CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE | DELHI NCR | PUNE

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).

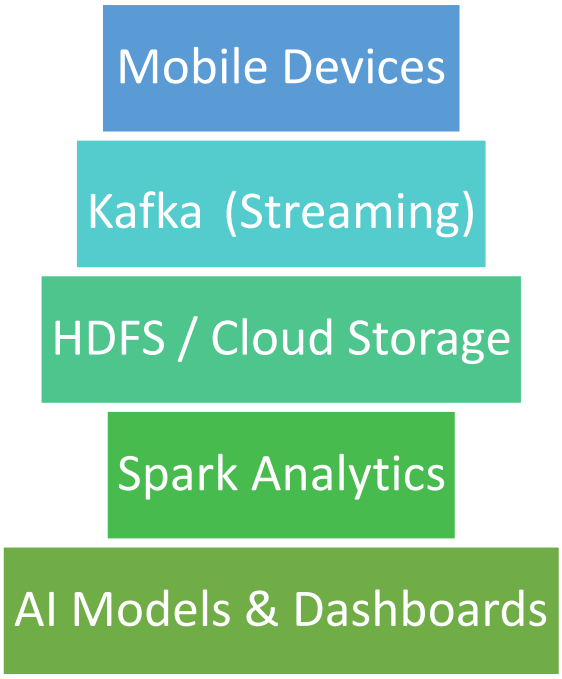
Que.3 Study any 2 Indian companies using Big Data and document their architecture.

1. Reliance Jio

Use Cases

- Network optimization
- Customer behavior analysis
- Personalized offers

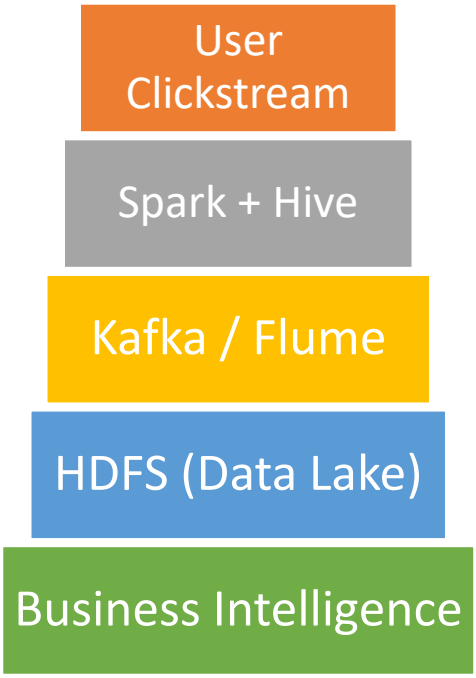
Architecture



2. Flipkart

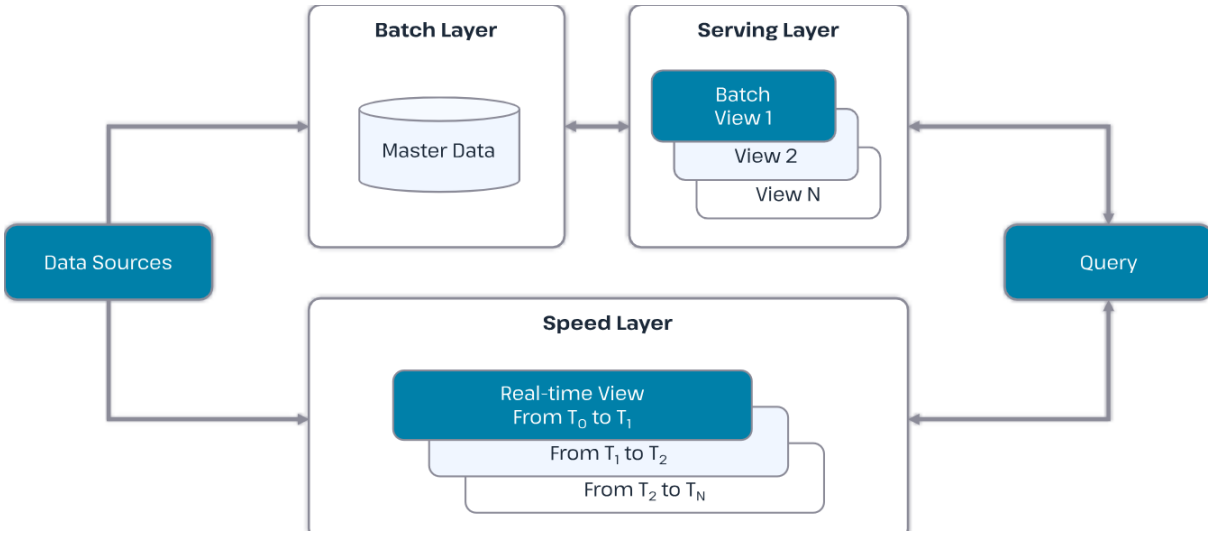
Use Cases

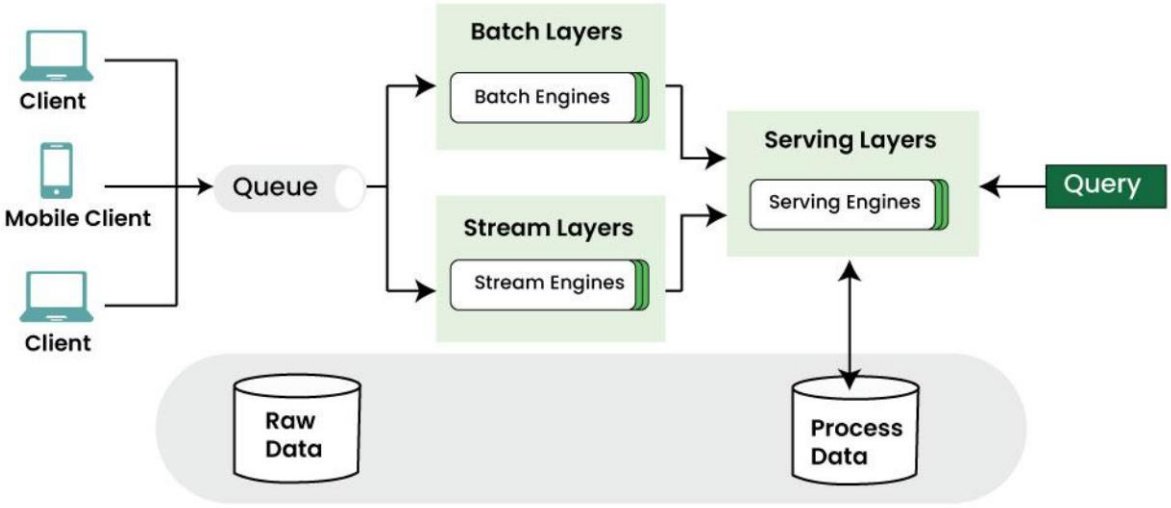
- **Dynamic pricing**
- **Recommendation engine**
- **Fraud detection**



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

Lambda Architecture





Components

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

Pros

Accurate

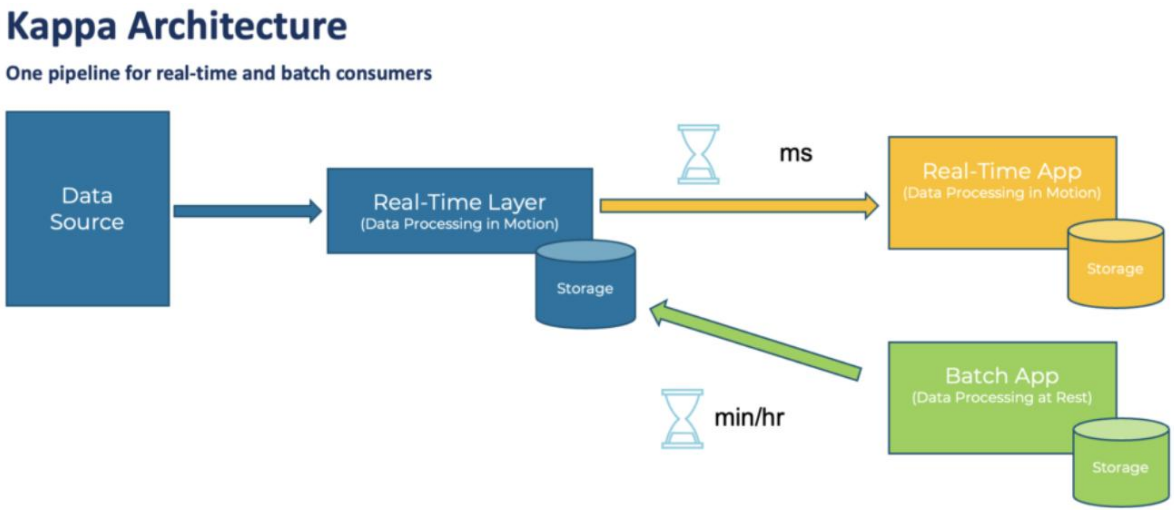
Fault-tolerant

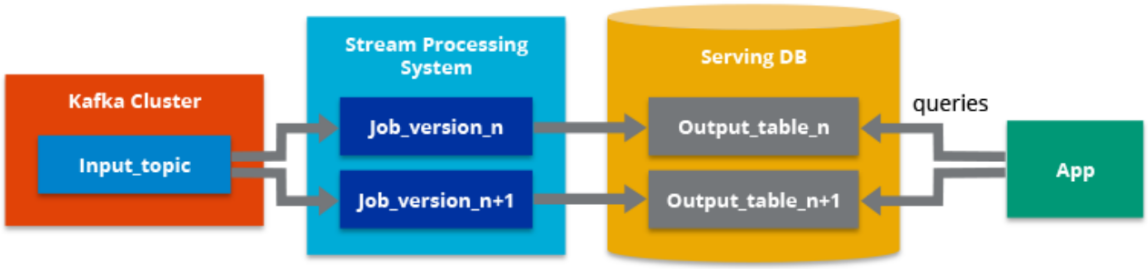
Cons

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.

