Big Data Analyst

MOST IMPORTANT

Unit 1

✅ 1. Different Types of Digital Data

Digital data can be broadly classified into **3 types**:

1. Structured Data

- **Definition:** Data that follows a pre-defined model and is stored in tabular form.
- **Examples:** SQL databases, Excel files.
- **Characteristics:**
  - Easy to enter, store, query, and analyze.
  - Organized into rows and columns.
- **Use Cases:** Online transaction systems, banking records, student databases.

2. Semi-Structured Data

- **Definition:** Data that doesn't follow strict tabular structure but still has some organizational properties.
- **Examples:** XML, JSON, email (with metadata), NoSQL databases.
- **Characteristics:**
  - Contains tags or markers for separation.
  - More flexible than structured data.

3. Unstructured Data

- **Definition:** Data without a specific format or structure.
- **Examples:** Text files, images, audio, video, social media posts.

- **Characteristics:**
  - Most challenging to process and analyze.
  - Makes up around **80%** of the world's data.

✅ 2. Key Characteristics of Big Data (3Vs and 5Vs)

The core characteristics of Big Data are explained using **Vs**:

🔸 **3Vs (Classic definition)**

1. **Volume** – Massive amount of data (terabytes, petabytes).
2. **Velocity** – Speed at which data is generated and processed.
3. **Variety** – Different types of data (structured, semi-structured, unstructured).

🔹 **Extended to 5Vs**

4. **Veracity** – Data quality and trustworthiness.
   - Data might be incomplete, noisy, or inconsistent.
5. **Value** – Worth derived from analyzing big data.
   - Not all data is useful, but valuable insights can lead to better decision-making.

Example: Social media generates huge **volume** of unstructured **variety** of data at high **velocity**, but its **veracity** might be questionable unless filtered and analyzed to extract **value**.

✅ 3. Graph Databases and Their Differences from Relational Databases

🔹 **Graph Databases**

- **Definition:** A database that uses graph structures with nodes, edges, and properties to represent and store data.
- **Popular Examples:** Neo4j, Amazon Neptune.

- **Structure:**
    - **Nodes** = entities (e.g., person, product)
    - **Edges** = relationships between entities
    - **Properties** = additional info about nodes/edges

🔸 **Relational Databases**

- **Definition:** Traditional databases that store data in tables (rows and columns).
- **Examples:** MySQL, Oracle, PostgreSQL.

📊 **Comparison Table**

Feature	Relational DB (RDBMS)	Graph DB
Data Structure	Tables	Graphs (nodes & edges)
Relationships	Through foreign keys & joins	Direct connections (edges)
Query Language	SQL	Cypher (Neo4j)
Speed for Complex Relationships	Slower (many joins)	Faster (traverse relationships directly)
Example Use Cases	Banking, ERP systems	Social networks, Recommendation engines

---

✅ 4. Difference Between Traditional Data and Big Data

Feature	Traditional Data	Big Data
Volume	Low to moderate (MBs to GBs)	Extremely large (TBs to PBs+)
Variety	Mostly structured (tables)	Structured, semi-structured, unstructured
Velocity	Slow to moderate data generation	High-speed streaming and real-time
Storage	Relational Databases (RDBMS)	Distributed storage (HDFS, NoSQL)

Processing	Centralized (single server)	Distributed (parallel processing)
Tools Used	SQL, Oracle, MySQL	Hadoop, Spark, Hive, Pig
Scalability	Vertical (adding power to one machine)	Horizontal (adding more machines)
Cost	Expensive due to high-end servers	Cost-effective using commodity hardware

🔍 Example:
Traditional systems work well for transactional data (e.g., banking).
Big Data is needed for analyzing social media, IoT sensor data, or clickstreams from websites.

✅ 5. Graph Databases and Their Differences from Relational Databases

🔷 Graph Databases:

- **Based on:** Graph theory (nodes and relationships/edges).
- **Best For:** Complex, interconnected data (social networks, fraud detection, recommendation systems).
- **Example:** Neo4j, Amazon Neptune.

🔶 Relational Databases (RDBMS):

- **Based on:** Tables (rows and columns).
- **Best For:** Structured data with clear schema (banking systems, payroll, inventory).
- **Example:** MySQL, PostgreSQL, Oracle.

📊 Comparison Table:

Feature	Relational DB	Graph DB
Data Representation	Tables	Nodes & Edges
Relationships	Foreign keys + Joins	Direct edges (relationships)

Query Language	SQL	Cypher (Neo4j)
Performance on Relationships	Slower with multiple joins	Fast for deep relationships
Schema	Fixed	Flexible
Use Case	Structured business data	Social media, logistics, recommendations

✅ **When to use Graph DB:**
If your query looks like “Find friends of friends of friends” — Graph DBs are much more efficient!

