VTU Question Paper Answer scheme & Solutions-June /July 2025

Sub:	BIG DATA ANALYTICS					Sub Code:	BCS601	Branch :	AInDS/CSDS
Date:	23/06/2025	Duration:	3hrs	Max Marks:	100	Sem	VI		OBE

		Answer any FIVE Questions	MARKS	CO	RBT
1	a	**Explain the classification of big data with a diagram**			

Big Data classification refers to the categorization of data based on different parameters such as the source, structure, and processing method.

Classification of Big Data

Big Data can be classified in three major ways:

Based on Data Format (Structure)

Type	Description
Structured	Data that is organized into rows and columns (e.g., SQL databases
Semi-Structured	Data that has some structure but not in a rigid format (e.g., XML,
Unstructured	Data with no predefined format (e.g., videos, images, social media

2. **Based on Data Source**

Type	Description
Human-Generated	Data created by people (e.g., emails, social media content, blo
Machine-Generated	Data produced by machines or sensors (e.g., logs, IoT sensors,

3. **Based on Processing Requirements**

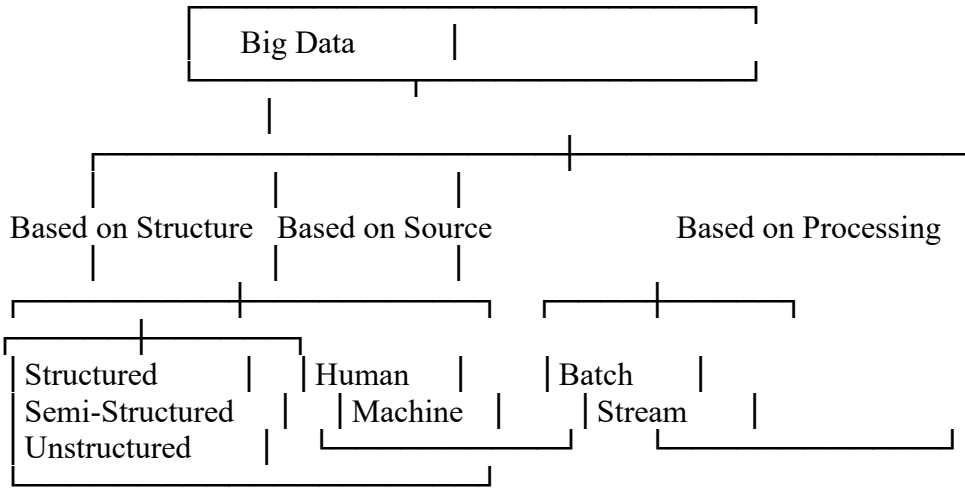
Type	Description
Batch Processing	Processing large volumes of data collected over time (e.g., Hadoop
Stream Processing	Real-time or near real-time processing of continuous data streams (e.g., Streaming, Apache Flink).

Classification of Big Data

Here's a simple conceptual diagram:

vbnet

CopyEdit



b

Define Big Data, explain its characteristics and challenges

Definition of Big Data

Big Data refers to extremely large datasets that are complex, high in volume, and generated at high speed, making traditional data processing tools inadequate to store, manage, and analyze them. It requires advanced technologies and frameworks for effective storage, processing, and analysis.

Characteristics of Big Data (The 5 V's)

Big Data is commonly described using five key characteristics, also known as the 5 Vs:

1. **Volume:** Refers to the massive amount of data generated every second.
   - Example: Facebook generates terabytes of data daily.
2. **Velocity:** Refers to the speed at which data is generated and processed.
   - Example: Real-time data from stock markets or IoT sensors.
3. **Variety:** Refers to the different types of data (structured, semi-structured, and unstructured).
   - Example: Emails, tweets, videos, logs, XML files, etc.
4. **Veracity:** Refers to the uncertainty or trustworthiness of the data.
   - Big Data can be noisy or biased, affecting data quality.
5. **Value:** Refers to the usefulness of data in decision-making.
   - The goal is to derive insights and value from raw data. Challenges of Big Data.

Big Data brings significant challenges:

1. **Data Storage**
   - Storing petabytes of data efficiently is a challenge.
   - Requires scalable storage systems like HDFS or cloud platforms.
2. **Data Processing**
   - Processing huge datasets in real-time or near real-time requires high computational power.
   - Frameworks like Apache Hadoop, Spark are used.
3. **Data Quality**
   - Ensuring accuracy, consistency, and completeness of data is difficult due to noise and redundancy.
4. **Security & Privacy**
   - Protecting sensitive data (especially user data) from breaches is critical.
   - Compliance with regulations like GDPR adds complexity.
5. **Integration**
   - Integrating data from multiple sources and formats into a single system can be complex and time-consuming.

2	a	**What is NoSQL explain different types of NoSQL with example**
		
		NoSQL stands for "Not Only SQL".
		It is a non-relational database system designed to handle large volumes of unstructured, semi-structured, or structured data. Unlike traditional relational databases (RDBMS), NoSQL databases do not use fixed schema, allow horizontal scaling, and are optimized for performance, flexibility, and scalability.
		
		Key Features of NoSQL:
		• Schema-less (flexible data models)
		• High scalability (can handle large-scale distributed systems)
		• Fast for read/write operations
		• Supports unstructured and semi-structured data
		• Suited for Big Data and real-time applications
		
		Types of NoSQL Databases with Examples
		NoSQL databases are mainly classified into four types:
		
		1. **Document-based NoSQL**
		• Stores data in documents (usually JSON, BSON, or XML formats).
		• Each document is a self-contained unit with its own structure.
		• Highly flexible and good for hierarchical data.
		
		Feature Description
		Format JSON / BSON
		Example MongoDB, CouchDB
		Use Case Content management systems, blogging platforms, catalogs
		Example (MongoDB):
		
		{
		"name": "Alice",
		"age": 25,
		"skills": ["Python", "ML"]
		}
		
		2. **Key-Value Stores**
		• Stores data as key-value pairs (like a dictionary or hash table).
		• Very fast and scalable, but limited querying capabilities.
		
		Feature Description
		Format Key: "user123", Value: "{name: John}"
		Example Redis, DynamoDB, Riak
		Use Case Caching, session management, user profiles
		Example (Redis):
		
		SET user:1234 "{name: 'John', age: 30}"
		
		3. **Column-based Stores (Wide-Column Stores)**
		• Stores data in columns instead of rows, allowing fast read/write for specific columns.			

