Contents

Introduction to Data Science

What is Data Science?

Data Science is an interdisciplinary field that combines statistical methods, machine learning, and data analysis techniques to extract insights from structured and unstructured data. It involves data cleaning, exploration, modeling, and interpretation to solve real-world problems.

Who is a Data Scientist?

A Data Scientist is a professional who collects, processes, and analyzes data to extract valuable insights. They use programming (Python, R, SQL), statistical methods, and machine learning to make data-driven decisions.

Data Science Skill Set

1. **Programming** – Python, R, SQL
2. **Mathematics & Statistics** – Probability, Linear Algebra
3. **Machine Learning** – Supervised & Unsupervised Learning
4. **Data Manipulation** – Pandas, NumPy, Excel
5. **Big Data Technologies** – Hadoop, Spark
6. **Data Visualization** – Matplotlib, Seaborn, Tableau

Data Science Job Roles

- **Data Analyst** – Analyzes structured data for business insights
- **Data Engineer** – Builds and maintains data pipelines
- **Machine Learning Engineer** – Develops ML models
- **Data Scientist** – Extracts insights and builds predictive models

Data Life Cycle

1. **Data Collection** – Gathering raw data
2. **Data Cleaning** – Handling missing values and outliers
3. **Data Exploration** – Identifying patterns and relationships
4. **Data Modeling** – Applying statistical or machine learning models
5. **Evaluation & Deployment** – Assessing model performance and integrating into production

Statistics & Probability

Categories of Data

1. Qualitative Data (Categorical Data)

Non-numeric data that represents categories or labels.

- **Nominal Data**: Categories with no inherent order (e.g., Gender: Male, Female; Colors: Red, Blue, Green).
- **Ordinal Data**: Categories with a meaningful order but no fixed interval (e.g., Education Level: High School < Bachelor's < Master's).

2. Quantitative Data (Numerical Data)

Numeric values that can be measured and used for calculations.

- **Discrete Data**: Countable numbers (e.g., Number of students in a class).
- **Continuous Data**: Measurable values with infinite possibilities within a range (e.g., Height, Weight, Temperature).

Basic Terminologies in Statistics

- **Population** – The entire dataset of interest
- **Sample** – A subset of the population used for analysis
- **Variable** – Any measurable characteristic in the dataset

---

📉 **Sampling Techniques**

- **Random Sampling** – Each member of the population has an equal chance of being selected
- **Systematic Sampling** – Selecting every nth individual from a list
- **Stratified Sampling** – Dividing population into subgroups and sampling from each

---

📈 **Types of Statistics**

1. Descriptive Statistics

Summarizes data using measures like:

- Mean (Average): Sum of values divided by the count.
- Median: The middle value in an ordered dataset.

- Mode: The most frequently occurring value.
- Standard Deviation: Measure of data dispersion.

2. Inferential Statistics

Uses sample data to make predictions about a population. Example methods:

- Confidence Intervals: Range within which a population parameter likely falls.
- Hypothesis Testing: Checking if a sample result applies to the population.

📏 Measures of Spread

- **Range** – Difference between max and min values
- **Interquartile Range (IQR)** – Spread of the middle 50% of the data
- **Variance** – Average squared deviation from the mean
- **Standard Deviation** – Square root of variance, measures data dispersion

---

🎲 Probability Concepts

Probability is the measure of the likelihood of an event occurring, ranging from 0 (impossible) to 1 (certain).

- **Probability Distribution** – A function that describes the likelihood of different outcomes
- **Probability Density Function (PDF)** – Describes the probability of continuous variables
- **Normal Distribution** – A symmetric bell-shaped distribution where most data points cluster around the mean
- **Central Limit Theorem** – The sampling distribution of the sample mean approaches normality as sample size increases

---

🔢 Types of Probability

- **Marginal Probability** – Probability of a single event occurring
- **Joint Probability** – Probability of two events occurring together
- **Conditional Probability** – Probability of one event occurring given another event has occurred

Bayes Theorem:

P(A|B)=P(B|A)P(A)P(B)P(A|B) = \frac{P(B|A) P(A)}{P(B)}P(A|B)=P(B)P(B|A)P(A)

Used to calculate conditional probabilities and is the basis for Naive Bayes classifiers.

📊 Inferential Statistics & Hypothesis Testing

Inferential Statistics

Inferential statistics use sample data to make generalizations about a population. Methods include confidence intervals, regression, and hypothesis testing.

Hypothesis Testing

A statistical method to determine if there is enough evidence to reject a null hypothesis.

1. **Null Hypothesis (H_0)** – Assumes no effect or relationship
2. **Alternative Hypothesis (H_1)** – Suggests a significant effect or relationship
3. **p-value** – Probability of observing data given H_0 is true. If **$p < 0.05$**, reject H_0.

Machine Learning

What is Machine Learning?

Machine Learning (ML) enables systems to learn from past experiences (data) and improve their performance over time. Instead of following a strict set of instructions, ML models find patterns in data to make decisions.

Types of Machine Learning:

1. **Supervised Learning** – The model is trained on labeled data (input-output pairs).
2. **Unsupervised Learning** – The model finds hidden patterns in unlabeled data.
3. **Reinforcement Learning** – The model learns by interacting with an environment and receiving rewards or penalties.

Machine Learning Process

1. Data Collection

- Gather relevant data from databases, APIs, or web scraping.
- Example: Collecting customer purchase history for recommendation systems.

2. Data Preprocessing

- Handle missing values, remove duplicates, normalize/standardize data.
- Feature selection and engineering to improve model accuracy.

3. Model Selection

- Choose an appropriate algorithm (e.g., Linear Regression for predicting sales, Decision Trees for classification).

4. Model Training

- Train the model using the dataset.
- Adjust parameters to minimize errors.

5. Model Evaluation

- Use metrics like accuracy, precision, recall, and RMSE to check model performance.

6. Deployment

- Deploy the model for real-world use (e.g., on websites, apps, or embedded systems).

📈 Supervised Learning Algorithms

1 Regression Algorithms 🔢

Used when the target variable is continuous (e.g., predicting house prices, stock prices).

🔹 1. Linear Regression

✅ **Concept:**

- Establishes a linear relationship between independent variables (**X**) and a dependent variable (**Y**).

🔹 2. Ridge & Lasso Regression (Regularized Linear Regression)

Used when there is **multicollinearity** or overfitting.

- **Ridge Regression:** Uses L2 regularization (penalizes large coefficients).
- **Lasso Regression:** Uses L1 regularization (shrinks some coefficients to zero, feature selection).

🔹 3. Polynomial Regression (For Non-Linear Relationships)

✅ **Concept:**

- Instead of fitting a straight line, it fits a polynomial curve.
- Uses **PolynomialFeatures** to create higher-degree terms.

