Important

- **Save Time** → No need for manual data pulls or processing.
- **Reduce Mistakes** → Less human error, consistent results.
- **Faster** → Fresh data delivered in near real-time or scheduled runs.
- **Scalable** → Same system can handle growing data volume.
- **Reliable** → Automated retries & monitoring improve stability.






Workflow Automation

- **Workflow Automation is the process of automating repetitive tasks and data pipelines to ensure tasks run reliably, on schedule, and without manual intervention.**
- In data engineering, it's used to automate ETL/ELT processes, data cleaning, model training, reporting, and more.

◆ Why Workflow Automation is Important

1. **Saves Time** ⌚
 - Tasks run automatically, reducing manual effort.
2. **Reduces Errors** ✓
 - Consistent execution avoids human mistakes.
3. **Ensures Timely Execution** ⌚
 - Data pipelines and reports are always updated on schedule.
4. **Scalable** 📈
 - Can handle complex, multi-step pipelines across multiple systems.
5. **Enhances Reliability**
 - Automated retries and monitoring improve system robustness.

Steps in Workflow Automation

1. Define Task 
 - Identify the tasks that need automation (e.g., data extraction, transformation, loading).
2. Design Workflow 
 - Arrange tasks in a logical sequence, with dependencies clearly defined.
3. Schedule 
 - Decide when and how often tasks should run (daily, hourly, on-event).
4. Execute & Monitor 
 - Automate task execution.
 - Track status, logs, and errors.
5. Handle Failures 
 - Set retries, alerts, or fallback mechanisms for failed tasks.

SQL Basics – CRUD Operations

CRUD = **Create, Read, Update, Delete** → Core operations in databases.

1. CREATE

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    City VARCHAR(50)  
);  
  
INSERT INTO Customers (CustomerID, Name, City)  
VALUES (1, 'Alice', 'New York');
```

2. READ (SELECT)

- Used to **retrieve data**.

-- All rows

```
SELECT * FROM Customers;
```

-- Specific columns

```
SELECT Name, City FROM Customers;
```

-- With condition

```
SELECT * FROM Customers WHERE City = 'New York';
```

3. UPDATE

- Used to **modify existing records**.

```
UPDATE Customers
```

```
SET City = 'Los Angeles'
```

```
WHERE CustomerID = 1;
```

4. DELETE

- Used to **remove records**.

-- Delete specific row

```
DELETE FROM Customers WHERE CustomerID = 1;
```

-- Delete all rows (⚠ careful!)

```
DELETE FROM Customers;
```

JOIN

The JOIN clause in SQL is used to combine rows from two or more tables based on a related column between them. Different types of joins exist:

- **INNER JOIN:**

Returns only the rows where there is a match in both tables.

- **LEFT (OUTER) JOIN:**

Returns all rows from the left table and the matching rows from the right table. If there is no match in the right table, NULL values are returned for the right table's columns.

- **RIGHT (OUTER) JOIN:**

Returns all rows from the right table and the matching rows from the left table. If there is no match in the left table, NULL values are returned for the left table's columns.

- **FULL (OUTER) JOIN:**

Returns all rows when there is a match in either the left or right table. If there is no match, NULL values are returned for the non-matching side.

- **CROSS JOIN:**

Returns the Cartesian product of the two tables, meaning every row from the first table is combined with every row from the second table.

GROUP BY

The GROUP BY clause is used to group rows that have the same values in specified columns into a summary row. It is typically used with aggregate functions (like SUM(), AVG(), COUNT(), MAX(), MIN()) to perform calculations on each group.

```
SELECT department, SUM(salary)
FROM employees
GROUP BY department;
```

Reading Data from Various Sources

Common Data Sources:

➔ CSV / TSV / Text Files

```
import pandas as pd
df = pd.read_csv('file.csv')
```

➔ Excel Files

```
df = pd.read_excel('file.xlsx', sheet_name='Sheet1')
```

➔ Databases (SQL)

```
import sqlalchemy
engine = sqlalchemy.create_engine('mysql+pymysql://user:pass@host/db')
df = pd.read_sql('SELECT * FROM table_name', engine)
```

➔ JSON Files / APIs