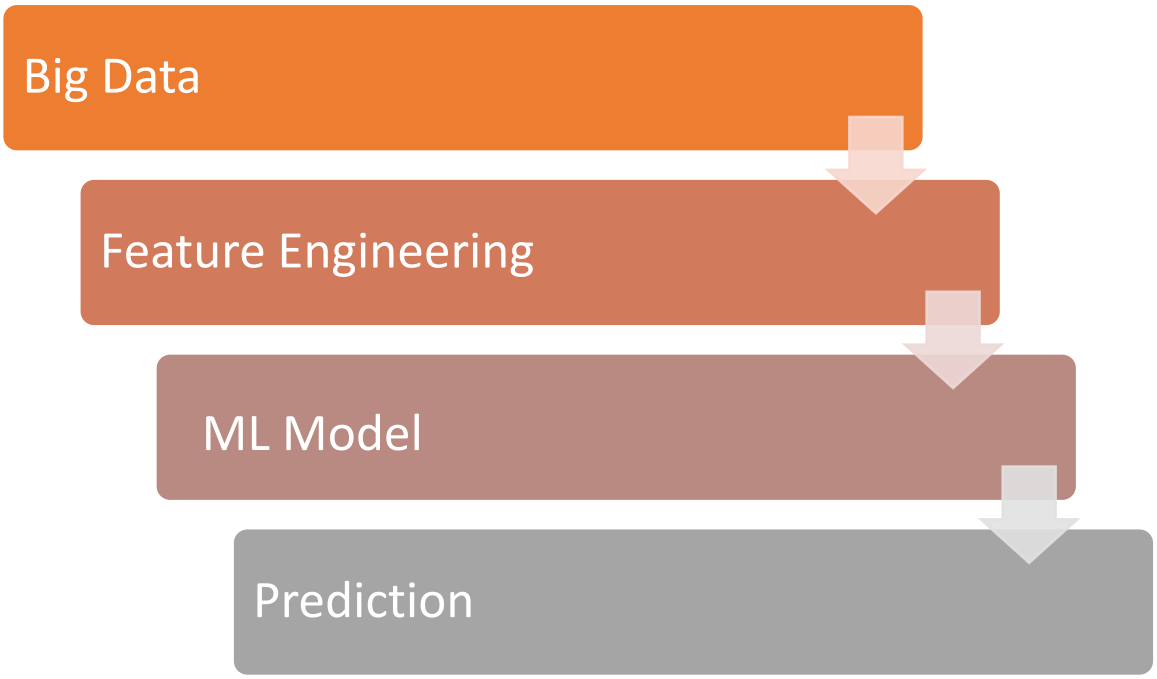
Big Data provides:

- **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).

```
kanishka@KANISHKAANAND:/mnt/c/Users/kanis$ sudo apt update
[sudo] password for kanishka:
Sorry, try again.
[sudo] password for kanishka:
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
```

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)**

```
kanishka@KANISHKAANAND:/mnt/c/Users/kanis$ sudo apt install -y openjdk-11-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

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)

```
kanishka@KANISHKAANAND:/mnt/c/Users/kanis$ java -version
openjdk version "11.0.29" 2025-10-21
OpenJDK Runtime Environment (build 11.0.29+7-post-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 11.0.29+7-post-Ubuntu-1ubuntu124.04, mixed mode, sharing)
kanishka@KANISHKAANAND:/mnt/c/Users/kanis$
```

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

4. Install Hadoop (Download)

```
kanishka@KANISHKAANAND:/mnt/c/Users/kanis$ cd /opt
kanishka@KANISHKAANAND:/opt$ sudo wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
--2025-12-12 13:06:58--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.208.237, 135.181.214.104, 2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.208.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: ‘hadoop-3.3.6.tar.gz’
```

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)

```
2025-12-12 13:09:26 (4.75 MB/s) - ‘hadoop-3.3.6.tar.gz’ saved [730107476/730107476]

kanishka@KANISHKAANAND:/opt$ sudo tar -xzf hadoop-3.3.6.tar.gz
kanishka@KANISHKAANAND:/opt$
```

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

6. Hadoop environment configuration

```
kanishka@KANISHKAANAND:/opt$ sudo tar -xzf hadoop-3.3.6.tar.gz
kanishka@KANISHKAANAND:/opt$ sudo mv hadoop-3.3.6 hadoop
kanishka@KANISHKAANAND:/opt$ sudo chown -R $USER:$USER /opt/hadoop
kanishka@KANISHKAANAND:/opt$ echo 'export HADOOP_HOME=/opt/hadoop' >> ~/.bashrc
kanishka@KANISHKAANAND:/opt$ echo 'export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin' >> ~/.bashrc
kanishka@KANISHKAANAND:/opt$ echo 'export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc
kanishka@KANISHKAANAND:/opt$ source ~/.bashrc
kanishka@KANISHKAANAND:/opt$ mkdir -p ~/hadoopdata/namenode
kanishka@KANISHKAANAND:/opt$ mkdir -p ~/hadoopdata/datanode
kanishka@KANISHKAANAND:/opt$ chmod +x /opt/hadoop/bin/* /opt/hadoop/sbin/*
kanishka@KANISHKAANAND:/opt$ nano /opt/hadoop/etc/hadoop/core-site.xml
kanishka@KANISHKAANAND:/opt$ nano /opt/hadoop/etc/hadoop/hdfs-site.xml
kanishka@KANISHKAANAND:/opt$ hdfs namenode -format
WARNING: /opt/hadoop/logs does not exist. Creating.
log4j:ERROR Could not instantiate class [org.apache.hadoop.log.metrics.EventCounter].
java.lang.ClassNotFoundException: org.apache.hadoop.log.metrics.EventCounter
        at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:581)
        at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:527)
        at java.base/java.lang.Class.forName0(Native Method)
        at java.base/java.lang.Class.forName(Class.java:315)
```

setting up Hadoop environment variables such as HADOOP_HOME, JAVA_HOME, and updating the system PATH. These configurations enable Hadoop commands to run globally.

```
kanishka@KANISHKAANAND:/opt$ source ~/.bashrc
kanishka@KANISHKAANAND:/opt$ jps
```

7. Hadoop installation verification

```
kanishka@KANISHKAANAND:/opt$ jps
3952 DataNode
4615 NameNode
4139 SecondaryNameNode
5022 Jps
kanishka@KANISHKAANAND:/opt$
```

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

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -mkdir -p /user/$USER/mydirs
2025-12-12 13:24:07,563 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
```

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

```
kanishka@KANISHKAANAND:/opt$ for i in {1..10}; do hdfs dfs -mkdir -p /user/$USER/mydirs/dir$i; done
2025-12-12 13:24:13,185 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
tin-java classes where applicable
2025-12-12 13:24:14,280 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
```

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

10. Recursive listing of directories

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -ls -R /user/$USER/mydirs
2025-12-12 13:25:01,401 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir1
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir10
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir2
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir3
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir4
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir5
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir6
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir7
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir8
drwxr-xr-x   - kanishka supergroup          0 2025-12-12 13:24 /user/kanishka/mydirs/dir9
```

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

11. HDFS File System Commands (du)

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -du -h /user/$USER/mydirs
2025-12-12 13:25:07,045 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
0  0  /user/kanishka/mydirs/dir1
0  0  /user/kanishka/mydirs/dir10
0  0  /user/kanishka/mydirs/dir2
0  0  /user/kanishka/mydirs/dir3
0  0  /user/kanishka/mydirs/dir4
0  0  /user/kanishka/mydirs/dir5
0  0  /user/kanishka/mydirs/dir6
0  0  /user/kanishka/mydirs/dir7
0  0  /user/kanishka/mydirs/dir8
0  0  /user/kanishka/mydirs/dir9
```

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)

```
kanishka@KANISHKAANAND:/opt$ cat > ~/sample.csv <<'CSV'
id,name,age
1,Alice,30
2,Bob,25
3,Charlie,28
CSV
```

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)**

```
kanishka@KANISHKAANAND:/opt$ cat > ~/readme.txt <<'TXT'
This is a sample text file for HDFS upload test.
Line two of the file.
TXT
```

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)

```
kanishka@KANISHKAANAND:/opt$ cat > ~/data.json <<'JSON'
{"id":1,"name":"Alice"}
{"id":2,"name":"Bob"}
{"id":3,"name":"Charlie"}
JSON
```

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

```
kanishka@KANISHKAANAND:/opt$ dd if=/dev/urandom of=~/photo.jpg bs=1024 count
=100
100+0 records in
100+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.000335793 s, 305 MB/s
```

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)

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -mkdir -p /user/$USER/myfiles
2025-12-12 13:26:01,365 WARN util.NativeCodeLoader: Unable to load native-ha
doop library for your platform... using builtin-java classes where applicabl
e
```

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

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -put ~/sample.csv ~/readme.txt ~/data.
json ~/photo.jpg /user/$USER/myfiles/
2025-12-12 13:26:06,913 WARN util.NativeCodeLoader: Unable to load native-ha
doop library for your platform... using builtin-java classes where applicabl
e
```

