****

**Big Data Analytics**



Date: 17-12-2025

**Submitted To: Yogesh Sir Submitted by: Kanishka Anand (24122016)**

**Research questions**

**Que.1** Compare Traditional Data Processing vs Big Data Processing with 3 real-world

Examples?

Answer. **Comparison with 3 Real-World Examples**

|  |  |  |
| --- | --- | --- |
| Aspect | Traditional Data Processing | Big Data Processing |
| Data size | GBs | TBs–PBs |
| Data type | Structured (tables) | Structured, Semi-structured, Unstructured |
| Processing | Batch | Batch + Real-time |
| Storage | RDBMS | HDFS, Cloud storage |
| Scalability | Vertical (limited) | Horizontal (highly scalable) |

**Real-World Examples**

**Example 1: Banking**

* Traditional: Daily transaction reports using SQL databases
* Big Data: Fraud detection in real time using Spark Streaming

**Example 2: Retail**

* Traditional: Monthly sales analysis in Excel
* Big Data: Amazon analyzes clickstream + purchase behavior instantly

**Example 3: Healthcare**

* Traditional: Patient records stored in hospital DB
* Big Data: Wearable + medical image analysis using distributed systems

**Que.2** Research and explain the 5 V’s of Big Data using industry case studies.

|  |  |  |
| --- | --- | --- |
| V | Meaning | Industry Example |
| Volume | Huge data size | Facebook processes PBs of data daily |
| Velocity | Speed of data generation | Stock market live feeds |
| Variety | Different data formats | Text, video, images on YouTube |
| Veracity | Data quality & accuracy | Removing fake reviews on Amazon |
| Value | Useful insights | Netflix recommendations |

**Case Study – Netflix**  
Netflix analyzes massive user viewing data (Volume), streaming behavior (Velocity), video + logs (Variety) to improve recommendations (Value).

**Que.3** Study any 2 Indian companies using Big Data and document their architecture.

**1. Reliance Jio**

**Use Cases**

* Network optimization
* Customer behavior analysis
* Personalized offers

**Architecture**

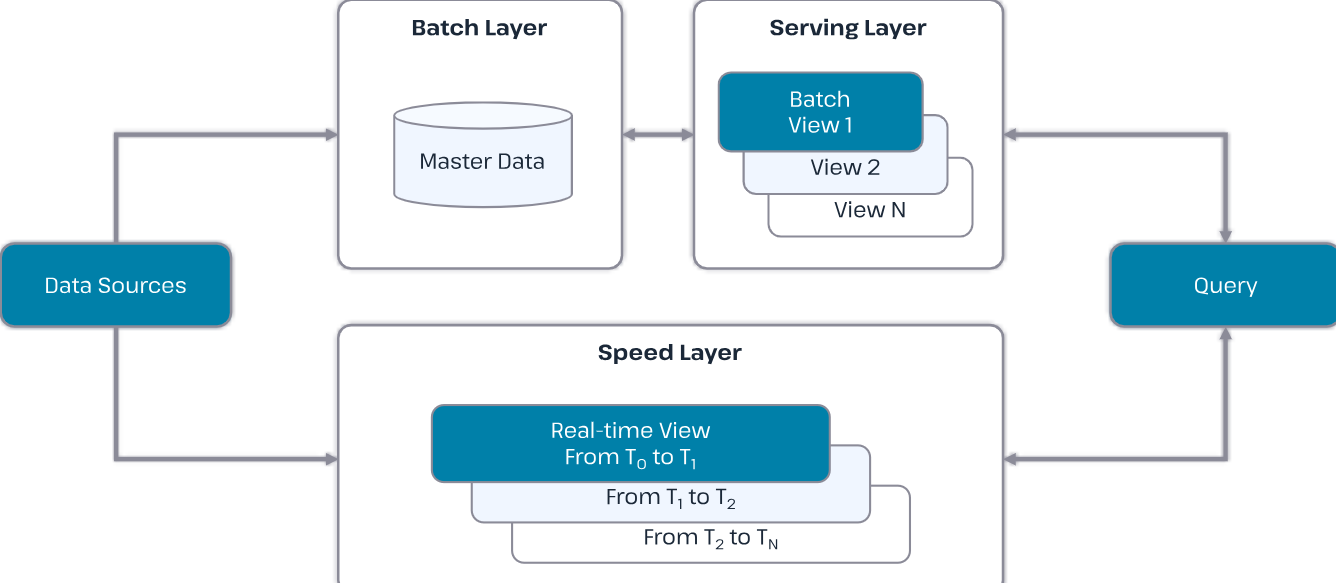
**2. Flipkart**

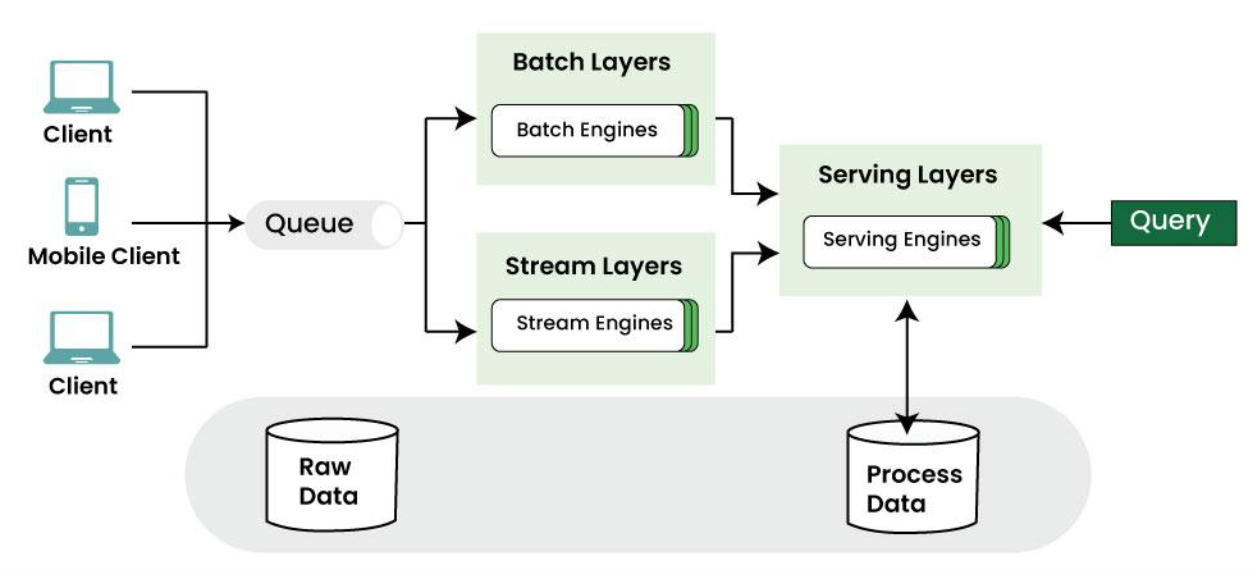
**Use Cases**

* **Dynamic pricing**
* **Recommendation engine**
* **Fraud detection**

**Que.4** Explain Lambda Architecture vs Kappa Architecture with diagrams.

**Lambda Architecture**





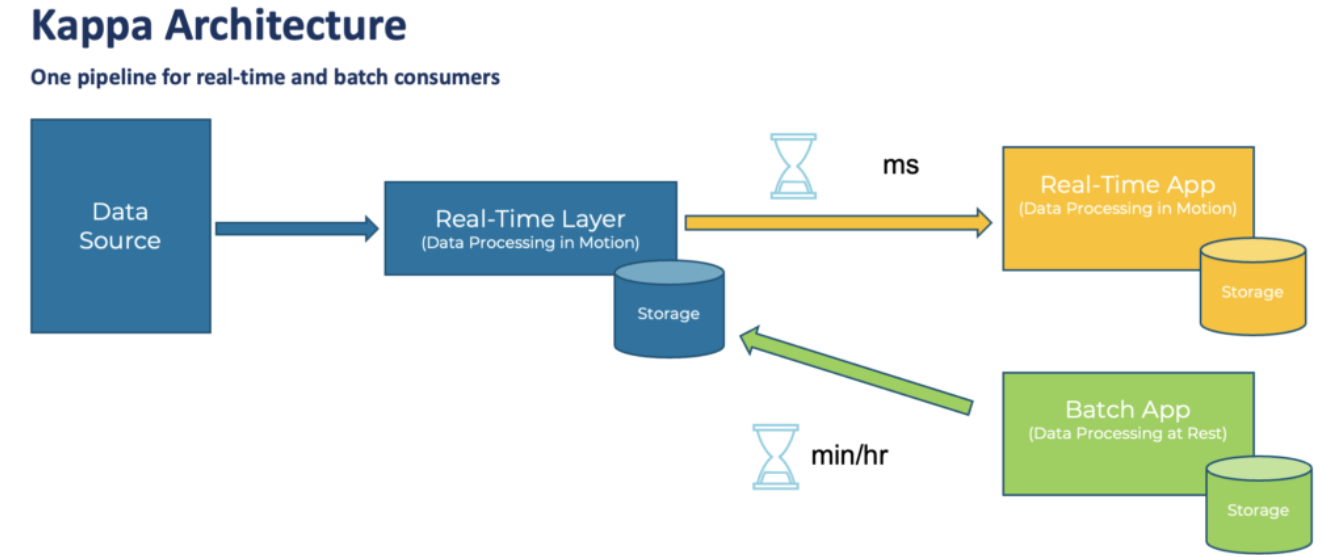
**Components**

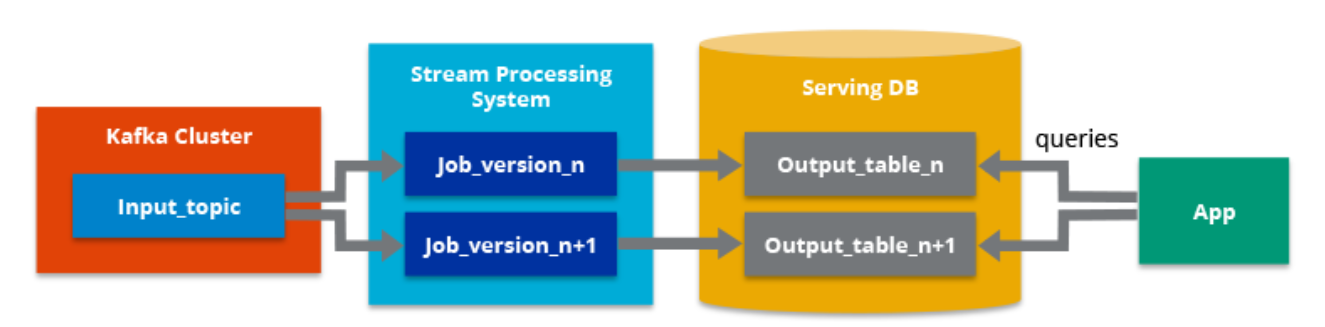
* Batch Layer (Hadoop)
* Speed Layer (Spark Streaming)
* Serving Layer

**Pros**Accurate  
Fault-tolerant

**Cons**Complex  
Duplicate code

**Kappa Architecture**

****

****

**Components**

Single streaming pipeline (Kafka + Spark)

**Pros**Simple  
Less maintenance

**Cons**Reprocessing historical data is difficult

|  |  |  |
| --- | --- | --- |
| Feature | Lambda | Kappa |
| Layers | 3 | 1 |
| Complexity | High | Low |
| Real-time focus | Medium | High |

**Que.5** Compare Hadoop vs Spark for real-time analytics.

|  |  |  |
| --- | --- | --- |
| Feature | Hadoop | Spark |
| Processing | **Disk-based** | **In-memory** |
| Speed | **Slower** | **Faster (10–100x)** |
| Real-time | **Not suitable** | **Excellent** |
| Use Case | **Batch jobs** | **Streaming + ML** |

**Example**

* **Hadoop: Log analysis overnight**
* **Spark: Live recommendation updates**

**Que.6** Study the role of Big Data in AI & Machine Learning.

Big Data provides:

* **Training data** for ML models
* **Feature extraction**
* **Model accuracy improvement**

**Examples**

* Google Maps traffic prediction
* Amazon product recommendations
* Face recognition systems

**Que.7** Explain data lake vs data warehouse with tools used in each.

|  |  |  |
| --- | --- | --- |
| Aspect | Data Lake | Data Warehouse |
| Data type | Raw (all formats) | Structured |
| Schema | Schema-on-read | Schema-on-write |
| Cost | Low | High |
| Users | Data Scientists | Business Analysts |

**Tools Used**

* **Data Lake**: HDFS, AWS S3, Azure Data Lake
* **Data Warehouse**: Snowflake, Redshift, BigQuery

**Que**.8 Research how Netflix/YouTube/Zomato uses Big Data.

**Netflix**

* Recommendation engine
* Viewer retention analysis
* Streaming quality optimization

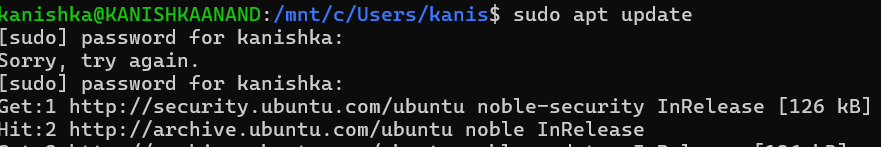
**YouTube**

* Video ranking
* Ad targeting
* Content moderation

**Zomato**

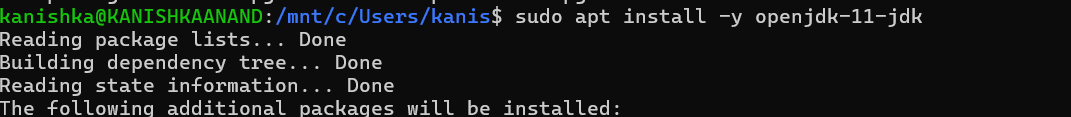
* Demand forecasting
* Restaurant recommendations
* Delivery route optimization

**HADOOP & HDFS HANDS-ON**

**1. Install Hadoop in Standalone or Pseudo-Distributed Mode (with proof).**

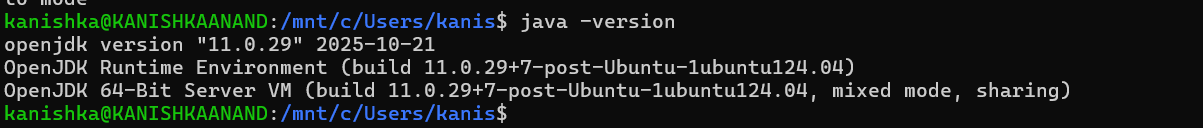
updating the Ubuntu package list using the **sudo apt update** command. This step ensures the system repositories are up to date before installing Java and Hadoop dependencies.