Unit 2

✅ 1. HDFS Client Interface / Command Line Usage (HDFS CLI)

HDFS (Hadoop Distributed File System) allows interaction through a **command line interface (CLI)**, similar to Linux shell commands.

🔧 Common HDFS CLI Commands

Command	Description
`hdfs dfs -ls /`	List files and directories in root directory
`hdfs dfs -mkdir /mydir`	Create a new directory
`hdfs dfs -put localfile.txt /mydir`	Upload file from local system to HDFS
`hdfs dfs -get /mydir/file.txt`	Download file from HDFS to local
`hdfs dfs -cat /mydir/file.txt`	Display contents of a file
`hdfs dfs -rm /mydir/file.txt`	Remove a file
`hdfs dfs -du /`	Display space usage of files
`hdfs dfs -cp /src /dest`	Copy file or directory in HDFS
`hdfs dfs -mv /src /dest`	Move file or directory in HDFS

📁 **Example Flow:**

bash
CopyEdit

```
hdfs dfs -mkdir /student_data
hdfs dfs -put students.csv /student_data
hdfs dfs -ls /student_data
hdfs dfs -cat /student_data/students.csv
```

> 💡 HDFS commands are essential for uploading/downloading data to/from the Hadoop file system and managing files in distributed storage.

Unit 3

✅ 1. MapReduce Job Process (Steps of Execution)

MapReduce is a **programming model** used to process large-scale data in a **distributed** environment like Hadoop.

🔄 Steps of a MapReduce Job Execution:

1. **Input Splitting**
    - Input data is split into blocks (e.g., 128MB each).
    - Each block is processed in parallel.
2. **Map Phase**
    - Mapper function processes input splits.
    - Converts data into key-value pairs (`<key, value>`).
    - Example: Word count → `("hello", 1)`
3. **Shuffling & Sorting**
    - Output of mapper is sorted by key.
    - Keys are grouped and sent to the correct reducer.
    - Ensures same keys go to the same reducer.

4. **Reduce Phase**
    - Reducer receives grouped key-value pairs.
    - Processes them to give final output.
    - Example: `("hello", [1,1,1])` → `("hello", 3)`
5. **Output Writing**
    - Final results are written to **HDFS**.

🛠️ Real-world Example (Word Count):

📄 Input File:

```
hello world
hello Hadoop
```

👨‍💻 **Mapper Output:**

```
hello → 1
world → 1
hello → 1
Hadoop → 1
```

🔀 **Shuffling + Sorting:**

```
hello → [1, 1]
world → [1]
Hadoop → [1]
```

🧠 **Reducer Output:**

```
hello → 2
world → 1
Hadoop → 1
```

✅ 2. Different Types of Failures in MapReduce

Failures can happen at different stages of a MapReduce job. Hadoop handles failures gracefully.

⚠️ Common Failures:

Failure Type	Description	Handling
Task Tracker Failure	A node running tasks crashes.	JobTracker reassigns the task to another node.
DataNode Failure	DataNode storing HDFS blocks fails.	Replication in HDFS provides data from another node.
Job Tracker Failure	The master (JobTracker) fails.	In Hadoop 1.x, it's a **single point of failure**. In Hadoop 2.x (YARN), ResourceManager can recover.
Map Task Failure	A mapper task fails due to code or hardware.	Re-run on a different node.
Reduce Task Failure	A reducer crashes before writing output.	Re-run from the intermediate data.
Network Issues	Data transfer between map & reduce fails.	Retried automatically or rerun the task.

🧠 Hadoop is fault-tolerant: it **detects and reassigns** failed tasks automatically!

✅ 3. Different Phases of MapReduce Job Execution

Here's a visual breakdown of phases:

`INPUT → Splitting → Mapping → Shuffling → Reducing → OUTPUT`

Phase	What Happens	Purpose
Input Splits	Input is divided into splits (chunks).	Enables parallel processing
Map Phase	Mapper processes input and emits `<key, value>`.	Extracts useful data

Shuffle & Sort	Keys are sorted and grouped across nodes.	Prepares for reduction
Reduce Phase	Reducer processes and merges values per key.	Produces final result
Output Phase	Results are saved in HDFS.	Final storage

🔄 MapReduce Flow Recap:

1. **Client** submits job.
2. **JobTracker/ResourceManager** assigns tasks.
3. **InputSplit** happens.
4. **Map** task runs → emits key-values.
5. **Shuffle & Sort** organizes data.
6. **Reduce** task aggregates values.
7. **Output** is saved in HDFS.