- Ideal for analytical queries on large datasets.

Example (Cassandra):

UserID | Name | Email | Age
1001 | Alice | alice@mail.com | 24

4. **Graph Databases**

- Stores data as nodes and relationships (edges).
- Excellent for querying connected data (e.g., social networks, fraud detection).

Feature	Description
Format	Node-Edge Graph
Example	Neo4j, ArangoDB, OrientDB
Use Case	Social networks, recommendation systems, fraud detection

Example (Neo4j):

(Alice)-[:FRIEND]->(Bob)

b

Explain shared nothing architecture, CAP theorem and its advantage

1. Shared Nothing Architecture

Definition:

Shared Nothing Architecture is a distributed computing architecture where each node (server) is independent and self-sufficient, and does not share memory or disk storage with other nodes.

Each node:

- Has its own memory
- Has its own storage
- Communicates with others via a network

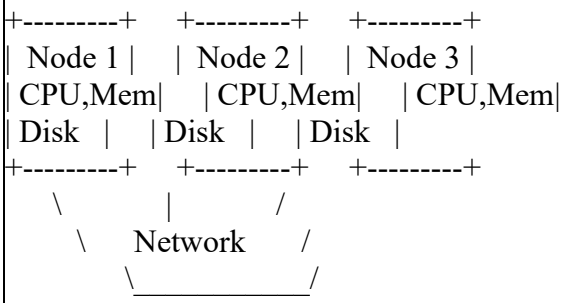
Key Features:

- No contention for resources (CPU, disk, RAM)
- Highly scalable (can add more nodes easily)
- Fault isolation (failure in one node doesn't affect others)

Examples:

- Hadoop Distributed File System (HDFS)
- Google File System (GFS)
- Cassandra, MongoDB, and Amazon DynamoDB

Diagram:



2. CAP Theorem (Brewer's Theorem)

Definition:

The CAP Theorem states that a distributed database system can satisfy only two out of the following three guarantees at the same time:

△ **C - Consistency**

- All nodes see the same data at the same time.
- Like traditional RDBMS behavior.

△ **A - Availability**

- Every request receives a non-error response, even if some nodes are down.

△ **P - Partition Tolerance**

- The system continues to operate despite network failures between nodes.

CAP Trade-offs:

Combination	Description
CA (No Partition Tolerance)	Good for centralized systems, not practical in distribut
CP (No Availability)	Prioritizes consistency during network issues, but may requests.
AP (No Consistency)	Ensures uptime, but may serve stale data.

📌 In distributed systems, Partition Tolerance (P) is a must. So the real trade-off is between Consistency (C) and Availability (A).

Examples:

System	Model	CAP Choice
MongoDB	AP	High availability, eventual consistency
HBase	CP	Strong consistency, can sacrifice availability
Cassandra	AP	High availability, tunable consistency
Google BigTable	CP	Consistency prioritized over availability

Advantages of Shared Nothing Architecture

Advantage	Explanation
✅ High Scalability	Add more nodes without performance bottlenecks.
✅ Fault Isolation	Node failures don't affect other nodes.
✅ No Resource Contention	Each node uses its own memory and disk, avoiding c
✅ Cost-Efficient	Uses commodity hardware; scales horizontally.
✅ Improved Performance	Parallel processing allows faster execution for big dat workloads.

3	a	**Differentiate between RDBMS and Hadoop** **Maper and Reducer** 1. **RDBMS vs Hadoop** Feature / RDBMS / Hadoop (HDFS + MapR Data Type / Structured data only / Structured, semi-structured, an unstructured data Schema / Fixed schema; predefined tables / Schema-less or flexible schema Storage / Centralized (single or few servers) / Distributed storage across man (HDFS) Processing / Single-node or limited cluster processing / Distributed parallel processing MapReduce Scalability / Vertical (scale up) / Horizontal (scale out by adding Fault Tolerance / Limited; mostly manual recovery / Built-in fault tolerance through replication Cost / Often expensive (licensed) / Open-source and runs on comm hardware Use Case / OLTP (transactional systems) / Big data analytics, large-scale processing Examples / MySQL, Oracle, SQL Server / Apache Hadoop, Hive, Pig 2. **Mapper vs Reducer (MapReduce Components)** Feature / Mapper / Reducer Role / Processes input data and produces key-value pairs / Aggregates and summar output Execution Time / First phase of MapReduce / Second phase after shuf Input Format / Raw data from HDFS / Output of the Mapper (k pairs) Output Format / Intermediate key-value pairs / Final result (aggregated pairs) Number of Tasks / One per input split (many mappers) / Fewer reducers (usually Example / Parsing log files, counting words / Summing word counts f mappers **Example (Word Count in MapReduce):** • Mapper Input: "ChatGPT is smart" • Mapper Output: ("ChatGPT", 1), ("is", 1), ("smart", 1) • Reducer Input: ("ChatGPT", [1, 1, 1]) • Reducer Output: ("ChatGPT", 3			

b

Identify and explain key aspects and components of Hadoop
Here's a complete and easy-to-understand explanation of the key aspects and components of Hadoop, useful for exams, interviews, and presentations.

What is Hadoop?

Hadoop is an open-source framework by the Apache Foundation used for storing and processing large volumes of data in a distributed and fault-tolerant manner across clusters of computers.

It follows a distributed computing model, allowing big data to be stored and processed efficiently using commodity hardware.

Key Aspects of Hadoop

Aspect	Description
Open-source	Freely available under the Apache license
Scalable	Can handle petabytes of data by adding more nodes
Fault-tolerant	Automatically replicates data to handle hardware failures
Distributed Storage	Data is split and stored across many nodes using HDFS
Parallel Processing	Uses MapReduce for fast and efficient processing
Cost-effective	Works on low-cost commodity hardware

Core Components of Hadoop Ecosystem

Hadoop has four main core components:

1. **HDFS (Hadoop Distributed File System)**

Purpose:

- Storage layer of Hadoop.
- Stores large data files across multiple machines in a distributed manner.

Key Concepts:

Concept	Description
Block	Files are split into fixed-size blocks (default: 128MB/256MB).
NameNode	Master node; manages metadata, file structure, and directory tree.
DataNode	Worker node; stores actual data blocks.
Replication	Data blocks are replicated across nodes (default: 3 copies)

2. **MapReduce**

Purpose: Processing layer of Hadoop.
A programming model for processing large datasets in parallel.