2. **Install Hadoop (Java installation)**



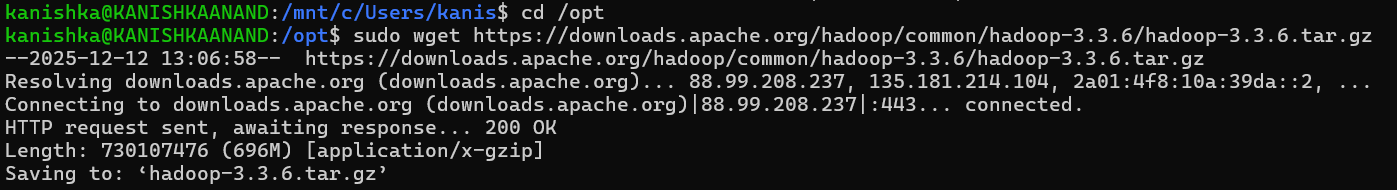
the installation of OpenJDK 11, which is a mandatory prerequisite for running Hadoop. Java is required for Hadoop daemons such as NameNode and DataNode.

**3. Install Hadoop (Verification)**



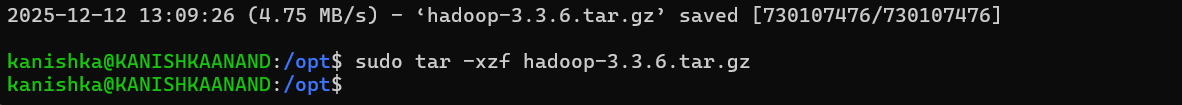
verifies the successful installation of Java by displaying the installed OpenJDK version using the java -version command.

**4. Install Hadoop (Download)**



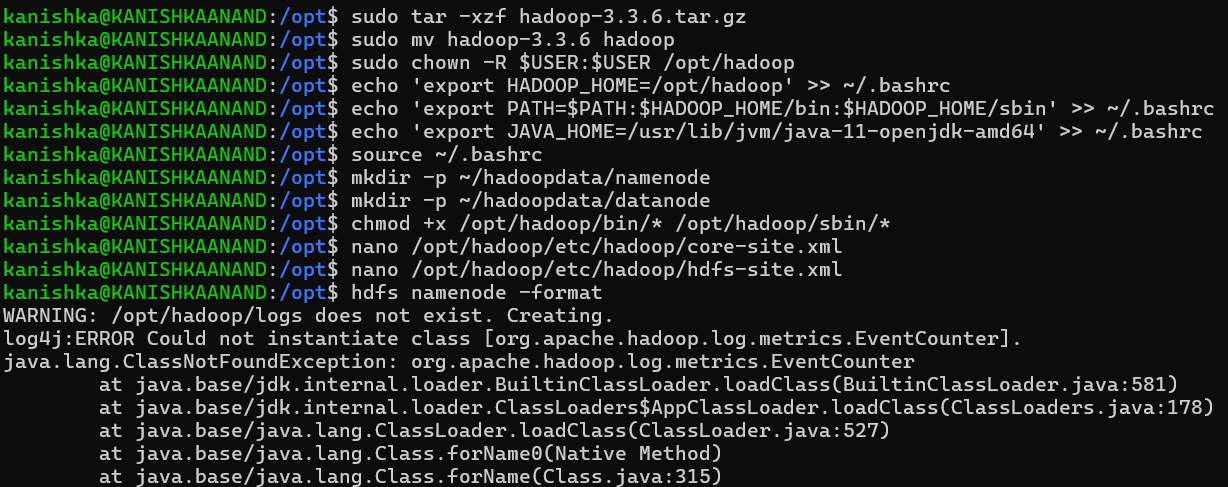
downloading Hadoop version 3.3.6 from the Apache official website using the wget command, confirming successful retrieval of the Hadoop binary archive.

**5. Install Hadoop (Configuration)**



shows extraction of the Hadoop tar file using the tar -xzf command, which prepares Hadoop binaries for configuration and execution.

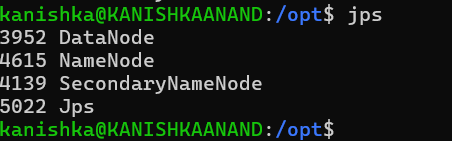
**6. Hadoop environment configuration**



setting up Hadoop environment variables such as HADOOP\_HOME, JAVA\_HOME, and updating the system PATH. These configurations enable Hadoop commands to run globally.

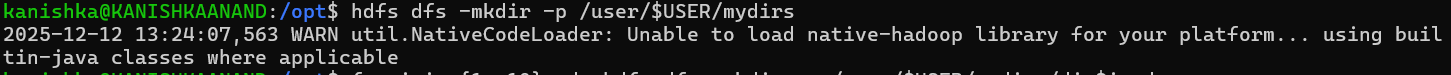


**7. Hadoop installation verification**



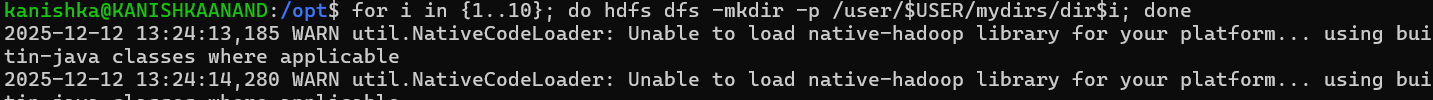
shows the output of the jps command displaying running Hadoop daemons such as **NameNode**, **DataNode**, and **SecondaryNameNode**, confirming successful Hadoop setup in pseudo-distributed mode.

**8. Create directories in HDFS**



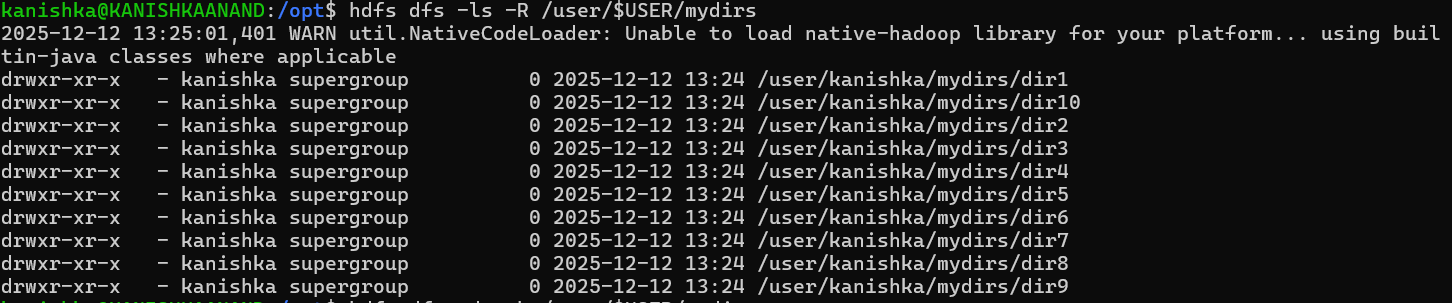
the creation of a directory in HDFS using the hdfs dfs -mkdir command. This verifies successful interaction with the Hadoop Distributed File System.

**9. Create 10 directories using loop**



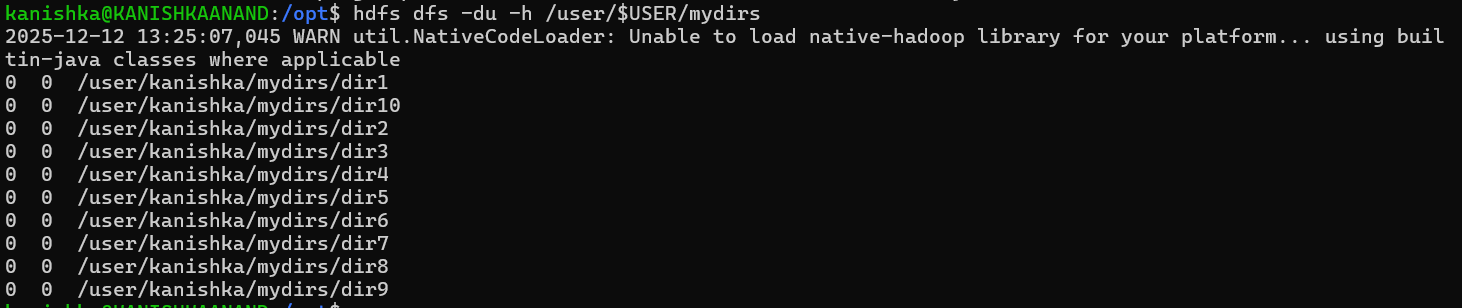
shows automated creation of 10 directories inside HDFS using a shell loop, demonstrating efficient directory management within HDFS.

**10. Recursive listing of directories**



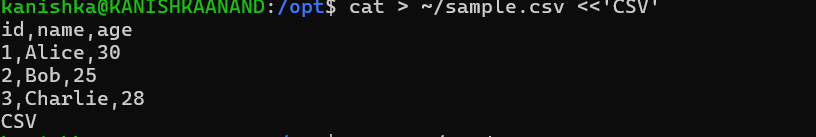
the recursive listing of all created directories inside HDFS, confirming successful creation and proper directory structure.

**11. HDFS File System Commands (du)**



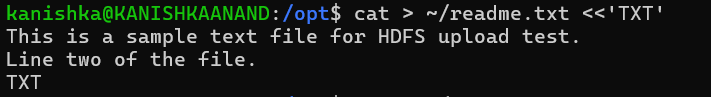
shows the use of the hdfs dfs -du -h command to display disk usage of directories inside HDFS. It confirms that the directories were created successfully and currently occupy zero space.

**12. Upload CSV file to HDFS (File preparation)**



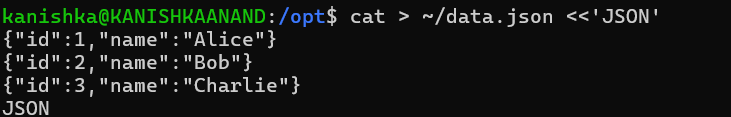
shows the creation of a CSV file (sample.csv) using the cat command. The file contains structured tabular data, which will later be uploaded to HDFS.

13. **Upload TXT file to HDFS (File preparation)**



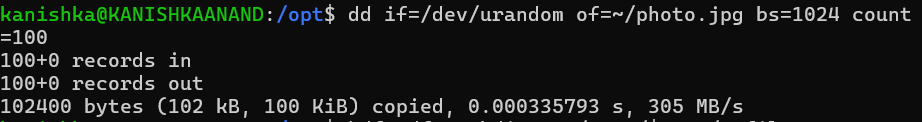
shows the creation of a text file (readme.txt) containing sample text data. This file is prepared for uploading into HDFS as part of different file format handling.

**14. Upload JSON file to HDFS (File preparation)**



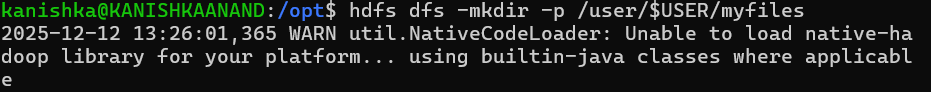
shows the creation of a JSON file (data.json) containing key-value pairs. This demonstrates handling of semi-structured data formats before uploading to HDFS.

**15. Create image file using dd**



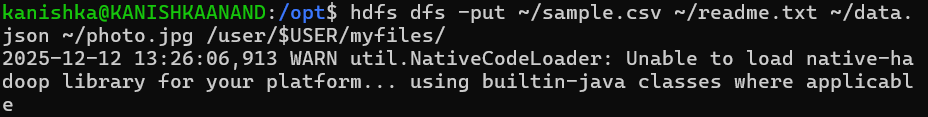
shows creation of an image file (photo.jpg) using the dd command with random data. This file simulates a binary image file for HDFS upload testing.

**16. Upload files to HDFS (Directory creation)**



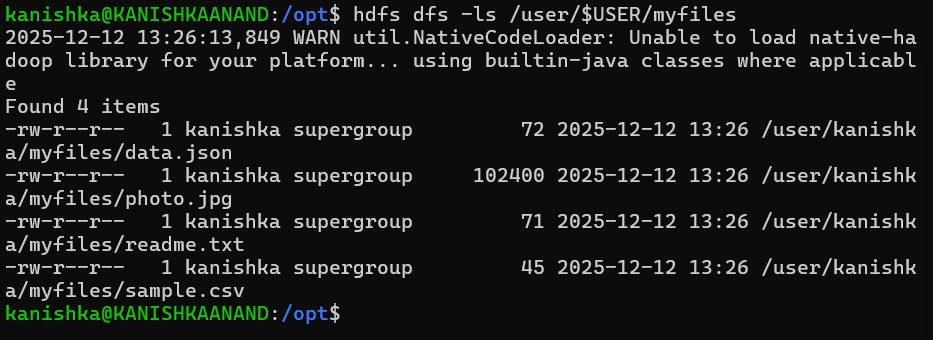
shows the creation of an HDFS directory (/user/$USER/myfiles) to store different file types including CSV, TXT, JSON, and image files.

**17. Upload multiple files to HDFS**



confirms successful upload of multiple file formats (CSV, TXT, JSON, and image) into HDFS using a single hdfs dfs -put command.