UNIT 4

✅ 1. Hadoop Ecosystem and Its Components

📌 What is Hadoop Ecosystem?

The **Hadoop Ecosystem** is a **framework of tools and technologies** built around the Hadoop platform to process, store, and analyze big data in a distributed manner.

Hadoop itself has two core parts:

- **HDFS (Storage)**
- **MapReduce (Processing)**
  But it needs more tools to be truly powerful — that's where the ecosystem comes in.

🌐 Key Components of the Hadoop Ecosystem

Category	Tool	Description
Storage	**HDFS**	Hadoop Distributed File System — stores data in distributed fashion
Data Processing	**MapReduce**	Programming model for processing large data sets
Query Languages	**Pig**	Scripting platform for data transformation using **Pig Latin**
	Hive	SQL-like interface for querying data using **HiveQL**
NoSQL Database	**HBase**	Column-based NoSQL database that runs on HDFS
Coordination	**Zookeeper**	Maintains configuration and synchronization across services
Workflow	**Oozie**	Workflow scheduler for Hadoop jobs
Data Ingestion	**Sqoop**	Transfers data between Hadoop and relational databases
	Flume	Ingests log and event data into HDFS
Data Serialization	**Avro**	For compact, fast, and efficient serialization of data
Resource Management	**YARN**	Allocates resources and schedules tasks
Machine Learning	**Mahout**	Scalable ML library
Visualization	**BigSheets (IBM)**	Spreadsheet-like tool to explore big data visually

---

🧠 Why is the Ecosystem Needed?

- HDFS alone can **store** and MapReduce can **process**, but for:
    - Data **ingestion** → Use Sqoop/Flume
    - **Querying** → Use Hive/Pig
    - **Scheduling** → Use Oozie
    - **Real-time processing** → Use tools like Spark/Storm

✅ 2. Apache Pig & Its Execution Modes + Comparison with Traditional Databases

🐷 What is Apache Pig?

- **Apache Pig** is a **high-level platform** for processing large datasets using a scripting language called **Pig Latin**.
- It simplifies **MapReduce programming** by abstracting complex code into simpler scripts.

💻 Execution Modes of Pig

Mode	Description
Local Mode	Runs on a single machine using local file system. Ideal for testing/debugging.
MapReduce Mode	Default. Pig converts scripts into MapReduce jobs that run on Hadoop cluster.
Tez Mode (Advanced)	Uses Apache Tez instead of MapReduce — faster and optimized.

🧾 Pig Latin Example:

pig

```
-- Load the file
data = LOAD 'sales.csv' USING PigStorage(',') AS (id:int,
amount:float);

-- Filter the data
high_sales = FILTER data BY amount > 1000;

-- Group and sum
grouped = GROUP high_sales BY id;
total = FOREACH grouped GENERATE group, SUM(high_sales.amount);
```

This is much simpler than writing MapReduce code manually!

🆚 Pig vs Traditional Databases

Feature	Pig (Big Data)	RDBMS (Traditional DB)
Language	Pig Latin	SQL
Data Size	Huge, unstructured/semi-structured	Structured, moderate size
Schema	Schema-on-read (flexible)	Schema-on-write (rigid)
Processing	Batch, parallel	Single node or limited parallelism
Execution	Converts to MapReduce jobs	Executes on RDBMS engine
Ideal for	ETL, large-scale data analysis	CRUD operations, transactions

---

UNIT 5

✅ 1. Supervised Learning: Classification vs. Regression

(Same as explained earlier — rephrased for completeness)

🧠 Supervised Learning

- Machine learns from **labeled data** (input + known output).
- Useful for making **predictions** or **decisions**.

🔴 Classification

- **Goal**: Predict a **class** or **category**.
- **Output**: Discrete labels (Yes/No, 0/1, Red/Blue).
- **Example**: Predict if an email is spam or not.

Algorithm Examples

Logistic Regression

Decision Trees

Naive Bayes

SVM

KNN

🔵 Regression

- **Goal**: Predict a **continuous value**.
- **Output**: Numeric values.
- **Example**: Predict house price based on size/location.

Algorithm Examples
Linear Regression
Lasso/Ridge Regression
SVR
Decision Trees for Regression

Feature	Classification	Regression
Output	Categorical	Numerical
Example	Spam/Not Spam	House Price
Algorithms	Logistic Regression	Linear Regression

✅ 2. Collaborative Filtering: User-Based vs. Item-Based

💡 What is Collaborative Filtering?

- A **recommendation system** approach that relies on **user behavior** (ratings, likes).

- Makes predictions based on **similarities**.

👤 A. User-Based Collaborative Filtering

- Finds users with **similar tastes**.
- Recommends what **similar users** liked.

