SIES COLLEGE OF ARTS,SCIENCE & COMMERCE
(EMPOWERED AUTONOMOUS)
SION(W),MUMBAI-22

DEPARTMENT OF INFORMATION TECHNOLOGY

MSc(IT), SEMESTER II

Practical Journal

For

the Subject

Big Data Analytics

Submitted by

Yogesh Chatrooram Sahu

FMIT2526179

For the Academic Year

2025-2026

SIES College of Arts, Science and Commerce (Empowered Autonomous),
Sion(W), Mumbai – 400022.
Department of Information Technology

CERTIFICATE

This is to certify that Mr. **Yogesh Chatrooram Sahu** of MSc [Information Technology] **Semester II**, Seat No. **FMIT2526179** has successfully completed the practical's for the subject of **Big Data Analytics** as a partial fulfillment of the degree **M.Sc. (IT)** during the academic year 2025-26.

Faculty-in-Charge

Anita Gupta

Course Co-ordinator

Sudha Bhagavatheeswaran

Date:

External Examiner

INDEX

SR.NO.	PRACTICAL TITLE	SIGNATURE
1	Installation of HADOOP	
2	Implement file management tasks in Hadoop System (HDFS).	
3	Basic CRUD operations in MongoDB	
4	Implement programs related to MapReduce	
5	Implement Clustering and Associated algorithms	
6	Implement Linear Regression.	
7	Implement Bloom Filters for Filter Stream Data	
8	Implement Time Series	

PRACTICAL NO 1

Aim: Install, configure and run Hadoop and HDFS ad explore HDFS

Steps:

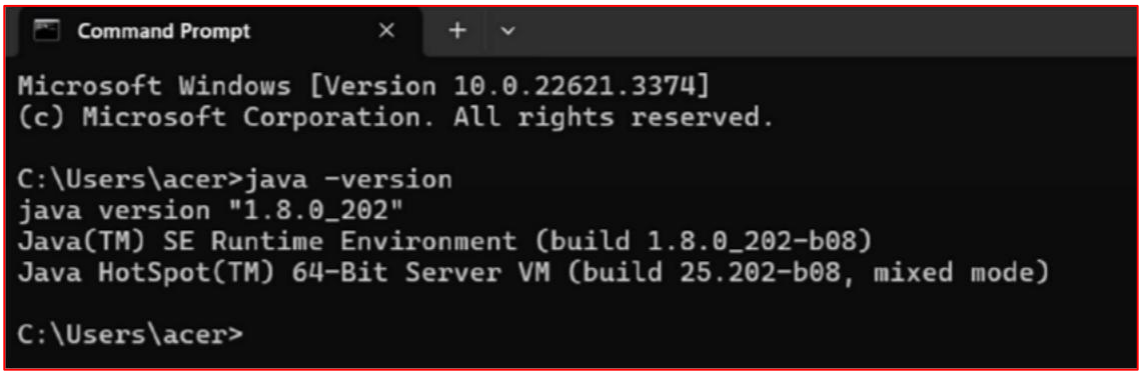
1) To install Hadoop, you should have Java version in your system Check your java version through the below given command in command prompt

(Link: https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html)