**18. List uploaded files in HDFS**



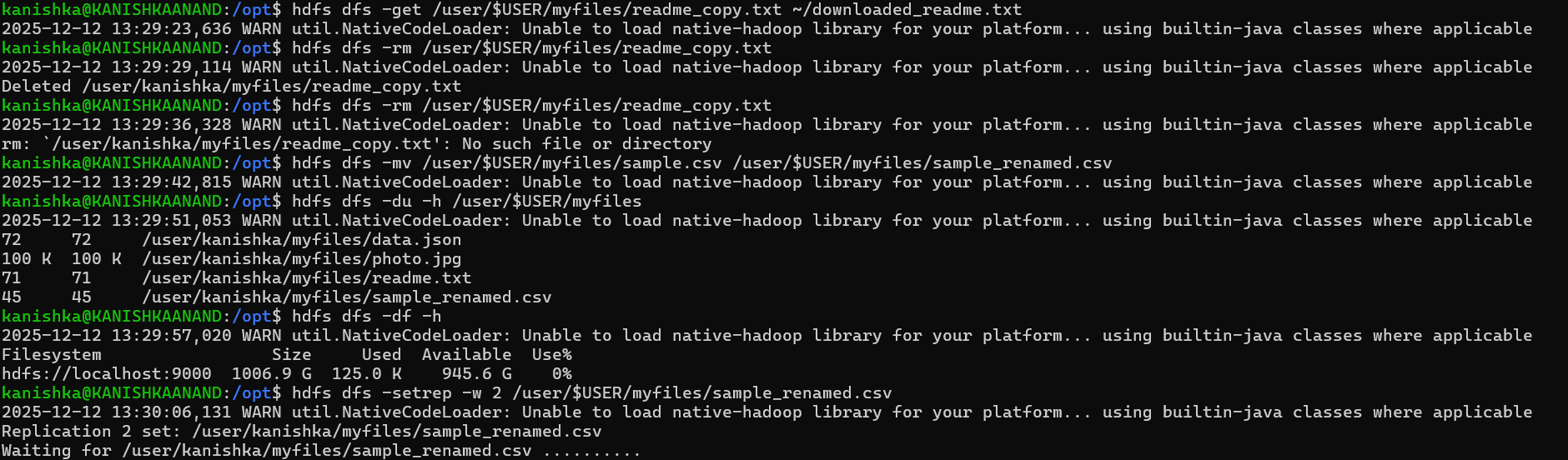
shows the listing of all uploaded files inside the HDFS directory /user/$USER/myfiles, verifying successful file upload and storage.

**19. HDFS File Operations (put)**



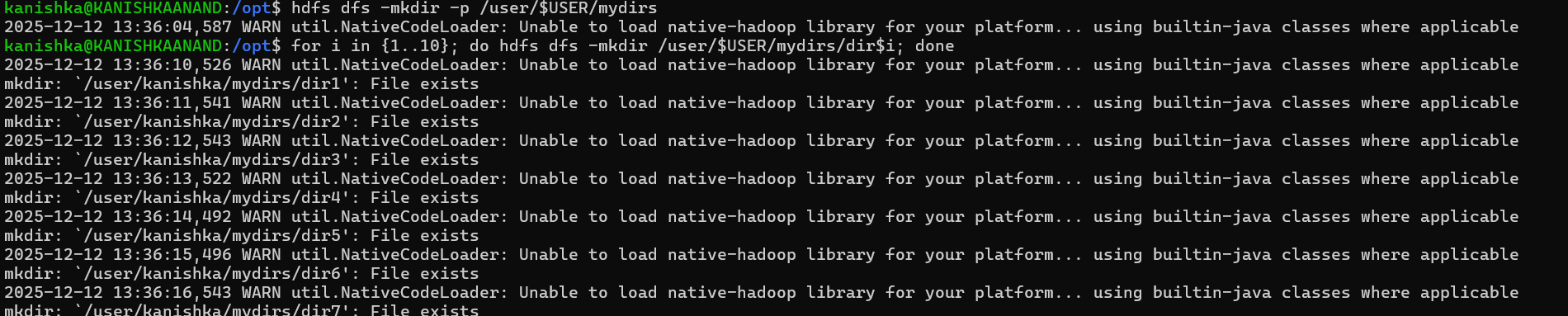
shows copying a local file into HDFS using the hdfs dfs -put command, creating a duplicate file inside the HDFS directory for further operations.

**20. HDFS File Operations (get)**



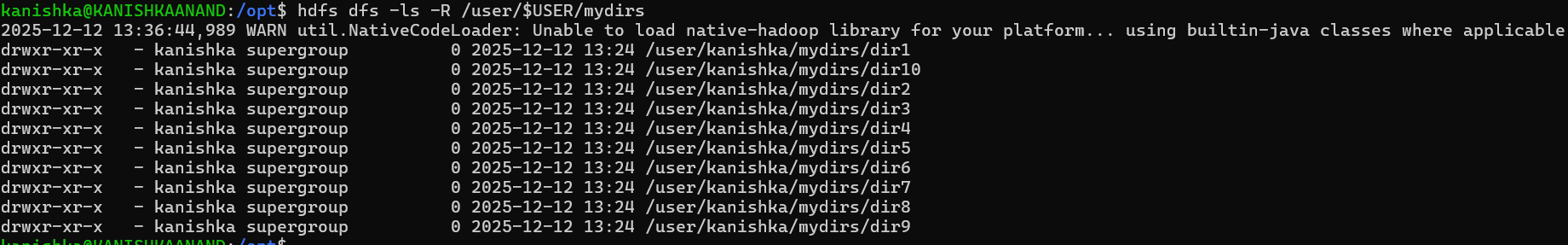
demonstrates downloading a file from HDFS to the local file system using the hdfs dfs -get command, confirming data retrieval from HDFS.

**21. HDFS File Operations (rm)**



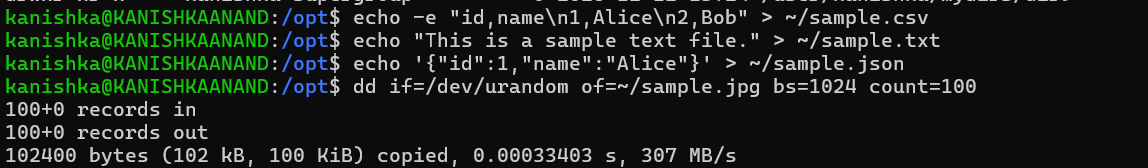
shows deletion of a file from HDFS using the hdfs dfs -rm command, confirming successful file removal.

**22. HDFS File Operations (move/rename)**



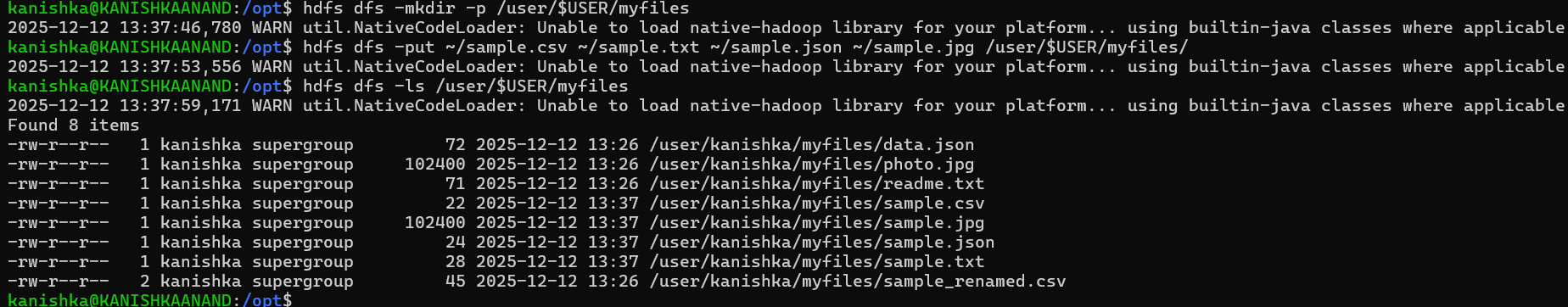
demonstrates renaming a file inside HDFS using the hdfs dfs -mv command, verifying file movement and renaming functionality.

**23. Create sample files using echo**



shows creating CSV, TXT, and JSON files using the echo command, demonstrating an alternative method for generating input files before HDFS upload.

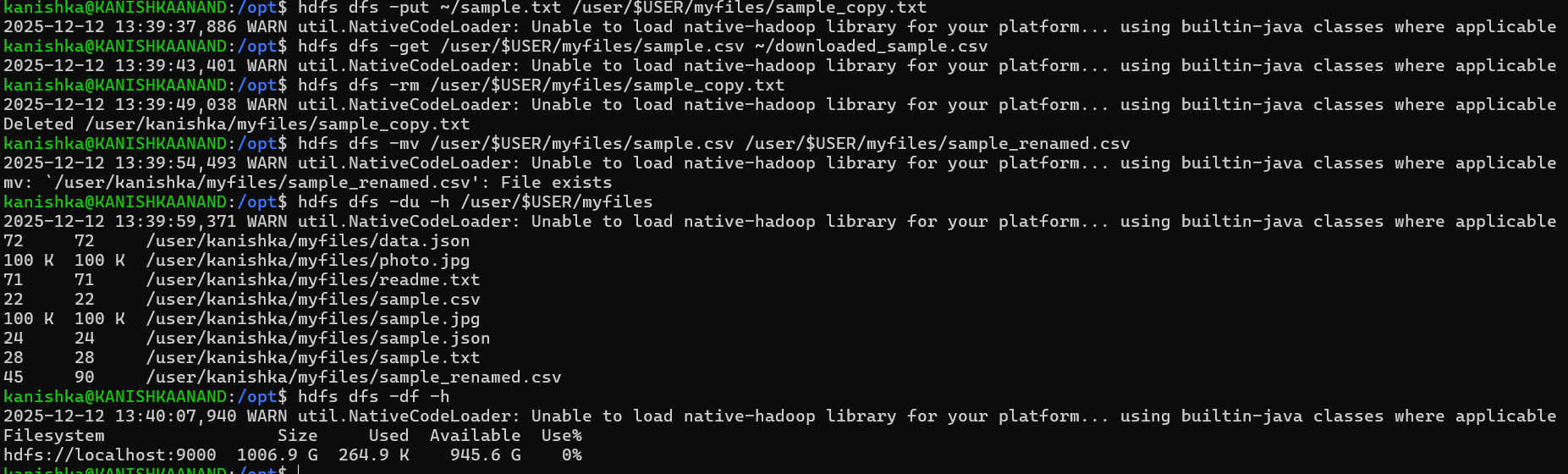
**24. HDFS File Operations (df)**



displays overall HDFS storage capacity, used space, and available space using the hdfs dfs -df -h command.

**25. hdfs dfs -du -h**

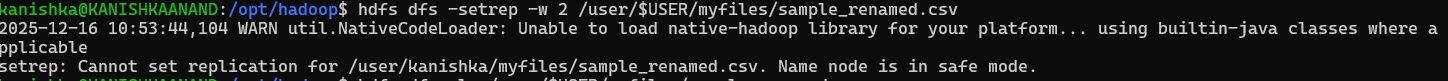
**hdfs dfs -df -h**



demonstrates multiple HDFS file operations including uploading a file (put), downloading it back to local system (get), deleting a file (rm), and renaming a file using the mv command. These operations confirm successful interaction with HDFS.

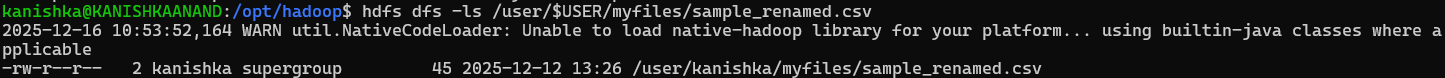
shows disk usage of files stored inside HDFS using the hdfs dfs -du -h command, displaying the size occupied by each file in the directory.

**26. Set Replication Factor**



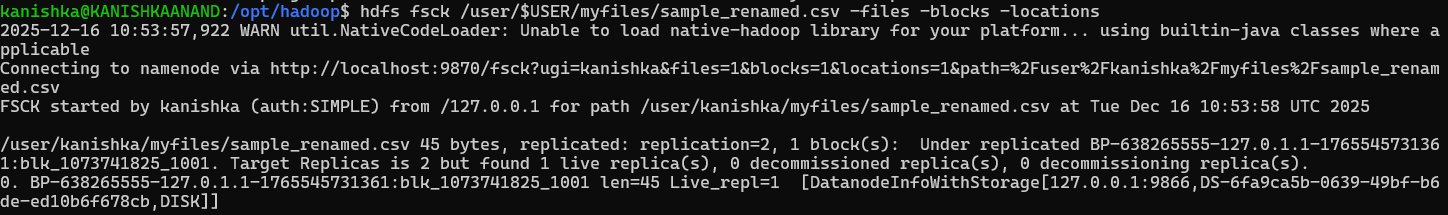
shows an attempt to set the replication factor of an HDFS file to 2. The message indicates that replication could not be updated because the NameNode was in Safe Mode.