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

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -ls /user/$USER/myfiles
2025-12-12 13:26:13,849 WARN util.NativeCodeLoader: Unable to load native-ha
doop library for your platform... using builtin-java classes where applicabl
e
Found 4 items
-rw-r--r--   1 kanishka supergroup         72 2025-12-12 13:26 /user/kanishk
a/myfiles/data.json
-rw-r--r--   1 kanishka supergroup     102400 2025-12-12 13:26 /user/kanishk
a/myfiles/photo.jpg
-rw-r--r--   1 kanishka supergroup         71 2025-12-12 13:26 /user/kanishk
a/myfiles/readme.txt
-rw-r--r--   1 kanishka supergroup         45 2025-12-12 13:26 /user/kanishk
a/myfiles/sample.csv
kanishka@KANISHKAANAND:/opt$
```

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)

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -put ~/readme.txt /user/$USER/myfiles/readme_copy.txt
2025-12-12 13:29:16,086 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

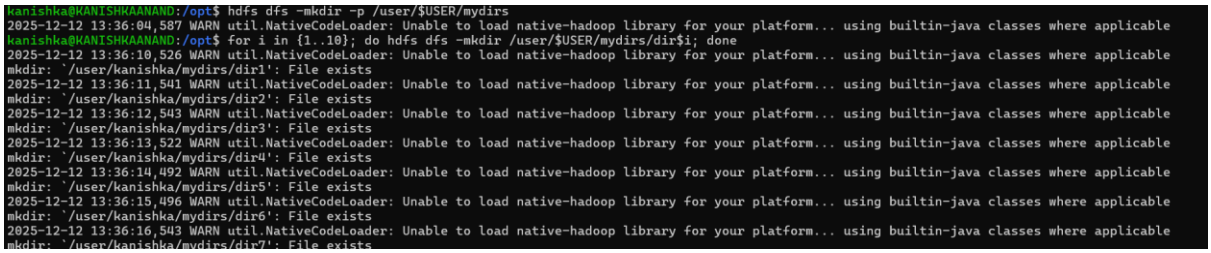
shows copying a local file into HDFS using the hdfs dfs -put command, creating a duplicate file inside the HDFS directory for further operations.

20. HDFS File Operations (get)

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -get /user/$USER/myfiles/readme_copy.txt ~/downloaded_readme.txt
2025-12-12 13:29:23,636 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt$ hdfs dfs -rm /user/$USER/myfiles/readme_copy.txt
2025-12-12 13:29:29,114 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Deleted /user/kanishka/myfiles/readme_copy.txt
kanishka@KANISHKAANAND:/opt$ hdfs dfs -rm /user/$USER/myfiles/readme_copy.txt
2025-12-12 13:29:36,328 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
rm: `/user/kanishka/myfiles/readme_copy.txt': No such file or directory
kanishka@KANISHKAANAND:/opt$ hdfs dfs -mv /user/$USER/myfiles/sample.csv /user/$USER/myfiles/sample_renamed.csv
2025-12-12 13:29:42,815 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt$ hdfs dfs -du -h /user/$USER/myfiles
2025-12-12 13:29:51,053 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
72     72     /user/kanishka/myfiles/data.json
100 K  100 K  /user/kanishka/myfiles/photo.jpg
71     71     /user/kanishka/myfiles/readme.txt
45     45     /user/kanishka/myfiles/sample_renamed.csv
kanishka@KANISHKAANAND:/opt$ hdfs dfs -df -h
2025-12-12 13:29:57,020 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Filesystem                Size    Used  Available  Use%
hdfs://localhost:9000  1006.9 G  125.0 K    945.6 G    0%
kanishka@KANISHKAANAND:/opt$ hdfs dfs -setrep -w 2 /user/$USER/myfiles/sample_renamed.csv
2025-12-12 13:30:06,131 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Replication 2 set: /user/kanishka/myfiles/sample_renamed.csv
Waiting for /user/kanishka/myfiles/sample_renamed.csv .........
```

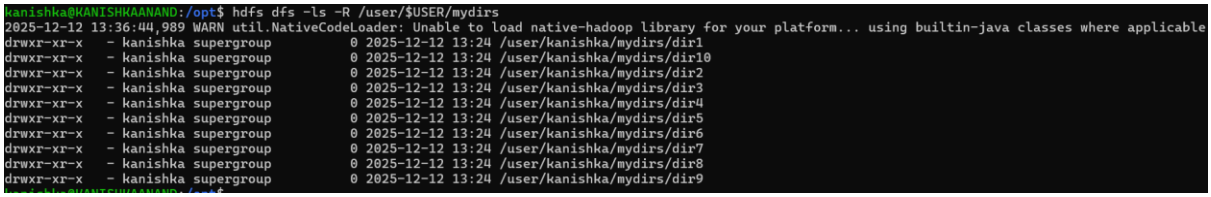
demonstrates downloading a file from HDFS to the local file system using the hdfs dfs -get command, confirming data retrieval from HDFS.

21. HDFS File Operations (rm)



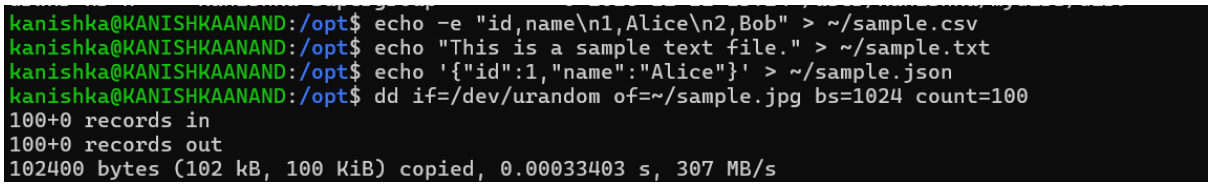
shows deletion of a file from HDFS using the hdfs dfs -rm command, confirming successful file removal.

22. HDFS File Operations (move/rename)



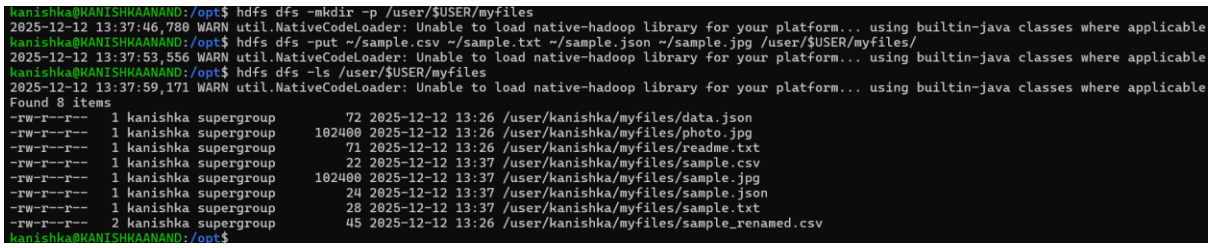
demonstrates renaming a file inside HDFS using the hdfs dfs -mv command, verifying file movement and renaming functionality.

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

```
kanishka@KANISHKAANAND:/opt$ hdfs dfs -put ~/sample.txt /user/$USER/myfiles/sample_copy.txt
2025-12-12 13:39:37,886 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt$ hdfs dfs -get /user/$USER/myfiles/sample.csv ~/downloaded_sample.csv
2025-12-12 13:39:43,401 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt$ hdfs dfs -rm /user/$USER/myfiles/sample_copy.txt
2025-12-12 13:39:49,038 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Deleted /user/kanishka/myfiles/sample_copy.txt
kanishka@KANISHKAANAND:/opt$ hdfs dfs -mv /user/$USER/myfiles/sample.csv /user/$USER/myfiles/sample_renamed.csv
2025-12-12 13:39:54,493 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
mv: `/user/kanishka/myfiles/sample_renamed.csv': File exists
kanishka@KANISHKAANAND:/opt$ hdfs dfs -du -h /user/$USER/myfiles
2025-12-12 13:39:59,371 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
72      72      /user/kanishka/myfiles/data.json
100 K   100 K   /user/kanishka/myfiles/photo.jpg
71      71      /user/kanishka/myfiles/readme.txt
22      22      /user/kanishka/myfiles/sample.csv
100 K   100 K   /user/kanishka/myfiles/sample.jpg
24      24      /user/kanishka/myfiles/sample.json
28      28      /user/kanishka/myfiles/sample.txt
45      90      /user/kanishka/myfiles/sample_renamed.csv
kanishka@KANISHKAANAND:/opt$ hdfs dfs -df -h
2025-12-12 13:40:07,940 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Filesystem                Size     Used  Available  Use%
hdfs://localhost:9000  1006.9 G  264.9 K    945.6 G    0%
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -setrep -w 2 /user/$USER/myfiles/sample_renamed.csv
2025-12-16 10:53:44,104 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
setrep: Cannot set replication for /user/kanishka/myfiles/sample_renamed.csv. Name node is in safe mode.
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -ls /user/$USER/myfiles/sample_renamed.csv
2025-12-16 10:53:52,164 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
-rw-r--r--   2 kanishka supergroup         45 2025-12-12 13:26 /user/kanishka/myfiles/sample_renamed.csv
```