🔁 Steps:

1. Find similar users (based on history)
2. Recommend items liked by them

🧠 Example:

- You and User X liked the same 3 songs.
- User X liked a 4th song you haven't heard → Recommend it.

📦 B. Item-Based Collaborative Filtering

- Finds **similar items** based on user ratings.
- Recommends items **similar to what user liked before**.

🔁 Steps:

1. Find items similar to what the user liked
2. Recommend those items

🧠 Example:

- You liked "Stranger Things".
- Recommend "Dark" because many users who liked one also liked the other.

Feature	User-Based CF	Item-Based CF
Based on	User similarity	Item similarity
Stability	Changes if user behavior changes	More stable
Use Case	Netflix, Goodreads	Amazon, Spotify

STILL IMPORTANT

UNIT 1

✅ 1. Impacts of Big Data on Processing and Decision-Making

💡 Overview

Big Data refers to data that's **large in volume**, **varied in type**, and **fast in generation** (3Vs: Volume, Variety, Velocity). It **impacts how we process information** and **make decisions** — both in real-time and strategically.

🔄 A. Impact on Data Processing

Traditional Processing	With Big Data
Limited to small data	Capable of handling terabytes/petabytes
Single system	Distributed processing (across clusters)
Batch jobs take hours/days	Real-time or near real-time
Limited parallelism	High parallelism using frameworks like Hadoop, Spark
Limited to structured data	Handles structured, semi-structured, unstructured data

Example:
Instead of analyzing 10,000 customer reviews manually or with Excel, Big Data tools like **Hadoop + NLP** can process **millions of reviews** for sentiment, keywords, and trends automatically.

📊 B. Impact on Decision-Making

Area	How Big Data Helps
Business Strategy	Predict future demand, optimize pricing, target right customers
Healthcare	Personalized treatment plans using patient history and genome data
Finance	Real-time fraud detection and investment decisions
Marketing	Targeted ads based on browsing/purchase behavior
Retail	Dynamic inventory control, sales forecasting

Example: Amazon uses Big Data to recommend products, forecast demand, and optimize warehouse logistics — saving millions and boosting user experience.

UNIT 2

✅ 2. Comparison of Functionalities: HDFS, MapReduce, YARN

Feature	HDFS	MapReduce	YARN
Role	Storage layer	Processing layer	Resource management
Full Form	Hadoop Distributed File System	-	Yet Another Resource Negotiator
Primary Function	Stores huge datasets across multiple machines	Processes large datasets via Map and Reduce steps	Manages cluster resources and job scheduling
Type	File system	Programming model	Resource manager
Interaction	Stores input/output of MapReduce jobs	Reads from HDFS, writes to HDFS	Coordinates MapReduce jobs across nodes
Components	NameNode, DataNode	Mapper, Reducer	ResourceManager, NodeManager
Handles	Data storage	Computation	Job execution, monitoring, and resource allocation

Failure Handling	Replication of blocks	Re-execution of failed tasks	Reassigns failed resources dynamically

---

📍 Simple Analogy:

Think of a **Pizza Factory**:

- **HDFS** = Warehouse (stores all ingredients)
- **MapReduce** = Workers (make pizzas)
- **YARN** = Manager (assigns workers to kitchen stations, monitors progress)

✅ 3. NameNode vs. DataNode (HDFS Components)

🔹 NameNode (Master Node)

- Acts as **the brain** of HDFS.
- **Stores metadata** (file names, directory structure, block locations).
- Keeps track of which DataNode holds what block.

🔸 DataNode (Slave Node)

- Stores **actual data blocks** (files split into blocks, usually 128MB).
- Periodically sends **heartbeats** and block reports to NameNode.
- Handles **read/write requests** from clients or MapReduce jobs.

🧱 Example Breakdown:

Suppose you store a 300MB file in HDFS.

- HDFS splits it into 3 blocks (128MB + 128MB + 44MB).
- Each block is replicated (default: 3 times).

- NameNode keeps track:
  - Block 1 → DataNode A, B, C
  - Block 2 → DataNode B, C, D
  - Block 3 → DataNode A, D, E

🆚 Comparison Table

Feature	NameNode	DataNode
Type	Master	Slave
Stores	Metadata	Actual file data
Responsibility	Directory tree, block locations	Store, retrieve blocks
Failure Impact	**Critical** — entire HDFS stops	Tolerated due to replication
Communication	Talks to clients and DataNodes	Talks to NameNode only
Heartbeat Check	Receives heartbeats	Sends heartbeats

🚨 What happens if a DataNode fails?