🔹 4. Support Vector Regression (SVR)

✅ **Concept:**

- Uses Support Vector Machine principles to predict continuous values.
- Works well for **small datasets** and **high-dimensional data**.

🔹 5. Decision Tree Regression

✅ **Concept:**

- Splits data into regions based on feature values (like decision trees for classification).
- Can capture **non-linear relationships** well.

🔹 6. Random Forest Regression

✅ **Concept:**

- Uses **multiple Decision Trees** and averages their predictions.
- Handles **overfitting** better than a single Decision Tree.

---

2 Classification Algorithms 🎯

Used when the target variable is **categorical** (e.g., spam detection, fraud detection).

🔹 1. Logistic Regression

✅ **Concept:**

- Used for **binary classification** (0 or 1).
- Uses the **sigmoid function** to map predictions between 0 and 1.

🔹 2. K-Nearest Neighbors (KNN)

✅ **Concept:**

- **Non-parametric algorithm** that classifies based on the majority vote of its **K nearest neighbors**.
- Works well for small datasets.

🔹 3. Support Vector Machine (SVM)

✅ **Concept:**

- Finds the best **hyperplane** that separates classes.
- Uses **kernels** to handle **non-linear** data.

🔹 4. Decision Tree Classifier

✅ **Concept:**

- Works by splitting data using **if-else conditions** on feature values.
- Can **overfit**, so **pruning** techniques are needed.

🔹 5. Random Forest Classifier

✅ **Concept:**

- Uses **multiple decision trees** and takes the majority vote.
- Reduces **overfitting** compared to a single Decision Tree.

🔹 6. Naive Bayes Classifier

✅ **Concept:**

- **Probabilistic classifier** based on **Bayes' theorem**.
- Works well for **text classification** (spam detection).

🔹 7. Gradient Boosting & XGBoost

✅ **Concept:**

- **Boosting technique** that trains models sequentially to reduce errors.
- **XGBoost** is optimized for speed and performance.

📌 Summary

Algorithm	**Type**	**Best For**
Linear Regression	Regression	Simple linear data
Ridge/Lasso Regression	Regression	Regularization
Polynomial Regression	Regression	Non-linear relationships
Decision Tree	Regression & Classification	Interpretable models
Random Forest	Regression & Classification	High accuracy, low overfitting
SVM	Classification	High-dimensional data

Algorithm	Type	Best For
Naive Bayes	Classification	Text classification
XGBoost	Classification	Large datasets

📊 Unsupervised Learning

1 Clustering Algorithms 🏷️

Used when we want to group similar data points based on certain patterns.

🔹 1. K-Means Clustering

✅ **Concept:**

- Divides data into **K clusters** based on feature similarity.
- Uses **centroids** to assign points to the nearest cluster.
- Works best when clusters are **spherical** and **well-separated**.

⚠ **Challenges:**

- Sensitive to the **choice of K**.
- Affected by outliers.

🔹 2. Hierarchical Clustering

✅ **Concept:**

- Builds a hierarchy of clusters **(dendrogram)**.
- Can be **Agglomerative** (bottom-up merging) or **Divisive** (top-down splitting).

⚠ **Challenges:**

- Computationally expensive for large datasets.
- Needs **cutoff criteria** to determine clusters.

🔹 3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

✅ **Concept:**

- Groups points based on **density** (areas with high point concentration).

- Can find **arbitrarily shaped** clusters.
- Identifies **outliers** as noise.

⚠ **Challenges:**

- Requires careful tuning of **epsilon (ε)** and **minimum points (MinPts)**.

---

2 Dimensionality Reduction 🏗

Used when we have **high-dimensional data** and want to **reduce the number of features** while keeping important information.

🔹 **1. Principal Component Analysis (PCA)**

✅ **Concept:**

- Transforms data into **new axes (principal components)** that capture maximum variance.
- Reduces redundancy and noise.
- Used in **image compression, finance, and gene expression analysis**.

⚠ **Challenges:**

- Works best for **linearly correlated** data.
- Difficult to interpret transformed features.

🔹 **2. t-SNE (t-Distributed Stochastic Neighbor Embedding)**

✅ **Concept:**

- Preserves **local structure** while reducing dimensions.
- Great for **visualization** (e.g., plotting high-dimensional data in 2D).

⚠ **Challenges:**

- Computationally expensive.
- Sensitive to **hyperparameters**.

🔹 **3. UMAP (Uniform Manifold Approximation and Projection)**

✅ **Concept:**

- Faster and better at **preserving global structure** than t-SNE.
- Used in **bioinformatics, NLP, and visualization**.

⚠ **Challenges:**

- Needs **careful parameter tuning**.

---

3 Association Rule Learning 🔗

Finds **hidden relationships** in data (e.g., Market Basket Analysis).

🔹 1. Apriori Algorithm

✅ **Concept:**

- Identifies **frequent item sets** and creates **association rules** (e.g., "People who buy milk also buy bread").
- Used in **retail, e-commerce, and recommendation systems**.

⚠ **Challenges:**

- Can be **computationally expensive**.
- Needs a **minimum support threshold** to work effectively.

🔹 2. FP-Growth (Frequent Pattern Growth)

✅ **Concept:**

- Faster than **Apriori** as it doesn’t generate candidate sets.
- Builds a **tree structure** to store frequent patterns.

⚠ **Challenges:**

- Can be **memory-intensive** for large datasets.

📌 Summary

Algorithm	Type	Best For
K-Means	Clustering	Well-separated clusters
Hierarchical	Clustering	Nested clusters (hierarchies)

Algorithm	Type	Best For
DBSCAN	Clustering	Arbitrary-shaped clusters, noise detection
PCA	Dimensionality Reduction	Removing redundant features
t-SNE	Dimensionality Reduction	Data visualization
UMAP	Dimensionality Reduction	Faster than t-SNE, better global structure
Apriori	Association Rule Learning	Market Basket Analysis
FP-Growth	Association Rule Learning	Faster frequent itemset mining

---

🎯 Reinforcement Learning

1 Key Concepts in RL

🔹 1. Agent & Environment

- **Agent**: The learner or decision-maker (e.g., a robot, a self-driving car).
- **Environment**: The system the agent interacts with (e.g., a game, a stock market).

🔹 2. State, Action, Reward

- **State (S)**: The current situation (e.g., position of a robot).
- **Action (A)**: The choices available to the agent (e.g., move left, right).
- **Reward (R)**: A numerical value given to the agent after an action (e.g., +10 for winning, -5 for failing).

🔹 3. Policy (π)

- A **policy** is a strategy that the agent follows to choose actions.
- **Deterministic Policy**: The same action is taken for the same state.
- **Stochastic Policy**: The action is chosen based on a probability distribution.

🔹 4. Value Function (V) & Q-Value (Q)

- **Value Function (V)**: Measures how good a state is in terms of expected rewards.