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

28. Block & Replica Information

```
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs fsck /user/$USER/myfiles/sample_renamed.csv -files -blocks -locations
2025-12-16 10:53:57,922 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
Connecting to namenode via http://localhost:9870/fsck?ugi=kanishka&files=1&blocks=1&locations=1&path=%2Fuser%2Fkanishka%2Fmyfiles%2Fsample_renam
ed.csv
FSCK started by kanishka (auth:SIMPLE) from /127.0.0.1 for path /user/kanishka/myfiles/sample_renamed.csv at Tue Dec 16 10:53:58 UTC 2025

/user/kanishka/myfiles/sample_renamed.csv 45 bytes, replicated: replication=2, 1 block(s):  Under replicated BP-638265555-127.0.1.1-176554573136
1:blk_1073741825_1001. Target Replicas is 2 but found 1 live replica(s), 0 decommissioned replica(s), 0 decommissioning replica(s).
0. BP-638265555-127.0.1.1-1765545731361:blk_1073741825_1001 len=45 Live_repl=1  [DatanodeInfoWithStorage[127.0.0.1:9866,DS-6fa9ca5b-0639-49bf-b6
de-ed10b6f678cb,DISK]]
```

```
Status: HEALTHY
 Number of data-nodes:  1
 Number of racks:               1
 Total dirs:                    0
 Total symlinks:                0

Replicated Blocks:
 Total size:    45 B
 Total files:   1
 Total blocks (validated):      1 (avg. block size 45 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       1 (100.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              1 (50.0 %)
 Blocks queued for replication: 0

Erasure Coded Block Groups:
 Total size:    0 B
 Total files:   0
 Total block groups (validated):        0
 Minimally erasure-coded block groups:  0
 Over-erasure-coded block groups:       0
 Under-erasure-coded block groups:      0
 Unsatisfactory placement block groups: 0
 Average block group size:      0.0
 Missing block groups:          0
 Corrupt block groups:          0
 Missing internal blocks:       0
 Blocks queued for replication: 0
FSCK ended at Tue Dec 16 10:53:58 UTC 2025 in 8 milliseconds


The filesystem under path '/user/kanishka/myfiles/sample_renamed.csv' is HEALTHY
kanishka@KANISHKAANAND:/opt/hadoop$
```

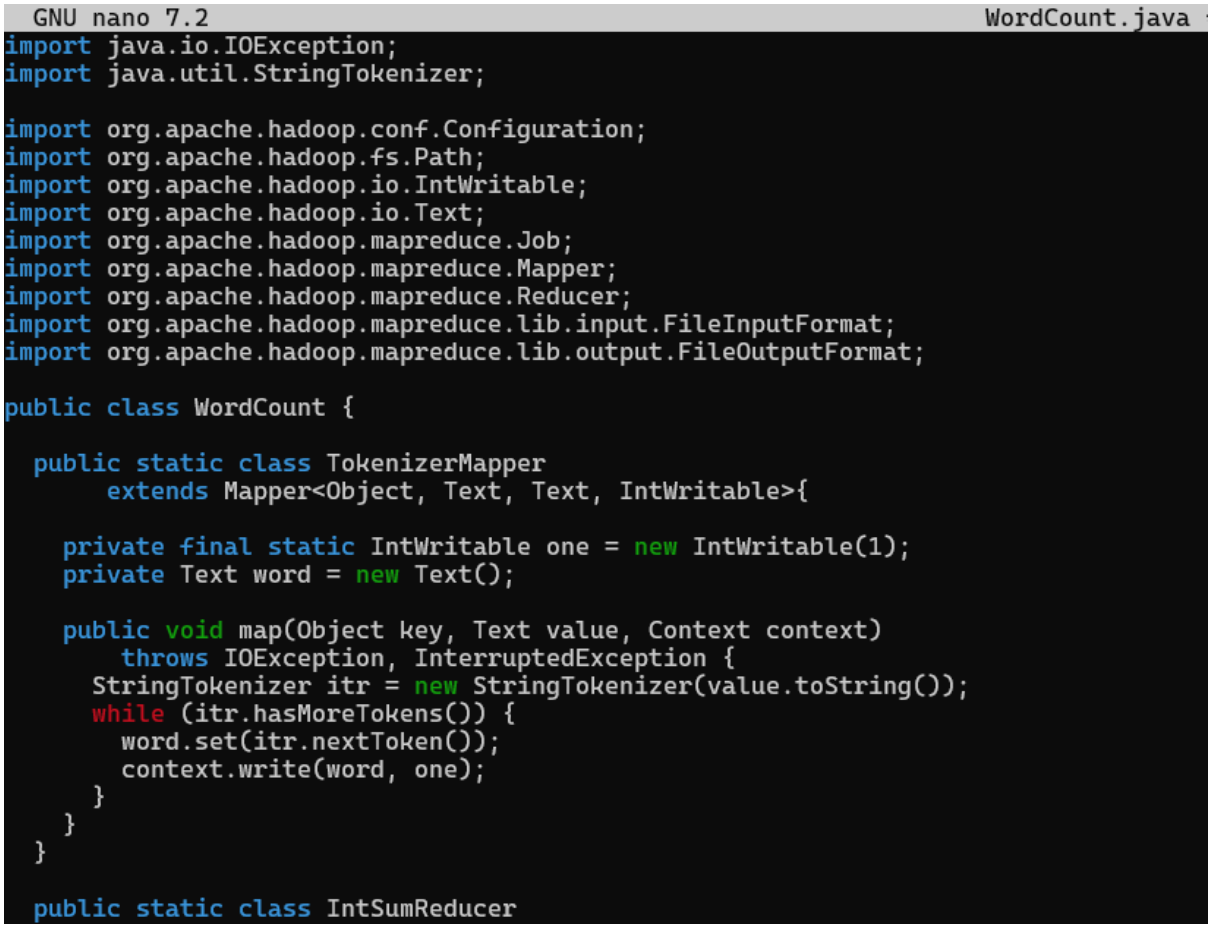
displays HDFS block information using the fsck command, showing block ID, replica count, and DataNode location, verifying block placement in HDFS.

29. Editing WordCount Program

```
kanishka@KANISHKAANAND:/opt/hadoop$ nano WordCount.java
```

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

30. WordCount MapReduce Program



```
  GNU nano 7.2                                                  WordCount.java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ javac -classpath `hadoop classpath` -d wc_classes WordCount.java
kanishka@KANISHKAANAND:/opt/hadoop$ jar -cvf wordcount.jar -C wc_classes/ .
added manifest
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 764)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 825)(deflated 45%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 749)(deflated 57%)
kanishka@KANISHKAANAND:/opt/hadoop$
```

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-

```
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -mkdir -p /user/$USER/wcinput
2025-12-16 10:58:37,358 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -put /user/$USER/myfiles/sample.txt /user/$USER/wcinput
2025-12-16 10:58:44,519 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
put: `/user/kanishka/myfiles/sample.txt': No such file or directory
kanishka@KANISHKAANAND:/opt/hadoop$ hadoop jar wordcount.jar WordCount /user/$USER/wcinput /user/$USER/wcoutput
2025-12-16 10:58:56,615 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
2025-12-16 10:58:57,017 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2025-12-16 10:58:57,076 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2025-12-16 10:58:57,076 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2025-12-16 10:58:57,186 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and e
xecute your application with ToolRunner to remedy this.
2025-12-16 10:58:57,294 INFO input.FileInputFormat: Total input files to process : 0
2025-12-16 10:58:57,321 INFO mapreduce.JobSubmitter: number of splits:0
2025-12-16 10:58:57,512 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local825947907_0001
2025-12-16 10:58:57,513 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-12-16 10:58:57,657 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2025-12-16 10:58:57,660 INFO mapreduce.Job: Running job: job_local825947907_0001
2025-12-16 10:58:57,662 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2025-12-16 10:58:57,676 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2025-12-16 10:58:57,677 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2025-12-16 10:58:57,677 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignor
e cleanup failures: false
2025-12-16 10:58:57,678 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2025-12-16 10:58:57,718 INFO mapred.LocalJobRunner: Waiting for map tasks
2025-12-16 10:58:57,718 INFO mapred.LocalJobRunner: map task executor complete.
2025-12-16 10:58:57,724 INFO mapred.LocalJobRunner: Waiting for reduce tasks
2025-12-16 10:58:57,725 INFO mapred.LocalJobRunner: Starting task: attempt_local825947907_0001_r_000000_0
2025-12-16 10:58:57,748 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2025-12-16 10:58:57,749 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2025-12-16 10:58:57,749 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignor
e cleanup failures: false
```

```
output
2025-12-16 10:58:57,879 INFO mapred.LocalJobRunner: reduce > reduce
2025-12-16 10:58:57,880 INFO mapred.Task: Task 'attempt_local825947907_0001_r_000000_0' done.
2025-12-16 10:58:57,883 INFO mapred.Task: Final Counters for attempt_local825947907_0001_r_000000_0: Counters: 30
        File System Counters
                FILE: Number of bytes read=3114
                FILE: Number of bytes written=642272
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=0
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Combine input records=0
                Combine output records=0
                Reduce input groups=0
                Reduce shuffle bytes=0
                Reduce input records=0
                Reduce output records=0
                Spilled Records=0
                Shuffled Maps =0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=6
                Total committed heap usage (bytes)=174063616
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=0
2025-12-16 10:58:57,883 INFO mapred.LocalJobRunner: Finishing task: attempt_local825947907_0001_r_000000_0
2025-12-16 10:58:57,884 INFO mapred.LocalJobRunner: reduce task executor complete.
```

```
2025-12-16 10:58:57,883 INFO mapred.LocalJobRunner: Finishing task: attempt_local825947907_0001_r_000000_0
2025-12-16 10:58:57,884 INFO mapred.LocalJobRunner: reduce task executor complete.
2025-12-16 10:58:58,670 INFO mapreduce.Job: Job job_local825947907_0001 running in uber mode : false
2025-12-16 10:58:58,671 INFO mapreduce.Job:  map 0% reduce 100%
2025-12-16 10:58:58,672 INFO mapreduce.Job: Job job_local825947907_0001 completed successfully
2025-12-16 10:58:58,677 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=3114
                FILE: Number of bytes written=642272
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=0
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Combine input records=0
                Combine output records=0
                Reduce input groups=0
                Reduce shuffle bytes=0
                Reduce input records=0
                Reduce output records=0
                Spilled Records=0
                Shuffled Maps =0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=6
                Total committed heap usage (bytes)=174063616
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=0
```

```
                Merged Map outputs=1
                GC time elapsed (ms)=5
                Total committed heap usage (bytes)=541065216
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=28
        File Output Format Counters
                Bytes Written=24
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -cat /user/$USER/wcoutput_stopwords/part-r-00000
2025-12-16 11:21:19,852 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
asses where applicable
file.   1
sample  1
text    1
kanishka@KANISHKAANAND:/opt/hadoop$
```