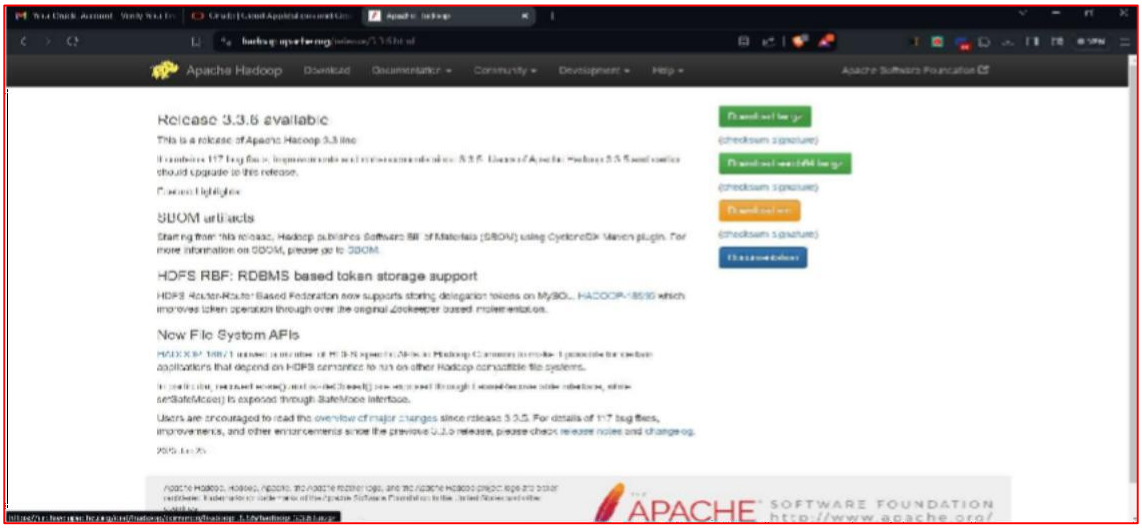
Command:

java -version

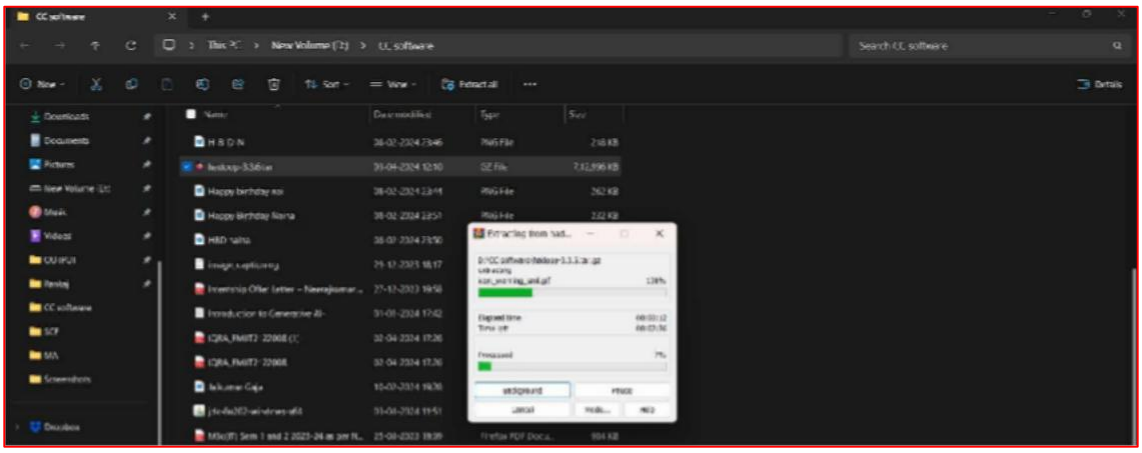
Output:



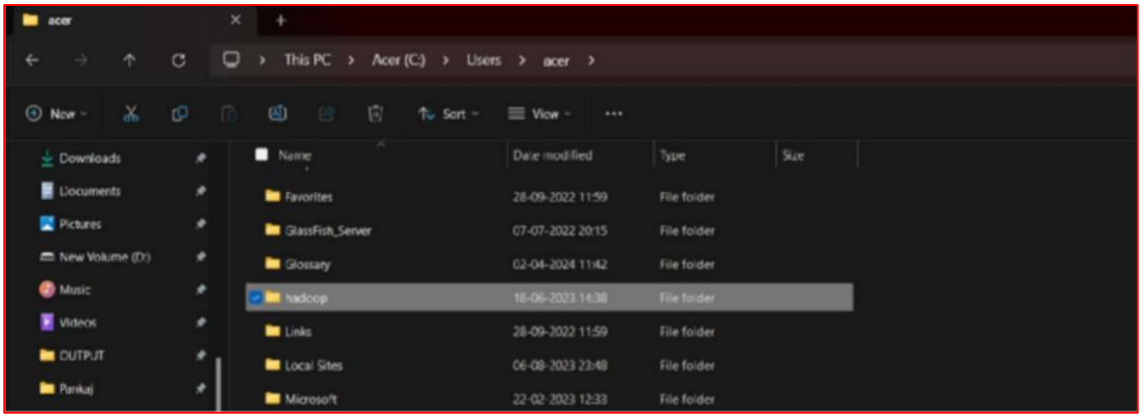
2) After downloading java version 1.8, download Hadoop version 3.3.6 from the given link. (Link: https://hadoop.apache.org/release/3.3.6.html)



