Data Engineering with Python: Data Pipelines & Workflow Automation

What is Data Engineering?

Data Engineering is the practice of designing, building, and maintaining systems that move, clean, and prepare data for analysis or machine learning.

- Focus on data pipelines: collecting → transforming → storing → serving.
- Ensures data is reliable, scalable, and accessible.

Data Pipelines

A data pipeline is an automated process that moves data from sources to destinations through a series of transformations.

Key Stages in a Data Pipeline

1. **Data Ingestion**
    - Collect raw data from APIs, databases, files, IoT devices, logs, streaming sources.
    - Tools: requests, pandas, Kafka, Flume.
2. **Data Storage**
    - Store ingested data in data lakes, warehouses, or relational databases.
    - Tools: PostgreSQL, MySQL, MongoDB, Hadoop HDFS, AWS S3.
3. **Data Transformation**
    - Clean, validate, enrich, aggregate, or normalize data.
    - Tools: Pandas, PySpark, Dask, SQL queries.
4. **Data Orchestration**

    - Automate pipeline execution, scheduling, dependency management.
    - Tools: Apache Airflow, Luigi, Prefect.

5. **Data Serving**
    - Deliver processed data to BI tools, ML models, or dashboards.
    - Tools: Tableau, Power BI, Flask APIs.

Workflow Automation

Workflow automation in data engineering means creating repeatable, reliable processes for moving and transforming data without manual effort.

Benefits

- Reduces manual errors.
- Saves time and improves efficiency.
- Ensures consistency across jobs.
- Enables real-time or scheduled data processing.

Automation Methods

- **Scheduling**: Automate tasks at intervals (cron jobs, Airflow DAGs).
- **Triggers**: Automate on events (e.g., new file in S3).
- **Orchestration**: Manage dependencies across tasks.
- **Monitoring & Logging**: Track job status, retry failures automatically.

Python in Data Engineering

Python is widely used due to its rich ecosystem of libraries:

Libraries for Data Ingestion

- **pandas** – Read CSV, Excel, SQL, JSON.
- **requests** – Fetch data from APIs.
- **pyodbc**, **sqlalchemy** – Database connectivity.

- **Kafka-python** – Stream processing.

Libraries for Data Transformation

- **pandas** – Cleaning, filtering, aggregations.
- **PySpark** – Distributed processing for big data.
- **dask** – Parallel data processing.
- **numpy** – Numerical transformations.

Workflow Orchestration Tools (Python-based)

- **Apache Airflow** – DAGs for scheduling and dependency management.

- **Luigi** – Build and track batch pipelines.
- **Prefect** – Modern orchestration, supports cloud workflows.

Automation & Scheduling

- **schedule** – Lightweight job scheduling in Python.
- **apscheduler** – Advanced scheduling for periodic jobs.
- **Cron jobs** (OS-level).

Deployment & Monitoring

- **Logging** (logging library).
- **Alerts** (Slack API, Email, PagerDuty).
- **Dashboards** – Grafana, Kibana.

Best Practices in Data Engineering Pipelines

1. **Modular Design** – Separate extraction, transformation, and loading.
2. **Error Handling** – Implement logging and retry mechanisms.
3. **Scalability** – Use Spark/Dask for large datasets.

4. **Version Control** – Git for pipeline scripts.

5. **Testing** – Unit tests for data transformations.
6. **Monitoring** – Track pipeline status (success/failure).
7. **Documentation** – Clear docs for maintainability.

Applications in Data Science & Analytics

- **Data Warehousing** – ETL pipelines into Snowflake, Redshift, BigQuery.
- **Real-Time Analytics** – Kafka + Spark Streaming pipelines.
- **ML Pipelines** – Automate feature engineering, model training, and deployment.
- **IoT Data Pipelines** – Process sensor data in real-time.
- **Business Dashboards** – Automate feeds to Tableau/Power BI.