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

```
        File Output Format Counters
                Bytes Written=24
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -cat /user/$USER/wcoutput_stopwords/part-r-00000
2025-12-16 11:21:19,852 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
file.   1
sample  1
text    1
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -cp /user/$USER/myfiles/*.txt /user/$USER/wcinput/
2025-12-16 11:23:17,868 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
cp: `/user/kanishka/wcinput/sample.txt': File exists
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -ls /user/$USER/wcinput
2025-12-16 11:23:24,396 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Found 2 items
-rw-r--r--   1 kanishka supergroup         71 2025-12-16 11:23 /user/kanishka/wcinput/readme.txt
-rw-r--r--   1 kanishka supergroup         28 2025-12-16 11:20 /user/kanishka/wcinput/sample.txt
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -rm -r /user/$USER/wcoutput_multi
2025-12-16 11:23:29,870 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
rm: `/user/kanishka/wcoutput_multi': No such file or directory
kanishka@KANISHKAANAND:/opt/hadoop$ hadoop jar wordcount.jar WordCount /user/$USER/wcinput /user/$USER/wcoutput_multi
2025-12-16 11:23:35,045 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2025-12-16 11:23:35,504 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2025-12-16 11:23:35,605 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2025-12-16 11:23:35,605 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2025-12-16 11:23:35,716 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool in
```

```
                Bytes Written=74
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -cat /user/$USER/wcoutput_multi/part-r-00000
2025-12-16 11:23:41,545 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
file    1
file.   2
for     1
hdfs    1
line    1
sample  2
test.   1
text    2
two     1
upload  1
kanishka@KANISHKAANAND:/opt/hadoop$
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ cd /opt/hadoop
sbin/stop-dfs.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [KANISHKAANAND]
2025-12-16 11:36:06,223 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
kanishka@KANISHKAANAND:/opt/hadoop$ jps
9443 Jps
kanishka@KANISHKAANAND:/opt/hadoop$ rm -rf /home/kanishka/hadoopdata
rm -rf /tmp/hadoop-*
kanishka@KANISHKAANAND:/opt/hadoop$ nano /opt/hadoop/etc/hadoop/core-site.xml
kanishka@KANISHKAANAND:/opt/hadoop$ nano /opt/hadoop/etc/hadoop/hdfs-site.xml
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs namenode -format
2025-12-16 11:37:27,293 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = KANISHKAANAND/127.0.1.1
STARTUP_MSG:   args = [-format]
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [KANISHKAANAND]
2025-12-16 11:37:43,324 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
kanishka@KANISHKAANAND:/opt/hadoop$ jps
9808 DataNode
10049 SecondaryNameNode
9672 NameNode
10175 Jps
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -ls /
2025-12-16 11:37:53,569 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
kanishka@KANISHKAANAND:/opt/hadoop$
```

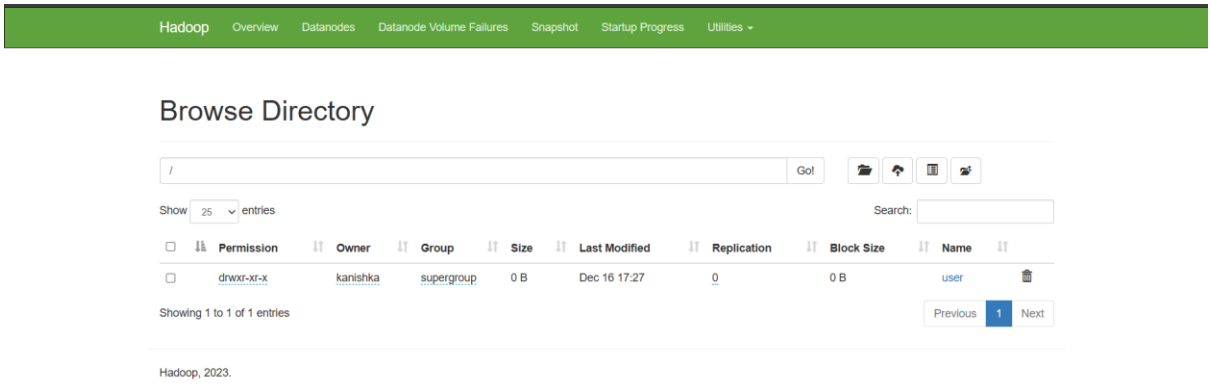
```
kanishka@KANISHKAANAND:/opt/hadoop$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [KANISHKAANAND]
2025-12-16 11:56:51,691 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt/hadoop$ jps
15987 NameNode
16126 DataNode
16367 SecondaryNameNode
16495 Jps
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -ls /
2025-12-16 11:57:01,353 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
kanishka@KANISHKAANAND:/opt/hadoop$ hdfs dfs -mkdir -p /user/kanishka
hdfs dfs -ls /user
2025-12-16 11:57:08,508 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2025-12-16 11:57:09,902 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
drwxr-xr-x   - kanishka supergroup          0 2025-12-16 11:57 /user/kanishka
kanishka@KANISHKAANAND:/opt/hadoop$
```

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

```
kanishka@KANISHKAANAND:/opt/hadoop$ dd if=/dev/urandom of=~/testfile.dat bs=1M count=50
50+0 records in
50+0 records out
52428800 bytes (52 MB, 50 MiB) copied, 0.125935 s, 416 MB/s
kanishka@KANISHKAANAND:/opt/hadoop$ time cp ~/testfile.dat ~/testfile_local.dat

real    0m0.049s
user    0m0.000s
sys     0m0.036s
kanishka@KANISHKAANAND:/opt/hadoop$ time cp ~/testfile.dat ~/testfile_local.dat

real    0m0.060s
user    0m0.000s
sys     0m0.038s
kanishka@KANISHKAANAND:/opt/hadoop$
```

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:

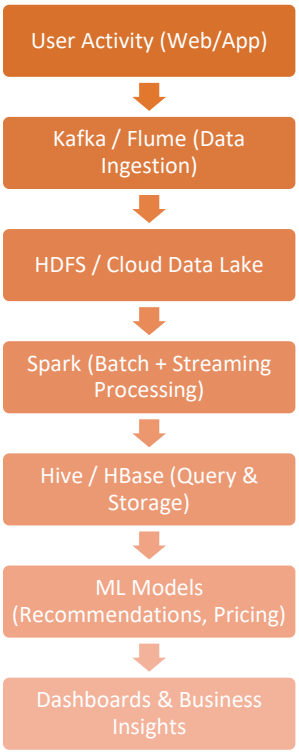
- Handling **huge volumes of user clickstream data**
- Providing **real-time product recommendations**
- Managing **dynamic pricing during sales**
- Detecting **fraudulent transactions**
- Forecasting **demand and inventory**

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

2. **Storage**

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

3. **Processing**

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

4. **Query & Access**

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

5. **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

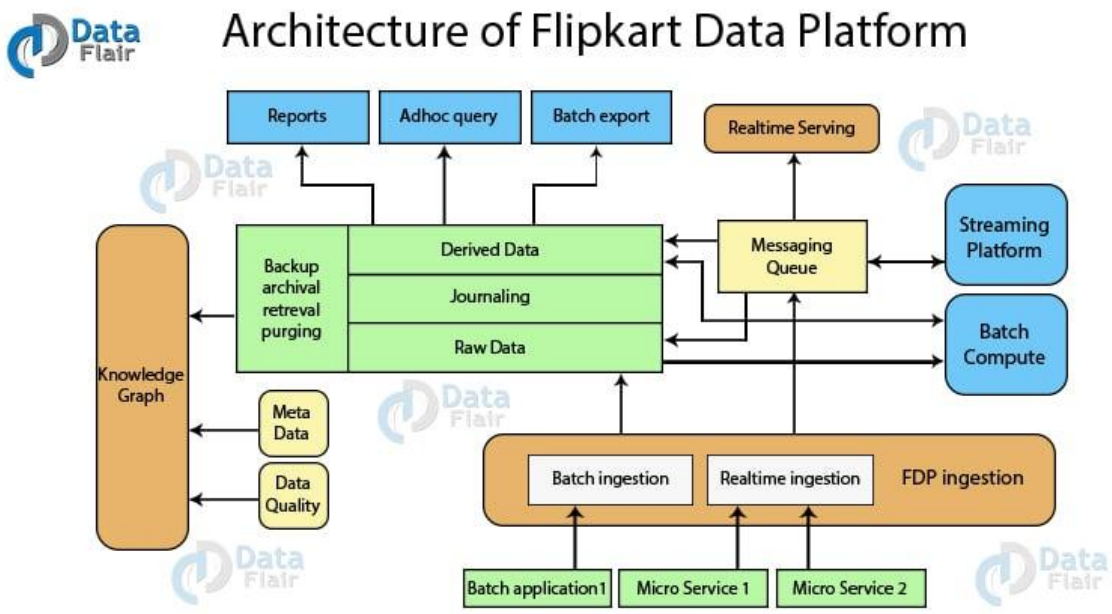
- Predicts demand
- Optimizes warehouse stocking
- Reduces delivery time