- NameNode detects no heartbeat.
- Automatically re-replicates blocks from other copies.

🧠 Real-Life Analogy:

- **NameNode** = Librarian (knows where every book is, doesn't hold them)
- **DataNode** = Bookshelves (holds the actual books)

⚠️ Tip for Viva/Exam:

Be ready to draw this simple diagram 👇

```
Client
   |
NameNode (Metadata)
   |     \
   |      \
DataNode1 (Block A)
DataNode2 (Block B)
DataNode3 (Block C)
```

UNIT 3

✅ 1. Shuffling and Sorting in MapReduce

🔄 What is Shuffling?

- **Shuffling** is the **process of transferring data** from the **Map phase to the Reduce phase**.
- It takes **intermediate key-value pairs** output by the Mapper and **groups all values by key** before passing them to the Reducer.

📦 What is Sorting?

- Before sending data to the Reducer, the framework **automatically sorts** the intermediate keys (ascending order by default).
- Sorting helps Reducers process records **in a predictable order**.

🧠 Example:

Imagine a WordCount job.

📤 Mapper Output (intermediate key-value pairs):

```
("apple", 1), ("banana", 1), ("apple", 1), ("cherry", 1)
```

🔁 After Shuffling + Sorting:

```
("apple", [1, 1]), ("banana", [1]), ("cherry", [1])
```

This gouped and sorted data is sent to the Reducers.

🎯 Why it's Important:

Feature	Role
Shuffling	Distributes correct data to Reducers
Sorting	Ensures Reducers receive sorted keys
Optimization	Helps in efficient grouping and performance

UNIT 4

✅ 1. Pig's Role in the Hadoop Ecosystem

🐷 What is Apache Pig?

- Apache Pig is a **high-level platform** for creating **MapReduce programs** using a **scripting language called Pig Latin**.
- It helps **analyze large data sets** without writing complex Java code like in traditional MapReduce.

🧑‍💻 Pig Latin Example:

pig

```
data = LOAD 'input.txt' AS (word:chararray);
grouped = GROUP data BY word;
counts = FOREACH grouped GENERATE group, COUNT(data);
```

🎯 Pig's Role:

Feature	Pig
Abstraction	Hides low-level MapReduce complexity

Ease of Use	Uses SQL-like syntax (Pig Latin)
Execution	Translates Pig Latin to MapReduce jobs
Extensibility	Allows UDFs (User Defined Functions) in Java, Python
Modes	Local Mode (single machine), MapReduce Mode (Hadoop cluster)

🔄 Pig vs. Traditional DBMS

Feature	Apache Pig	Traditional RDBMS
Language	Pig Latin	SQL
Schema	Flexible (Schema-on-read)	Fixed (Schema-on-write)
Storage	HDFS	Structured databases
Data Type Support	Semi/unstructured + structured	Mostly structured
Processing	Batch via MapReduce	Query-based
Performance	Optimized for big data	Not optimized for huge unstructured data

---

✅ 2. Hive Architecture and Comparison with RDBMS

🐝 What is Hive?

- Apache Hive is a **data warehouse tool** built on top of Hadoop.
- It allows you to **query large datasets** stored in HDFS using **HQL (Hive Query Language)**, similar to SQL.

🏗️ Hive Architecture Components:

1. **HiveQL**: Language used to write queries.
2. **Driver**: Manages the lifecycle of a HiveQL statement (parsing, compiling).
3. **Compiler**: Converts queries to **MapReduce** jobs.

4. **Metastore**: Stores schema and metadata (table names, columns, location in HDFS).
5. **Execution Engine**: Executes the query using Hadoop MapReduce.

🔁 Hive Query Flow (Simple Diagram):

```
HiveQL → Driver → Compiler → Execution Engine → Hadoop (MapReduce) → HDFS
                          ↘
                           Metastore
```

🆚 Hive vs RDBMS

Feature	Hive	RDBMS
Query Language	HiveQL (SQL-like)	SQL
Schema	Schema-on-read	Schema-on-write
Data	Large, unstructured/semi-structured	Structured
Speed	Slower (batch processing)	Fast (transactional)
Transaction Support	Limited	Full ACID compliance
Indexing	Basic or none	Advanced indexing
Use Case	Analytical processing on Big Data	OLTP (Online Transaction Processing)

💡 Example Query

```
-- Hive
SELECT category, COUNT(*) FROM products GROUP BY category;
```

📦 Summary:

Component	Pig	Hive
Language	Pig Latin	HiveQL

Use Case	Data transformation (ETL)	Data analysis & reporting
Processing Type	Procedural	Declarative
Developer	Yahoo!	Facebook

✅ 3. Big SQL vs. Hive (Comparison and Advantages)

🔍 What is Big SQL?