Phases:

Phase	Description
Map	Splits input data and processes into key-value pairs
Shuffle	Sorts and groups data by key between Map and Reduce
Reduce	Aggregates results and writes final output

3. **YARN (Yet Another Resource Negotiator)**

Purpose:

- Resource Management layer of Hadoop.
- Manages and schedules resources across the cluster.

Key Components:

Component	Role
ResourceManager (RM)	Allocates resources to applications
NodeManager (NM)	Runs on each node to monitor resources
ApplicationMaster (AM)	Manages the lifecycle of individual jobs

4. **Hadoop Common**

Purpose: Provides common utilities and libraries used by other Hadoop modules.

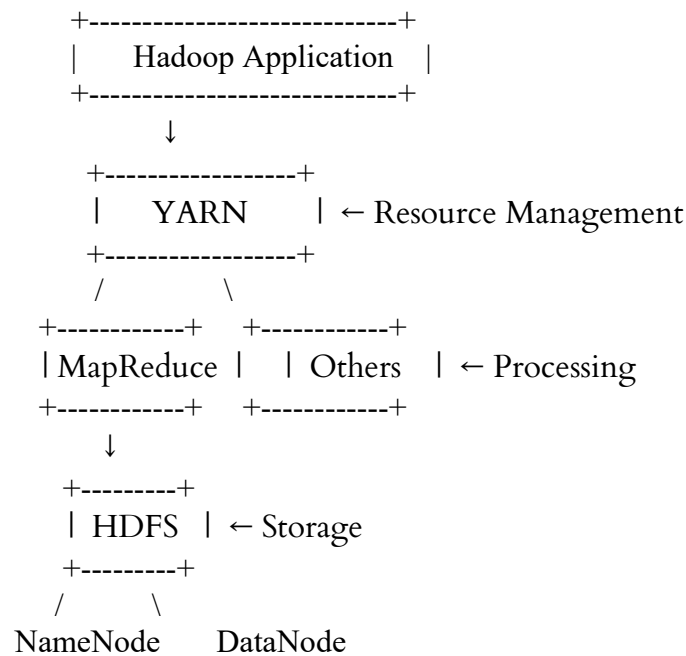
Includes:

- File system and I/O utilities
- Serialization, Java libraries, configuration files, etc.

Optional Tools in Hadoop Ecosystem

Tool	Function
Hive	SQL-like query engine for Hadoop
Pig	High-level scripting for MapReduce
HBase	Column-oriented NoSQL database on top of HDFS
Sqoop	Transfers data between Hadoop and RDBMS
Flume	Ingests streaming data into HDFS
Oozie	Workflow scheduler for Hadoop jobs

📌 **Diagram: Hadoop Architecture Overview**



4 a

Explain HDFS with a diagram

What is HDFS?

HDFS (Hadoop Distributed File System) is the storage component of the Hadoop ecosystem. It allows storage of large files across multiple machines (nodes) in a distributed, fault-tolerant manner.

HDFS is highly fault-tolerant, scalable, and suitable for applications with large datasets.

Key Features of HDFS

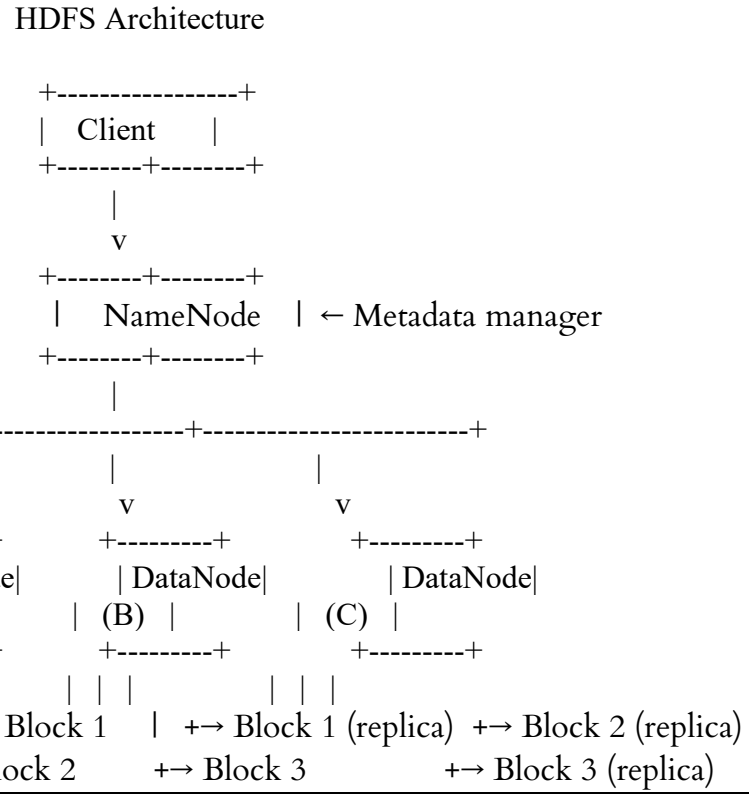
Feature	Description
Distributed Storage	Data is split and stored across multiple machines
Block-Based	Files are broken into large blocks (default size: 128 MB o
Replication	Each block is replicated across multiple nodes (default re 3)
Fault Tolerance	If one node fails, data is available from replicas
Write Once, Read Many	HDFS is optimized for high-throughput data access, not u

Components of HDFS

Component	Role
NameNode	The master node: Manages metadata, file-to-block mapping, a namespace
DataNode	The worker nodes: Store actual data blocks and serve read/wri
Secondary NameNode	Periodically takes snapshots of NameNode's metadata (for che not a backup)

Simple Diagram of HDFS Architecture

Here's a conceptual diagram of how HDFS works:



How HDFS Works (Step-by-Step)

Write Operation:

1. Client requests to write a file.
2. NameNode splits the file into blocks and assigns DataNodes to store each block.
3. Blocks are replicated to ensure fault tolerance.
4. Data is stored on DataNodes as per assignment.

Read Operation:

1. Client requests a file.
2. NameNode sends block locations (metadata).
3. Client reads blocks directly from DataNodes

Example:

- File size = 300 MB
- Block size = 128 MB
- File will be split into 3 blocks:
  - Block 1 = 128 MB
  - Block 2 = 128 MB
  - Block 3 = 44 MB
- Each block is replicated 3 times across different DataNodes.

b

What is CRUD operation operation in Mongo DB ?
Explain the following commands insert method update an array save method

What is CRUD in MongoDB?