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

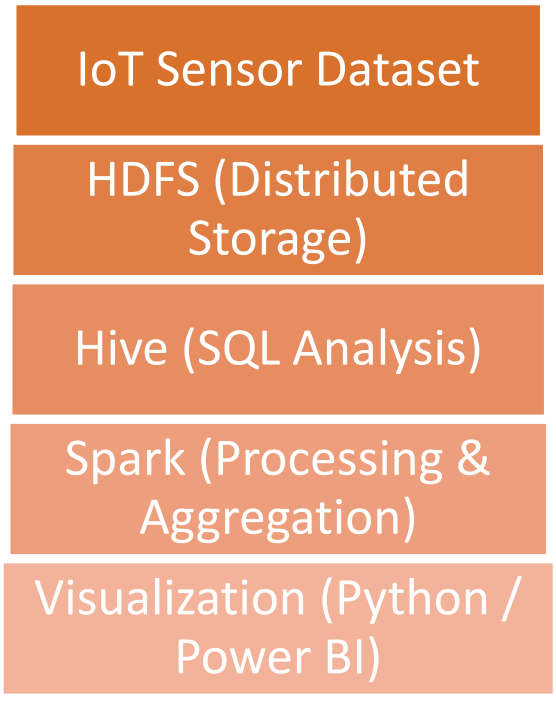
Dataset Description

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)
```

```
df.sh...     inferSchema=True
... )
ow(5)
df.printSchema()

>>>
>>> df.show(5)
+-----------------+----------+-----------------+-----------------+-----+-------------------+------+-------------------+-----------------+
|               ts|    device|               co|         humidity|light|                lpg|motion|              smoke|             temp|
+-----------------+----------+-----------------+-----------------+-----+-------------------+------+-------------------+-----------------+
|1.5945120943859746E9|b8:27:eb:bf:9d:51|0.004955938648391245|             51.0|false| 0.007650822270557 19| false| 0.020411270122412 92|             22.7|
|1.5945120947355676E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0|false|0.005114383400977071| false|0.013274836704851536|19.700000762939453|
|1.5945120980735729E9|b8:27:eb:bf:9d:51|0.004976012340421658|             50.9|false|0.007673227406398091| false| 0.020475125576178 24|             22.6|
| 1.594512099589146E9|1c:bf:ce:15:ec:4d|0.004403026829699689|76.80000305175781| true|0.007023337145877314| false|0.018628225377018803|             27.0|
| 1.594512101761235E9|b8:27:eb:bf:9d:51|0.004967363641908952|             50.9|false|0.007663577282372411| false|0.020447620810233658|             22.6|
+-----------------+----------+-----------------+-----------------+-----+-------------------+------+-------------------+-----------------+
only showing top 5 rows
```

Average temperature per location

avg_temp = df.groupBy("location").avg("temperature")

avg_temp.show()

```
>>> avg_temp_df.show()
+-----------------+------------------+
|           device|          avg_temp|
+-----------------+------------------+
|00:0f:00:70:91:0a| 19.36255246911829|
|b8:27:eb:bf:9d:51|22.279969165275187|
|1c:bf:ce:15:ec:4d|26.025511285357403|
+-----------------+------------------+
```

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)

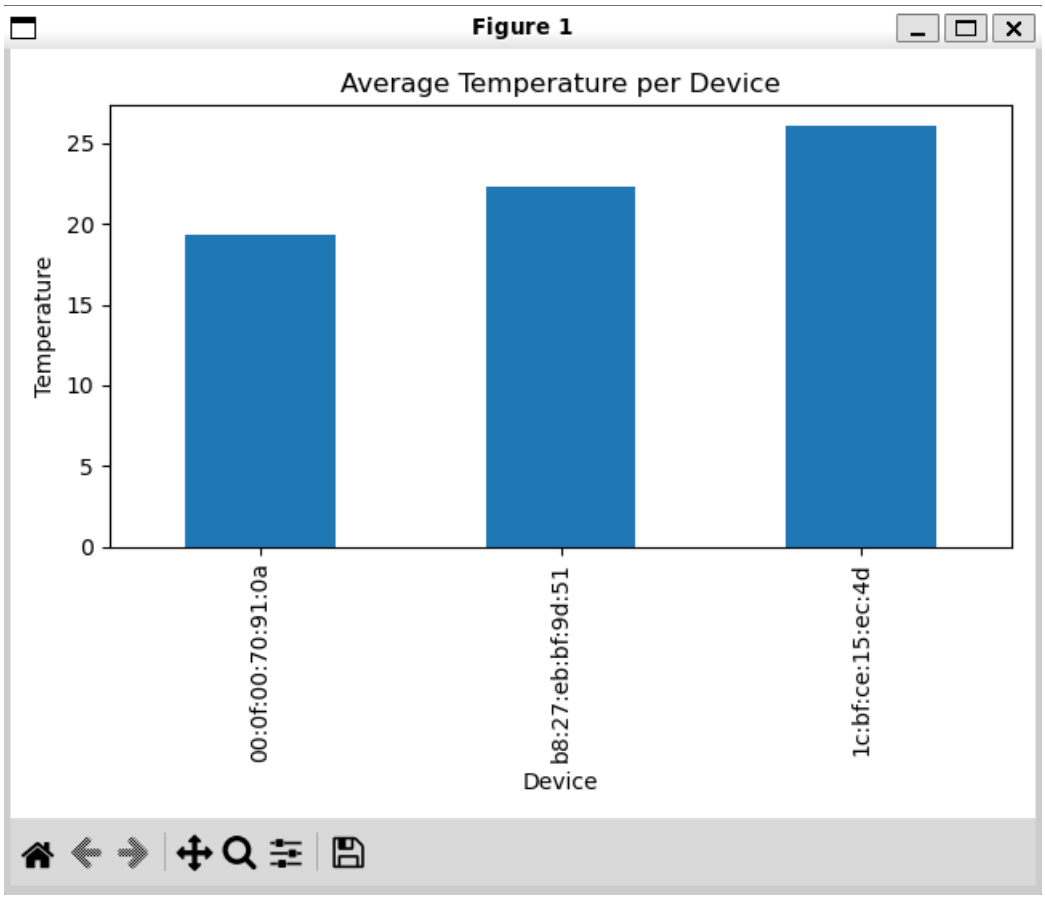
import pandas as pd

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.

```
spark.sql("CREATE DATABASE IF NOT EXISTS iot_db")
spark.sql("USE iot_db")
```

```
DataFrame[]
```

External table -1

```
spark.sql("""
CREATE TABLE IF NOT EXISTS sensors_main (
  ts STRING,
  device STRING,
  co DOUBLE,
  humidity DOUBLE,
  light DOUBLE,
  lpg DOUBLE,
  motion INT,
  smoke DOUBLE,
  temp DOUBLE
)
USING CSV
OPTIONS (
  path '/content/iot_telemetry_data.csv',
  header 'true'
);
""")
```

```
DataFrame[]
```

External Table 2

```
spark.sql("""
CREATE TABLE IF NOT EXISTS sensors_temperature (
  ts STRING,
  device STRING,
  temp DOUBLE,
  humidity DOUBLE
)
USING CSV
OPTIONS (
  path '/content/iot_telemetry_data.csv',
  header 'true'
);
""")
```

```
DataFrame[]
```

External Table 3

```
spark.sql("""
CREATE TABLE IF NOT EXISTS sensors_air (
  ts STRING,
  device STRING,
  co DOUBLE,
  lpg DOUBLE,
  smoke DOUBLE
)
USING CSV
OPTIONS (
  path '/content/iot_telemetry_data.csv',
  header 'true'
);
""")
```

```
DataFrame[]
```

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

```
spark.sql("SELECT COUNT(*) FROM sensors_main").show()
```

```
+--------+
|count(1)|
+--------+
|  405184|
+--------+
```

Que.3. Perform:

- **GROUP BY**

```
spark.sql("""
SELECT device, AVG(temp) AS avg_temp
FROM sensors_main
GROUP BY device
""").show()
```

```
+-----------------+------------------+
|           device|          avg_temp|
+-----------------+------------------+
|00:0f:00:70:91:0a| 19.36255246911829|
|b8:27:eb:bf:9d:51|22.279969165275897|
|1c:bf:ce:15:ec:4d|26.025511285357403|
+-----------------+------------------+
```

- **ORDER BY**

```
spark.sql("""
SELECT device, AVG(temp) AS avg_temp
FROM sensors_main
GROUP BY device
ORDER BY avg_temp DESC
""").show()
```

```
+-----------------+------------------+
|           device|          avg_temp|
+-----------------+------------------+
|1c:bf:ce:15:ec:4d|26.025511285357403|
|b8:27:eb:bf:9d:51|22.279969165275897|
|00:0f:00:70:91:0a| 19.36255246911829|
+-----------------+------------------+
```

- **CLUSTER BY**