- **Big SQL** is IBM's SQL engine for Hadoop.
- It allows **SQL queries on big data** stored in **HDFS** using **standard ANSI SQL**.
- It supports **JDBC/ODBC**, integrates with **Hive, HBase, ORC, Parquet**, and can **run federated queries** across **RDBMS + Hadoop** together.

🐝 What is Apache Hive?

- Hive is a **data warehouse system** built on Hadoop.
- It enables **SQL-like queries (HQL)** on large datasets using **batch processing (MapReduce)**.
- It's ideal for **batch analysis and summarizing** big data.

🔁 Big SQL vs. Hive: Comparison Table

Feature	Big SQL (IBM)	Hive (Apache)
Developer	IBM	Apache
SQL Compliance	ANSI SQL (full)	Partial SQL (HiveQL)
Query Performance	High-speed via **in-memory execution**	Slower, batch-based via **MapReduce/Tez**
Concurrency	Supports **multiple users efficiently**	Limited concurrency

Data Format Support	Parquet, ORC, Avro, Text, JSON	Same
Integration	Supports RDBMS + Hadoop (federated queries)	Primarily Hadoop ecosystem only
Execution Engine	IBM Common SQL Engine (parallel, optimized)	MapReduce / Tez / Spark
ACID Support	Strong (transactional)	Basic (only with newer Hive versions)

✅ Advantages of Big SQL:

- High performance, **low-latency queries**.
- Can query data **across platforms** (Oracle + Hadoop).
- **Standards-based ANSI SQL**, good for existing SQL developers.
- Integrated with **security and governance** features in IBM systems.

UNIT 5

✅ 1. Difference between Supervised and Unsupervised Learning

Feature	Supervised Learning	Unsupervised Learning
Definition	Learns from **labeled data** (input + output)	Learns from **unlabeled data**
Goal	Predict outcomes or classify data	Discover patterns, clusters, or structure
Input Data	Labeled (e.g., features + target values)	Unlabeled (just features, no target values)
Examples	Email spam detection, house price prediction	Customer segmentation, topic modeling
Algorithms	Linear regression, decision tree, SVM, KNN	K-Means, Hierarchical Clustering, PCA

Evaluation	Accuracy, precision, recall, RMSE	No true “accuracy”; evaluated via **clusters/patterns**

👇 Examples:

- **Supervised**: Predicting if an email is spam (Yes/No).
- **Unsupervised**: Grouping customers based on shopping behavior.

✅ 2. Applications of Machine Learning

Machine learning powers many technologies you use daily. Here's where it’s applied:

🎯 Real-World Applications:

Domain	Application Example
Healthcare	Disease diagnosis, predicting patient readmission
Finance	Fraud detection, credit scoring
Retail	Product recommendations (like Amazon/Flipkart)
Social Media	Content recommendation (YouTube, Instagram)
Marketing	Targeted ads, customer churn prediction
Transportation	Self-driving cars, route optimization
Cybersecurity	Threat detection, anomaly detection
Education	Student performance prediction, adaptive learning tools

---

🧠 ML is used for:

- Classification
- Regression
- Clustering
- Dimensionality Reduction

- Natural Language Processing (NLP)
- Image and Video Processing

✅ 3. Big Data Analytics with BigR

🔍 What is BigR?

- **BigR** is a tool (by IBM) that integrates the **R programming language** with **Hadoop** to allow **data scientists to perform statistical analysis on large datasets stored in HDFS**.
- Combines the simplicity of **R** with the **scalability of Hadoop**.

🔧 Features of BigR:

Feature	Description
Language	Uses R syntax and functions
Data Location	Operates directly on HDFS (big data storage)
Execution Model	Converts R commands to Hadoop MapReduce jobs
Use Case	Running **R-based analytics** on massive datasets
Libraries	Supports R packages and custom scripts

🎯 Advantages of BigR:

- No need to move big data into memory (R is memory-limited).
- BigR brings **R to big data** without rewriting the code in Java/Python.
- Parallel processing = better performance.
- Ideal for **data analysts familiar with R**, but working with large Hadoop datasets.

📌 Example Use Case:

```
# BigR command (pseudocode)
bigr.load("hdfs://data/sales.csv")
bigr.aggregate(data, by = "Region", FUN = "mean")
```

🧠 Summary:

Tool	Purpose
BigR	Run R code on big data (Hadoop)
Big SQL	Run ANSI SQL on Hadoop + relational databases

IF U GOT SOME TIME….