```
import json
```

```
df = pd.read_json('file.json')
```

Data Ingestion Techniques: Batch vs Streaming

What is Data Ingestion?

- The process of **collecting and importing data** from various sources (APIs, DBs, files, IoT devices, logs, etc.) into a system for processing & storage.
- Two main modes: **Batch** and **Streaming**.

2. Batch Data Ingestion

- Data is collected over a period of time and **ingested in bulk**.
- Suitable for **large volumes** of static or periodic data.

Characteristics:

- Scheduled (hourly, daily, weekly).
- High throughput.
- Latency = minutes to hours.
- Simpler and cheaper to implement.

Example:

- Uploading daily sales transactions from CSV → Database → Data Warehouse.

3. Batch Processing Architecture Components

1. Data Sources

- Databases, logs, CSV/JSON files, APIs.

2. Data Ingestion Layer

- Collect data at scheduled intervals.
- Tools: Apache Sqoop (DB → HDFS), Apache NiFi, Python scripts.

3. Data Storage Layer

- Raw data stored in a **Data Lake** (HDFS, S3, Azure Data Lake).
- Processed data stored in a **Data Warehouse** (Snowflake, BigQuery, Redshift).

4. Processing Layer

- Uses batch frameworks to process/transform data.

- Tools:
 - **Hadoop MapReduce** (older, disk-based).
 - **Apache Spark** (modern, in-memory).
 - Python (pandas, dask).

5. Orchestration Layer

- Automates ETL jobs & workflows.
- Tools: Apache Airflow, Oozie, Luigi.

6. Output Layer

- Processed data made available to:
 - BI tools (Power BI, Tableau).
 - Data Science models.
 - Reports & dashboards.

Streaming Processing

- **Continuous ingestion and processing** of data as it is generated.
- Focus on **real-time insights** (milliseconds to seconds).
- Ideal for **time-sensitive use cases** like fraud detection, IoT, live dashboards.

Streaming Architecture Components

1. Data Sources

- IoT devices, app logs, transactions, social media feeds, sensors.

2. Ingestion Layer (Message Broker)

- Collects real-time events and stores temporarily.
- **Apache Kafka** (most popular), RabbitMQ, AWS Kinesis, Google Pub/Sub.

3. Stream Processing Layer

- Performs filtering, aggregations, joins, and analytics in real time.
- Tools:
 - **Apache Spark Streaming**
 - **Apache Flink**
 - **Kafka Streams**
 - **Storm**

4. Storage Layer

- Hot Storage (fast, real-time): NoSQL DBs like Cassandra, MongoDB, Elasticsearch.

- Cold Storage (historical): HDFS, S3, Data Warehouse.

5. Serving/Output Layer

- Dashboards (Grafana, Kibana, Power BI).
- Alerts (fraud detection, anomaly alerts).
- Real-time APIs (recommendation systems).

Kafka

- **Distributed Event Streaming Platform.**
- Designed to handle **real-time, high-throughput, fault-tolerant** data streams.
- Used for building **data pipelines, event-driven applications, and streaming analytics**

Kafka Core Concepts

1. Producer

- Application that **sends messages (events)** into Kafka.
- Example: E-commerce app sends “new order placed”.

2. Topic

- Named **channel** where events are stored.
- Example: orders, payments, logs.

3. Partition

- A topic is split into **partitions** for scalability.
- Each message in a partition has a unique **offset** (ID).

4. Broker

- Kafka server that stores partitions.
- Kafka cluster = multiple brokers.

5. Consumer

- Application that **reads messages** from a topic.
- Example: Fraud detection system reads transactions topic.

6. Consumer Group

- Set of consumers working together to process a topic's partitions.

7. Offset

- Position of a message in a partition.
- Consumers use offsets to know "where they left off".

8. ZooKeeper / KRaft

- Used for cluster coordination & leader election.
- In newer Kafka versions → **KRaft** replaces ZooKeeper.

Producers → [Topics → Partitions] → Brokers → Consumers

ETL (Extract → Transform → Load)

- Data is extracted from source → transformed (cleaning, aggregation, formatting) → loaded into target system (Data Warehouse).