CRUD stands for:

Operation	Description
C – Create	Insert new documents into a collection
R – Read	Query and retrieve documents
U – Update	Modify existing documents
D – Delete	Remove documents from a collection

MongoDB supports these operations via its methods and commands on collections (like tables in RDBMS).

1. insert() **Method**

Purpose: Used to add new documents to a collection.

Syntax:

```
db.collection.insert(document)
```

Example:

```
db.students.insert({
  name: "Alice",
  age: 21,
  skills: ["Python", "MongoDB"]
})
```

- Adds a new student document to the students collection.
- If _id is not provided, MongoDB automatically generates it.

You can also insert multiple documents:

```
db.students.insert([
  { name: "Bob", age: 22 },
  { name: "Carol", age: 23 }
])
```

2. **Update an Array in a Document**

MongoDB allows modifying arrays within documents using operators like $push, $addToSet, $pull, etc.

a) **Add to Array using** $push

```
db.students.update(
  { name: "Alice" },
  { $push: { skills: "Node.js" } }
)
```

- Adds "Node.js" to the skills array of Alice.

b) **Add only if not present using** $addToSet

```
db.students.update(
  { name: "Alice" },
  { $addToSet: { skills: "Python" } }
)
```

- Adds "Python" only if it doesn't already exist in the array.

c) **Remove an item from an array using** $pull

```
db.students.update(
  { name: "Alice" },
  { $pull: { skills: "MongoDB" } }
)
```

- Removes "MongoDB" from the skills array.

d) **Update element at a specific position**

```
db.students.update(
  { name: "Alice" },
  { $set: { "skills.1": "Express.js" } }
)
```

- Changes the second element (index 1) in the array to "Express.js".

3. save() **Method**

Purpose:

Inserts a new document or updates it if it already exists (based on _id).

Syntax:

```
db.collection.save(document)
```

Example:

```
db.students.save({
  _id: ObjectId("60a7..."),
  name: "Alice",
  age: 22
})
```

- If _id exists, it replaces the entire document.
- If _id doesn't exist, it inserts the new document.

5 a

What is CRUD operation in MongoDB? Explain the following commands.
I. Insert method
II. Update on array
III. Save method

CRUD stands for the four basic types of operations you can perform on a database:

- **C** – Create (Insert documents)
- **R** – Read (Find/query documents)
- **U** – Update (Modify existing documents)
- **D** – Delete (Remove documents)

MongoDB, a NoSQL database, performs these operations on **collections** of **JSON-like documents** using intuitive syntax and powerful query operators.

The `insert()` method is used to **add new documents** to a MongoDB collection.

Syntax:

```
db.collection.insert({ key1: value1, key2: value2, ... })
```

Example:

```
db.students.insert({ name: "Alice", age: 22, subjects: ["Math", "CS"] })
```

- Adds a new document to the `students` collection.
- Automatically creates the collection if it doesn't exist.
- `_id` field is added automatically if not provided.

`insertOne()` and `insertMany()` are modern alternatives to insert a single or multiple documents.

MongoDB allows updating **specific elements inside an array** in a document using `$set`, `$push`, `$addToSet`, or positional operators like `$`.

Example: Update a specific element in an array

```
db.students.update(
  { name: "Alice", "subjects": "Math" },
  { $set: { "subjects.$": "Physics" } }
)
```

- Finds the document with `name: "Alice"` and `subjects` array containing `"Math"`.
- Replaces `"Math"` with `"Physics"`.

Example: Push a new value to an array

```
db.students.update(
  { name: "Alice" },
  { $push: { subjects: "English" } }
)
```

- Adds `"English"` to the `subjects` array.

The `save()` method is used to **insert a new document** or **update an existing document** based on `_id`.

		Syntax: db.collection.save({ _id: ObjectId("..."), key1: value1, ... }) **Example:** db.students.save({ _id: ObjectId("64fba..."), name: "Alice", age: 23 }) ● If the `_id` exists, the document is **updated** (replaced). ● If the `_id` doesn't exist, a **new document is inserted**. ● `save()` is deprecated in newer MongoDB drivers. Use `insertOne()` or `replaceOne()` instead.			

Determine and explain the characteristics of MongoDB

b

MongoDB is a popular open-source **NoSQL** database designed for storing, managing, and retrieving **large volumes of unstructured or semi-structured data**. Unlike traditional relational databases, MongoDB stores data in **JSON-like documents** called **BSON (Binary JSON)**, offering flexibility, scalability, and high performance.

Key Characteristics of MongoD

1. Document-Oriented Storage Data is stored in **collections** as **documents** (similar to JSON objects).
   - Each document can have a different structure (schema-less).

Example:

```
{
 "_id": 1,
 "name": "Alice",
 "age": 24,
 "skills": ["Python", "MongoDB"]
}
```

2. Schema-Less (Flexible Schema)
   - Documents in the same collection **can have different fields**.
   - Makes it easy to evolve applications without strict table definitions.

3. High Performance
   - Fast **read/write operations**.
   - Supports **embedded documents** and **indexes**, reducing the need for expensive JOINs.

4. Horizontal Scalability (Sharding)
   - MongoDB supports **automatic sharding**, distributing data across multiple servers.
   - Useful for applications that need to handle **big data** or high traffic.

5. Powerful Query Language
   - Supports rich and expressive **queries** using:
     - `find()`, `update()`, `aggregate()`
     - Operators like `$gt`, `$lt`, `$in`, `$regex`, `$and`, `$or`, etc.

6. Indexing Support
   - Fields in documents can be **indexed** to improve search performance.
   - Supports single field, compound, geospatial, and text indexes.

7. Aggregation Framework
   - Provides a way to perform complex **data processing and transformation** in pipelines.
   - Example operations: `$group`, `$sort`, `$match`, `$project`.

		8. Replication and High Availability ● MongoDB uses **replica sets** for redundancy and fault tolerance. ● Automatically promotes a secondary to primary if the main node fails. 9. Security Features ● Supports **authentication**, **authorization**, **encryption**, and **role-based access control**. 10. Cross-Platform & Cloud-Ready ● Runs on Windows, Linux, macOS, and supports cloud platforms like **MongoDB Atlas**. ● Easily integrates with programming languages like Python, Node.js, Java, etc.			

6 a.

Write short notes on
I. Aggregate function
II. Map reduce function in MongoDB

I. Aggregate Function in MongoDB

◇ Definition:

The **aggregate function** in MongoDB is used to **process data records** and **return computed results**. It allows users to perform **data transformation**, **grouping**, **filtering**, and **summarization** using a **pipeline** approach.