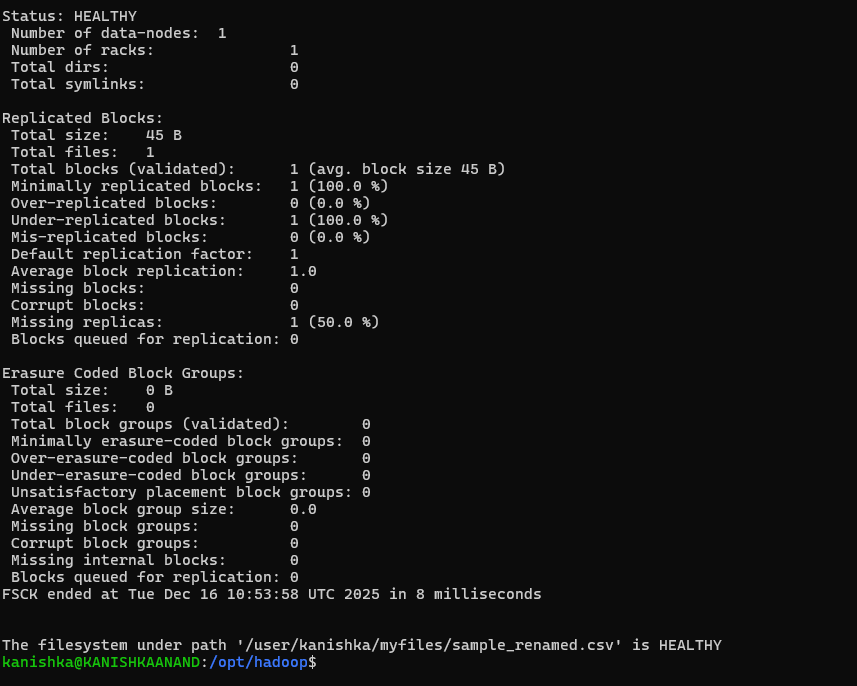
**27. Verify File Details**



shows file metadata including permissions, owner, group, and file size, confirming the presence of the file in HDFS.

**28. Block & Replica Information**





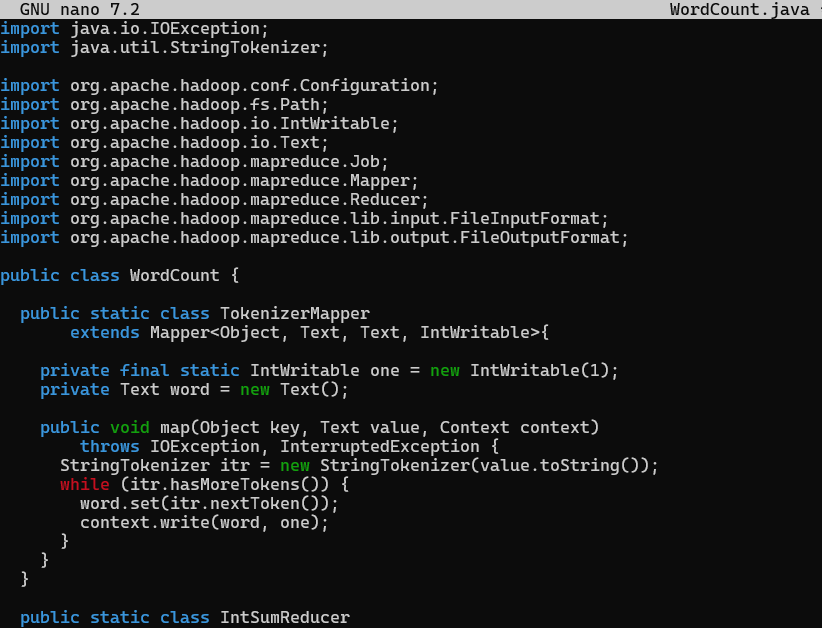
displays HDFS block information using the fsck command, showing block ID, replica count, and DataNode location, verifying block placement in HDFS.

**29. Editing WordCount Program**



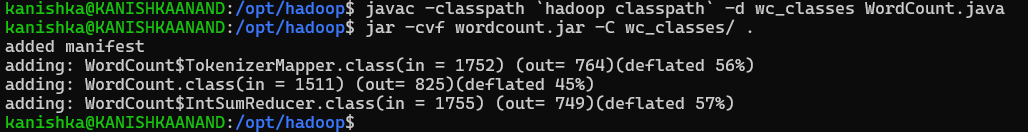
shows editing the WordCount.java program using the nano editor, marking the beginning of the MapReduce WordCount implementation.

**30. WordCount MapReduce Program**



shows the implementation of the WordCount MapReduce program in Java using Hadoop libraries. The code defines the TokenizerMapper and IntSumReducer classes to count word occurrences. It also shows successful compilation of the program using javac and creation of the executable JAR file using the jar command.

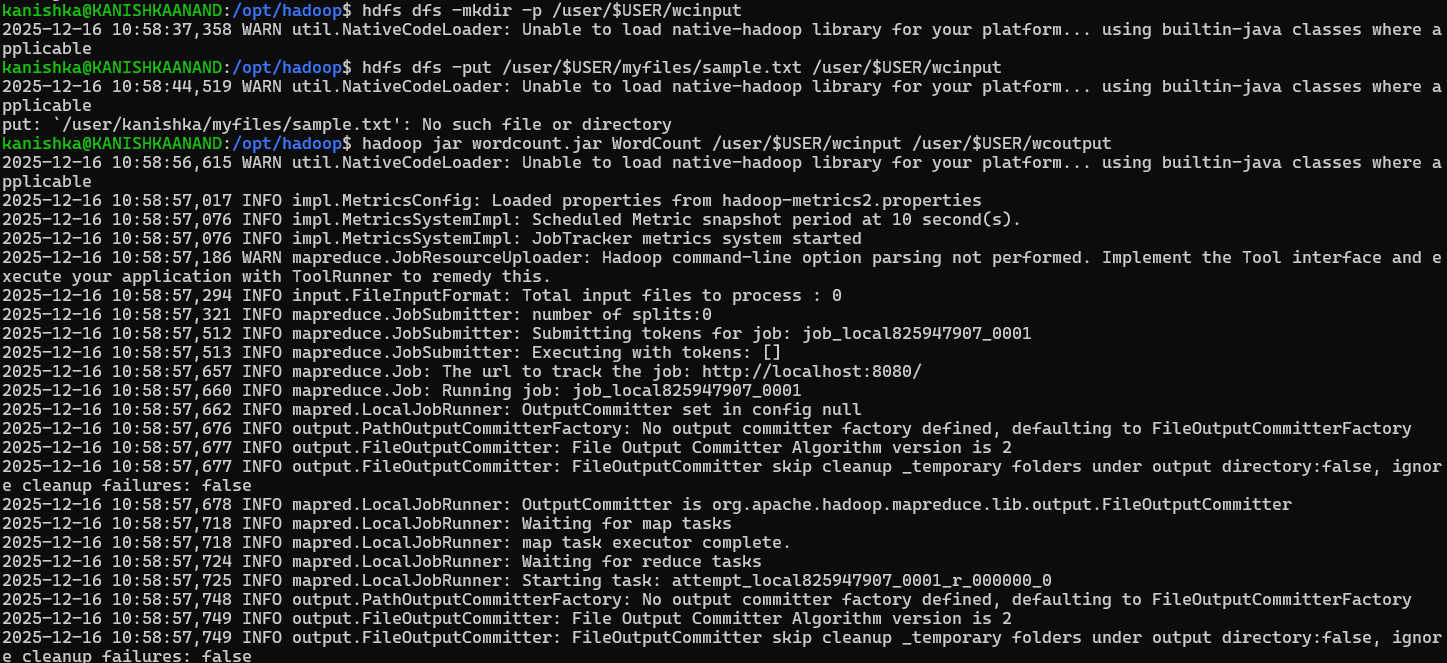
**31. Recompile & create JAR for modified WordCount**

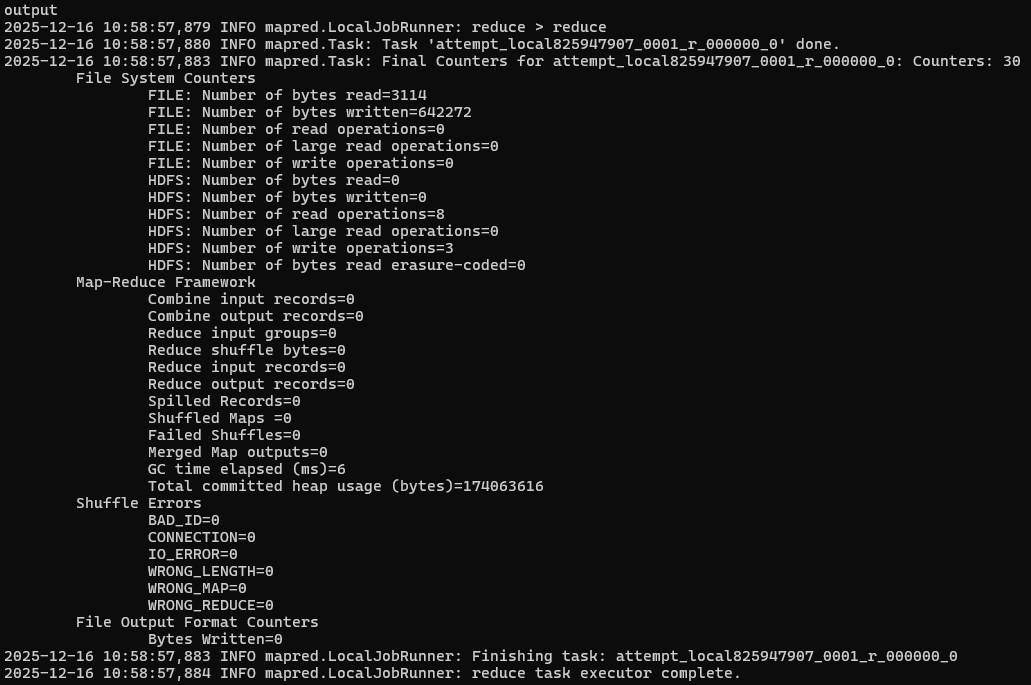


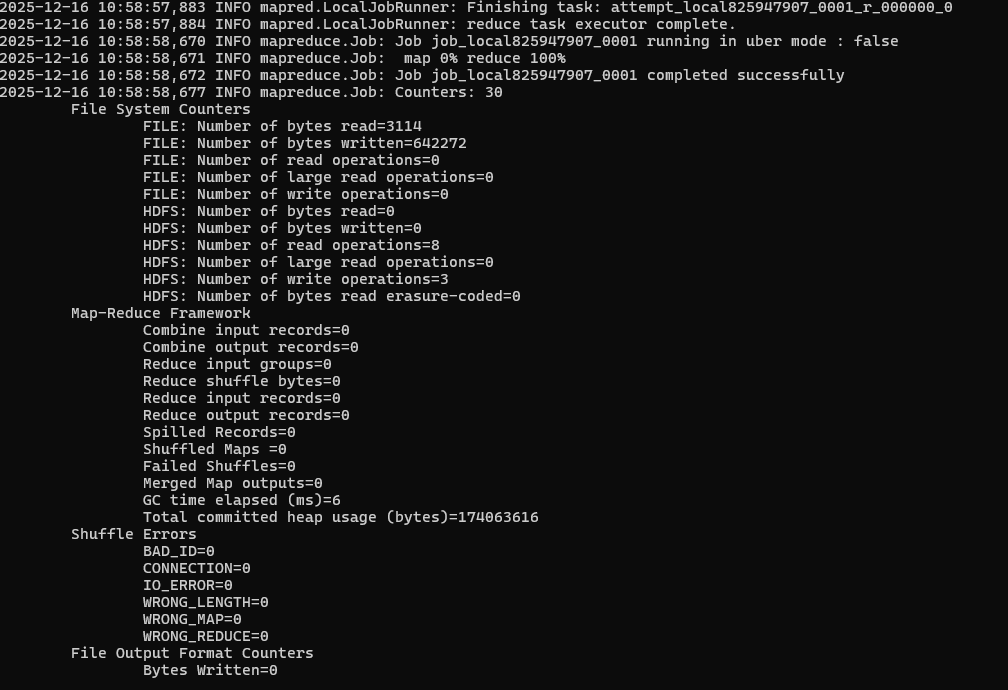
shows successful recompilation of the modified WordCount.java program (with stop-word removal logic) using the Hadoop classpath. It also shows packaging of the compiled classes into a JAR file (wordcount.jar) using the jar command, preparing the program for execution on HDFS.

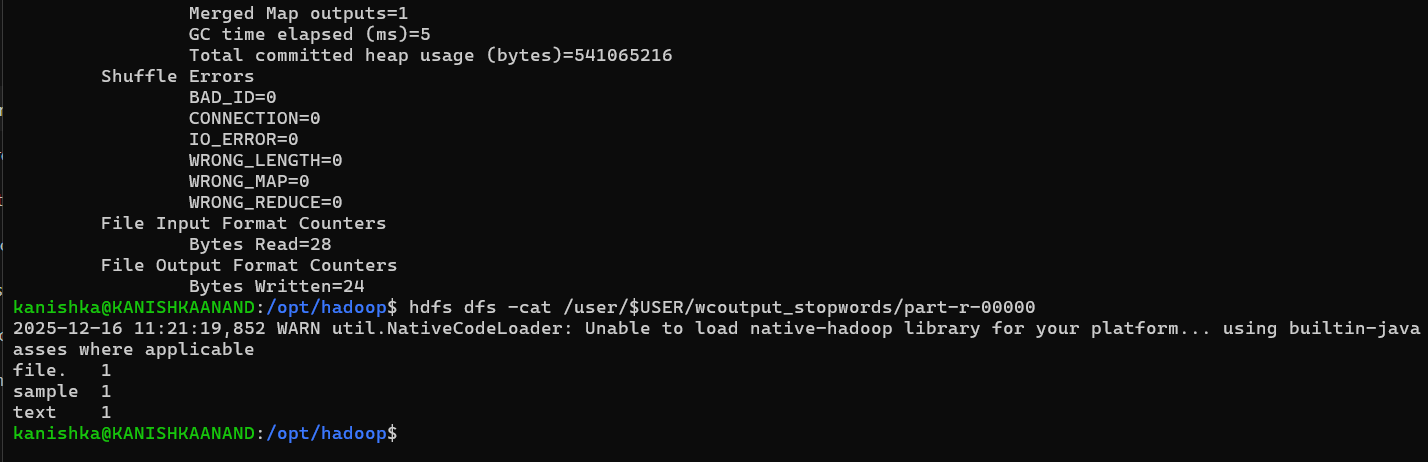
**32.** **Prepare input directory & run WordCount job**