- Transformations happen before loading.

Best for:

- Traditional Data Warehouses (limited compute).
- Small/medium datasets.
- Complex transformations outside the warehouse.

ELT (Extract → Load → Transform)

- Data is extracted from source → loaded into target system first → then transformed inside the warehouse.
- Relies on the compute power of modern cloud warehouses (Snowflake, BigQuery, Redshift).

Best for:

- Big Data pipelines.
- Cloud-native warehouses.
- Faster ingestion + scalable in-warehouse transformations.

D.A.G

- DAGs ensure clear, ordered steps from start to finish.

- Directed: Means the steps have a clear flow, from start to finish.
- Acyclic: Means no loops in task execution
- Graph: It's a visual map of all your tasks and how they connect.
- Python Code!: And the coolest part? You write these recipes in Python!"
- Key takeaway: DAGs are your foolproof, step-by-step workflow blueprints

Operators

- Operators are specialized tools for tasks.
- They are pre-defined templates for jobs.
- Examples include Python and Bash Operators.
- Hundreds more exist for various services.

Task

- Tasks are instances of an Operator.
- They represent a specific unit of work.
- Tasks execute within your defined DAGs.
- Each task performs a concrete operation.
- They are configured with specific parameters.

The Scheduler

- Scheduler is the heart of Airflow.
- It continuously monitors all defined workflows.
- It triggers tasks based on dependencies and schedules.
- The scheduler acts as an alarm clock and project manager

Workers/Executors

- Workers are managed by Executors in Airflow.
- Executors assign specific tasks to the Workers.
- Airflow can scale Workers for simultaneous task execution.
- Workers perform the heavy lifting of data jobs.

Key Advantages

- Automation on Steroids: "No more manual clicks or forgotten scripts! Airflow automates your factory from end to end."
- Code-First Brilliance: "Your recipes (DAGs) are just Python code. This means they are:
 - Versionable: Like saving different versions of your recipe.
 - Testable: You can test your recipes before the big cook-off.
 - Maintainable: Easy to update and fix."
- The Crystal Ball (Powerful UI): "Airflow comes with a beautiful web interface. It's your factory's control panel! See which lines are running, which jobs failed, view logs, and troubleshoot like a pro."
- Reliability & Resilience: "What if a blender breaks? Airflow can automatically retry a task, or alert you immediately. Your factory keeps running smoothly, even with hiccups."
- Scale Up, Not Out (Scalability): "Need to make more meals? Just add more workers (executors/workers)! Airflow is built to handle massive, growing workloads."
- Community & Extensibility: "Airflow has a huge, supportive community, and tons of pre-built 'appliances' (operators) for almost every data tool out there. It's like an ever-expanding kitchen gadget store!"

FEATURE ENGINEERING

Feature Engineering is the process of **transforming raw data into meaningful features** that improve the performance of machine learning models.

- It involves **selecting, modifying, or creating new features** from existing data.

A **feature** is an **input variable** that helps a model **learn patterns** in data.

Examples:

- Raw: Date of Birth → Feature: Age
- Raw: Transaction Time → Feature: Hour of Day

Improves model accuracy

Reduces overfitting

simplifies the model

enhances interpretability

Workflow in Feature Engineering

1. Understand data and types of variables.
2. Handle missing values and outliers.
3. Transform features (scaling, encoding, binning).
4. Create new features (derived from existing ones).

5. Select important features (remove irrelevant/noisy features)

Tableau Dashboard

- A **visual interface** that combines multiple visualizations (charts, tables, KPIs) into a single interactive view.
- Helps **analyze data, identify patterns, and share insights**.