◇ Key Concepts:

- The aggregation framework processes documents **in stages**, called a **pipeline**.
- Each stage transforms the data and passes it to the next stage.

◇ Common Stages:

Stage	Purpose
`$match`	Filters documents (like WHERE)
`$group`	Groups data by a field and performs operations like `sum`, `avg`, `count`
`$sort`	Sorts the documents
`$project`	Selects specific fields to return
`$limit`	Limits the number of documents

◇ Example:

```
db.orders.aggregate([
  { $match: { status: "delivered" } },
  { $group: { _id: "$customer", total: { $sum: "$amount" } } }
])
```

II. MapReduce Function in MongoDB

◇ Definition:

MapReduce is a powerful **data processing paradigm** in MongoDB that allows developers to process large volumes of data in two phases:

- **Map**: Extract and emit key-value pairs
- **Reduce**: Combine values with the same key

◆ Syntax:

```
db.collection.mapReduce(
```

```
  mapFunction,
  reduceFunction,
  {
    out: "output_collection"
  }
)
```

- **Example:**

```
var mapFunction = function() {
  emit(this.category, 1);
};

var reduceFunction = function(key, values) {
  return Array.sum(values);
};

db.products.mapReduce(mapFunction, reduceFunction, { out: "category_counts" });
```

b

Determine and explain creation of database, dropping of database

Creating and Dropping a Database in MongoDB

MongoDB handles databases dynamically — you don't need to create one explicitly before using it. You simply switch to a database using the `use` command, and it is created when you **insert the first collection or document**.

I. Creating a Database

◇ **Syntax:**

use database_name

- If the database doesn't exist, MongoDB prepares to create it.
- The database is **created only after inserting data** into it.

Example:

use studentDB

This switches to (or prepares to create) a database called `studentDB`.

Insert to Create:

db.students.insertOne({ name: "Alice", age: 21 })

Now `studentDB` is officially created with a collection named `students`.

II. Dropping a Database

To permanently **delete a database**, use the `dropDatabase()` command.

◇ **Syntax:**

db.dropDatabase()

- This command removes the **current active database** and all its collections.
- Be careful — **this action cannot be undone**.

Example:

use studentDB
db.dropDatabase()
This deletes `studentDB` along with all its data.

7 a

Define Hive. Explain its main task and features

Apache Hive is a data warehouse system built on top of Hadoop that facilitates querying, analyzing, and managing large datasets stored in Hadoop Distributed File System (HDFS) using a SQL-like language called HiveQL. Hive allows data analysts to perform SQL-style queries on big data without needing to write complex MapReduce programs.
The primary task of Hive is to:

- Convert SQL-like queries (HiveQL) into low-level MapReduce jobs (or Spark/Tez jobs)
- Allow users to perform data summarization, ad-hoc queries, and analysis on large-scale structured datasets in Hadoop.

Key Features of Hive

Feature	Description
HiveQL (SQL-like Language)	Supports familiar SQL syntax for data analysis
Schema on Read	Data schema is applied only during query time, not when data is loaded
Large-Scale Data Handling	Designed for querying petabytes of data in Hadoop
Execution Engine	Uses MapReduce (default), can also run on Apache Tez or Apache Spark
Partitioning and Bucketing	Improves query performance by organizing data into parts
Storage Integration	Works seamlessly with HDFS, Apache HBase, and other Hadoop file systems
Extensibility	Supports user-defined functions (UDFs) for custom operations
Data Warehouse Support	Ideal for data summarization, OLAP operations, and batch processing

b

Analyze Hive architecture with its diagram

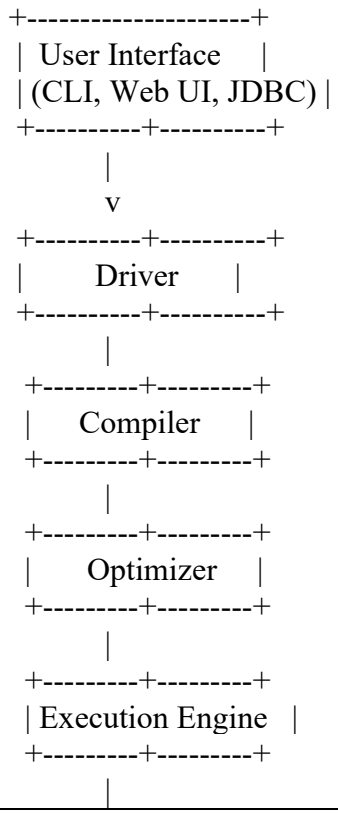
Hive Architecture

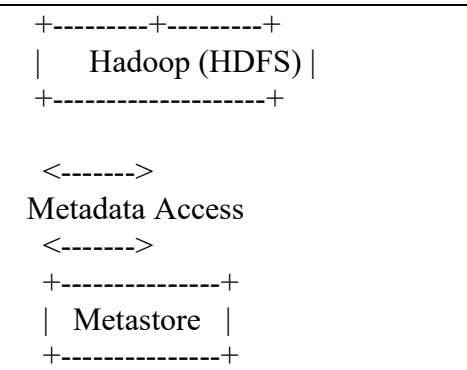
Hive is a data warehousing system built on top of Hadoop, designed to enable users to query and analyze large datasets stored in HDFS using HiveQL, a SQL-like query language. The architecture of Hive consists of various components that interact to convert HiveQL queries into executable MapReduce, Tez, or Spark jobs.

Main Components of Hive Architecture

Component	Description
1. User Interface	Allows users to interact with Hive using Hive CLI, Beeline, or JDBC/ODBC clients
2. Driver	Manages the lifecycle of a HiveQL statement: parsing, compiling, optimizing, and executing
3. Compiler	Converts HiveQL queries into DAGs of stages and ultimately into MapReduce/Spark/Tez jobs
4. Metastore	Stores metadata about tables, schemas, partitions, and data types in an RDBMS like MySQL/PostgreSQL
5. Execution Engine	Communicates with Hadoop to execute queries and fetch results
6. HDFS (Storage)	Hive stores its data in Hadoop Distributed File System (HDFS) for scalability and fault tolerance

Hive Architecture Diagram





Workflow Summary: How Hive Executes a Query

1. User submits HiveQL via UI (CLI/Beeline).
2. Driver receives query and passes it to the Compiler.
3. Compiler parses, validates, and converts HiveQL into an execution plan.
4. Optimizer improves execution using techniques like predicate pushdown.
5. Execution Engine sends jobs to Hadoop (MapReduce/Spark/Tez).
6. Hive interacts with Metastore to fetch metadata (table schemas, partitions, etc.).
7. Results are fetched from HDFS and sent back to the user.