3) Extract Hadoop to a local drive



4) Rename it “hadoop”.



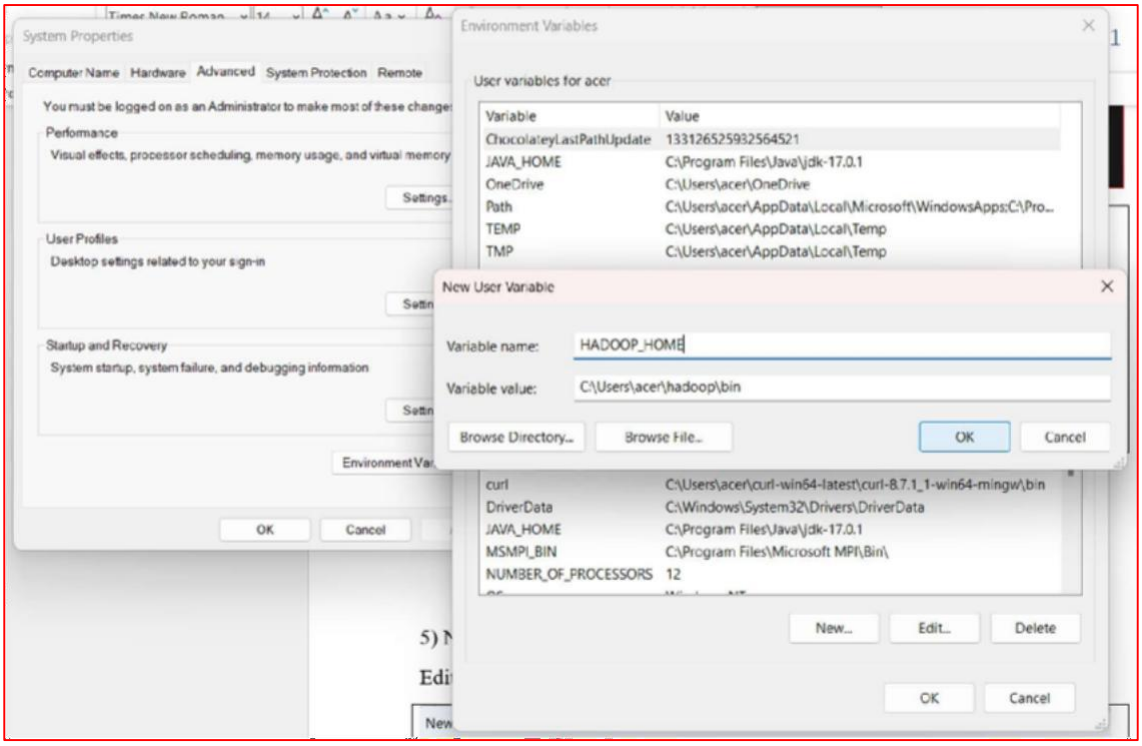
5) Now set the path open environment variables