- Key goal: Make **data storytelling easy and intuitive**.

Key Principles of Dashboard Design

1. Know your audience

- Business stakeholders → Focus on KPIs and trends.
- Data scientists → More detailed charts and filters.

2. Clarity over complexity

- Avoid cluttered dashboards.
- Only include **relevant metrics**.

3. Use proper chart types

- Bar/Column → Compare categories
- Line → Trends over time
- Scatter → Relationship between variables
- Heatmap → Correlations or intensity
- KPI Cards → Single number metrics

4. Effective use of color

- Use colors **consistently**.
- Highlight important insights (red for negative, green for positive).

5. Interactivity

- Filters → Select category, region, or time period
- Drill-down → Explore details
- Tooltips → Show extra info without clutter

6. Layout & hierarchy

- Place most important insights **at the top-left**.
- Group related charts together.

◆ 3. Steps to Design a Tableau Dashboard

1. **Define objective**
 - What problem are you solving? What questions should it answer?
2. **Select relevant data**
 - Prepare your dataset (clean, transformed).
 - Identify key metrics and dimensions.
3. **Build individual visualizations**
 - Create charts for trends, comparisons, distributions, and KPIs.
4. **Combine visualizations**
 - Drag sheets into a dashboard.
 - Adjust size, layout, and alignment.
5. **Add interactivity**
 - Filters, actions, parameter controls.
6. **Test & optimize**
 - Ensure dashboard is readable, intuitive, and responsive.
7. **Publish & share**
 - Tableau Server, Tableau Online, or Tableau Public.

Data Warehouse

Data warehousing is the process of **collecting, integrating, storing, and managing data from multiple sources** in a central repository to enable efficient querying, analysis, and reporting.

Goal:

Support decision-making by providing **clean, consistent, and timely access to historical data**, ensuring fast retrieval even for massive datasets.

Need for Data Warehousing

1. **Handling Large Data Volumes:** Can store and analyze TBs of historical data.
2. **Centralized Data Storage:** Combines data from multiple sources for a unified view.
3. **Trend Analysis:** Enables analyzing historical trends and predicting future outcomes.

4. **Business Intelligence Support:** Works with BI tools for quick insights.

Key Components

1. **Data Sources:** Systems providing raw internal or external data.
2. **ETL Process:** Extract, Transform, Load – prepares data for storage.
3. **Data Warehouse Database:** Central repository of cleaned and transformed data.
4. **Metadata:** Describes structure, source, and usage of data.
5. **Data Marts:** Focused subsets of the data warehouse for specific business areas.
6. **OLAP Tools:** Enable multidimensional data analysis.
7. **End-User Access Tools:** Reporting and BI tools like dashboards.

Characteristics of Data Warehousing

- Centralized Data Storage
- Query & Analysis
- Data Transformation
- Data Mining
- Data Security

OLTP (Online Transaction Processing)

- **Definition:** OLTP systems manage **day-to-day transactional data** and support routine operations in real time.
- **Purpose:** To **process large numbers of small, short-duration transactions quickly and accurately.**
- **Data Type:** Operational, current data (e.g., customer orders, account balances).
- **Focus:** **Speed, reliability, and consistency** of transactions.
- **Typical Operations:**
 - Insert, update, delete, and retrieve small amounts of data
 - Frequent, repetitive queries
- **Database Type:** Usually **relational databases** like MySQL, PostgreSQL, Oracle.
- **Key Features:**
 - **ACID compliance** (Atomicity, Consistency, Isolation, Durability) ensures reliability

- **High concurrency** to handle multiple users simultaneously
- Real-time transaction processing
- **Examples:**
 - Banking: Fund transfers, ATM withdrawals
 - E-commerce: Placing orders, updating inventory