Key Advantages of Hive Architecture

- Separates query logic from data storage
- Scales to big data volumes
- Leverages Hadoop's fault tolerance and distributed processing
- Uses metadata for fast planning and schema enforcement

8 a

Define Pig. Explain its feature, anatomy and philosophy

Apache Pig is a high-level platform built on top of Hadoop for analyzing large datasets. It uses a language called Pig Latin, which is a data flow scripting language designed to simplify the process of writing MapReduce jobs.

Features of Apache Pig

Feature	Description
High-Level Language (Pig Latin)	Simplifies the creation of MapReduce jobs using a simple script-like syntax
Handles Structured & Semi-Structured Data	Supports data from HDFS, HBase, JSON, CSV, etc.
Extensible	Allows user-defined functions (UDFs) in Java, Python, etc.
Automatic Optimization	Optimizes execution plans internally before job submission
Platform Independent	Can run in local mode (single machine) or distributed mode (Hadoop cluster)
Schema Flexibility	Works even with partial or unknown schema (optional typing)
Interoperable with Hadoop Ecosystem	Works well with Hive, HDFS, and HCatalog

Anatomy of Apache Pig

The anatomy of Pig refers to the components involved in writing, compiling, and executing a Pig script.

◇ 1. Pig Latin Script

- Written by the user to define the data flow (load → transform → store)

◇ 2. Parser

- Parses the script to check for syntax errors
- Produces a logical plan

◇ 3. Logical Optimizer

- Performs rule-based optimizations (e.g., projection pushdown)

◇ 4. Physical Plan Generator

- Converts the logical plan into a physical plan (series of physical operators)

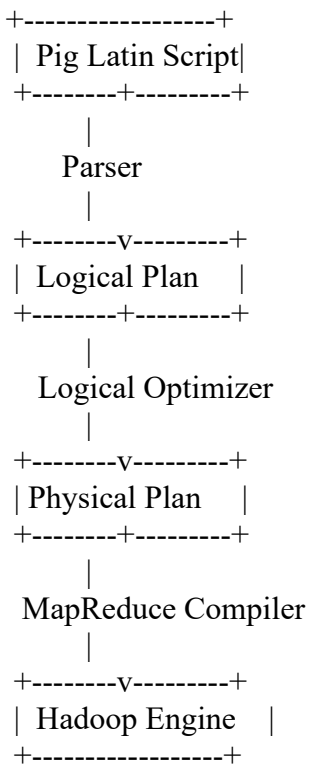
◇ 5. MapReduce Plan Generator

- Translates physical plan into MapReduce jobs

◇ 6. Execution Engine

- Submits the job(s) to Hadoop MapReduce
- Collects results and returns output

Pig Architecture Diagram (Simplified)



Apache Pig is designed with a data flow and procedural philosophy, contrasting with SQL's declarative nature.

◇ 1. Think Like a Pipeline

- Pig processes data as a sequence of steps, similar to a pipeline.
- Each step applies transformations to the data (e.g., filter, group, join).

◇ 2. Script-Driven Programming

- Pig favors writing scripts that describe how data should be transformed, rather than just what output is needed (as in SQL).

◇ 3. Handles Complex Workflows

- Designed to manage ETL tasks, data preparation, and batch processing, making it powerful for big data transformation.

◇ 4. Developer-Focused: Built for programmers, not analysts; integrates easily with code and custom logic via UDFs.

b

Examine identifiers, keywords, data types and operators of PIG with an example

1. Identifiers in Pig

Identifiers are the names used for relations, fields, aliases, and variables in Pig Latin scripts.

- Must start with a letter or underscore (_)
- Can contain letters, digits, and underscores
- Case-sensitive

Example:

```
data = LOAD 'students.csv' USING PigStorage(',') AS (name:chararray, age:int);
```

- `data` and `name` are identifiers

2. Keywords in Pig

Keywords are reserved words that have a special meaning in Pig Latin. They cannot be used as identifiers.

Common Keywords:

LOAD, STORE, FILTER, GROUP, FOREACH, GENERATE, ORDER, LIMIT, BY, USING, AS

Example:

```
students = LOAD 'students.csv' USING PigStorage(',') AS (name:chararray, marks:int);
passed = FILTER students BY marks > 35;
```

- `LOAD`, `FILTER`, `BY`, `AS` are keywords

Data Types in Pig

Pig supports both primitive and complex data types.

Primitive Data Types:

Type	Description	Example
`int`	32-bit integer	10
`long`	64-bit integer	100000L
`float`	32-bit float	3.14f
`double`	64-bit float	3.14159
`chararray`	String of characters	"Alice"

`bytearray`	Raw binary data	-
`boolean`	True/False	true

Complex Data Types:

Type	Description	Example
`Tuple`	Ordered set of fields	`(10, "Alice")`
`Bag`	Collection of tuples (like a table)	`{(10, "Alice"), (20, "Bob")}`
`Map`	Key-value pairs	`[name#'Alice', age#22]`

4. Operators in Pig

Operators perform actions like loading, filtering, joining, grouping, etc.

📌 Types of Operators:

Category	Operators & Usage
Relational	`LOAD`, `STORE`, `FILTER`, `GROUP`, `JOIN`, `UNION`
Arithmetic	`+`, `-`, `*`, `/`, `%`
Comparison	`==`, `!=`, `<`, `>`, `<=`, `>=`
Logical	`AND`, `OR`, `NOT`
Casting	`(type)` → `(int)myField`

Example Script: Pig Latin Overview

```
-- Load the dataset
students = LOAD 'students.csv' USING PigStorage(',')
        AS (name:chararray, age:int, marks:int);

-- Filter students who passed
passed_students = FILTER students BY marks >= 40;

-- Group by age
grouped = GROUP passed_students BY age;

-- Calculate average marks
result = FOREACH grouped GENERATE group AS age,
AVG(passed_students.marks) AS avg_marks;
```

		-- Store the result STORE result INTO 'output' USING PigStorage(','); Analysis: ● `students`, `passed_students`, `grouped`, and `result` are identifiers. ● `LOAD`, `FILTER`, `GROUP`, `FOREACH`, `STORE` are keywords. ● `chararray`, `int` are data types. ● `>=`, `AVG()` are operators.			