- **Q-Value (Q)**: Measures how good an action is in a given state.

---

2 Types of Reinforcement Learning Algorithms

🔹 1. Model-Based vs. Model-Free RL

- **Model-Based RL**: The agent builds a model of the environment and plans actions accordingly.
- **Model-Free RL**: The agent learns directly by interacting with the environment without building a model.

🔹 2. Dynamic Programming (DP)

- Uses **Bellman Equations** to compute the best policy.
- Requires a **complete model** of the environment (i.e., transition probabilities).
- **Example Methods**:
  - **Policy Iteration**: Evaluates and improves the policy repeatedly.
  - **Value Iteration**: Computes the best possible value function to derive an optimal policy.

⚠ **Challenges**:

- Computationally expensive for large environments.

🔹 3. Monte Carlo Methods

- Learns **by sampling episodes** and updating values based on returns.
- Doesn’t require a complete model of the environment.

⚠ **Challenges**:

- Requires full episode completion before updating values.

🔹 4. Temporal Difference (TD) Learning

- Learns **from incomplete episodes** using the difference between predicted and actual rewards.
- **Example Methods**:

    - **TD(0)**: Updates values after each step.
    - **SARSA (State-Action-Reward-State-Action)**: Learns a **soft policy** (follows its own learned policy).
    - **Q-Learning**: Learns an **optimal policy** (chooses the best action regardless of the current policy).

⚠ **Challenges**:

- Can be unstable if learning rates are not tuned properly.

---

3 Advanced RL Techniques

🔹 **1. Deep Q-Networks (DQN)**

- Combines **Q-Learning** with **Deep Learning** to handle high-dimensional states.
- Used in **Atari game playing** and **robotics**.

🔹 **2. Policy Gradient Methods**

- Instead of learning **value functions**, learns **directly the best policy**.
- Example: **REINFORCE Algorithm**.

⚠ **Challenges**:

- Requires **large amounts of training data**.

🔹 **3. Actor-Critic Methods**

- Combines **Value-Based (Critic) and Policy-Based (Actor) Methods**.
- **Actor**: Chooses actions.
- **Critic**: Evaluates actions.

🚀 **Example Algorithms:**

- **A3C (Asynchronous Actor-Critic)**
- **PPO (Proximal Policy Optimization)**
- **DDPG (Deep Deterministic Policy Gradient)**

Deep Learning

Deep Learning (DL) is a subset of Machine Learning that uses artificial **neural networks** to learn from data. Unlike traditional algorithms, DL models can automatically extract **features** from raw data, making them powerful for tasks like **image recognition, natural language processing (NLP), and autonomous systems**.

---

1 Fundamentals of Deep Learning

◆ 1.1 Artificial Neural Networks (ANNs)

- Inspired by the human brain, consisting of **neurons** (nodes) connected by **weights**.
- **Input Layer** → Receives data
- **Hidden Layers** → Process information (extract features)
- **Output Layer** → Provides predictions

◆ 1.2 Activation Functions

These determine whether a neuron should be activated or not:

- **Sigmoid** → Outputs between (0,1), used in binary classification.
- **ReLU (Rectified Linear Unit)** → Commonly used in deep networks.
- **Leaky ReLU** → Fixes the dying ReLU problem.
- **Softmax** → Converts values into probabilities for multi-class classification.

◆ 1.3 Loss Functions

Measure how well the model's predictions match the actual data.

- **Mean Squared Error (MSE)** → Regression tasks.
- **Binary Cross-Entropy** → Binary classification.
- **Categorical Cross-Entropy** → Multi-class classification.

◆ 1.4 Optimization Algorithms

Help minimize the loss function:

- **Gradient Descent** → Updates weights in the direction of the steepest descent.
- **SGD (Stochastic Gradient Descent)** → Updates weights after each data sample.
- **Adam (Adaptive Moment Estimation)** → Popular optimizer for deep learning.

---

2 Deep Learning Architectures

🔹 2.1 Convolutional Neural Networks (CNNs) 🖼️

Best for **image processing** tasks.

- **Convolutional Layers** → Extract features from images.
- **Pooling Layers** → Reduce dimensionality, preventing overfitting.
- **Fully Connected Layers** → Make predictions based on extracted features.

📌 **Applications**: Object detection, facial recognition, medical imaging.

🔹 2.2 Recurrent Neural Networks (RNNs) ⏳

Best for **sequential data** (e.g., time series, speech, text).

- Uses previous information to make better predictions.
- Suffers from **vanishing gradient problem**, making it hard to learn long-term dependencies.

📌 **Applications**: Language modeling, speech recognition, stock price prediction.

🔹 2.3 Long Short-Term Memory (LSTM) & Gated Recurrent Unit (GRU)

- **LSTM** → Solves vanishing gradient issue, remembers long-term dependencies.
- **GRU** → A simplified version of LSTM with fewer parameters.

📌 **Applications**: Chatbots, machine translation, weather forecasting.

🔹 2.4 Transformers & Attention Mechanisms 🧠

- Unlike RNNs, Transformers process entire sequences **at once**, making them faster.
- **Self-Attention Mechanism** → Allows models to focus on important words in a sentence.

📌 **Applications**: NLP (e.g., GPT, BERT), text summarization, speech recognition.

🔹 2.5 Autoencoders & Variational Autoencoders (VAEs)

Used for **dimensionality reduction, anomaly detection, and generative models**.

- **Autoencoder** → Encodes data into a compressed form and reconstructs it.
- **VAE** → A more powerful version that learns probability distributions.

📌 **Applications**: Image denoising, feature extraction, synthetic data generation.

🔹 2.6 Generative Adversarial Networks (GANs) 🎨

Used to generate realistic images, videos, and text.

- **Generator** → Creates fake data.
- **Discriminator** → Detects if the data is real or fake.

📌 **Applications**: AI-generated art, deepfake videos, image super-resolution.

---

3 Model Training & Regularization

🔹 3.1 Overfitting & Underfitting

- **Overfitting** → Model performs well on training data but poorly on new data.
- **Underfitting** → Model is too simple to capture patterns in data.

🔹 3.2 Regularization Techniques

Prevent overfitting and improve generalization.

- **Dropout** → Randomly drops neurons during training.
- **L1 & L2 Regularization** → Adds penalty to large weights.
- **Batch Normalization** → Normalizes inputs to stabilize learning.

---

4 Deployment & Optimization

🔹 4.1 Model Compression

Making deep learning models lightweight for deployment.

- **Pruning** → Removes unnecessary neurons.
- **Quantization** → Reduces precision of weights to lower memory usage.

📌 **Used for**: Mobile AI, IoT, Edge computing.

🔹 4.2 Model Deployment

How to use trained models in real-world applications.