SQL Basics and CRUD Operations

Introduction to SQL

- SQL (Structured Query Language) is the standard language for interacting with relational databases.
- It is used to store, manipulate, and retrieve data stored in tables.
- SQL commands are categorized into different types:
    - DDL (Data Definition Language): Create, alter, drop structures (tables, databases).
    - DML (Data Manipulation Language): Insert, update, delete records.
    - DQL (Data Query Language): Select data from tables.
    - DCL (Data Control Language): Grant or revoke access.

    - TCL (Transaction Control Language): Commit, rollback transactions.

What is CRUD?

- CRUD stands for:
    - C → Create (Insert data): Used to add new records into a table
    - R → Read (Retrieve data)/(Select Data): Used to fetch data from a database.
    - U → Update (Modify data): Used to change existing records
    - D → Delete (Remove data): Used to delete records from a table.

Clauses with SELECT:

- WHERE → filtering rows
- ORDER BY → sorting results
- GROUP BY → grouping rows
- HAVING → filtering groups

SQL: JOIN, GROUP BY & WINDOW FUNCTIONS

JOIN

Definition:

- JOIN is used to combine data from two or more tables based on a related column (usually a primary key–foreign key relationship).

- Helps in querying across multiple tables.

Types of Joins:

1. INNER JOIN → Returns only matching rows from both tables.
2. LEFT JOIN (LEFT OUTER JOIN) → Returns all rows from the left table + matching rows from the right.
3. RIGHT JOIN (RIGHT OUTER JOIN) → Returns all rows from the right table + matching rows from the left.
4. FULL JOIN (FULL OUTER JOIN) → Returns all rows when there is a match in one of the tables.
5. CROSS JOIN → Cartesian product (all combinations).

GROUP BY

Definition:

- GROUP BY is used to group rows with the same values in specified columns and apply aggregate functions (SUM, COUNT, AVG, MAX, MIN).
- Often used with HAVING to filter aggregated results.

WINDOW FUNCTIONS

Definition:

- Unlike GROUP BY, Window Functions perform calculations across a set of rows related to the current row without collapsing them into one row.
- Use OVER() clause with optional PARTITION BY and ORDER BY.

Common Window Functions

1. ROW_NUMBER() → Assigns unique sequential number per row
2. RANK() → Assigns rank, with gaps if ties exist.

3. DENSE_RANK() → Similar to RANK but without gaps.
4. NTILE(n) → Divides rows into n buckets.
5. Aggregate Functions with OVER()
    - SUM(), AVG(), COUNT(), MAX(), MIN() can be applied as window functions.

Data Handling and PySpark Basics

Reading Data from Various Sources:

Data in real-world projects comes from many sources. Python (Pandas, NumPy) and PySpark provide built-in connectors to handle them.

Common Data Sources:

- CSV files → Most common flat file format.

```
df = pd.read_csv("data.csv")
```

- Excel files → Used in business data.

```
df = pd.read_excel("data.xlsx", sheet_name="Sheet1")
```

- Databases (SQL) → Use connectors like sqlite3, SQLAlchemy, or PySpark JDBC.

```
import sqlite3

conn = sqlite3.connect("mydb.db")

df = pd.read_sql("SELECT * FROM Students", conn)
```

- JSON files / APIs → Structured semi-structured data

```
df = pd.read_json("data.json")
```

- Big Data Sources (in PySpark) → HDFS, Hive, Parquet, ORC, Avro.

```
df = spark.read.csv("hdfs://path/data.csv", header=True, inferSchema=True)
```

Array:

- Provided by **NumPy** (numpy.array).
- Stores **homogeneous elements** (all of the same data type).
- Supports vectorized operations (fast computation).

Use case → Scientific computing, numerical analysis.

DataFrame

- **Pandas DataFrame**:
    - 2D labeled data structure (rows + columns).
    - Like a spreadsheet or SQL table.

- **PySpark DataFrame**:
    - Distributed version of Pandas DataFrame.
    - Handles **large-scale data** across clusters.

Use case → Tabular structured data, data analysis, machine learning pipelines.

Vector

- In machine learning, a vector = numeric representation of data (features).
- PySpark uses MLlib Vectors (DenseVector, SparseVector) for feature representation.