UNIT 1

✅ 1. Diagram Showing the Evolution of Big Data

📊 Evolution of Big Data

Here's a **simple, clean diagram** that shows how Big Data evolved:

pgsql
CopyEdit

```
+---------------------+-----------------------+------------------------+
|      Era            |     Data Type         |    Technology Used     |
+---------------------+-----------------------+------------------------+
| Traditional Era     | Structured Data       | RDBMS, Spreadsheets    |
| (~1980s - 2000)     | (Tabular Format)      | SQL, Excel             |
+---------------------+-----------------------+------------------------+
| Web Era             | Semi-Structured Data  | XML, JSON, Web Logs    |
| (2000 - 2010)       | (Emails, Logs, HTML)  | NoSQL, Apache Log      |
+---------------------+-----------------------+------------------------+
| Social Media Era    | Unstructured Data     | Text, Images, Videos   |
| (2010 - 2020)       | (Social posts, Videos)| Hadoop, Spark, Cloud   |
+---------------------+-----------------------+------------------------+
| IoT / AI Era        | Real-time + All Types | Sensors, ML, Streams   |
| (2020 - present)    | (Massive-scale data)  | Big Data + AI          |
+---------------------+-----------------------+------------------------+
```

🌐 Summary:

- **Structured Data**: Tables, rows, columns → SQL
- **Semi-Structured**: JSON, XML, web logs → NoSQL
- **Unstructured**: Videos, images, audio → Hadoop/Spark
- **Real-time & AI-driven**: IoT, Sensors, ML → Streaming Analytics

---

UNIT 2

✅ 2. Explain Avro with Example

📦 What is Avro?

- **Avro** is a **data serialization system** developed in Hadoop ecosystem.
- It helps in **efficient storage and exchange of large volumes of data** across Hadoop applications.
- It is **language-neutral** and uses **JSON for schema** and **compact binary for data**.

🧠 Key Features:

- **Schema-based**: Uses **JSON schema**
- **Compact and fast**: Binary data is very lightweight
- **Dynamic typing**: Each data file stores its own schema
- **Supports rich data types**: arrays, maps, unions, etc.
- **Compatible with many languages**: Java, Python, C, etc.

🧪 Example:

✅ Avro Schema (in JSON):

```
{
  "type": "record",
```

```
  "name": "Student",
  "fields": [
    {"name": "rollNo", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "marks", "type": "float"}
  ]
}
```

✅ **Data using Avro:**

```
{
  "rollNo": 101,
  "name": "Praveen",
  "marks": 87.5
}
```

This schema + data gets converted into **compressed binary** and stored in HDFS, reducing storage and improving performance.

🔁 Why Avro?

Feature	Benefit
Self-describing	Schema is stored with the data
Compact binary format	Saves space
Interoperable	Works across languages/platforms
Integrates with	Hive, Pig, Spark, Flume, Kafka

UNIT 3

✅ 1. What is MapReduce? (with basic + diagram)

⚙️ Definition:

MapReduce is a **programming model** in Hadoop to process large datasets in a **distributed parallel** way across a Hadoop cluster.

It has two main functions:

- **Map()** – filters and sorts data (key-value pair generation)
- **Reduce()** – aggregates and summarizes data

🔁 Working Process:

1. **Input Splitting**: Input data is divided into blocks (128MB by default).
2. **Mapping**: Each block is processed by a **Mapper** – converts data into `(key, value)` pairs.
3. **Shuffling & Sorting**: Hadoop automatically groups keys and sends them to respective **Reducers**.
4. **Reducing**: Aggregation happens in the **Reducer** function.
5. **Final Output**: Result is written back to HDFS.

📈 Simple Example:

🎯 **Problem: Count word frequency in a document.**

✅ **Input:**

```
"Big data is big"
```

✅ **Map Output (key-value pairs):**

```
("Big", 1)
("data", 1)
("is", 1)
("big", 1)
```

✅ **After Shuffle & Sort:**

```
("Big", [1])
("big", [1])
("data", [1])
("is", [1])
```

✅ **Reduce Output:**

```
("Big", 1)
("big", 1)
("data", 1)
("is", 1)
```

📊 MapReduce Diagram:

```
Input Splits
     ↓
+-------------+
|   Mapper    |
+-------------+
     ↓  (Key, Value)
 Shuffle & Sort
     ↓
+-------------+
|   Reducer   |
+-------------+
     ↓
Final Output in HDFS
```

🔧 Why MapReduce?

Feature	Advantage
Scalability	Handles petabytes of data
Fault Tolerant	Retries failed tasks automatically
Parallel Processing	Faster processing via clusters
Simplified Model	Easy to write simple jobs

✅ Summary:

Topic	Key Idea
Big Data Evolution	Moved from tabular → unstructured → real-time
Avro	Schema-based serialization for Hadoop
MapReduce	Distributed processing with Map and Reduce

✅ 2. Task Execution in MapReduce

🔁 What is Task Execution?