- **TensorFlow Serving** → Deploying models as web APIs.
- **TFLite & ONNX** → Running models on mobile devices and edge devices.
- **Streamlit & FastAPI** → Building AI-powered web applications.

📌 **Used in**: Chatbots, fraud detection, AI assistants.

Introduction to Big Data

🔹 Introduction to Big Data

Big Data refers to vast and complex datasets that traditional data processing tools cannot efficiently manage. It encompasses structured, semi-structured, and unstructured data generated at high velocity from various sources, such as social media, sensors, transactions, and logs.

🔹 What is Big Data?

Big Data is a field that deals with analyzing, processing, and extracting valuable insights from extremely large datasets. It helps organizations make data-driven decisions by leveraging advanced analytics and machine learning techniques.

Big Data is characterized by its ability to handle:

- **Large volumes of data** from diverse sources
- **High-velocity data streams** that require real-time processing
- **Complex data structures**, including text, images, videos, and logs

---

🔹 5 V's of Big Data

Big Data is often defined using the **5 V's** framework:

1 **Volume** – The sheer amount of data generated every second (e.g., terabytes, petabytes of data from social media, IoT devices, and transactions).

2 **Velocity** – The speed at which data is generated and processed (e.g., real-time streaming data from stock markets, social media feeds).

3 **Variety** – The diversity of data formats, including structured (databases), semi-structured (JSON, XML), and unstructured (videos, images, text).

4 **Veracity** – The accuracy, reliability, and trustworthiness of data. Poor data quality can lead to misleading insights.

5 **Value** – The usefulness of the data in decision-making. Extracting meaningful patterns from Big Data enables businesses to gain a competitive edge.

🔹 Big Data as an Opportunity

Big Data presents significant opportunities for businesses and organizations, enabling them to:

- **Enhance customer experience** – Companies like Amazon and Netflix use Big Data to provide personalized recommendations.
- **Optimize operations** – Businesses use data analytics to improve supply chain efficiency and reduce costs.
- **Improve healthcare** – Hospitals analyze patient data for better disease prediction and personalized treatment plans.
- **Predict market trends** – Financial institutions use Big Data to forecast stock market movements and mitigate risks.
- **Enable smart cities** – Governments use real-time data to manage traffic, energy consumption, and public safety efficiently.

🔹 IBM Big Data Analytics Use-Case

IBM is a leader in Big Data solutions, providing tools like **IBM Watson, IBM BigInsights, and IBM Cloud Pak for Data** to analyze massive datasets.

🔹 **Example Use Case:**
A retail company using **IBM Big Data Analytics** can:

- Analyze customer purchasing patterns
- Optimize inventory management
- Personalize marketing campaigns using AI-driven insights
- Improve fraud detection by identifying suspicious transactions

IBM's Big Data solutions help industries like **banking, healthcare, manufacturing, and telecommunications** enhance decision-making through real-time data analytics.

🔹 Big Data Analytics & Use-Case

Big Data Analytics refers to the techniques used to process and analyze large datasets to uncover patterns, trends, and insights. It involves:
✅ **Descriptive Analytics** – Summarizing past data
✅ **Diagnostic Analytics** – Understanding why events happened
✅ **Predictive Analytics** – Forecasting future trends using machine learning
✅ **Prescriptive Analytics** – Recommending actions based on predictive models

Example Use Case:
A **healthcare provider** uses Big Data Analytics to:

- Analyze patient medical records
- Predict disease outbreaks
- Enhance drug development using AI models

Big Data Analytics transforms raw data into actionable insights, enabling businesses to **make data-driven decisions and gain a competitive edge**.

Big Data Analytics

🔹 What is Big Data Analytics?

Big Data Analytics refers to the process of examining large and complex datasets to uncover patterns, trends, correlations, and insights. It uses various analytical techniques, including **machine learning, artificial intelligence (AI), and statistical methods**, to derive valuable information from data.

Big Data Analytics enables businesses to:

✅ Make data-driven decisions
✅ Identify customer behavior patterns
✅ Optimize operations and reduce costs
✅ Enhance fraud detection and security

Companies like **Google, Amazon, Netflix, and Facebook** leverage Big Data Analytics to personalize services, improve marketing strategies, and optimize supply chains.

🔹 Stages of Big Data Analytics

The process of Big Data Analytics follows a structured pipeline, consisting of:

1 **Data Collection** – Gathering raw data from multiple sources (social media, IoT devices, transactions, logs, etc.).
2 **Data Storage** – Storing massive datasets in **Hadoop Distributed File System (HDFS), cloud storage, or NoSQL databases**.
3 **Data Processing** – Cleaning and transforming raw data into a usable format using tools like **Apache Spark, Hadoop MapReduce, and Apache Flink**.
4 **Data Analysis** – Applying statistical methods, machine learning algorithms, and AI to extract insights.
5 **Data Visualization** – Presenting insights through charts, dashboards, and reports using **Tableau, Power BI, or Matplotlib**.
6 **Decision Making** – Businesses use analytics results to **predict trends, optimize processes, and drive innovation**.

🔹 Types of Big Data Analytics

Big Data Analytics is categorized into four types based on its objective:

✅ **Descriptive Analytics** – Summarizes historical data to understand what happened.

- Example: A retail company analyzing past sales to identify best-selling products.

✅ **Diagnostic Analytics** – Explains why an event occurred by identifying patterns and correlations.

- Example: A bank analyzing transaction data to detect fraud patterns.

✅ **Predictive Analytics** – Uses machine learning and statistical models to forecast future trends.

- Example: Netflix recommending movies based on user preferences.

✅ **Prescriptive Analytics** – Suggests the best course of action based on data insights.

- Example: An airline optimizing flight routes based on weather conditions and fuel efficiency.

🔹 Big Data Analytics in Different Domains

Big Data Analytics is used across various industries to drive innovation and improve efficiency:

💡 **Healthcare** – Predicting disease outbreaks, personalizing treatments, and analyzing patient records.
💡 **Finance** – Fraud detection, algorithmic trading, and risk assessment.
💡 **Retail & E-commerce** – Personalized recommendations, demand forecasting, and supply chain optimization.
💡 **Manufacturing** – Predictive maintenance, quality control, and production optimization.
💡 **Smart Cities** – Traffic management, energy efficiency, and crime prevention using IoT data.

🔹 Problems with Big Data: Restaurant Analogy

Understanding Big Data challenges can be simplified using a **restaurant analogy**:

🍽️ **Volume Challenge** (Too Much Data) – Imagine a restaurant with an overwhelming number of customers, making it difficult to serve food efficiently. Similarly, handling huge datasets requires scalable storage solutions like Hadoop.

🍽️ **Velocity Challenge** (Speed of Data) – If food orders come in too quickly, the kitchen struggles to prepare meals in time. Likewise, Big Data requires **real-time processing** using tools like Apache Kafka and Spark Streaming.