Use case → ML models (regression, classification, clustering).

Series

- **Pandas Series**:
    - 1D labeled array (like a single column).

Use case → Time series, single-variable operations.

Introduction to PySpark

What is PySpark?

- PySpark is the Python API for Apache Spark (big data framework).
- Used for distributed computing on large datasets across clusters.
- Supports SQL, streaming, ML, and graph processing.

Features:

- **Resilient Distributed Dataset (RDD):** Low-level distributed collection.
- **DataFrame API:** High-level structured data API (like SQL + Pandas).
- **MLlib:** Machine learning library.
- **Spark SQL:** Query structured data using SQL.
- **GraphX / GraphFrames:** Graph computation.

Why PySpark?

- Handles big data (beyond single machine).
- Faster (in-memory computation).
- Integrates with Hadoop ecosystem.
- Supports machine learning & streaming.

Data Ingestion Techniques: Batch vs. Streaming Data (Kafka)

What is Data Ingestion?

- Data ingestion = Process of collecting and moving data from various sources into a storage/processing system (like a data lake, data warehouse, or real-time analytics platform).

- Goal: Make data available for analysis, reporting, or machine learning.

Two main approaches: Batch ingestion and Streaming ingestion.

Batch Data Ingestion

Definition:

- Data is collected and transferred in large chunks (batches) at scheduled intervals (hourly, daily, weekly).
- System processes all the data at once.

Workflow:

1. Collect data (logs, transactions, files).
2. Store temporarily (CSV, JSON, staging DB).
3. Load into data warehouse or Hadoop/Spark for processing.

Example Tools:

- Apache Sqoop (RDBMS → Hadoop)
- Apache Nifi
- ETL pipelines (Informatica, Talend, AWS Glue)

Advantages:

- Efficient for large volumes of data.
- Suitable for historical analysis, reports, dashboards.
- Simple to implement.

Limitations:

- Not real-time → delays between data generation and availability.
- Not suitable for applications needing instant decisions.

Streaming Data Ingestion

Definition:

- Data is collected, processed, and delivered continuously in real time.
- Each record/event is ingested as soon as it is generated.

Workflow:

1. Data generated from sensors, transactions, logs.
2. Pushed into a streaming platform (e.g., Kafka).
3. Consumers process events immediately.

Example Tools:

- Apache Kafka (most popular for real-time ingestion).
- Apache Flume, Apache Pulsar, AWS Kinesis.
- Processing frameworks: Apache Spark Streaming, Flink, Storm.

Advantages:

- Low latency (near real-time).
- Suitable for IoT, fraud detection, stock trading, monitoring systems.
- Scalable (Kafka handles millions of events/sec).

Limitations:

- More complex architecture.
- Requires strong fault tolerance & scaling.
- Costlier compared to batch.

Apache Kafka

Apache Kafka is a publish-subscribe messaging system that lets applications send and receive messages between processes, servers, or services. Messages are organized into topics (categories), which can carry any type of information—from blog events to simple text—that can trigger further processing.

What is Kafka?

Kafka is an open-source messaging system that was created by LinkedIn and later donated to the Apache Software Foundation. It's built to handle large amounts of data in real time, making it perfect for creating systems that respond to events as they happen.

Core Components of Apache Kafka

1. Kafka Broker

A Kafka broker is a server that runs Kafka and stores data. Typically, a Kafka cluster consists of multiple brokers that work together to provide scalability, fault tolerance, and high availability. Each broker is responsible for storing and serving data related to topics.

2. Producers

A producer is an application or service that sends messages to a Kafka topic. These processes **push data** into the Kafka system. Producers decide which topic the message should go to, and Kafka efficiently handles it based on the partitioning strategy.

3. Kafka Topic

A topic in Kafka is a category or feed name to which messages are published. Kafka messages are always associated with topics, and when you want to send a message, you send it to a specific topic. Topics are divided into partitions, which allow Kafka to scale horizontally and handle large volumes of data.

4. Consumers and Consumer Groups

A **Consumer** reads messages from Kafka topics. In consumer groups, multiple consumers can share a topic, but each message is processed by only one consumer, enabling load balancing and flexible offset reading.