 - Reservation systems: Airline or hotel bookings
- **Goal:** Ensure that operational data is **processed efficiently, accurately, and securely** for business continuity.

OLAP (Online Analytical Processing)

- A technology used to organize large business databases.
- Supports Business Intelligence (BI).
- Data stored in cubes for easy analysis.
- Cube administrator designs cubes for reports (PivotTable, PivotChart, etc.).

Characteristics of OLAP

1. Fast:
 - Most queries answered within 5 seconds.
 - Simple analysis in <1 sec, complex <15 sec.
2. Analysis
 - Adapts to business logic & statistical needs.
 - Allows Ad-hoc calculations & reporting.
3. Share:
 - Provides security & multi-user access.
 - Supports concurrent updates securely.
4. Multidimensional:
 - Offers a multidimensional view of data.
 - Supports hierarchies for analysis.
5. Information:

- Can store all required application data.
- Handles data sparsity efficiently.

Data Cube

A **Data Cube** is a **multidimensional representation of data**, used in **OLAP (Online Analytical Processing)** to analyze large volumes of data efficiently.

Think of it as a **3D spreadsheet**, where each axis represents a **dimension** and the cell contains **measures** (numerical data).

Operations

- Slice** Select a **single dimension value** to view a 2D slice.
- Dice** Select **specific values from multiple dimensions**.
- Roll-up** Aggregate data **up the hierarchy** (e.g., Month → Year).
- Drill-down** Go **down the hierarchy** to more detailed data.
- Pivot** Rotate cube to view data from a different perspective.

Types

Multidimensional Data Cube

Relational Data Cube

MOLAP (Multidimensional OLAP)

- Stores data in **multidimensional cubes**.
- **Fast query performance** due to pre-aggregation.
- Best for **small to medium datasets**.
- Examples: Sales analysis, financial reporting.

ROLAP (Relational OLAP)

- Stores data in **relational databases** (tables).
- Queries are generated **on-the-fly using SQL**.
- Handles **large datasets** efficiently.
- Slightly slower than MOLAP due to dynamic computation.

HOLAP (Hybrid OLAP)

- Combines **MOLAP** and **ROLAP** approaches.
- Stores **aggregated data in cubes** (MOLAP) and **detailed data in relational tables** (ROLAP).
- Balances **speed and storage efficiency**.

Dimensional Data Modeling (DDM)

◆ Definition

- Dimensional Data Modeling is a **design technique for data warehouses** that structures data to make it **easy to retrieve, analyze, and report**.
- Organizes data into **facts** (measurable data) and **dimensions** (contextual data).

Components

a) Fact Table

- Stores **measurable, quantitative data** (metrics).
- Usually **numerical values** that can be aggregated.
- Examples: Sales Amount, Quantity Sold, Revenue.
- Characteristics:
 - Primary key = combination of foreign keys from dimension tables.
 - Contains **additive, semi-additive, or non-additive measures**.

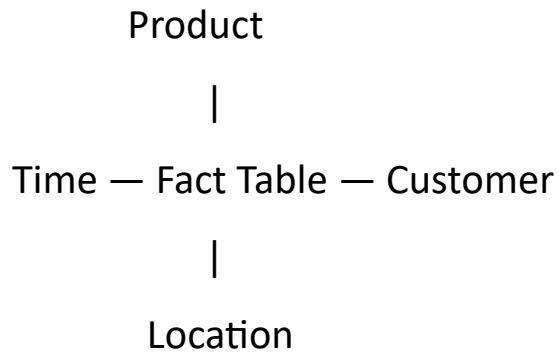
b) Dimension Table

- Stores **descriptive, contextual information** to provide meaning to facts.
- Examples: Product, Customer, Time, Location.
- Characteristics:
 - Contains **attributes** for filtering, grouping, and labeling.
 - Usually denormalized for fast querying.

Schema Types

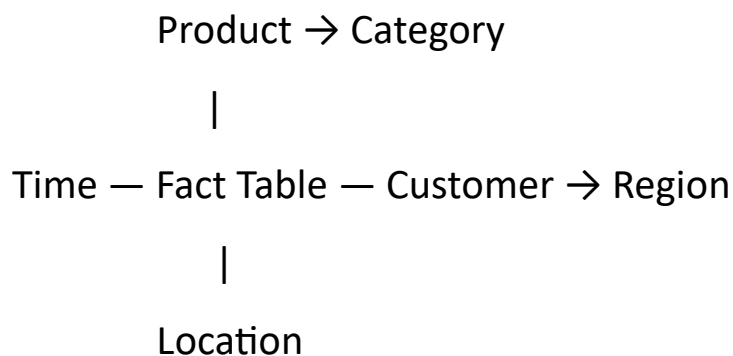
a) Star Schema

- Central **fact table** connected to multiple **dimension tables**.
- Dimension tables are **denormalized** (flat).
- **Advantages:** Simple, fast queries, easy to understand.



b) Snowflake Schema

- Dimensions are **normalized** into multiple related tables.
- Reduces data redundancy.
- **Advantages:** Saves storage, maintains hierarchy.
- **Disadvantages:** More complex queries, slower performance.



Galaxy Schema / Fact Constellation

- Multiple **fact tables share dimension tables**.
- Useful for complex data warehouses.
- Example: Sales fact table and Inventory fact table sharing Product and Time dimensions.