🍽️ **Variety Challenge** (Different Data Types) – Customers order different cuisines (structured and unstructured data), and the restaurant must accommodate all types efficiently. Similarly, Big Data involves managing diverse formats like text, images, videos, and logs.

🍽️ **Veracity Challenge** (Data Quality) – If the restaurant receives incorrect orders (inaccurate data), it affects customer satisfaction. In Big Data, ensuring **data accuracy and reliability** is critical to making informed decisions.

🍽️ **Value Challenge** (Extracting Insights) – If a restaurant doesn't analyze customer preferences, it may fail to serve the best dishes. Similarly, businesses must extract meaningful insights from Big Data to drive **growth and innovation**.

Hadoop & Distributed Computing

🔹 Apache Hadoop

Apache Hadoop is an open-source framework that enables distributed storage and processing of large datasets across a cluster of computers. It is designed to handle **Big Data** efficiently by breaking it into smaller chunks and distributing the workload among multiple machines.

💡 Why Hadoop?

- Traditional databases (RDBMS) struggle to process massive data volumes.
- Hadoop provides **scalability, fault tolerance, and cost-effectiveness** using commodity hardware.
- It processes both **structured and unstructured** data.

✅ Key Components of Hadoop:

1. **Hadoop Distributed File System (HDFS)** – A scalable storage system.
2. **MapReduce** – A processing framework for parallel computation.
3. **Yet Another Resource Negotiator (YARN)** – Manages cluster resources.
4. **Hadoop Common** – Utilities supporting Hadoop modules.

🔹 Hadoop Master-Slave Architecture

Hadoop follows a **Master-Slave** architecture, where:

✅ **Master Node** – Manages and controls the cluster.
✅ **Slave Nodes** – Perform actual data storage and processing tasks.

🔸 Components in Master-Slave Architecture:

1 **Master Node (NameNode)** – Manages metadata (file structure, location, permissions).
2 **Slave Nodes (DataNodes)** – Store actual data blocks.
3 **Secondary NameNode** – Supports checkpointing to avoid metadata loss.

🔹 HDFS (Hadoop Distributed File System)

HDFS is a **highly fault-tolerant distributed storage system** that allows large-scale data storage across multiple machines.

💡 **Key Features of HDFS:**
✅ **Distributed Storage** – Data is split into **blocks** and distributed across multiple nodes.
✅ **Fault Tolerance** – Automatically replicates data across different machines.
✅ **Scalability** – Handles petabytes of data efficiently.
✅ **High Throughput** – Optimized for batch processing of large files.

🔹 NameNode & DataNode

HDFS follows a **master-slave architecture** with two main components:

1 **NameNode (Master)**

- Stores metadata (file structure, permissions, locations).
- Keeps track of where file blocks are stored in DataNodes.
- Does **not store actual data**, only manages the filesystem.

2 **DataNodes (Slaves)**

- Store actual file blocks.
- Continuously send health reports to the NameNode.
- Responsible for **data replication** and recovery in case of failures.

📌 **Example:**
If a 300MB file is stored in HDFS (assuming block size = 128MB):

- It is split into **three blocks** (128MB, 128MB, 44MB).
- These blocks are distributed across multiple DataNodes.

🔹 Secondary NameNode & Checkpointing

🔹 The **Secondary NameNode** is **not a backup NameNode** but a **helper node** for metadata management.

Functions of Secondary NameNode:
✅ Periodically merges **FsImage (snapshot of metadata)** and **EditLogs (recent changes)** to create a new checkpoint.
✅ Reduces the workload of the **Primary NameNode** by managing metadata snapshots.
✅ Prevents metadata loss in case of NameNode failure.

📌 **Why is Checkpointing Needed?**

- The **NameNode constantly updates metadata** in EditLogs, causing them to grow.
- The **Secondary NameNode merges these logs** periodically to avoid excessive storage and speed up recovery.

🔹 **HDFS Data Blocks**

🔹 HDFS **stores large files as fixed-size blocks** (default size: **128MB or 256MB**).

✅ **Why Blocks?**

- Handling large files in smaller chunks improves efficiency.
- Data distribution across multiple nodes ensures fault tolerance.
- Blocks allow parallel processing, speeding up computations.

📌 **Example:**
A **500MB file** is stored in HDFS with a block size of **128MB**:

- Block 1: 128MB
- Block 2: 128MB
- Block 3: 128MB
- Block 4: 128MB
- Block 5: 4MB

These blocks are distributed across different **DataNodes**.

🔹 **HDFS Replication**

🔹 To **prevent data loss**, HDFS replicates each block multiple times.

✅ **Default Replication Factor**: **3**

- Each block is stored in **three different nodes** for redundancy.
- If one node fails, data can be recovered from other nodes.

📌 **Example:**
If **Block A** is stored on **Node 1**, it is also replicated to **Node 2 and Node 3**.

💡 **HDFS Replication Rules:**

1. One replica is stored on the same rack (local copy).

2. Another replica is placed on a different rack for reliability.
3. The third replica is placed randomly across the cluster.

🔹 HDFS Read/Write Mechanism

HDFS follows a **client-server** model for reading and writing data.

✅ **Write Process:**
1 The client requests the **NameNode** to create a new file.
2 The **NameNode assigns DataNodes** to store file blocks.
3 The client writes data in chunks, which are replicated.
4 Once all blocks are written, the **file is closed**.

✅ **Read Process:**
1 The client requests the **NameNode** to read a file.
2 The **NameNode provides block locations** from different DataNodes.
3 The client reads the blocks directly from the **nearest DataNodes**.

💡 **Optimized for Sequential Reads** – HDFS is designed for batch processing and is efficient for reading large files sequentially.

Hadoop Processing Frameworks

🔹 **MapReduce**

What is MapReduce?

MapReduce is a **distributed data processing framework** in Hadoop that enables parallel computation across multiple nodes. It follows the **Divide & Conquer** strategy:

✅ **Map Phase** – Splits the input data into key-value pairs.
✅ **Shuffle & Sort** – Groups and sorts data based on keys.
✅ **Reduce Phase** – Aggregates and processes the mapped data.

💡 **Why MapReduce?**

- Processes large datasets efficiently across multiple nodes.
- Works in a **fault-tolerant** and **scalable** manner.
- Handles **structured, semi-structured, and unstructured** data.

MapReduce Word Count Program

💡 A classic MapReduce example that counts the frequency of words in a dataset.

Steps in Word Count Program:

1 **Mapper Function:**

- Reads text input and emits key-value pairs (word, 1).

2 **Shuffling & Sorting:**

- Groups key-value pairs by words and sorts them.

3 **Reducer Function:**

- Aggregates word counts.