**OUTPUT-**



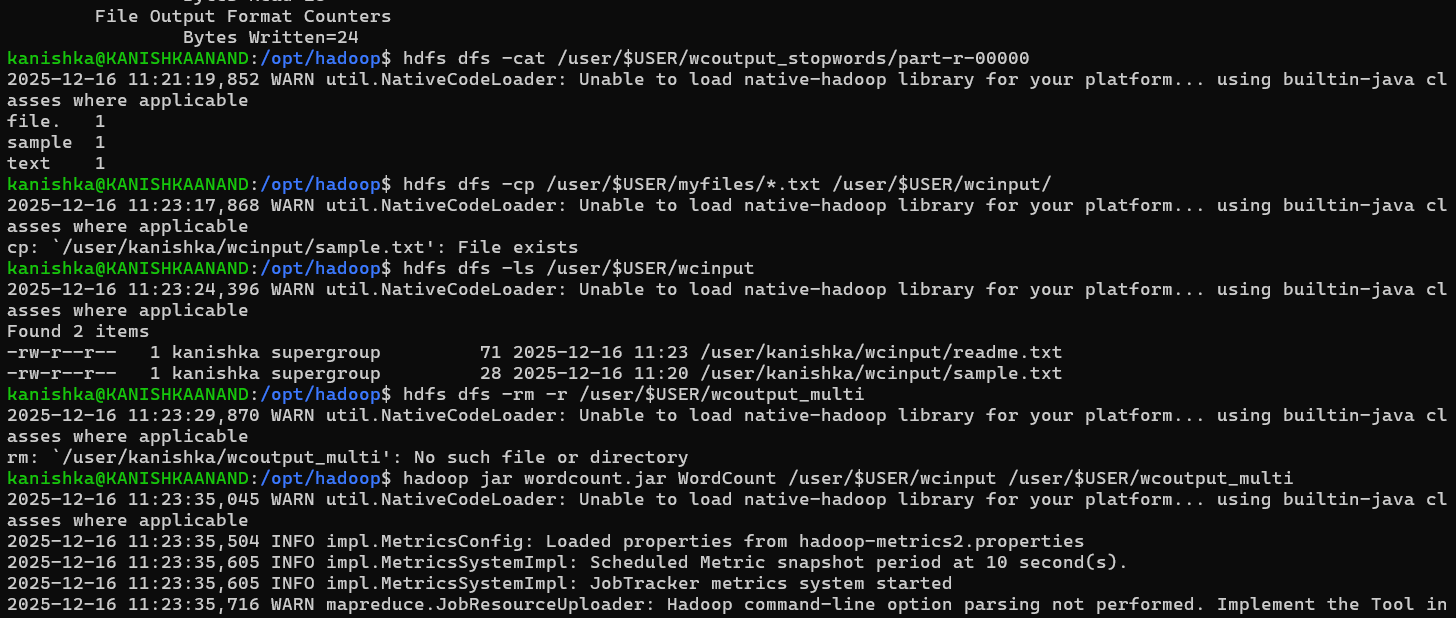


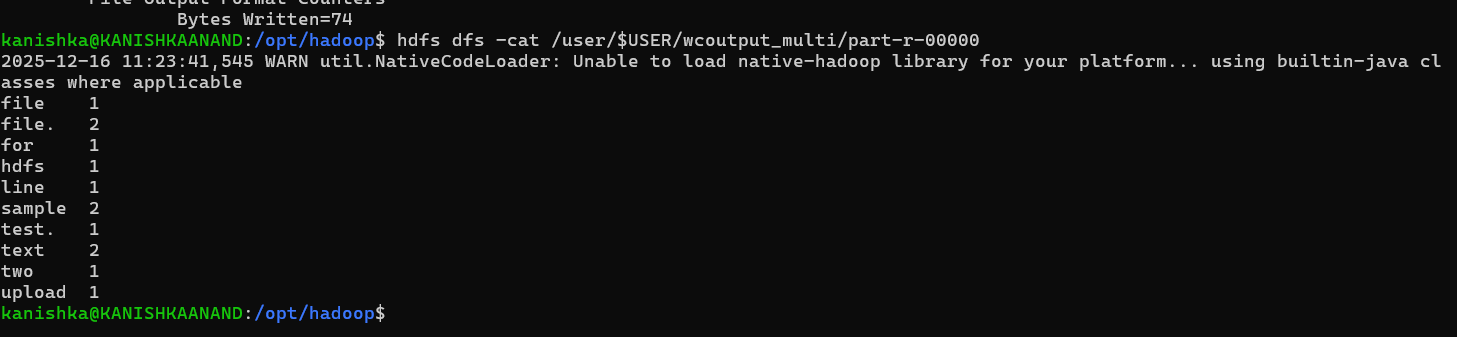




shows creation of the input directory in HDFS, uploading input text files, and execution of the WordCount MapReduce job using the Hadoop JAR file. The logs confirm successful submission and execution of the MapReduce job.

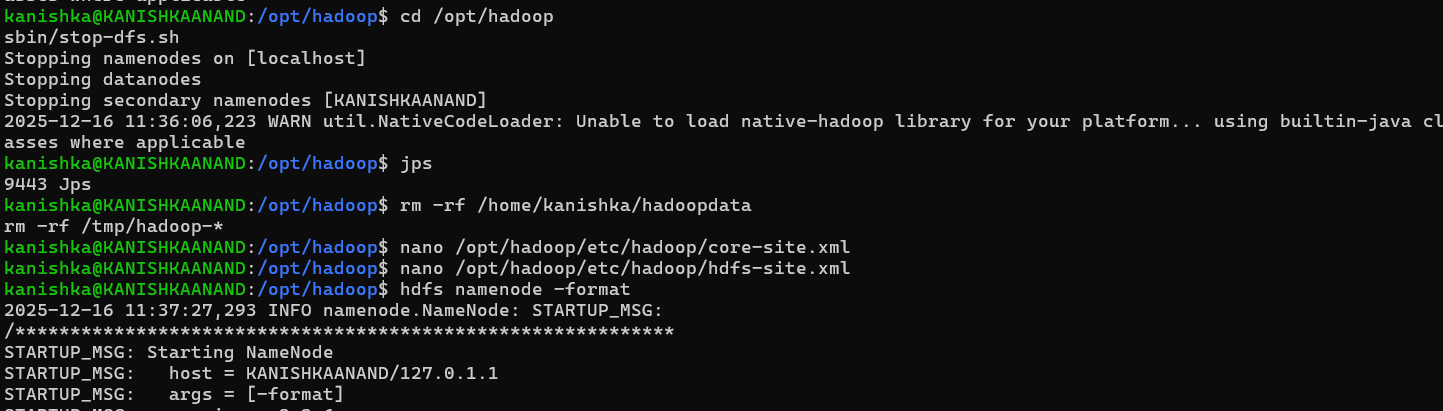
**33. Stop-word WordCount output**





shows the output of the modified WordCount MapReduce program where common stop words are ignored. The output confirms that only meaningful words are counted, verifying successful implementation of stop-word filtering logic.

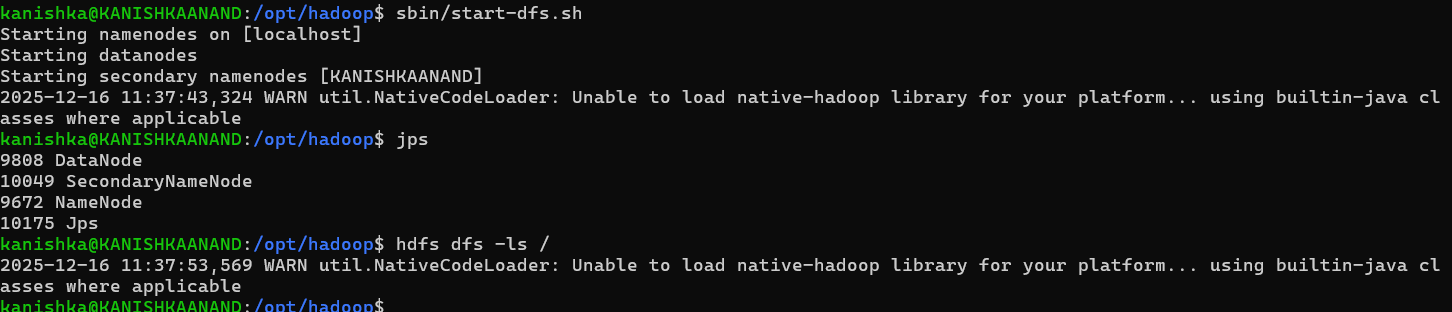
**34. Hadoop HDFS Re-initialization & Service Restart**

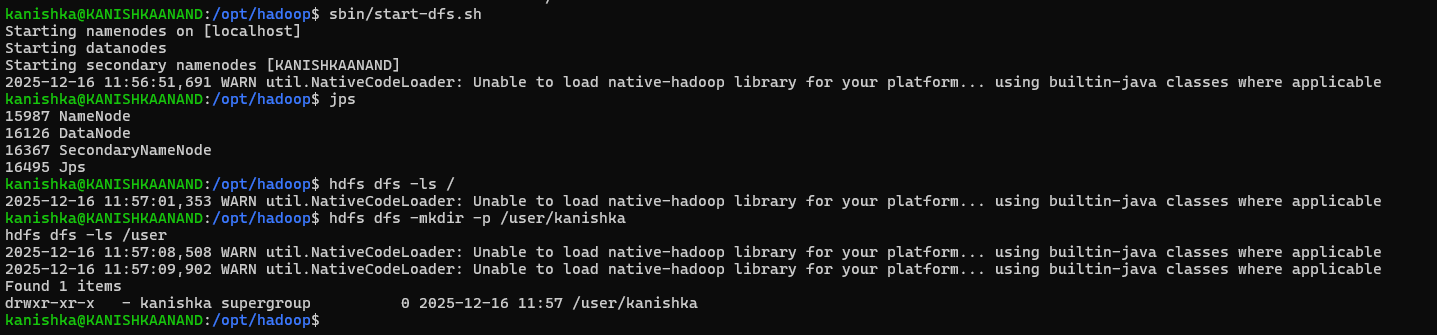


shows the process of **stopping Hadoop services, cleaning old HDFS metadata, re-formatting the NameNode, and restarting HDFS successfully**.

**35. Checking HDFS Root Directory**

**Verifying Running Hadoop Daemons  
Checking HDFS Root Directory**



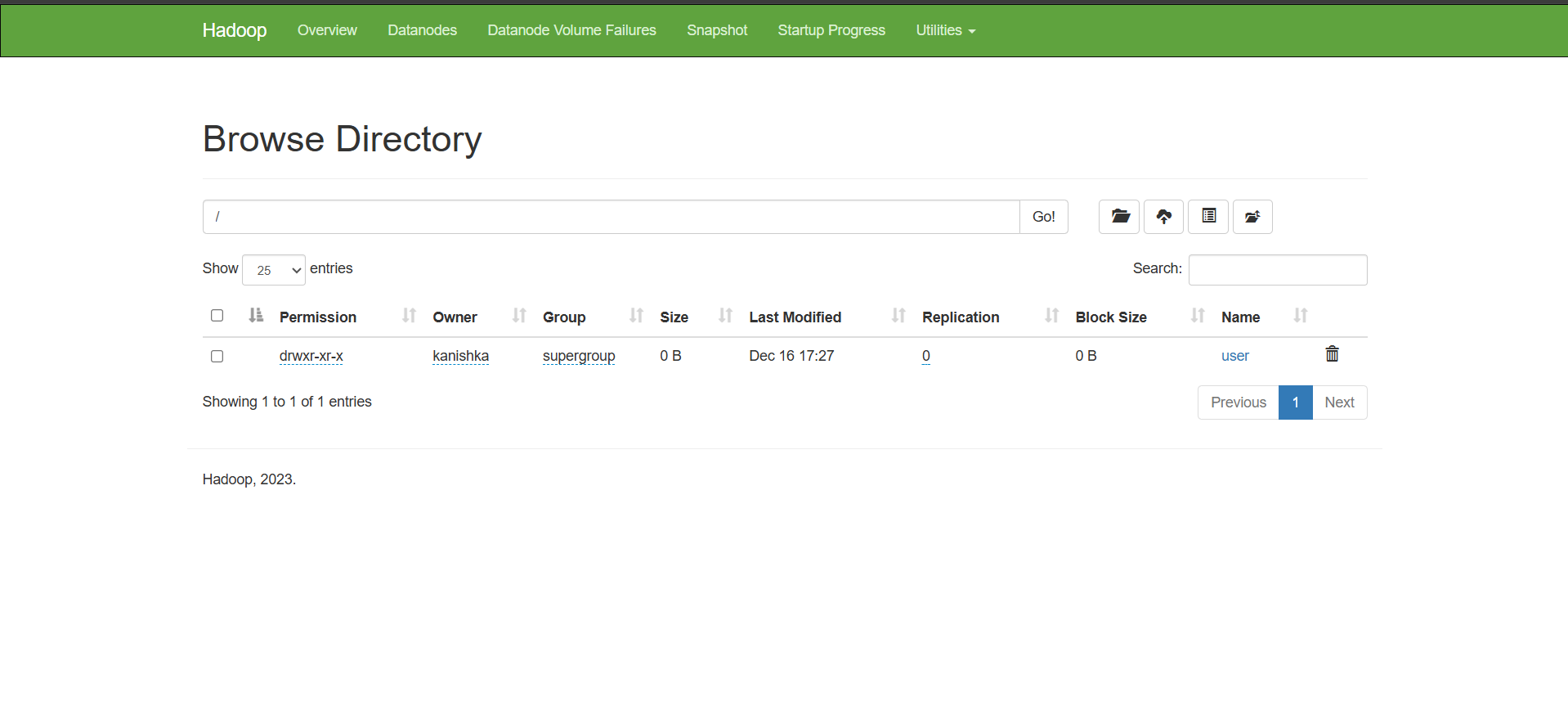


Hadoop services are successfully launched.

Confirms HDFS is running correctly.

Confirms successful filesystem initialization.