```
SELECT * FROM sensors_main
CLUSTER BY device
""").show()
```

```
+-------------------+----------------+-------------------+-----------------+-----+--------------------+------+--------------------
|                 ts|          device|                 co|         humidity|light|                 lpg|motion|                smok
+-------------------+----------------+-------------------+-----------------+-----+--------------------+------+--------------------
|1.5945120947355676E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
| 1.594512106869076E9|00:0f:00:70:91:0a|0.002938115626660...|             76.0| NULL|0.005241481841731117|  NULL|0.01362752113201919
|1.5945121227857318E9|00:0f:00:70:91:0a|0.002905014756555...|75.80000305175781| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121314763765E9|00:0f:00:70:91:0a|0.002938115626660...|75.80000305175781| NULL|0.005241481841731117|  NULL|0.01362752113201919
|1.5945121352890863E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121396414526E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121488186827E9|00:0f:00:70:91:0a|0.002905014756555...| 75.9000015258789| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121531727257E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121591454568E9|00:0f:00:70:91:0a|0.002872341154862943|             76.0| NULL|0.005156332935627952|  NULL|0.01339117678217600
|1.5945121672564228E9|00:0f:00:70:91:0a|0.002905014756555...| 75.9000015258789| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121753714526E9|00:0f:00:70:91:0a|0.002872341154862943|             76.0| NULL|0.005156332935627952|  NULL|0.01339117678217600
|1.5945121807944636E9|00:0f:00:70:91:0a|0.002840088607101...| 75.9000015258789| NULL|0.005114383400977071|  NULL|0.01327483670485153
| 1.594512184619436E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121932924936E9|00:0f:00:70:91:0a|0.002905014756555...| 76.0999984741211| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122003289607E9|00:0f:00:70:91:0a|0.002840088607101...|             76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945122057552145E9|00:0f:00:70:91:0a|0.002880499447453191|             76.0| NULL|0.005166923396135935|  NULL|0.01342055789269630
| 1.594512211184731E9|00:0f:00:70:91:0a|0.002905014756555...|             76.0| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122160754442E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945122225777857E9|00:0f:00:70:91:0a|0.002905014756555...|             76.0| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122269381864E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
+-------------------+----------------+-------------------+-----------------+-----+--------------------+------+--------------------
only showing top 20 rows
```

- **DISTRIBUTE BY**

```
spark.sql("""
SELECT * FROM sensors_main
DISTRIBUTE BY device
""").show()
```

```
+-------------------+---------------+--------------------+----------------+-----+-------------------+------+-------------------+
|                 ts|         device|                  co|        humidity|light|                lpg|motion|               smok
+-------------------+---------------+--------------------+----------------+-----+-------------------+------+-------------------+
|1.5945120947355676E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
| 1.594512106869076E9|00:0f:00:70:91:0a|0.002938115626660...|            76.0| NULL|0.005241481841731117|  NULL|0.01362752113201919
|1.5945121227857318E9|00:0f:00:70:91:0a|0.002905014756555...|75.80000305175781| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121314763765E9|00:0f:00:70:91:0a|0.002938115626660...|75.80000305175781| NULL|0.005241481841731117|  NULL|0.01362752113201919
|1.5945121352890863E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121396414526E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121488186827E9|00:0f:00:70:91:0a|0.002905014756555...| 75.9000015258789| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121531727257E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121591454568E9|00:0f:00:70:91:0a|0.002872341154862943|            76.0| NULL|0.005156332935627952|  NULL|0.01339117678217600
|1.5945121672564228E9|00:0f:00:70:91:0a|0.002905014756555...| 75.9000015258789| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945121753714526E9|00:0f:00:70:91:0a|0.002872341154862943|            76.0| NULL|0.005156332935627952|  NULL|0.01339117678217600
|1.5945121807944636E9|00:0f:00:70:91:0a|0.002840088607101...| 75.9000015258789| NULL|0.005114383400977071|  NULL|0.01327483670485153
| 1.594512184619436E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945121932924936E9|00:0f:00:70:91:0a|0.002905014756555...| 76.0999984741211| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122003289607E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945122057552145E9|00:0f:00:70:91:0a|0.002880499447453191|            76.0| NULL|0.005166923396135935|  NULL|0.01342055789269630
| 1.594512211184731E9|00:0f:00:70:91:0a|0.002905014756555...|            76.0| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122160754442E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
|1.5945122225777857E9|00:0f:00:70:91:0a|0.002905014756555...|            76.0| NULL|0.005198697479294309|  NULL|0.01350873332955624
|1.5945122269381864E9|00:0f:00:70:91:0a|0.002840088607101...|76.19999694824219| NULL|0.005114383400977071|  NULL|0.01327483670485153
+-------------------+---------------+--------------------+----------------+-----+-------------------+------+-------------------+
```

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

```
spark.sql("""
CREATE TABLE sensors_partitioned (
  ts STRING,
  device STRING,
  temp DOUBLE
)
USING CSV
PARTITIONED BY (year STRING);
""")
```

```
DataFrame[]
```

```
spark.sql("""
INSERT INTO sensors_partitioned
PARTITION (year='2023')
SELECT ts, device, temp
FROM sensors_main;
""")
```

```
DataFrame[]
```

Que 5. Create Bucketed Table and perform join.

```
spark.sql("""
CREATE TABLE sensors_bucketed (
  device STRING,
  temp DOUBLE
)
CLUSTERED BY (device) INTO 4 BUCKETS;
""")
```

```
DataFrame[]
```

```
spark.sql("""
INSERT INTO sensors_bucketed
SELECT device, temp FROM sensors_main;
""")
```

```
DataFrame[]
```

```
FROM sensors_main a
JOIN sensors_bucketed b
ON a.device = b.device
""").show()
```

```
+-----------------+----+
|           device|temp|
+-----------------+----+
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
|b8:27:eb:bf:9d:51|22.7|
+-----------------+----+
only showing top 20 rows
```

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

```
# 1. Maximum temperature per device
spark.sql("SELECT device, MAX(temp) AS max_temp FROM sensors_main GROUP BY device").show()

# 2. Devices with high temperature and low humidity
spark.sql("SELECT * FROM sensors_main WHERE temp > 30 AND humidity < 50").show()

# 3. Devices having more than 1000 records
spark.sql("SELECT device, COUNT(*) AS record_count FROM sensors_main GROUP BY device HAVING COUNT(*) > 1000").show()

# 4. Average temperature where humidity > 60
spark.sql("SELECT device, AVG(temp) AS avg_temp FROM sensors_main WHERE humidity > 60 GROUP BY device").show()

# 5. Top 5 highest temperature readings
spark.sql("SELECT device, temp FROM sensors_main ORDER BY temp DESC LIMIT 5").show()
```

```
+----------------+-----------------+
|          device|         max_temp|
+----------------+-----------------+
|00:0f:00:70:91:0a|20.200000762939453|
|b8:27:eb:bf:9d:51|             24.1|
|1c:bf:ce:15:ec:4d|30.600000381469727|
+----------------+-----------------+

+-------------------+----------------+-------------------+-----------------+-----+-------------------+------+-----------------
|                 ts|          device|                 co|         humidity|light|                lpg|motion|              smo
+-------------------+----------------+-------------------+-----------------+-----+-------------------+------+-----------------
|1.5951069668590875E9|1c:bf:ce:15:ec:4d|0.004492535123085711|2.9000000953674316| NULL|0.007126334076582767|  NULL|0.0189201787637720
+-------------------+----------------+-------------------+-----------------+-----+-------------------+------+-----------------
```

```
|1.5951069668590875E9|1c:bf:ce:15:ec:4d|0.004492535123085711|2.9000000953674316| NULL|0.007126334076582767|  NULL|0.0189201787637720
+-------------------+----------------+-------------------+-----------------+-----+-------------------+------+-----------------

+----------------+------------+
|          device|record_count|
+----------------+------------+
|00:0f:00:70:91:0a|      111815|
|b8:27:eb:bf:9d:51|      187451|
|1c:bf:ce:15:ec:4d|      105918|
+----------------+------------+

+----------------+-----------------+
|          device|         avg_temp|
+----------------+-----------------+
|00:0f:00:70:91:0a|19.36384146238063|
|1c:bf:ce:15:ec:4d|25.500870994636674|
|b8:27:eb:bf:9d:51|22.42485207100592|
+----------------+-----------------+

+----------------+-----------------+
|          device|             temp|
+----------------+-----------------+
|1c:bf:ce:15:ec:4d|30.600000381469727|
|1c:bf:ce:15:ec:4d|30.600000381469727|
|1c:bf:ce:15:ec:4d|30.600000381469727|
|1c:bf:ce:15:ec:4d|30.600000381469727|
|1c:bf:ce:15:ec:4d|30.600000381469727|
+----------------+-----------------+
```

Que 7. Compare Hive vs MySQL query execution time.

```
spark.sql("""
SELECT device, AVG(temp)
FROM sensors_main
GROUP BY device
""").show()
```

```
+----------------+-----------------+
|          device|         avg(temp)|
+----------------+-----------------+
|00:0f:00:70:91:0a|19.36255246911829|
|b8:27:eb:bf:9d:51|22.279969165275897|
|1c:bf:ce:15:ec:4d|26.025511285357403|
+----------------+-----------------+
```

```
spark.sql("""
SELECT device, AVG(temp)
FROM sensors_main
GROUP BY device
""").show()
```

```
+-----------------+------------------+
|           device|         avg(temp)|
+-----------------+------------------+
|00:0f:00:70:91:0a| 19.36255246911829|
|b8:27:eb:bf:9d:51|22.279969165275897|
|1c:bf:ce:15:ec:4d|26.025511285357403|
+-----------------+------------------+
```

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

```
spark = SparkSession.builder \
    .appName("HiveSparkMode") \
    .enableHiveSupport() \
    .getOrCreate()
```

```
spark.sql("SHOW TABLES").show()
```

```
+---------+------------------+-----------+
|namespace|         tableName|isTemporary|
+---------+------------------+-----------+
|   iot_db|           sensors|      false|
|   iot_db|       sensors_air|      false|
|   iot_db|  sensors_bucketed|      false|
|   iot_db|      sensors_main|      false|
|   iot_db|sensors_partitioned|      false|
|   iot_db|sensors_temperature|      false|
|         |          iot_data|       true|
+---------+------------------+-----------+
```

APACHE SPARK & PySpark TASKS

29. Install Spark & run in Standalone Mode.