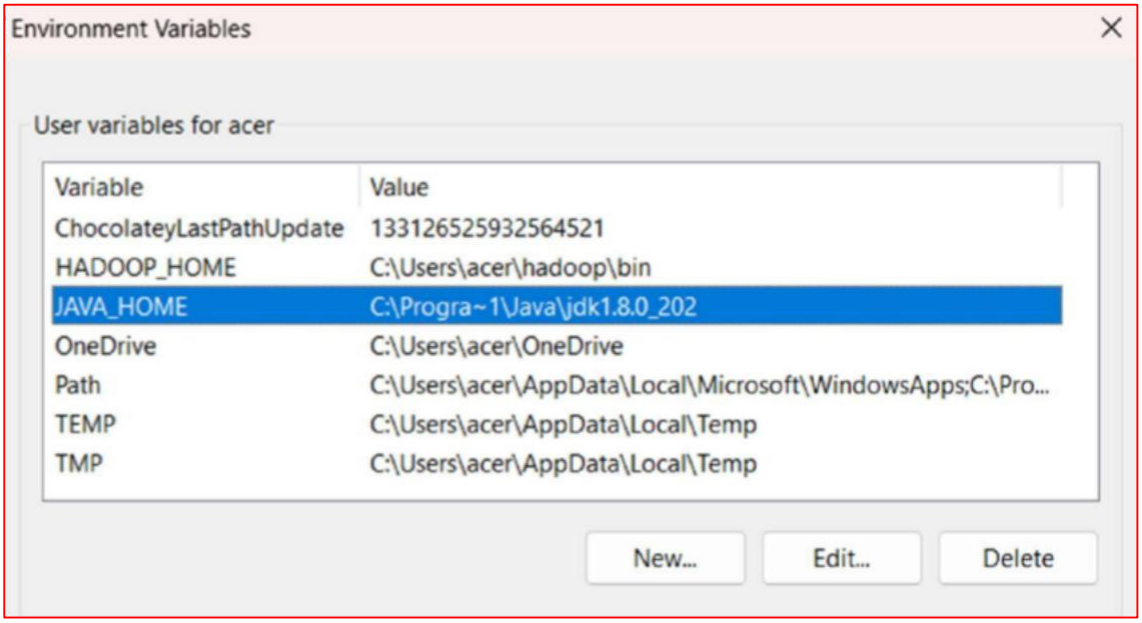
Search Edit the system environment variable on windows and click on it >> Click on Environment Variable >> Click on New >> Add name as HADOOP_HOME and paste the location as (C:\Users\acer\hadoop\bin) value

Do the same for both user variable as well as system variable and add both HADOOP_HOME and JAVA_HOME.

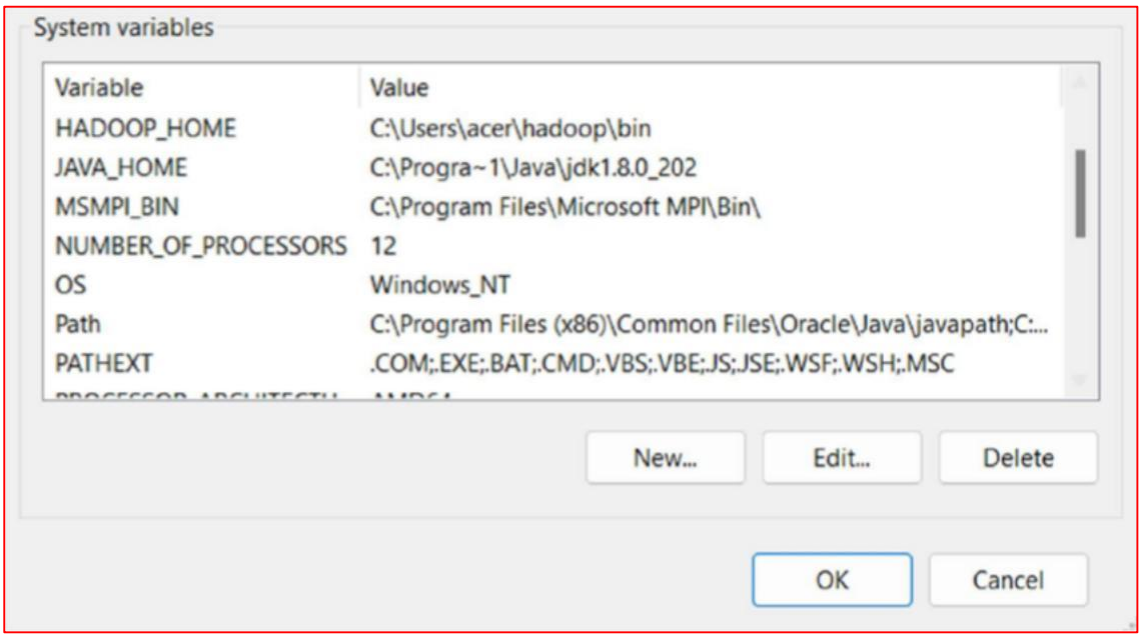
- C:\Users\acer\hadoop\bin (also add in system variable path)
- C:\Users\acer\hadoop\sbin (also add in system variable path in order to run the start-dfs.cmd command properly)
- C:\Progra~1\Java\jdk1.8.0_202