9 a

Analyze spark SQL and Pandas for data analysis

Analyzing Spark SQL vs Pandas for Data Analysis

Both Spark SQL and Pandas are popular tools used for data manipulation and analysis, but they are designed for different use cases and data scales.

Tool	Description
Pandas	A Python library for data analysis and manipulation in-memory, optimized for small to medium datasets.
Spark SQL	A module in Apache Spark that allows querying structured data using SQL and DataFrame APIs, optimized for big data and distributed computing.

2. Core Comparison Table

Feature	Pandas	Spark SQL
Data Size	In-memory (best under ~1-2 GB)	Handles terabytes to petabytes of data
Performance	Fast for small data; single machine	Distributed & parallel; scales with cluster
Language	Python	SQL, Scala, Python, Java, R
Execution Engine	Single-core/multi-core (limited by RAM)	Cluster-based distributed engine (Spark Core)
Syntax (Query)	Pythonic (e.g., `df[df['col'] > 5]`)	SQL-style or DataFrame API (`df.filter()`)
Speed (Large Data)	Slow or crashes	Optimized with Catalyst and Tungsten engines
Ease of Use	Simple for beginners	Requires knowledge of Spark setup
Join & Grouping	Slower on large joins	Optimized join strategies in Spark SQL
I/O Support	CSV, Excel, SQL, JSON, etc.	HDFS, Hive, Parquet, ORC, Avro, JDBC, etc.
Deployment	Local systems, notebooks	Big data clusters, cloud platforms
Missing Value Handling	Excellent built-in support	Supported with `fillna`, `dropna`, etc.

3. Code Comparison Example

Pandas:

```
import pandas as pd

df = pd.read_csv('sales.csv')
filtered = df[df['region'] == 'East']
summary = filtered.groupby('category')['sales'].sum()
print(summary)
```

Spark SQL:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("SalesAnalysis").getOrCreate()
df = spark.read.csv("sales.csv", header=True, inferSchema=True)
df.createOrReplaceTempView("sales")