📌 **Example:**

🔹 **Input:**

Hadoop is powerful

Hadoop is scalable

🔹 **Mapper Output (Key-Value Pairs):**

(Hadoop, 1)

(is, 1)

(powerful, 1)

(Hadoop, 1)

(is, 1)

(scalable, 1)

🔹 **Reducer Output (Final Count):**

(Hadoop, 2)

(is, 2)

(powerful, 1)

(scalable, 1)

🔹 **YARN (Yet Another Resource Negotiator)**

YARN is Hadoop's **resource management framework** that schedules and manages computing resources.

✅ **Why YARN?**

- **Decouples resource management from computation.**
- Supports **multiple processing frameworks** (not just MapReduce).
- Optimizes **job execution and resource allocation.**

MapReduce Job Workflow

✅ **How a MapReduce job runs on Hadoop:**
1 The **client submits the job** to the YARN ResourceManager.
2 The **ResourceManager** assigns it to an **ApplicationMaster**.
3 The **ApplicationMaster** requests resources from **NodeManagers**.
4 **Mappers** process input data and generate intermediate key-value pairs.
5 The **Reduce tasks** aggregate results to produce the final output.
6 The output is stored in **HDFS**.

YARN Architecture

Components of YARN:
1 **ResourceManager (Master)** – Allocates cluster resources.
2 **NodeManager (Worker Nodes)** – Manages execution on individual nodes.
3 **ApplicationMaster** – Oversees execution of a job.
4 **Container** – An execution unit where tasks run.

🔹 Hadoop Ecosystem & Installation

Hadoop Architecture

✅ **Hadoop has a layered architecture with three key components:**

1. **HDFS (Storage Layer)** – Stores large files across multiple nodes.
2. **YARN (Resource Management Layer)** – Allocates and manages computing resources.
3. **MapReduce (Processing Layer)** – Performs distributed data processing.

Hadoop Ecosystem

The Hadoop ecosystem consists of several tools that extend its capabilities:

✅ **Data Storage & Processing:**

- **HDFS** – Distributed File System
- **HBase** – NoSQL Database

✅ **Data Ingestion:**

- **Sqoop** – Transfers data between Hadoop & RDBMS
- **Flume** – Ingests streaming data

✅ **Data Processing:**

- **MapReduce** – Batch processing framework
- **Apache Pig** – High-level scripting for Hadoop
- **Apache Hive** – SQL-like querying

✅ **Workflow & Resource Management:**

- **YARN** – Resource Management
- **Oozie** – Workflow Scheduling

Hadoop Cluster Mode

Hadoop can be deployed in different modes:
1 **Standalone Mode** – Runs on a single machine (for testing).
2 **Pseudo-Distributed Mode** – Simulates a cluster on a single machine.
3 **Fully Distributed Mode** – Runs on multiple machines (production).

Hadoop Installation

📌 **Basic Steps to Install Hadoop:**

1. Install **Java** (JDK).
2. Download & extract **Hadoop binaries**.
3. Configure **core-site.xml, hdfs-site.xml, yarn-site.xml**.
4. Format **HDFS** and start Hadoop services.
5. Verify installation with basic commands (hadoop fs -ls /).

🔹 MapReduce Examples

Weather Data Set Analysis

✅ **Objective:** Analyze temperature trends from historical weather data.

📌 **Process:**

- **Mapper:** Extracts temperature data from records.
- **Reducer:** Computes **max/min temperatures** per year.

Example Output:

(2010, 35°C)

(2011, 38°C)

(2012, 37°C)

MapReduce Last.FM Example

✅ **Objective:** Analyze song play counts from **Last.FM music streaming data**.

📌 **Process:**

- **Mapper:** Extracts song & play count from logs.
- **Reducer:** Aggregates play counts per song.

Example Output:

(Imagine Dragons - Believer, 15000 plays)

(Coldplay - Fix You, 12000 plays)

Data Ingestion Tools

Big Data systems require **data ingestion** tools to transfer data from different sources into Hadoop. Two popular tools for this are **Apache Sqoop** (for structured data) and **Apache Flume** (for streaming data).

🔹 Apache Sqoop

What is Sqoop?

Apache Sqoop is a **data transfer tool** that efficiently moves **structured data** between **relational databases (RDBMS) and Hadoop (HDFS, Hive, HBase, etc.)**.

💡 **Why Sqoop?**
✅ Automates bulk data transfers between Hadoop & databases.
✅ Supports data import/export from **MySQL, PostgreSQL, Oracle, etc.**
✅ Optimized for **parallel processing** using MapReduce.

Features of Sqoop

✅ **Efficient Data Transfer** – Uses **parallelization** for fast imports/exports.
✅ **Data Compression** – Supports gzip, bzip2 compression for efficiency.
✅ **Incremental Import** – Imports only **new or updated records**.
✅ **Export Capabilities** – Moves processed data back to relational databases.
✅ **Hadoop Integration** – Works with **HDFS, Hive, HBase**.

Sqoop Architecture

📌 **How Sqoop Works:**
1 **Client** sends a request to transfer data.
2 **Sqoop Driver** interacts with the relational database.
3 **Connectors** (JDBC) fetch data from the source database.
4 **MapReduce Jobs** handle parallel data transfer.
5 The data is stored in **HDFS, Hive, or HBase**.

🔹 **Key Components:**

- **JDBC Connector** – Connects to relational databases.
- **Sqoop CLI** – Command-line interface for executing jobs.
- **HDFS/Hive/HBase Integration** – Stores data in Hadoop.

Sqoop Commands

Import Data from RDBMS to HDFS

```
sqoop import --connect jdbc:mysql://localhost/db_name --username user --password pass \
--table employees --target-dir /hdfs_path
```

✅ Transfers the **employees** table from MySQL to **HDFS**.

Export Data from HDFS to RDBMS

```
sqoop export --connect jdbc:mysql://localhost/db_name --username user --password pass \
--table employees --export-dir /hdfs_path
```

✅ Moves processed data from **HDFS back to MySQL**.

List Databases in RDBMS

```
sqoop list-databases --connect jdbc:mysql://localhost --username user --password pass
```

✅ Displays all databases in MySQL.

🔹 **Apache Flume**

What is Flume?

Apache Flume is a **real-time data ingestion tool** designed for **streaming data** from sources like logs, social media, and sensors into **Hadoop**.

💡 **Why Flume?**
✅ Handles **large-scale real-time event data.**
✅ Uses **distributed architecture** for scalability.
✅ Supports **data transformation & filtering** before ingestion.

Flume Architecture

Apache Flume has a **pipeline-based architecture** with three key components:

- 🔹 **Source** – Collects data from logs, web servers, social media, etc.
- 🔹 **Channel** – Temporarily buffers data before processing.
- 🔹 **Sink** – Sends data to **HDFS, HBase, Hive, or Kafka**.