Note:- for JAVA_HOME set the value as C:\Progra~1\Java\jdk1.8.0_202 (i.e. Program Files is replaced as Progra~1) otherwise it will give you error in command prompt when you will try to execute hadoop version.



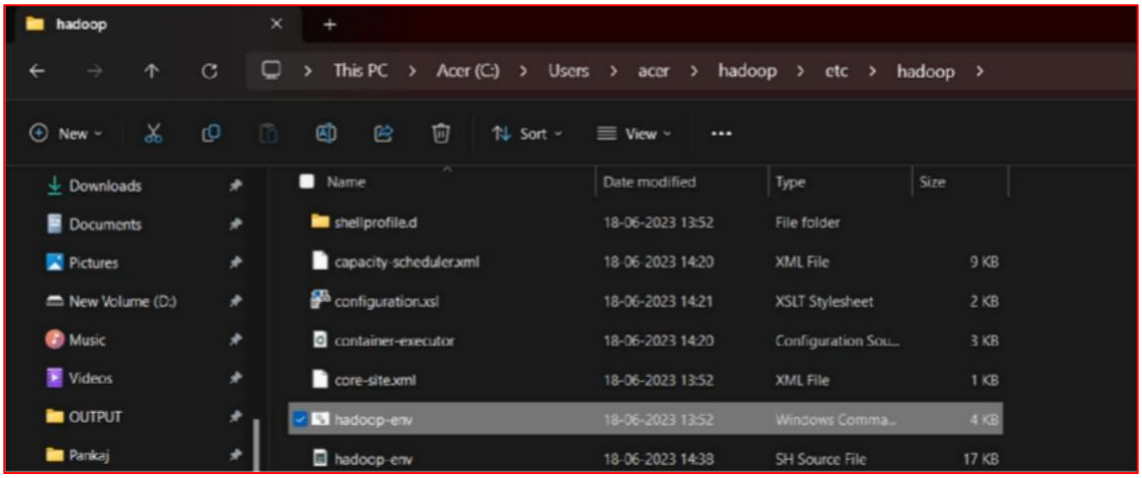


6) Now set the system variables

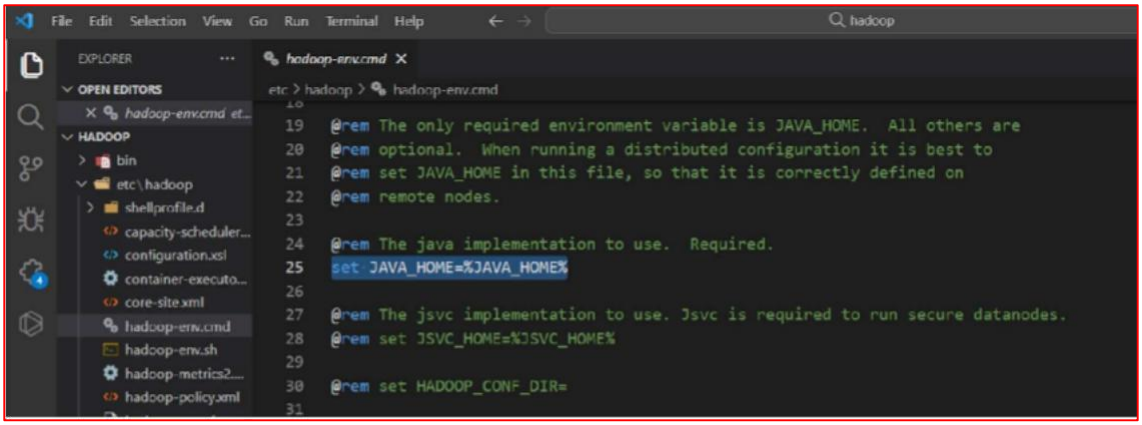


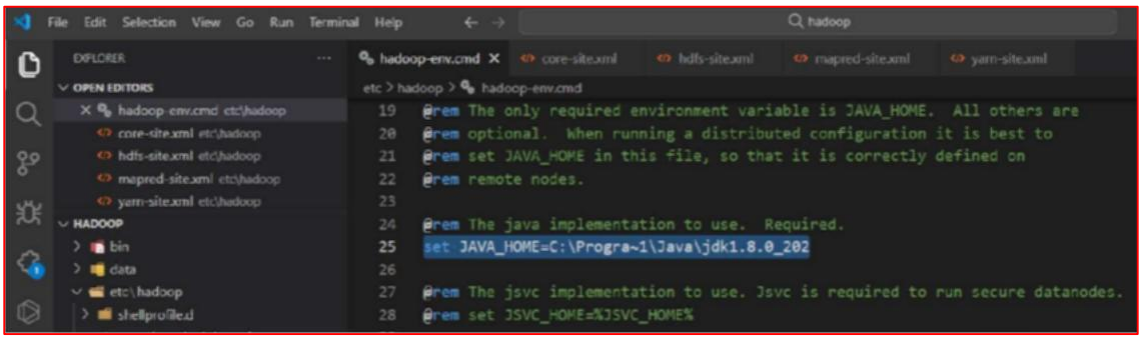
7) Now go to "**hadoop"** folder open "**etc"** folder in it then open "**hadoop"** folder inside it and then select the "**hadoop-env"** windows command open it and make the in changes according to below given code.

Note:- Open the file using notepad or any other editor here we have used visual studio code.



8) Edit **hadoop-env.cmd** and replace **%JAVA_HOME%** with the path of the java folder where your jdk 1.8 is installed i.e. **C:\Program Files\Java\jdk1.8.0_202**.





9) Make the changes in code of below given files by going to the **"etc"** folder in **"hadoop"**