5. Zookeeper

Kafka uses Apache ZooKeeper to manage metadata, control access to Kafka resources, and handle leader election and broker coordination**.** ZooKeeper ensures high availability by making sure the Kafka cluster remains functional even if a broker fails.

Important Concepts of Apache Kafka

- **Topic partition**: Kafka topics are divided into a number of partitions, which allows you to split data across multiple brokers.
- **Consumer Group**: A consumer group includes the set of consumer processes that are subscribing to a specific topic.
- **Node**: A node is a single computer in the Apache Kafka cluster.
- **Replicas:** A replica of a partition is a "backup" of a partition. Replicas never read or write data. They are used to prevent data loss.
- **Producer**: Application that sends the messages.
- **Consumer**: Application that receives the messages.

How Apache Kafka Works:

1. **Producers send data** – Applications create messages (logs, events, transactions) and send them to Kafka, which splits data into partitions for easier handling.
2. **Kafka stores data** – Messages are saved in **topics** with replication across servers for durability; data isn’t deleted immediately.
3. **Consumers read data** – Applications subscribe to topics, often in **consumer groups** to avoid duplicate processing, and can start reading from any offset.

4. **Kafka balances load** – ZooKeeper manages servers and ensures failover if one goes down.
5. **Data is processed** – Consumers can store, analyze, or trigger events; Kafka integrates with Spark, Flink, Hadoop, etc.

ETL vs ELT: Building Robust Data Pipelines using Apache Airflow

ELT Process

Extraction, Load and Transform (ELT) is the technique of extracting raw data from the source, storing it in the data warehouse of the target server and preparing it for end-stream users.

ELT consists of three different operations performed on the data:

1. **Extract**: Extracting data is the process of identifying data from one or more sources. The sources may include databases, files, ERP, CRM, or any other useful source of data.
2. **Load**: Loading is the process of storing the extracted raw data in a data warehouse or data lake.
3. **Transform**: Data transformation is the process in which the raw data from the source is transformed into the target format required for analysis

ETL Process

ETL is the traditional technique of extracting raw data, transforming it as required for the users and storing it in data warehouses. ELT was later developed, with ETL as its base. The three operations in ETL and ELT are the same, except that their order of processing is slightly different. This change in sequence was made to overcome some drawbacks.

1. **Extract**: It is the process of extracting raw data from all available data sources such as databases, files, ERP, CRM or any other.

2. **Transform:** The extracted data is immediately transformed as required by the user.
3. **Load**: The transformed data is then loaded into the data warehouse from where the users can access it.

ETL vs ELT Comparison

Feature	ETL	ELT
Transformation Timing	Before loading	After loading
Tools	Talend, Informatica, Python ETL scripts	BigQuery, Snowflake, Spark
Data Volume	Moderate	Large-scale
Processing Location	On ETL server	In data warehouse/lake
Use Case	Traditional warehouses	Cloud & Big Data systems

Building Robust Data Pipelines with Apache Airflow

What is Apache Airflow?

- Open-source workflow orchestration tool.
- Manages data pipelines as Directed Acyclic Graphs (DAGs).

- Handles scheduling, dependency management, retries, logging, and monitoring.

How Airflow Implements ETL/ELT

1. **Extract Task** → Pull data from source APIs, databases, or files.
2. **Transform Task** (ETL) → Clean, aggregate, and format data before loading. **OR**
   Transform Task (ELT) → Load raw data first, then use SQL or Spark inside warehouse.
3. **Load Task** → Load transformed or raw data into the target storage.
4. **Monitoring & Scheduling** → Airflow automatically retries failed tasks and logs results.

Benefits of Airflow Pipelines

- **Automation:** Runs workflows without manual intervention.
- **Scalability:** Works for small ETL jobs or large-scale ELT pipelines.
- **Reliability:** Handles retries, failure alerts, and logging.
- **Extensibility:** Integrates with cloud services, databases, APIs, Spark, Kafka, etc.