In Hadoop's MapReduce framework, every job is broken into smaller **tasks**, and those tasks are executed across nodes in a cluster.

🧱 Breakdown of Execution:

Let's say you submitted a MapReduce job:

➤ Step-by-step Execution:

1. **Job Submission**:
   - You submit a job to the Hadoop JobTracker (or ResourceManager in YARN).
2. **Input Splits**:
   - The input file is divided into blocks (e.g., 128MB), and each block is assigned to a Mapper.
3. **Map Task Execution**:
   - Each input split is processed by a **Map task**.
   - The Map function produces intermediate `(key, value)` pairs.
4. **Intermediate Output Storage**:
   - The output is written to local disk.
   - Then it moves to the **shuffle phase**.
5. **Shuffle and Sort**:
   - Intermediate keys are sorted and grouped.
   - Data is transferred to **Reduce tasks**.
6. **Reduce Task Execution**:
   - Each group of keys is processed by a Reduce task.
   - The final output is written back to HDFS.

⚙️ Types of Tasks:

Task Type	Role
Map Task	Processes input splits into key-value pairs
Reduce Task	Aggregates and summarizes intermediate output
Speculative Task	Backup task to handle slow-running nodes

UNIT 4

✅ 1. What is Hive? Key Components of Its Architecture

🐝 What is Hive?

Apache Hive is a **data warehouse infrastructure** built on top of Hadoop for processing structured data using **SQL-like queries** (HiveQL).

Hive allows you to write queries similar to SQL, which internally get converted into MapReduce or Tez or Spark jobs.

🧱 Hive Architecture Diagram (Simplified):

```
+-------------+
|   User      |
+-------------+
      ↓
+-------------+     +---------------+
| HiveQL CLI  | ---> |   Compiler    |
+-------------+     +---------------+
                           ↓
                   +---------------+
                   | Execution Plan|
                   +---------------+
                           ↓
                   +---------------+
                   | Execution Engine |
                   +---------------+
                           ↓
                    Hadoop (MapReduce/YARN)
                           ↓
                     HDFS Storage
```

🔑 Key Components:

Component	Description
HiveQL	SQL-like language to write queries
Driver	Manages lifecycle of a query
Compiler	Converts HiveQL to execution plan (MapReduce/Spark)
Metastore	Stores metadata (table schema, locations, partitions)
Execution Engine	Executes jobs using Hadoop framework
HDFS	Stores Hive table data in Hadoop

✅ 2. What is HBase? How is it different from RDBMS?

🗃️ What is HBase?

Apache HBase is a **column-oriented NoSQL database** that runs on top of HDFS. It's designed for **real-time read/write** access to large datasets.

Think of HBase as the **Hadoop version of Google's BigTable**.

📊 HBase vs. RDBMS

Feature	HBase (NoSQL)	RDBMS (SQL-based)
Data Model	Column-based (No fixed schema)	Table-based (Rows and Columns)
Schema	Dynamic (Flexible)	Rigid (Predefined)
Query Language	No SQL (uses API/Java/REST)	SQL
Scalability	Horizontal (across machines)	Vertical (within one server)
Performance	Optimized for large, sparse datasets	Optimized for smaller structured data
Use Case	Facebook Messaging, Time series logs	Banking systems, Inventory

---

📦 HBase Key Concepts:

- **Tables** with rows identified by **row keys**
- Each row has **column families** and **columns**
- Each cell can have **multiple versions (timestamps)**

UNIT 5

✅ 1. What is Machine Learning? (Definition and Types)

🤖 Definition:

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) where **machines learn from data** without being explicitly programmed.

Instead of writing rules, we feed data to algorithms and let the model learn patterns.

🔍 Types of Machine Learning:

Type	Description	Example
Supervised Learning	Model is trained with **labeled data** (input → output pairs).	Email Spam Detection, Credit Scoring
Unsupervised Learning	Model learns from **unlabeled data** to find hidden patterns.	Customer Segmentation, Clustering
Reinforcement Learning	Agent learns by interacting with environment, using **reward & penalty**.	Self-driving Cars, Game AI
Semi-supervised Learning	Combination of a small amount of labeled data + large amount of unlabeled data.	Image tagging when only a few are labeled

---

🔁 Summary Table:

Type	Input Type	Output Known?	Example
Supervised	Labeled	Yes	Regression, Classification

Unsupervised	Unlabeled	No	Clustering, Dimensionality Reduction
Reinforcement	State/Reward	Indirect	Game-playing agents