result = spark.sql("SELECT category, SUM(sales) FROM sales WHERE region =
'East' GROUP BY category")
result.show()
```

4. When to Use What?

Use Case	Choose
Small to medium datasets (< 1GB)	Pandas
Exploratory Data Analysis (EDA)	Pandas
Running on personal computer or notebook	Pandas
Massive datasets (GBs to TBs+)	Spark SQL
Distributed computing (cluster/cloud)	Spark SQL
Need SQL-like querying on large data	Spark SQL
Real-time or big data pipeline	Spark SQL

b

Determine and explain component, features of spark architecture.

Apache Spark is an open-source, distributed computing framework designed for fast processing of large-scale data. It supports batch processing, streaming, SQL, machine learning, and graph processing — all in a unified engine.

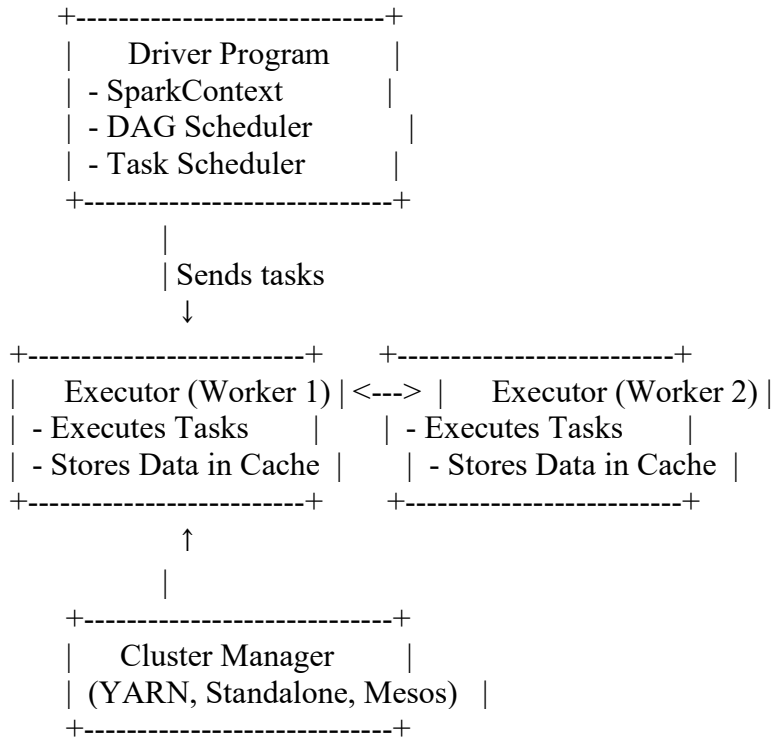
1. Components of Spark Architecture

Spark follows a master-slave architecture with a central driver program and distributed executors running on a cluster.

Key Components:

Component	Description
Driver Program	Main controller that runs the SparkContext, coordinates tasks, and keeps track of job progress.
Cluster Manager	Allocates resources (CPU & memory) to Spark applications. Types: Standalone, YARN, Mesos, Kubernetes.
Workers (Executors)	Run on cluster nodes. Execute tasks and return results to the driver. Each executor also caches data for reuse.
Tasks	Smallest unit of work sent to executors. A job is broken into multiple tasks.
RDD/DataFrame/Dataset	Abstractions to represent data for processing. RDD = Resilient Distributed Dataset.
Job	A complete computation triggered by an action (e.g., `.collect()` or `.save()`).
Stage	A group of tasks that can be executed in parallel. Spark divides jobs into stages using DAG.
DAG (Directed Acyclic Graph)	Represents logical execution flow. Spark optimizes and schedules tasks using DAG

Spark Architecture Diagram



2. Key Features of Apache Spark

Feature	Description
In-Memory Processing	Stores intermediate data in memory (RAM), making it 100x faster than Hadoop MapReduce.
Unified Framework	Supports batch, streaming, SQL, MLlib (Machine Learning), and GraphX (graph processing).
Fault Tolerant	Uses RDD lineage to recompute lost data automatically.
Lazy Evaluation	Operations are not executed until an action is called, which optimizes job execution.
DAG Execution Engine	Replaces MapReduce by representing computation as a DAG for better task optimization.
Language Support	Supports Scala, Java, Python (PySpark), and R.
MLlib Integration	Built-in library for scalable machine learning algorithms.
Spark Streaming	Enables real-time data processing from sources like Kafka, Flume, or socket.
SQL Support	Spark SQL allows querying structured data using familiar SQL syntax or DataFrame API

		.			

Analyze different phase of text mining process with a diagram.

Text Mining, also known as Text Data Mining or Text Analytics, is the process of extracting useful insights, patterns, and knowledge from unstructured text data using techniques from NLP (Natural Language Processing), Machine Learning, and Data Mining.

The text mining process consists of sequential and interdependent phases, each designed to transform raw text into structured, analyzable insights.

1. Text Preprocessing (Cleaning)

Goal: Prepare raw text for analysis by removing noise.

Includes:

- Tokenization – Splitting text into words or phrases
- Stop-word removal – Removing common words (e.g., "the", "is")
- Lowercasing – Converting all text to lowercase
- Stemming / Lemmatization – Reducing words to their root forms
- Noise removal – Removing punctuation, numbers, HTML tags, etc.

2. Text Transformation (Feature Generation)

Goal: Convert cleaned text into structured format.

Techniques:

10 a

- **Vectorization – Convert text to numeric format using:**
  - **Bag of Words (BoW)**
  - **TF-IDF (Term Frequency-Inverse Document Frequency)**
  - **Word Embeddings (e.g., Word2Vec, GloVe)**

3. Feature Selection / Dimensionality Reduction

Goal: Reduce feature space and improve performance.

- **Chi-square**
- **Information Gain**
- **PCA (Principal Component Analysis)**
- **LDA (Latent Dirichlet Allocation)**

4. Text Mining / Analysis

Goal: Extract patterns or build predictive models.

- **Classification (e.g., spam detection)**
- **Clustering (e.g., document grouping)**
- **Sentiment Analysis**
- **Topic Modeling**
- **Named Entity Recognition (NER)**

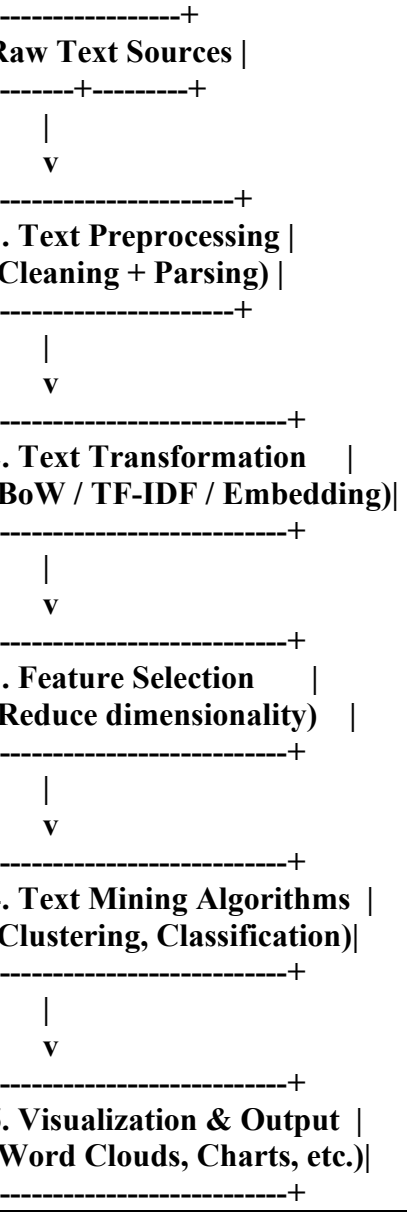
5. Interpretation & Visualization

Goal: Make results understandable and useful.

- **Word clouds**
- **Bar plots of term frequency**
- **Topic graphs**

- **Heatmaps for document-term matrices**

Text Mining Process Diagram



Identify and explain preprocessing steps mining task for content.

Preprocessing is the first and most crucial step in any content/text mining task. It involves transforming raw, unstructured text into a clean, structured, and analyzable format.

Without preprocessing, the text is too noisy and inconsistent for meaningful analysis.

Key Preprocessing Steps in Text Mining

◇ **1. Text Cleaning**

Removes unwanted characters, formatting, and non-textual content.

- **Remove punctuation (`!`, `,`, `.`)**
- **Remove special characters (`@`, `#`, `$`)**
- **Remove HTML tags (e.g., `<div>`)**
- **Remove numbers if irrelevant**

◆ 2. Tokenization

Breaks the text into individual units called tokens (usually words or phrases).

Example:
"Data mining is useful." → ["Data", "mining", "is", "useful"]

b ◆ 3. Lowercasing

Converts all characters to lowercase to avoid treating the same word differently.

Example:
"Machine" and "machine" → "machine"

◆ 4. Stop-word Removal

Removes commonly used words that do not carry much meaning in analysis.

Examples of stop words:

```
["the", "is", "at", "which", "on", "and"]
```

◆ 5. Stemming or Lemmatization

Reduces words to their root form.

Stemming	Lemmatization
"running" → "run"	"running" → "run"
"better" → "better"	"better" → "good"

- **Stemming cuts off prefixes/suffixes.**
- **Lemmatization uses vocabulary & grammar rules.**

◆ 6. Part-of-Speech (POS) Tagging *(optional but useful)*

Tags each word with its grammatical role (noun, verb, adjective, etc.).

Example:
"Data mining is interesting" → [("Data", NN), ("mining", VBG), ("interesting", JJ)]

◆ 7. Named Entity Recognition (NER)

Identifies names of people, places, organizations, dates, etc.

Example:
"Apple Inc. was founded by Steve Jobs" → [ORG: Apple Inc., PERSON: Steve Jobs]

◆ 8. Text Vectorization

Converts cleaned text into numeric format for ML algorithms.

- **Bag of Words (BoW)**
- **TF-IDF (Term Frequency-Inverse Document Frequency)**
- **Word Embeddings (Word2Vec, GloVe)**

◆ 9. Handling Negation *(contextual)*

Detects negative expressions like "not good", "wasn't happy" which invert sentiment. ◆ 10. Spelling Correction / Normalization *(optional)*

Corrects typos and standardizes slang or abbreviations.

Example:
"u r gr8" → "you are great"

Preprocessing Pipeline Summary

Raw Text
↓
Clean Text (remove noise)
↓
Tokenization
↓
Lowercasing
↓
Stopword Removal
↓
Stemming / Lemmatization
↓
Vectorization (BoW, TF-IDF, Word2Vec)
↓
→ Ready for Mining Task (Classification, Clustering, Sentiment Analysis, etc.)