Choosing Between ETL and ELT with Airflow:

The choice depends on factors like:

- Data Volume and Velocity: ELT is often preferred for high-volume, high-velocity data in modern cloud environments.
- Data Warehouse Capabilities: ELT leverages powerful data warehouses, while ETL might be necessary for less capable systems.

- Transformation Complexity and Security: ETL provides more control over data quality and security during transformation, while ELT offers flexibility for evolving analytical needs.
- Real-time vs. Batch Processing: ELT can better support near real-time analytics due to faster loading, while ETL is well-suited for batch processing.

Data Warehousing Concepts: OLAP, OLTP & Dimensional Modeling

What is Data Warehousing?

A data warehouse is a central repository of integrated data collected from different sources. It is used for reporting, analysis, and decision-making.

- Stores historical data.
- Optimized for querying and analysis, not transaction processing.
- Supports Business Intelligence (BI) activities like dashboards, reporting, and data mining.

Need for Data Warehousing:

- **Handling Large Data Volumes:** Traditional databases store limited data (MBs to GBs), while data warehouses are built to handle huge datasets (up to TBs), making it easier to store and analyze long-term historical data.
- **Centralized Data Storage:** A data warehouse combines data from multiple sources, giving a single, unified view for better decision-making.
- **Trend Analysis**: By storing historical data, a data warehouse allows businesses to analyze trends over time, enabling them to make strategic decisions based on past performance and predict future outcomes.
- **Business Intelligence Support:** Data warehouses work with BI tools to give quick access to insights.

Characteristics of Data Warehousing:

- **Centralized Data Storage**: Combines data from various sources into one place for a complete view.
- **Query & Analysis**: Supports fast and flexible data analysis for better

decision-making.

- **Data Transformation**: Cleans and formats data for consistency and quality.
- **Data Mining**: Finds hidden patterns to discover insights and predict trends.
- **Data Security**: Protects data with encryption, access control, and backups.

OLTP (Online Transaction Processing)

Definition:

OLTP systems are designed to manage day-to-day transactional operations (insert, update, delete).

Characteristics

- Purpose: Run business operations in real-time (e.g., banking, retail POS, airline reservations).
- Data: Current, highly detailed, operational data.
- Transactions: Short, frequent, and simple queries.
- Database Design: Highly normalized (3NF) to avoid redundancy.
- Response Time: Very fast (milliseconds).
- Users: Large number of concurrent users (e.g., cashiers, customers, employees).

Example:

- ATM withdrawals
- Online ticket booking
- E-commerce order placement

Types of OLTP:

- Batch Processing: Collects and processes data in bulk periodically.
- Real-time Processing: Updates data instantly.

- Distributed OLTP Systems: Data processing across multiple locations or servers.

Advantages of OLTP:

- User-friendly and accessible
- Fast read, write, and delete operations
- Immediate response to user actions
- Original source of data
- Help run fundamental business tasks
- Enhance customer base by simplifying processes

OLAP (Online Analytical Processing)

Definition:

OLAP systems are designed for data analysis and decision support. They work on the historical data stored in the data warehouse.

Characteristics:

- Purpose: Support business analysis and strategic decision-making.
- Data: Historical, aggregated, and summarized data.
- Transactions: Complex queries (multi-dimensional analysis).
- Database Design: De-normalized (star schema/snowflake schema) for faster querying.
- Response Time: Seconds to minutes (slower than OLTP, but optimized for large queries).
- Users: Business analysts, managers, executives.

OLAP Operations (Cube Analysis):

1. Roll-up – Summarize data (e.g., monthly → yearly sales).

2. Drill-down – Go into detailed data (e.g., yearly → monthly sales).

3. Slice – Extract a single dimension (e.g., sales in 2023 only).

4. Dice – Extract data on multiple dimensions (e.g., sales by region AND product).

5. Pivot (Rotate) – Reorient data view (e.g., swap rows and columns in analysis).

Example:

- Sales analysis by product, region, and time.
- Market basket analysis.
- Customer segmentation.