📌 **Data Flow:**
1 **Source** (Web servers, Twitter API) →
2 **Channel** (Memory, File, JDBC) →
3 **Sink** (HDFS, Hive, Kafka)

Flume Twitter Streaming

✅ Apache Flume can **capture real-time tweets** and store them in **HDFS**.

Example: Twitter Streaming to HDFS

1 **Set up Twitter API Keys.**
2 **Configure Flume Twitter Source.**
3 **Define HDFS Sink.**
4 **Run Flume Agent.**

📌 **Flume Configuration Example (twitter.conf)**

```
TwitterAgent.sources = Twitter

TwitterAgent.channels = MemChannel

TwitterAgent.sinks = HDFS


# Twitter API credentials

TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource

TwitterAgent.sources.Twitter.consumerKey = <your_key>

TwitterAgent.sources.Twitter.consumerSecret = <your_secret>

TwitterAgent.sources.Twitter.accessToken = <your_token>

TwitterAgent.sources.Twitter.accessTokenSecret = <your_token_secret>


# Channel Configuration

TwitterAgent.channels.MemChannel.type = memory

TwitterAgent.channels.MemChannel.capacity = 10000
```

```
# HDFS Sink

TwitterAgent.sinks.HDFS.type = hdfs

TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/twitter_data/

TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
```

✅ This configuration collects tweets, buffers them in memory, and stores them in HDFS.

Data Processing Tools

Big Data systems use **data processing tools** to analyze and transform large datasets. The two main tools are:

✅ **Apache Pig** – A high-level scripting language for parallel processing.

✅ **Apache Hive** – A SQL-based query engine for structured data in Hadoop.

🔹 Apache Pig

What is Apache Pig?

Apache Pig is a **high-level scripting framework** designed for processing large datasets on Hadoop using a simple scripting language called **Pig Latin**.

💡 **Why Pig?**

✅ **Simplifies complex MapReduce tasks** with fewer lines of code.

✅ **Optimized for large-scale parallel processing** on Hadoop.

✅ **Extensible** – Users can write UDFs (User-Defined Functions) in Java or Python.

Pig vs. MapReduce

Feature	Apache Pig	MapReduce
Language	Pig Latin (scripting)	Java (low-level coding)
Complexity	High-level (easier)	Low-level (harder)
Optimization	Optimized execution plan	Manual tuning needed
Readability	Simple & concise	Lengthy & complex

📌 **Example** – Word Count in Pig vs. MapReduce

```
-- Pig Latin (Easy)

data = LOAD 'input.txt' AS (line:chararray);

words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS word;

grouped = GROUP words BY word;

counted = FOREACH grouped GENERATE group, COUNT(words);

DUMP counted;
```

✅ **Fewer lines of code** compared to MapReduce.

Twitter Case Study with Pig

Apache Pig can **analyze social media data**, such as extracting trending hashtags from Twitter.

📌 **Example: Finding Trending Hashtags**

```
tweets = LOAD 'tweets.txt' AS (tweet:chararray);

hashtags = FOREACH tweets GENERATE FLATTEN(REGEX_EXTRACT_ALL(tweet, '#\\w+')) AS hashtag;

grouped = GROUP hashtags BY hashtag;

trending = FOREACH grouped GENERATE group AS hashtag, COUNT(hashtags) AS count;

DUMP trending;
```

✅ Finds the most frequently mentioned hashtags in tweets.

Pig Architecture & Components

🔹 **Parser** – Validates syntax & generates an abstract syntax tree.
🔹 **Optimizer** – Rewrites the logical plan for efficiency.
🔹 **Compiler** – Converts the logical plan into MapReduce jobs.
🔹 **Execution Engine** – Runs the compiled jobs on Hadoop.

Pig Data Models

✅ **Atom** – Single data value (e.g., string, number).
✅ **Tuple** – Ordered set of fields (like a row).
✅ **Bag** – Collection of tuples.
✅ **Map** – Key-value pairs (like a dictionary).

Pig Commands

📌 **Load Data:**

```
data = LOAD 'input.txt' AS (name:chararray, age:int);
```

📌 **Filtering Data:**

```
filtered = FILTER data BY age > 25;
```

📌 **Grouping & Counting:**

```
grouped = GROUP data BY name;

counted = FOREACH grouped GENERATE group, COUNT(data);
```

📌 **Storing Results:**

```
STORE counted INTO 'output.txt';
```

✅ **Easier than writing raw MapReduce code!**

🔹 **Apache Hive**

What is Hive?

Apache Hive is a **SQL-based data warehouse** built on Hadoop that allows users to write queries in **HiveQL** (similar to SQL).

💡 **Why Hive?**
✅ **SQL-like syntax** – No need for complex Java coding.
✅ **Optimized for analytical queries** on large datasets.
✅ **Integrates with HDFS, HBase, and other Big Data tools.**

PigLatin vs. HiveQL

Feature	**Apache Pig**	**Apache Hive**
Language	Pig Latin (procedural)	HiveQL (SQL-based)
Best for	Data transformation	Data querying
Complexity	Moderate	Easy
Use case	ETL & data preparation	Reporting & analytics

📌 **Example – Filtering data in Pig vs. Hive**

🔹 **Apache Pig (Procedural approach)**

```
data = LOAD 'employees.txt' AS (name:chararray, age:int, salary:int);

filtered = FILTER data BY salary > 50000;

DUMP filtered;
```

🔹 **Apache Hive (SQL approach)**

```
SELECT * FROM employees WHERE salary > 50000;
```

✅ **Hive is best for SQL-like querying!**

Hive Architecture & Components

🔹 **Metastore** – Stores metadata (table schemas, partitions).
🔹 **Compiler** – Converts HiveQL queries into execution plans.
🔹 **Execution Engine** – Runs queries using MapReduce, Tez, or Spark.
🔹 **HDFS Storage** – Stores large datasets.

Hive Commands & Setup

📌 **Create a Table:**

```
CREATE TABLE employees (
  id INT,
  name STRING,
  salary FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

📌 **Load Data into Table:**

```
LOAD DATA INPATH '/data/employees.csv' INTO TABLE employees;
```

📌 **Query Data:**

```
SELECT name, salary FROM employees WHERE salary > 50000;
```

📌 **Join Tables:**

```
SELECT e.name, d.department
FROM employees e
JOIN departments d
ON e.id = d.id;
```

✅ **Familiar SQL-like syntax makes it easy to use!**

Advanced Hive Concepts

Hive Partitioning

Partitions improve query performance by **organizing data** into separate directories.

```
CREATE TABLE sales (
    id INT,
    amount FLOAT
)
PARTITIONED BY (year INT, month INT);
```

✅ Queries are **faster** because only relevant partitions are scanned.

Bucketing in Hive

Buckets distribute data into **fixed-sized segments** for faster retrieval.