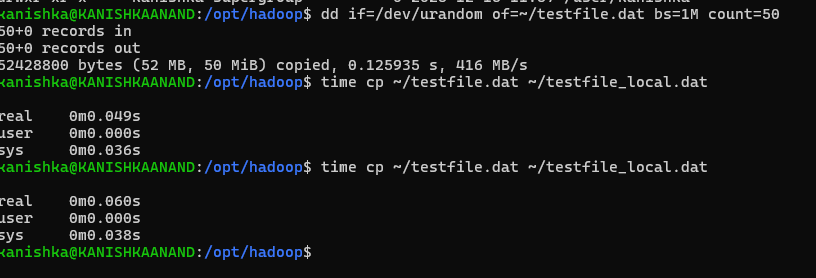
**36. Browse HDFS using Web UI**



shows the Hadoop NameNode Web UI used to browse the HDFS directory structure. The root (/) directory is displayed, containing the /user directory,confirming that HDFS is running correctly and accessible via the browser.

**37. Create a Large File for Performance Testing**

**Local File System Performance Test**



shows the Hadoop NameNode Web UI used to browse the HDFS directory structure. The root (/) directory is displayed, containing the /user directory owned by user *kanishka*, confirming that HDFS is running correctly and accessible via the browser.

demonstrates the creation of a large file (testfile.dat) using the dd command by reading random data from /dev/urandom. The file size is approximately 50 MB, which is commonly used for testing file system performance.

shows the execution of the time cp command to measure how long it takes to copy a large file within the local file system. The output displays real, user, and system time taken for the copy operation.

**38. Comparison Table – Local File System vs HDFS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Feature | Local File System | HDFS | | Storage | Single machine | Distributed | | Speed | Faster for small files | Optimized for large files | | Fault tolerance | None | Replication | | Scalability | Limited | High | | Best use case | Personal files | Big data processing | |

table compares the Local File System and HDFS based on storage type, speed, fault tolerance, scalability, and best use case. It highlights that HDFS is distributed, fault-tolerant, and scalable, making it suitable for big data processing, whereas the local file system is limited to single-machine use.

**Case Study Report**

**Flipkart Big Data Architecture**

**Introduction**

Flipkart is one of India’s largest e-commerce platforms, handling **millions of users, products, and transactions daily**. To manage large-scale data efficiently and deliver personalized user experiences, Flipkart uses a **robust Big Data architecture**.

Big Data enables Flipkart to:

* Analyze customer behavior
* Optimize supply chain & pricing
* Provide real-time recommendations
* Detect fraud and anomalies

**Business Problems Addressed Using Big Data**

Flipkart uses Big Data to solve the following challenges:

* Handling **huge volumes of user clickstream data**
* Providing **real-time product recommendations**
* Managing **dynamic pricing during sales**
* Detecting **fraudulent transactions**
* Forecasting **demand and inventory**

**Types of Data Handled –**

Flipkart processes multiple forms of data:

|  |  |
| --- | --- |
| Data Type | Examples |
| Structured | Orders, payments, user profiles |
| Semi-Structured | JSON logs, API responses |
| Unstructured | Reviews, images, search text |
| Streaming | Clickstream, cart activity |

**Flipkart Big Data Architecture**

**Key Technologies Used**

1. **Data Ingestion**

* **Apache Kafka** – real-time streaming data
* **Apache Flume** – log data collection

1. **Storage**

* HDFS – distributed storage
* Cloud Storage (AWS S3) – scalable data lake

1. **Processing**

* Apache Spark – fast analytics & ML
* MapReduce – batch processing

1. **Query & Access**

* Apache Hive – SQL-like querying
* Apache HBase – real-time NoSQL access

1. **Machine Learning**

* Recommendation systems
* Demand forecasting
* Fraud detection models

**Use Cases of Big Data at Flipkart**

**Recommendation System**

* Suggests products based on:
  + Browsing history
  + Past purchases
  + Similar user behavior

**Dynamic Pricing**

* Prices adjusted in real time during sales
* Based on demand, competition, and inventory

**Supply Chain Optimization**

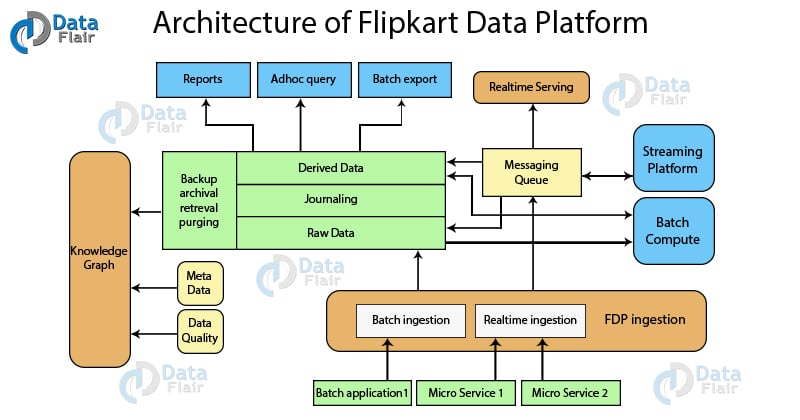
* Predicts demand
* Optimizes warehouse stocking
* Reduces delivery time

**Fraud Detection**

* Detects unusual transaction patterns
* Prevents fake orders and payment fraud

**Benefits Achieved**

* Faster decision-making
* Personalized customer experience
* Increased sales and conversions
* Reduced operational cost
* Scalable system during big sales (Big Billion Days)



**Challenges Faced**

|  |  |
| --- | --- |
| Challenge | Solution |
| High traffic load | **Distributed architecture** |
| Data variety | **Data Lake approach** |
| Real-time processing | **Spark Streaming** |
| Data reliability | **Fault-tolerant HDFS** |

**Conclusion**

Flipkart’s Big Data architecture plays a critical role in its success. By combining distributed storage, real-time processing, and machine learning, Flipkart delivers scalable, reliable, and personalized e-commerce services.

Big Data has transformed Flipkart from a traditional online store into a data-driven digital platform.

**Mini Project**

**Sensor / IoT Data Analytics**

**Project Overview**

With the growth of IoT devices, huge volumes of sensor data are generated continuously.  
This project focuses on analyzing large-scale IoT sensor data using Big Data tools like HDFS, Hive, and Apache Spark, followed by data visualization to extract meaningful insights.

**Objective of the Project**

Store large IoT dataset (>1GB) in HDFS

Perform analytical queries using Hive

Process and analyze data using Apache Spark

Visualize insights such as:

* Sensor activity trends
* Temperature / usage patterns
* Device-wise analysis

**Dataset Description**

**Dataset Type:** IoT / Sensor Telemetry Data  
**Size:** > 1 GB  
**Attributes (example):**

* device\_id
* timestamp
* temperature
* humidity
* location
* device\_status

Link - <https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k>

**System Architecture**

**Data Loading into HDFS (Mandatory)**

**Step 1:** Create HDFS directory

**hdfs dfs -mkdir /iot\_project**

**Step 2:** Upload dataset to HDFS

**hdfs dfs -put iot\_telemetry\_data.csv /iot\_project/**

**Step 3:** Verify upload

**hdfs dfs -ls /iot\_project**

**Hive Analysis (Mandatory)**

**Step 1:** Start Hive

**Hive**

**Step 2:** Create Hive table

CREATE DATABASE iot\_db;

USE iot\_db;

CREATE EXTERNAL TABLE iot\_data (

device\_id STRING,

timestamp STRING,

temperature DOUBLE,

humidity DOUBLE,

location STRING,

device\_status STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/iot\_project';

**Step 3: Sample Hive Queries**

**1. Average temperature per device**

SELECT device\_id, AVG(temperature)

FROM iot\_data

GROUP BY device\_id;

**2. Count of active devices**

SELECT device\_status, COUNT(\*)

FROM iot\_data

GROUP BY device\_status;

**3. Location-wise sensor count**

SELECT location, COUNT(device\_id)

FROM iot\_data

GROUP BY location;

**Spark Processing**

**Step 1: Start PySpark**

Pyspark

**Step 2: Spark Code**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("IoT Data Analytics") \

.enableHiveSupport() \

.getOrCreate()

# Load Hive table

df = spark.sql("SELECT \* FROM iot\_db.iot\_data")

# Show data

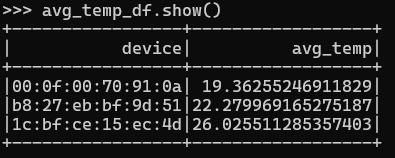
df.show(5)



# Average temperature per location

avg\_temp = df.groupBy("location").avg("temperature")

avg\_temp.show()



# High temperature alerts

alerts = df.filter(df.temperature > 50)

alerts.show()

# Save processed data

avg\_temp.write.mode("overwrite").csv("/iot\_project/output/avg\_temp")

**Visualization**

**Visualization using Python (Matplotlib)**

import pandas as pd

import matplotlib.pyplot as plt

data = {

'Location': ['Mumbai', 'Pune', 'Delhi'],

'AvgTemp': [32, 29, 35]

}

df = pd.DataFrame(data)

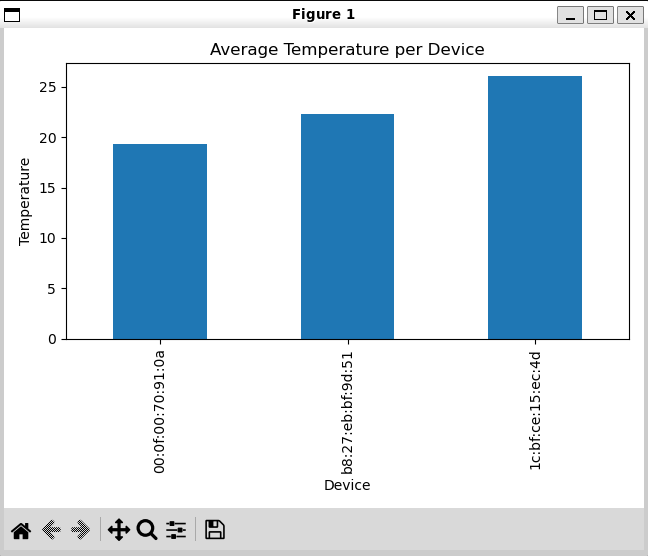
plt.bar(df['Location'], df['AvgTemp'])

plt.xlabel("Location")

plt.ylabel("Average Temperature")

plt.title("Location-wise Average Temperature")

plt.show()



**Visualizations included:**

* Location-wise temperature
* Active vs inactive devices
* Sensor usage trends over time

**Results & Insights**

* Identified **high-temperature zones**
* Found **device usage patterns**
* Detected **inactive/faulty sensors**
* Improved understanding of sensor behavior

**Conclusion**

This mini project demonstrates how **Big Data technologies** efficiently handle and analyze **large-scale IoT sensor data**.  
Using **HDFS, Hive, and Spark**, real-time and batch insights were generated successfully, proving the effectiveness of Big Data analytics in IoT systems.

**Tools & Technologies Used**

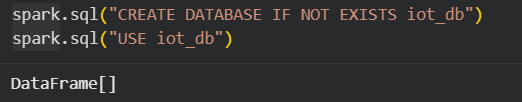
* Hadoop (HDFS)
* Hive
* Apache Spark
* Python (Matplotlib)
* Linux / WSL

**Future Scope**

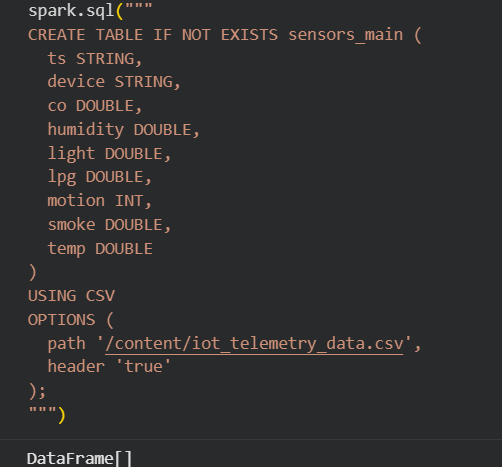
* Real-time streaming using Kafka
* Machine Learning for anomaly detection
* Cloud-based IoT analytics
* Dashboard integration (Power BI / Tableau)

**APACHE HIVE TASKS**

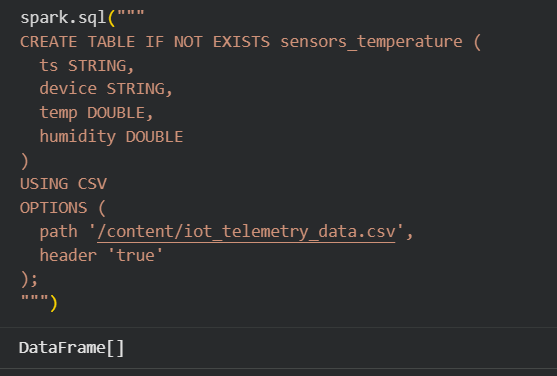
**Que.1 Create Hive database & 3 external tables using CSV.**