Advantages of OLAP:

- Faster query performance (optimized storage & indexing).
- Reduced storage (compression techniques).
- Automated computation of higher-level aggregates.
- Compact representation of multidimensional datasets.
- Efficient data extraction via pre-aggregated structures.

Disadvantages of OLAP:

- Long processing time during data load (for large datasets).
- Mitigated by incremental processing (only new data).
- Possible data redundancy in some OLAP methods.

Dimensional Data Modeling

Dimensional Modeling is a data design technique used in data warehouses to structure data for easy retrieval and analysis.
→It focuses on making the database understandable and optimized for OLAP (Online Analytical Processing) rather than for transactions (OLTP).

Instead of normalizing data (as in OLTP systems), it organizes data into:

- Facts → measurable business data (sales, revenue, quantity, etc.)
- Dimensions → descriptive attributes that give context to facts (time, product, customer, region, etc.)

Elements of Dimensional Data Model

Fact Table:

In a dimensional data model, the fact table is the central table that contains the measures or metrics of interest, surrounded by the dimension tables

Dimension Table:

A Dimension Table is a table that stores the descriptive attributes (contextual information) about the business entities used for analysis.

Attributes:

An attribute is a property characteristic or descriptive detail of a dimension that helps to provide context for analysis.

DATA DIMENSIONAL MODELS

1. Star schema

Structure: Central **Fact Table** surrounded by multiple **Dimension Tables**.Fact Table contains measurable data (metrics, key performance indicators).Dimension Tables contain descriptive attributes (like time, product, customer).

Advantages: Simple design, Fast query performance due to fewer joins.
Disadvantages: Data redundancy in dimension tables.

2. Snowflake Schema

Structure: Similar to Star Schema, but Dimension Tables are **normalized** (split

into multiple related tables).

Advantages:

- Reduces data redundancy.
- Saves storage space.

Disadvantages:

- More complex queries (more joins).
- Slightly slower query performance compared to Star Schema

3. Galaxy Schema (Fact Constellation)

Structure: Contains multiple fact tables that share dimension tables.

Advantages:

- Useful for complex systems.
- Supports multiple fact tables for various subject areas.

Disadvantages:

- Increased complexity.

Steps to Create Dimensional Data Modeling:

Step-1: Identifying the business objective: The first step is to identify the business objective. Sales, HR, Marketing, etc. are some examples of the need of the organization. Since it is the most important step of Data Modelling the selection of business objectives also depends on the quality of data available for that process.

Step-2: Identifying Granularity: Granularity is the lowest level of information stored in the table. The level of detail for business problems and its solution is described by Grain.

Step-3: Identifying Dimensions and their Attributes: Dimensions are objects

or things. Dimensions categorize and describe data warehouse facts and measures in a way that supports meaningful answers to business questions. A data warehouse organizes descriptive attributes as columns in dimension tables. For Example, the data dimension may contain data like a year, month, and weekday.

Step-4: Identifying the Fact: The measurable data is held by the fact table. Most of the fact table rows are numerical values like price or cost per unit, etc.

Step-5: Building of Schema: We implement the Dimension Model in this step. A schema is a database structure. There are two popular schemes: Star Schema and Snowflake Schema.

Feature Engineering and Visualization-Datatypes

1. What is Feature Engineering?

Feature Engineering is the process of creating, transforming, or selecting features from raw data to improve the performance of machine learning models.

Key Goals:

- Improve model accuracy & interpretability.
- Convert raw data into meaningful inputs.
- Handle issues like missing values, categorical encoding, scaling, etc.

Common Steps in Feature Engineering:

1. Handling Missing Values
    - Techniques: mean/median/mode imputation, forward-fill, backward-fill, or ML-based imputation.

2. Encoding Categorical Variables
    - One-Hot Encoding, Label Encoding, Ordinal Encoding, Target Encoding.

3. Feature Scaling
    - Normalization, Standardization (Z-score), Robust scaling.

4. Feature Transformation
    - Log transformation, square root, Box-Cox to handle skewed data.

5. Feature Extraction
    - PCA (Principal Component Analysis), Feature hashing, embeddings.

6. Feature Creation (Derived Features)
    - Combining existing variables (e.g., BMI = weight/height², Age from Date of Birth).