```
CREATE TABLE users (
    id INT,
    name STRING
)
CLUSTERED BY (id) INTO 10 BUCKETS;
```

✅ **Used for better load balancing in queries.**

External Tables

Hive can use external tables to query data without moving it.

```
CREATE EXTERNAL TABLE logs (
    event STRING,
    timestamp STRING
)
LOCATION '/hdfs_path/logs/';
```

✅ **Data remains in HDFS; Hive just references it.**

NoSQL & Apache HBase

Big Data applications often require **scalable and flexible databases** that can handle unstructured or semi-structured data efficiently. **NoSQL databases** like Apache HBase provide a solution for such use cases.

🔹 **NoSQL & Apache HBase**

Types of NoSQL Databases

NoSQL databases are categorized based on how they store and retrieve data.

Type	Description	Example Databases
Key-Value Stores	Store data as key-value pairs	Redis, DynamoDB
Document Stores	Store semi-structured documents (JSON, BSON, XML)	MongoDB, CouchDB
Column-Family Stores	Store data in columns instead of rows	Apache HBase, Cassandra
Graph Databases	Represent relationships using nodes and edges	Neo4j, ArangoDB

✅ **HBase is a Column-Family NoSQL database** optimized for Big Data applications.

History of HBase

Apache HBase is inspired by **Google Bigtable**, a distributed storage system designed for handling massive amounts of structured data.

📌 **Key Events:**

- **2006** – Google releases **Bigtable** paper.
- **2008** – Apache HBase introduced as an **open-source implementation**.
- **2011** – HBase becomes a **top-level Apache project**.
- **Now** – HBase is widely used in **real-time Big Data applications**.

HBase vs. RDBMS

Feature	HBase (NoSQL)	RDBMS (SQL)

Feature	HBase (NoSQL)	RDBMS (SQL)
Schema	Schema-less	Fixed schema
Data Model	Column-family	Table-based (rows & columns)
Scalability	Horizontally scalable	Vertically scalable
Transactions	No ACID compliance	ACID-compliant
Best for	Real-time Big Data	Structured data

📌 **Example:**

- **Use RDBMS (SQL)** if you need strong **transactions & relationships** (e.g., banking).
- **Use HBase (NoSQL)** if you need **fast lookups & horizontal scaling** (e.g., web analytics, IoT data).

Uses of HBase

🔹 **Real-time analytics** – Processing massive datasets quickly.
🔹 **IoT applications** – Storing and querying sensor data.
🔹 **Fraud detection** – Fast access to customer transactions.
🔹 **Social media analytics** – Storing user interactions and logs.

Companies Using HBase

✅ **Facebook** – Stores billions of messages for **Facebook Messenger**.
✅ **LinkedIn** – Stores user activity logs for **real-time analytics**.
✅ **Netflix** – Manages user recommendations with HBase.
✅ **Adobe** – Uses HBase for digital marketing data storage.

🔹 HBase Data Models & Architecture

HBase Data Model

Unlike RDBMS, **HBase stores data in column families** rather than rows.

Row Key	Column Family: Personal	Column Family: Contact
001	Name: Alice, Age: 25	Email: alice@mail.com, Phone: 12345

Row Key	**Column Family: Personal**	**Column Family: Contact**
002	Name: Bob, Age: 30	Email: bob@mail.com, Phone: 67890

📌 **HBase stores data in flexible columns instead of rigid tables.**

Row vs. Column-Oriented Databases

Feature	**Row-Oriented (SQL)**	**Column-Oriented (HBase)**
Storage	Data stored row by row	Data stored by columns
Best for	OLTP (transactional workloads)	OLAP (analytical workloads)
Query Speed	Faster for small datasets	Faster for large datasets

📌 **HBase is optimized for reading & writing large volumes of columnar data.**

HBase Physical Storage

HBase stores data in **HDFS** using the following structure:

1 **MemStore** – Stores data in memory before writing to disk.
2 **HFiles** – Stores persistent data on HDFS.
3 **Write-Ahead Log (WAL)** – Ensures durability in case of failure.

✅ **Data is first written to MemStore, then flushed to HFiles.**

HBase Architecture & Components

📌 **Key Components:**
🔹 **HMaster** – Manages regions and assigns them to RegionServers.
🔹 **RegionServers** – Handles read/write requests and stores data.
🔹 **ZooKeeper** – Ensures coordination and fault tolerance.

HBase Read & Write Mechanism

🔹 **Write Process:**

1. **Client writes data** → Stored in MemStore.
2. **MemStore fills up** → Data is flushed to **HFiles**.

3. **Write-Ahead Log (WAL)** ensures durability.

🔹 **Read Process:**

1. **Client sends a read request**.
2. **Data is fetched from MemStore (if available)**.
3. **If not in MemStore, it's read from HFiles** in HDFS.

✅ **This structure allows fast reads & writes in HBase!**

Compaction in HBase

HBase periodically **merges smaller HFiles** into larger files to optimize performance.

📌 **Types of Compaction:**

- **Minor Compaction** – Merges small HFiles to reduce fragmentation.
- **Major Compaction** – Merges all HFiles for efficient storage.

✅ **Reduces the number of disk reads and improves performance.**

HBase Shell & Client API

📌 **Create a Table:**

create 'users', 'personal', 'contact'

📌 **Insert Data:**

put 'users', '001', 'personal:name', 'Alice'

put 'users', '001', 'contact:email', 'alice@mail.com'

📌 **Read Data:**

scan 'users'

✅ **HBase Shell allows direct interaction with HBase tables.**

📌 **HBase Java API Example:**

Configuration config = HBaseConfiguration.create();

Connection connection = ConnectionFactory.createConnection(config);

Table table = connection.getTable(TableName.valueOf("users"));

✓ **Java API allows integration with enterprise applications.**

Big Data Projects & Distributed Processing

🔹 **Hadoop E-Commerce Projects**

📌 **Example Use Cases:**

✅ **Customer Segmentation** – Analyzing shopping patterns.

✅ **Recommendation Systems** – Predicting user preferences.

✅ **Fraud Detection** – Identifying fraudulent transactions.

✅ **Inventory Management** – Forecasting product demand.

🔹 **Distributed Cache in Hadoop**

📌 **What is Distributed Cache?**

A mechanism that **distributes and caches small files** across Hadoop nodes to improve performance.

✅ **Used for:**

- Storing lookup tables.
- Caching frequently accessed data.
- Speeding up **MapReduce jobs**.

📌 **Example:**

```
DistributedCache.addCacheFile(new URI("/cache/user_data.txt"), conf);
```

✅ **This reduces data movement across nodes!**

🔹 **Code Sections**

📌 **Example: Running a Hadoop Job with Distributed Cache**

```
hadoop jar myjob.jar -Dmapreduce.job.cache.files=/cache/lookup.txt
```

✅ **Cache files improve Hadoop job efficiency.**