****

**External table -1**

****

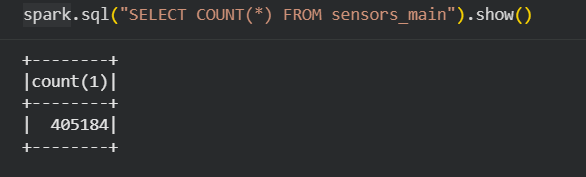
**External Table 2**

****

**External Table 3**

****

**Que.2 Load 1 lakh+ records dataset in Hive.**

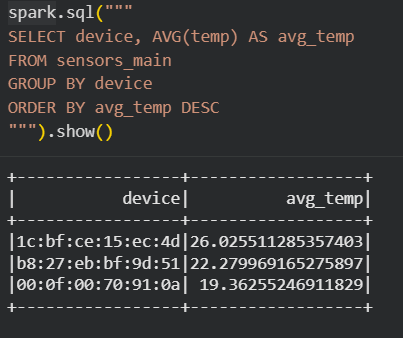
****

**Que.3. Perform:**

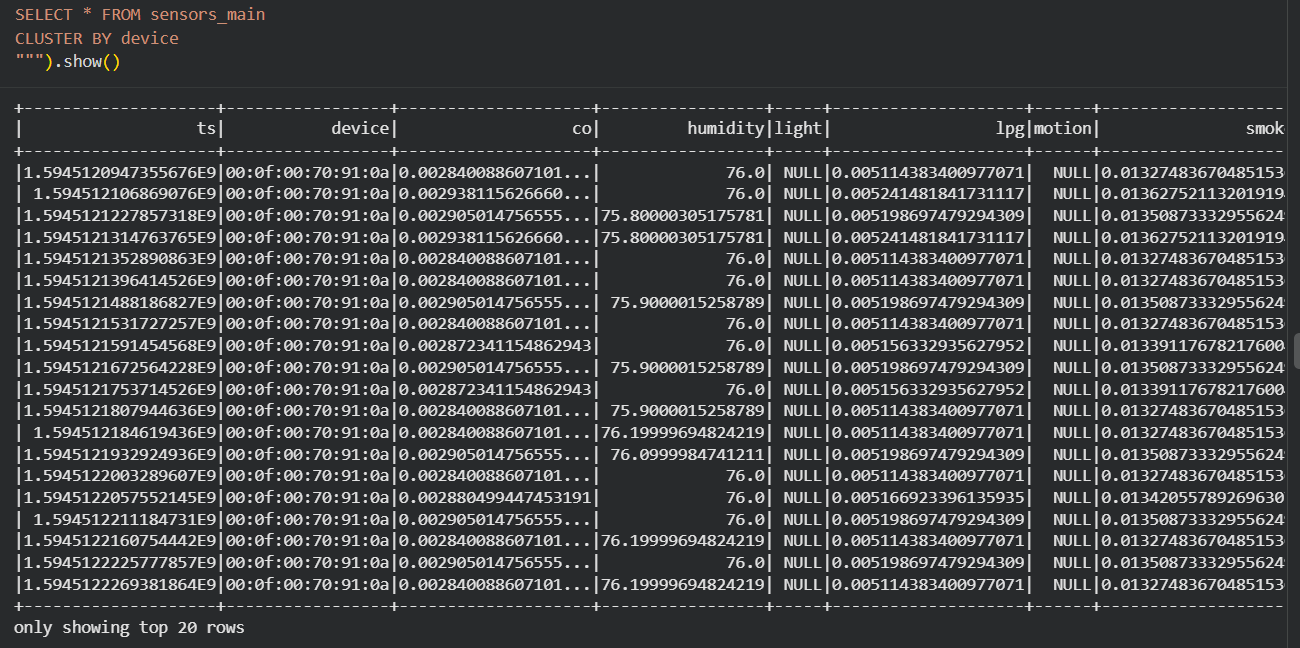
**• GROUP BY**

****

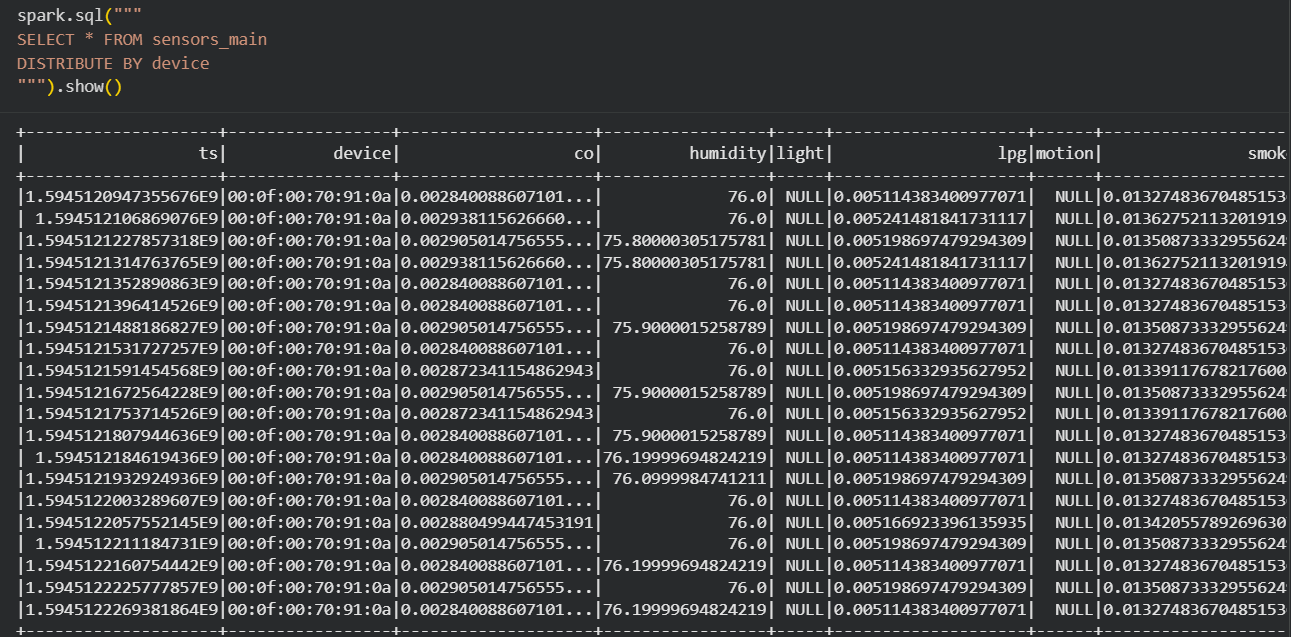
**• ORDER BY**

****

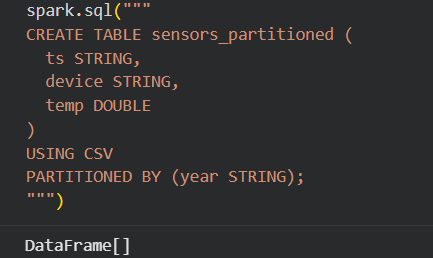
**• CLUSTER BY**

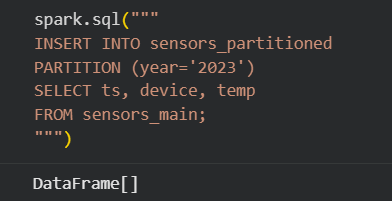
****

**• DISTRIBUTE BY**

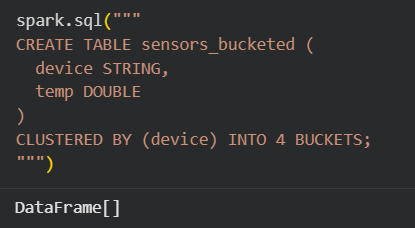
****

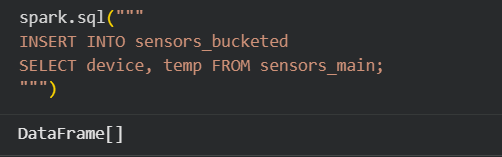
**Que 4. Create Partitioned Table & load year-wise data.**

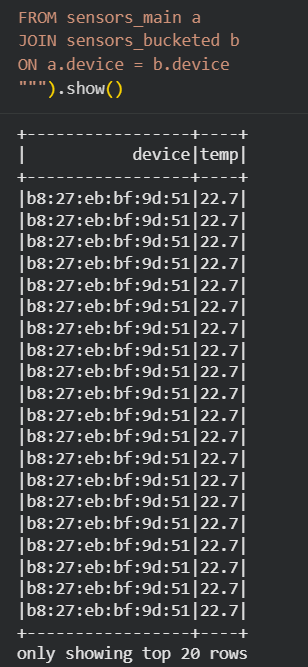
****

****

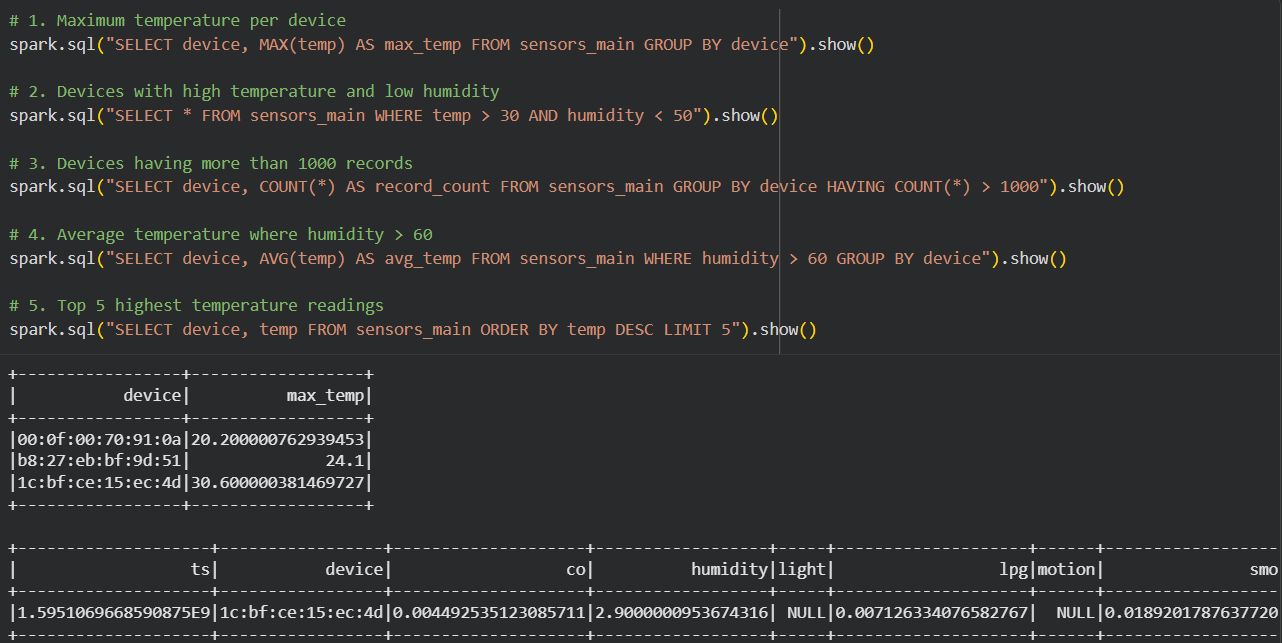
**Que 5. Create Bucketed Table and perform join.**

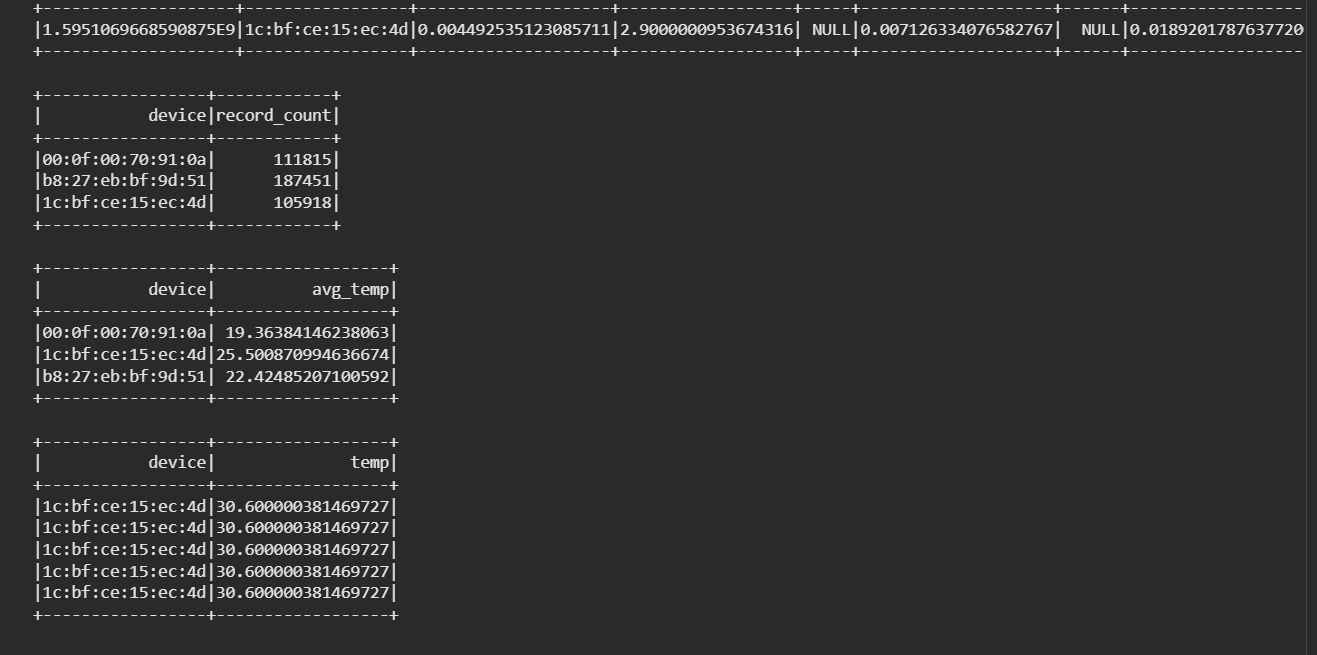
****

****

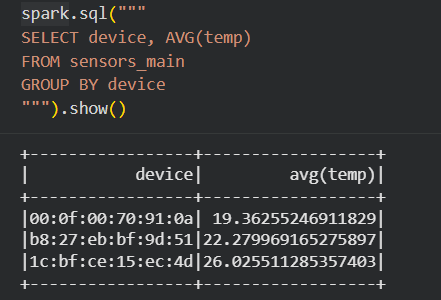
****

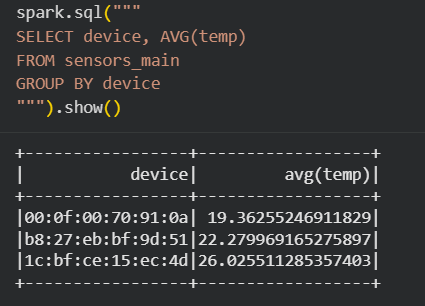
**Que 6. Write 5 complex SQL queries on Hive dataset.**