7. Handling Outliers
    - Capping, transformation, or removal.

8. Feature Selection
    - Techniques: correlation analysis, variance threshold, Lasso regularization, mutual information.

2. Data Types in Feature Engineering:

Understanding data types is crucial, since each type requires different preprocessing and visualization.

(a) Numerical (Quantitative) Data

- Represents numbers and quantities.

- Sub-types:
    - Continuous: Infinite values (height, weight, temperature).
    - Discrete: Countable values (number of children, defects, sales units).

Feature Engineering Techniques:

- Scaling (StandardScaler, MinMaxScaler).
- Log/Power transformation to fix skew.
- Outlier treatment (IQR method, Winsorization).

Visualization:

- Histogram (distribution).
- Boxplot (spread + outliers).
- Scatterplot (relationship with other variables).
- Line chart (trend over time).
-

(b) Categorical (Qualitative) Data:

- Represents labels or groups.
- Sub-types:
    - Nominal: No order (e.g., gender, city, color).
    - Ordinal: Ordered categories (e.g., low/medium/high, education levels).

Feature Engineering Techniques:

- Nominal → One-Hot Encoding.
- Ordinal → Label Encoding / Ordinal Encoding (preserve order).
- Grouping rare categories into “Other” class.

Visualization:

- Bar chart (frequency of categories).
  Pie chart (proportion of categories).
- Countplot (in Python: seaborn/matplotlib).

(c) Text Data

- Unstructured text like reviews, tweets, product descriptions.

Feature Engineering Techniques:

- Tokenization (splitting text into words).
- Stopword removal (e.g., "the", "is").
- Stemming / Lemmatization (reduce to root form).
- TF-IDF (Term Frequency – Inverse Document Frequency).
- Word embeddings (Word2Vec, GloVe, BERT).

Visualization:

- Word clouds (most frequent words).
- Bar charts of word frequency.
- N-gram distribution plots.

(d) Datetime/Time Series Data:

- Data related to time: timestamps, dates, intervals.

Feature Engineering Techniques:

- Extract parts of date: Year, Month, Day, Weekday, Hour.
- Create lag features (previous day's value).
- Rolling averages, moving windows.
- Time difference calculations (e.g., days between orders).
- Seasonality and trend decomposition.

Visualization:

- Line charts (time vs value).
- Seasonal plots.
- Autocorrelation plots (time series dependencies).
- Heatmaps (e.g., sales by weekday and hour).

(e) Boolean Data (True/False)

- Binary outcomes (yes/no, 0/1, pass/fail).

Feature Engineering Techniques:

- Convert to integers (0/1).
- Can be treated as categorical or numerical depending on model.

Visualization:

- Bar chart (count of True/False).
- Pie chart (percentage split).

(f) Image / Multimedia Data (Advanced)

- Used in deep learning applications.

Feature Engineering Techniques:

- Pixel normalization (0–1 scaling).
- Feature extraction using CNNs.
- Dimensionality reduction (PCA, autoencoders).

Visualization:

- Displaying raw images.
- Heatmaps for feature importance (Grad-CAM in CNNs).

3. Why Data Types Matter in Feature Engineering & Visualization?

- Correct preprocessing techniques depend on data type.
- Visualization varies (numerical → histogram, categorical → bar chart).
- Choosing the wrong method (e.g., one-hot encoding on continuous values) leads to poor model performance.

Tableau Dashboard Design for Data Science

What is a Tableau Dashboard?

A Tableau dashboard is a collection of visualizations, charts, and objects presented together on a single screen to provide insights from data.

- Combines multiple worksheets (charts, maps, KPIs).
- Helps monitor, analyze, and tell a data story.
- Interactive: users can filter, drill down, and highlight.

Role of Dashboards in Data Science:

In Data Science, dashboards bridge the gap between complex models and business users.