```
spark
```

SparkSession - hive

SparkContext

Spark UI

Version
v4.0.1
Master
local[*]
AppName
HiveAssignment

30. Create RDD using:

- **Parallelize**

```
rdd1 = sc.parallelize([1, 2, 3, 4, 5])
rdd1.collect()
```

```
[1, 2, 3, 4, 5]
```

- **External file**

```
rdd2 = sc.textFile("/content/iot_telemetry_data.csv")
rdd2.take(5)
```

```
['"ts","device","co","humidity","light","lpg","motion","smoke","temp"',
'"1.5945120943859746E9","b8:27:eb:bf:9d:51","0.004955938648391245","51.0","false","0.0076508222705571 9","false","0.02041127012241292
'"1.5945120947355676E9","00:0f:00:70:91:0a","0.0028400886071015706","76.0","false","0.005114383400977071","false","0.013274836704851
'"1.5945120980735729E9","b8:27:eb:bf:9d:51","0.004976012340421658","50.9","false","0.007673227406398091","false","0.0204751255761782
'"1.594512099589146E9","1c:bf:ce:15:ec:4d","0.004403026829699689","76.80000305175781","true","0.007023337145877314","false","0.01862
```

31. Perform:

- **map**

```
rdd_map = rdd1.map(lambda x: x * 2)
rdd_map.collect()
```

```
[2, 4, 6, 8, 10]
```

- **filter**

```
rdd_filter = rdd1.filter(lambda x: x % 2 == 0)
rdd_filter.collect()
```

```
[2, 4]
```

- **reduce**

```
rdd_reduce = rdd1.reduce(lambda a, b: a + b)
rdd_reduce
```

```
15
```

- **flatMap**

```
rdd_flatmap = sc.parallelize(["hello world", "spark rdd"]).flatMap(lambda x: x.split())
rdd_flatmap.collect()
```

```
['hello', 'world', 'spark', 'rdd']
```

32. Load CSV using Spark DataFrame.

```
df = spark.read.csv(
    "/iot_telemetry_data.csv",
    header=True,
    inferSchema=True
)

df.show(5)
df.printSchema()
```

```
+----------------+----------------+-------------------+----------------+-----+-------------------+------+--------------------
|              ts|          device|                 co|        humidity|light|                lpg|motion|               smoke
+----------------+----------------+-------------------+----------------+-----+-------------------+------+--------------------
|1.5945120943859746E9|b8:27:eb:bf:9d:51|0.004955938648391245|            51.0|false| 0.00765082227055719| false| 0.02041127012241292
|1.5945120947355676E9|00:0f:00:70:91:0a|0.002840088607101...|            76.0|false|0.005114383400977071| false|0.013274836704851536
|1.5945120980735729E9|b8:27:eb:bf:9d:51|0.004976012340421658|            50.9|false|0.007673227406398091| false| 0.02047512557617824
| 1.594512099589146E9|1c:bf:ce:15:ec:4d|0.004403026829699689|76.80000305175781| true|0.007023337145877314| false|0.018628225377018803
| 1.594512101761235E9|b8:27:eb:bf:9d:51|0.004967363641908952|            50.9|false|0.007663577282372411| false|0.020447620810233658
+----------------+----------------+-------------------+----------------+-----+-------------------+------+--------------------
only showing top 5 rows
root
 |-- ts: double (nullable = true)
 |-- device: string (nullable = true)
 |-- co: double (nullable = true)
 |-- humidity: double (nullable = true)
 |-- light: boolean (nullable = true)
 |-- lpg: double (nullable = true)
 |-- motion: boolean (nullable = true)
 |-- smoke: double (nullable = true)
```

33. Perform:

- **groupBy and agg**

```
from pyspark.sql.functions import avg

df.groupBy("device") \
  .agg(avg("temp").alias("avg_temp")) \
  .show()
```

```
+-----------------+-----------------+
|           device|         avg_temp|
+-----------------+-----------------+
|00:0f:00:70:91:0a|19.33904213634782|
|b8:27:eb:bf:9d:51|22.15932783887716|
|1c:bf:ce:15:ec:4d|25.34140694343986|
+-----------------+-----------------+
```

- **withColumn**

```
from pyspark.sql.functions import col

df2 = df.withColumn("temp_fahrenheit", col("temp") * 9/5 + 32)
df2.select("device", "temp", "temp_fahrenheit").show(5)
```

```
+-----------------+------------------+-----------------+
|           device|              temp|  temp_fahrenheit|
+-----------------+------------------+-----------------+
|b8:27:eb:bf:9d:51|              22.7|            72.86|
|00:0f:00:70:91:0a|19.700000762939453|67.46000137329102|
|b8:27:eb:bf:9d:51|              22.6|            72.68|
|1c:bf:ce:15:ec:4d|              27.0|             80.6|
|b8:27:eb:bf:9d:51|              22.6|            72.68|
+-----------------+------------------+-----------------+
only showing top 5 rows
```

34. Perform Spark SQL queries.

```
df.createOrReplaceTempView("iot_data")

spark.sql("""
SELECT device, MAX(temp) AS max_temp
FROM iot_data
GROUP BY device
ORDER BY max_temp DESC
""").show()
```

```
+-----------------+------------------+
|           device|          max_temp|
+-----------------+------------------+
|1c:bf:ce:15:ec:4d|28.600000381469727|
|b8:27:eb:bf:9d:51|              23.4|
|00:0f:00:70:91:0a|20.100000381469727|
+-----------------+------------------+
```

35. Compare Spark vs Hadoop WordCount performance.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("SparkAssignment") \
    .master("local[*]") \
    .getOrCreate()

sc = spark.sparkContext

text_rdd = sc.textFile("/iot_telemetry_data.csv")

spark_wordcount = (
    text_rdd.flatMap(lambda x: x.split(","))
            .map(lambda x: (x, 1))
            .reduceByKey(lambda a, b: a + b)
)

spark_wordcount.take(10)
```

```
[('"humidity"', 1),
 ('"light"', 1),
 ('"lpg"', 1),
 ('"motion"', 1),
 ('"smoke"', 1),
 ('"temp"', 1),
 ('"1.5945120943859746E9"', 1),
 ('"51.0"', 2735),
 ('"0.0076508222705719"', 169),
 ('"0.02041127012241292"', 169)]
```

36. Perform Missing Value Handling using Spark.

```
df_missing = df.na.fill({
    "temp": 0,
    "humidity": 0
})

df_missing.show(5)
```

```
+--------------------+----------------+--------------------+-----------------+-----+--------------------+------+--------------------
|                  ts|          device|                  co|         humidity|light|                 lpg|motion|                smok
+--------------------+----------------+--------------------+-----------------+-----+--------------------+------+--------------------
|1.5945120943859746E9|b8:27:eb:bf:9d:51|0.0049559386483912 45|             51.0|false| 0.0076508222705 5719| false| 0.0204112701224129
|1.5945120947355676E9|00:0f:00:70:91:0a|0.00284008860710 1...|             76.0|false|0.0051143834009770 71| false|0.0132748367048 5153
|1.5945120980735729E9|b8:27:eb:bf:9d:51|0.0049760123404216 58|             50.9|false|0.0076732274063980 91| false| 0.0204751255761782
| 1.594512099589146E9|1c:bf:ce:15:ec:4d|0.0044030268296996 89|76.80000305175781| true|0.0070233371458773 14| false|0.0186282253770188 0
| 1.594512101761235E9|b8:27:eb:bf:9d:51|0.0049673636419089 52|             50.9|false|0.0076635772823724 11| false|0.0204476208102336 5
+--------------------+----------------+--------------------+-----------------+-----+--------------------+------+--------------------
only showing top 5 rows
```

37. Implement Linear Regression using Spark MLlib.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

lr_data = df.select("humidity", "temp").dropna()

assembler = VectorAssembler(
    inputCols=["humidity"],
    outputCol="features"
)

lr_ready = assembler.transform(lr_data)

lr = LinearRegression(
    featuresCol="features",
    labelCol="temp"
)

lr_model = lr.fit(lr_ready)

print("Coefficient:", lr_model.coefficients)
print("Intercept:", lr_model.intercept)
```

```
Coefficient: [-0.09402777918578237]
Intercept: 27.888780659173502
```

38. Implement Classification Model using Spark.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.sql.functions import col

cls_data = df.select("temp", "humidity", col("motion").cast("int")).dropna()

assembler_cls = VectorAssembler(
    inputCols=["temp", "humidity"],
    outputCol="features"
)

cls_ready = assembler_cls.transform(cls_data)

log_reg = LogisticRegression(
    featuresCol="features",
    labelCol="motion"
)

cls_model = log_reg.fit(cls_ready)

print("Accuracy:", cls_model.summary.accuracy)
```

```
Accuracy: 0.9984365307440088
```