****

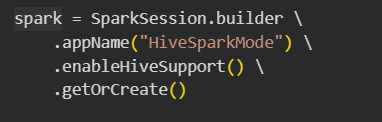
****

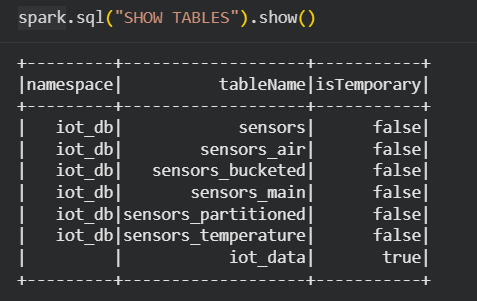
**Que 7. Compare Hive vs MySQL query execution time.**

****

****

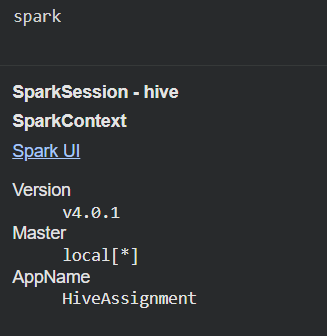
**Que 8. Enable and test Hive on Tez/Spark mode**

****

****

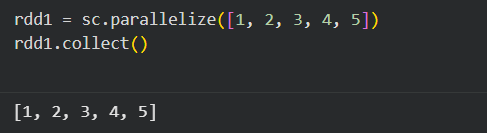
**APACHE SPARK & PySpark TASKS**

**29. Install Spark & run in Standalone Mode.**

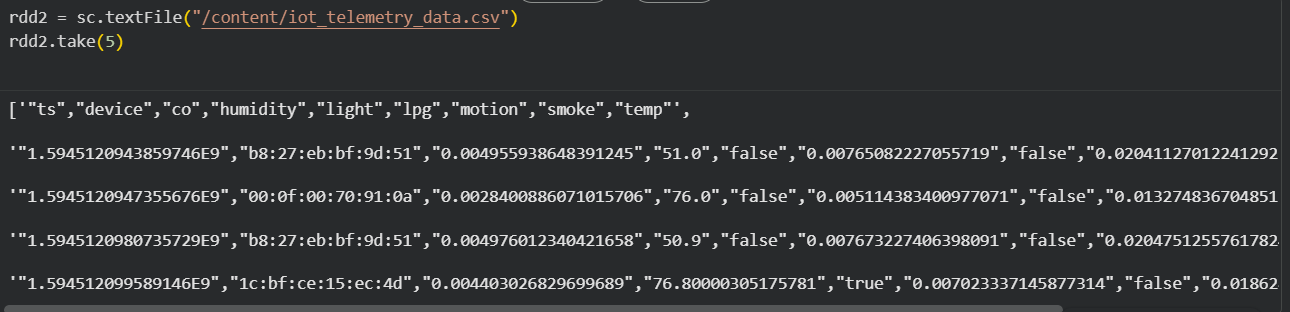
****

**30. Create RDD using:**

**• Parallelize**

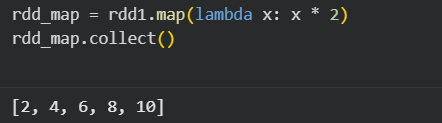
****

**• External file**

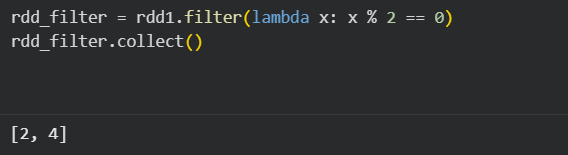
****

**31. Perform:**

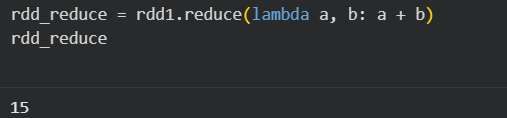
**• map**

****

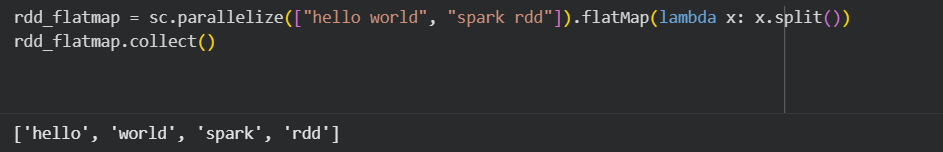
**• filter**

****

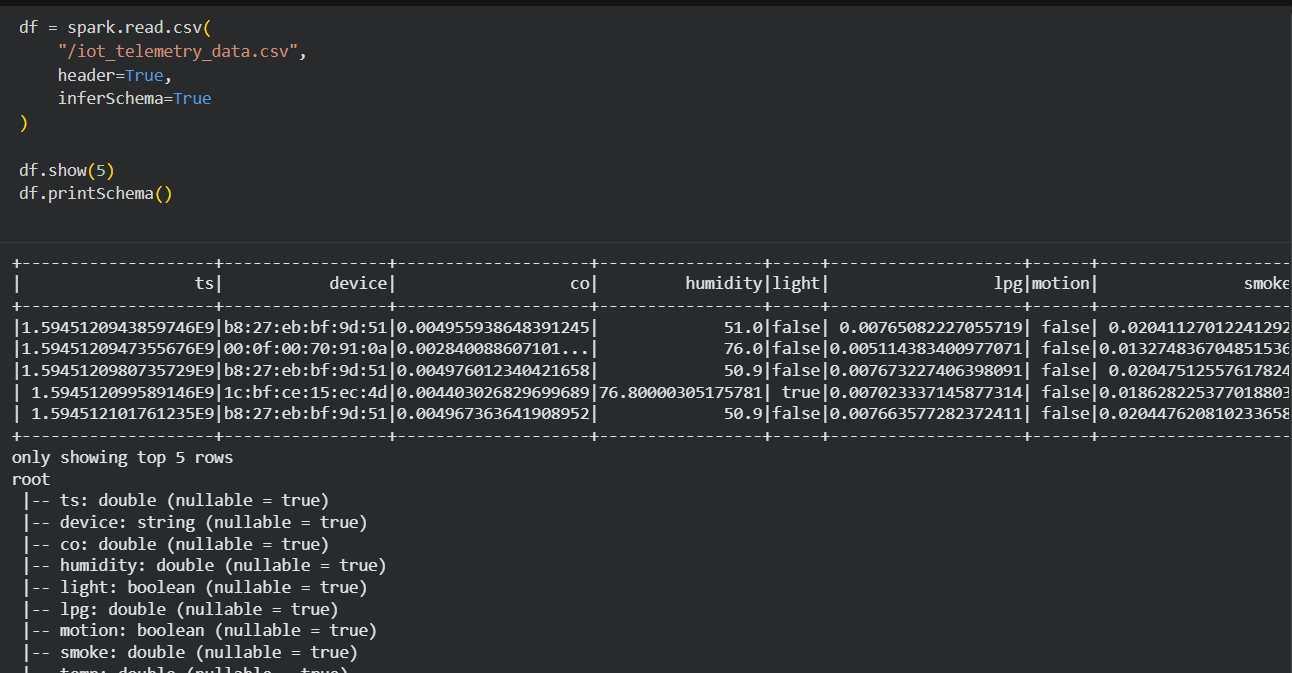
**• reduce**

****

**• flatMap**

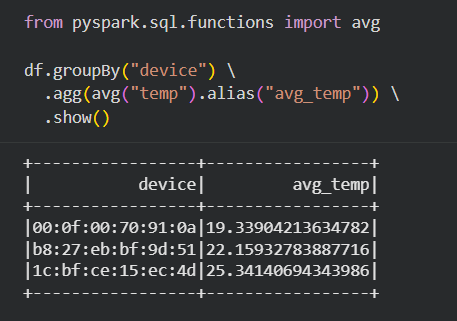
****

**32. Load CSV using Spark DataFrame.**

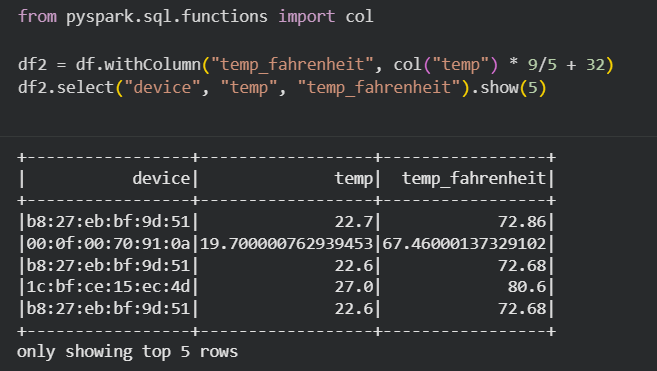
****

**33. Perform:**

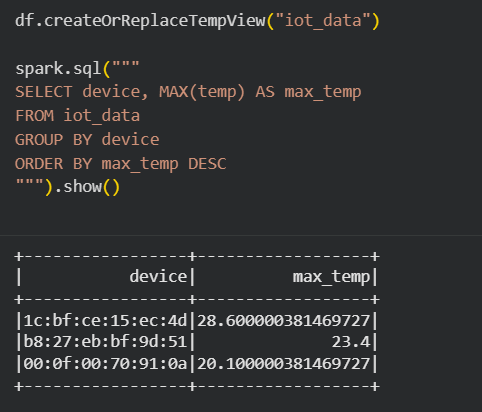
**• groupBy and agg**

****

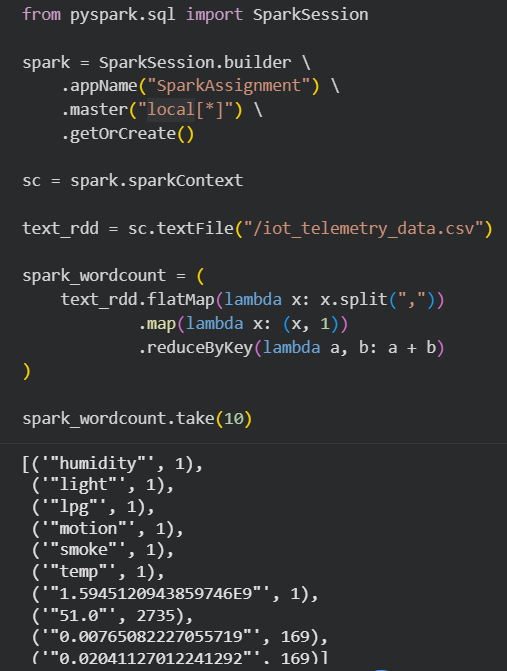
**• withColumn**

****

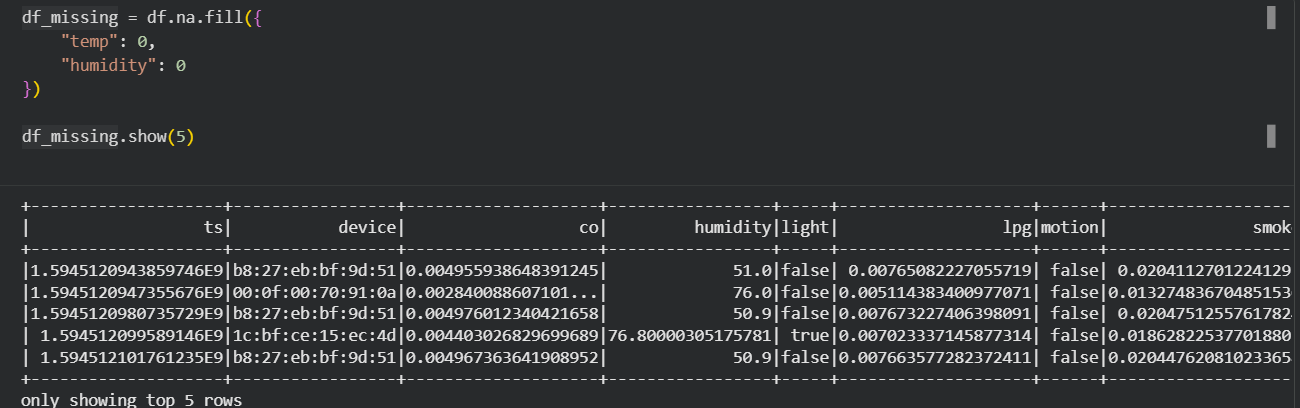
**34. Perform Spark SQL queries.**

****

**35. Compare Spark vs Hadoop WordCount performance.**

****

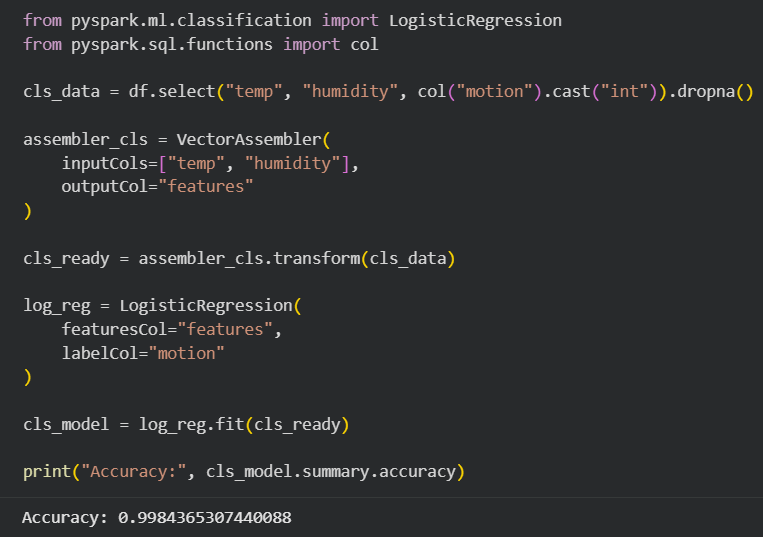
**36. Perform Missing Value Handling using Spark.**

****

**37. Implement Linear Regression using Spark MLlib.**

****

**38. Implement Classification Model using Spark.**

****