- Present model outputs (predictions, clusters, trends) in an easy-to-understand way.
- Enable exploratory data analysis (EDA) by visualizing raw features and engineered features.
- Support decision-making with KPI tracking, comparisons, forecasts.
- Allow real-time monitoring of processes, customer behavior, or ML system performance.

Components of a Tableau Dashboard:

1. Worksheets – Individual charts/visuals.
2. Filters – Allow users to narrow down data (by region, product, time).
3. Parameters – User-defined controls (what-if analysis, scenario testing).
4. Objects – Text, images, web content, buttons for navigation.
5. Layouts – Tiled or floating arrangements for design.

Dashboard Design Process in Data Science:

Step 1: Define Purpose & Audience

- Who will use the dashboard? (executives, analysts, operations team).
- What's the goal? (monitoring, exploration, storytelling, prediction results).

Step 2: Data Preparation

- Clean, transform, and engineer features.
- Ensure correct data types (numeric, categorical, datetime).
- Create calculated fields and KPIs (e.g., churn rate, sales growth, accuracy score).

Step 3: Choose the Right Visualizations

- Numerical → line charts, histograms, scatterplots.
- Categorical → bar charts, treemaps.
- Time-series → line chart, area chart.
- Spatial → maps.
- Advanced → bullet charts, KPI cards.

Step 4: Layout & Interactivity

- Arrange visuals logically (top → summary, bottom → details).
- Add filters, dropdowns, and highlight actions.
- Use drill-down to go from **overview → detail**.

Step 5: Testing & Feedback

- Ensure clarity (no clutter).
- Optimize for performance (avoid heavy queries).
- Validate with end-users before deployment.

Dashboard Design Principles (Best Practices)

(a) Clarity & Simplicity

- Avoid clutter – include only essential visuals.
- Use clear titles, labels, and legends.

- Apply consistent colors and formatting.

(b) Visual Hierarchy

- Place **important KPIs at the top/left** (eye's natural scanning order).
- Summaries first → details later.
- Use size, color, and position to emphasize key points.

(c) Effective Use of Colors

- Use color sparingly and meaningfully (e.g., red = risk, green = growth).
- Maintain consistency across all visuals.
- Avoid overuse of gradients that confuse readers.

(d) Interactivity & Storytelling

- Add filters, highlight actions, parameters for exploration.
- Use navigation buttons to create multi-page dashboard apps.
- Include annotations or captions to explain insights.

(e) Performance Optimization

- Extracts instead of live connections when possible.
- Avoid too many quick filters; use context filters.
- Minimize use of complex calculated fields in real-time.

Types of Tableau Dashboards in Data Science

1. **Exploratory Dashboards**
    - Used during EDA.
    - Show correlations, distributions, outliers.
    - Example: Customer demographics vs. purchase behavior.

2. **Explanatory Dashboards**
    - Present final results & insights.

- Example: Churn prediction results → “At-risk customers” segment.

3. **KPI Dashboards**
   - Monitor key performance indicators.
   - Example: Sales growth %, model accuracy, customer lifetime value.

4. **Real-Time Dashboards**
   - Connect to streaming data.
   - Example: Fraud detection system monitoring transactions.

5. **Story Dashboards**
   - Present a **narrative** with sequential views.
   - Example: "Market trends → Customer segments → Prediction outcomes".

Example Applications of Tableau Dashboards in Data Science

- **Customer Analytics**:
  Churn rate, segmentation, lifetime value visualization.
- **Predictive Modeling Output**:
  Show probability of churn, fraud, or conversion for each customer.
- **Operational Monitoring**:
  Real-time monitoring of supply chain or IoT sensor data.
- **Model Performance Dashboards**:
  Accuracy, precision, recall, ROC curves, confusion matrix visualization.

Advantages of Tableau Dashboards in Data Science

- **Interactive** – Non-technical users can explore data without coding.
- **Fast & Intuitive** – Drag-and-drop interface.
- **Scalable** – Connects to multiple data sources (databases, APIs, ML outputs).
- **Storytelling Power** – Communicates data insights effectively.
- **Collaboration** – Can be published to Tableau Server/Tableau Online for team use.