i. core-site.xml
ii. hdfs-site.xml
iii. mapred-site.xml
iv. yarn-site.xml

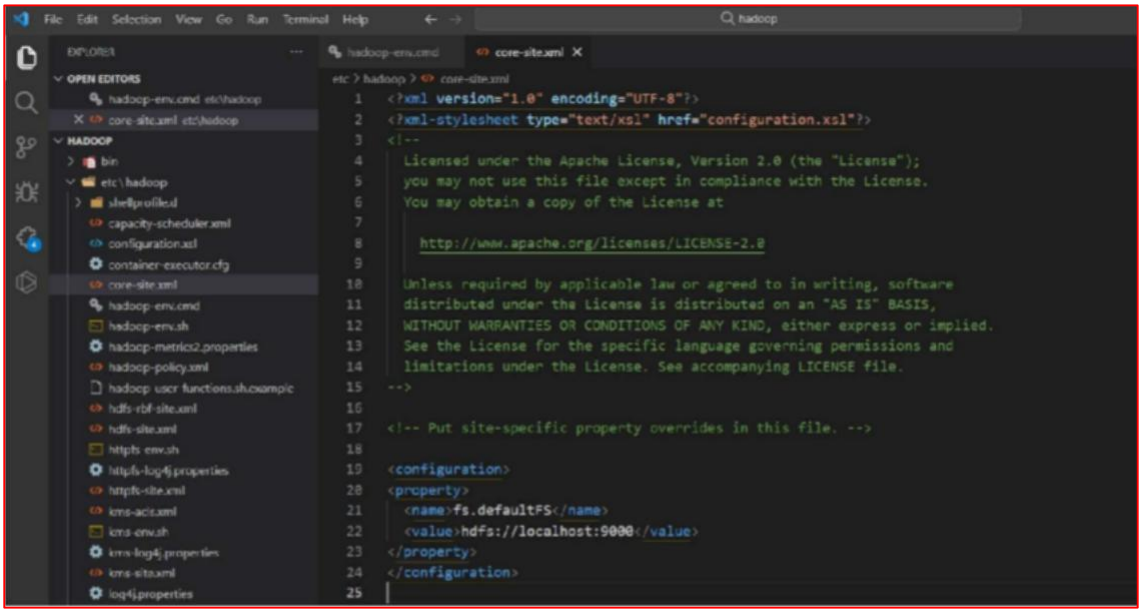
i) For core-site.xml

Code:

```
<configuration>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
```

</property>

</configuration>



ii) For hdfs-site.xml

Code:

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>C:\hadoop\data\namenode</value>

</property>

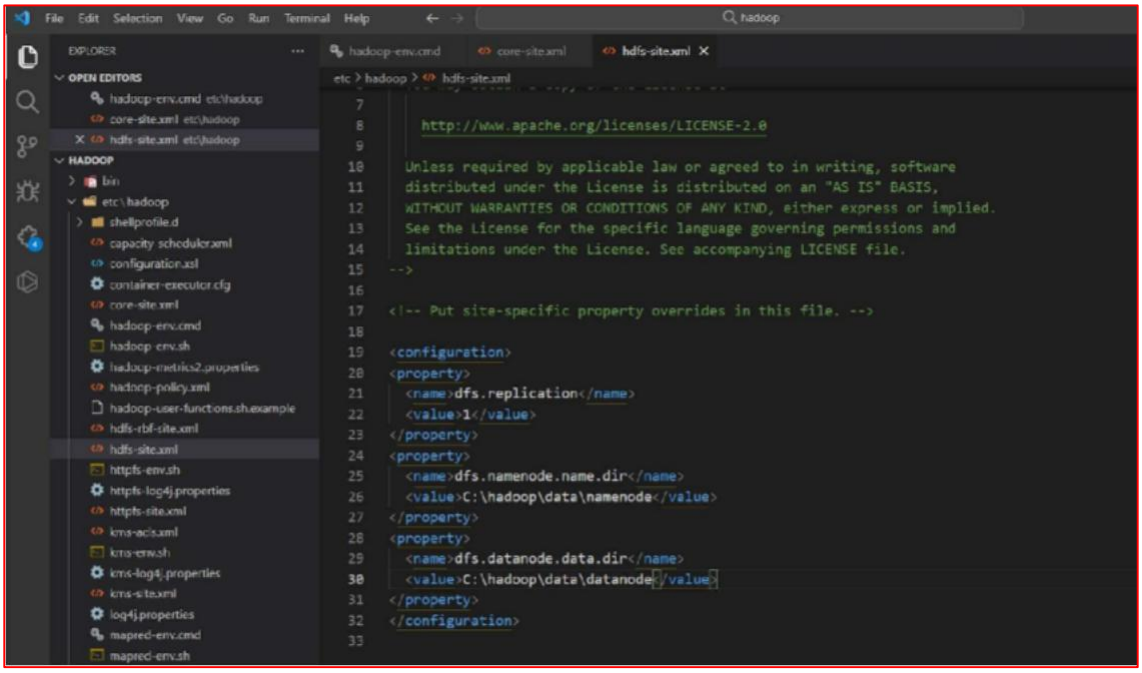
<property>

<name>dfs.datanode.data.dir</name>

<value>C:\hadoop\data\datanode</value>

</property>

</configuration>



iii) For mapred-site.xml

Code:

<configuration>

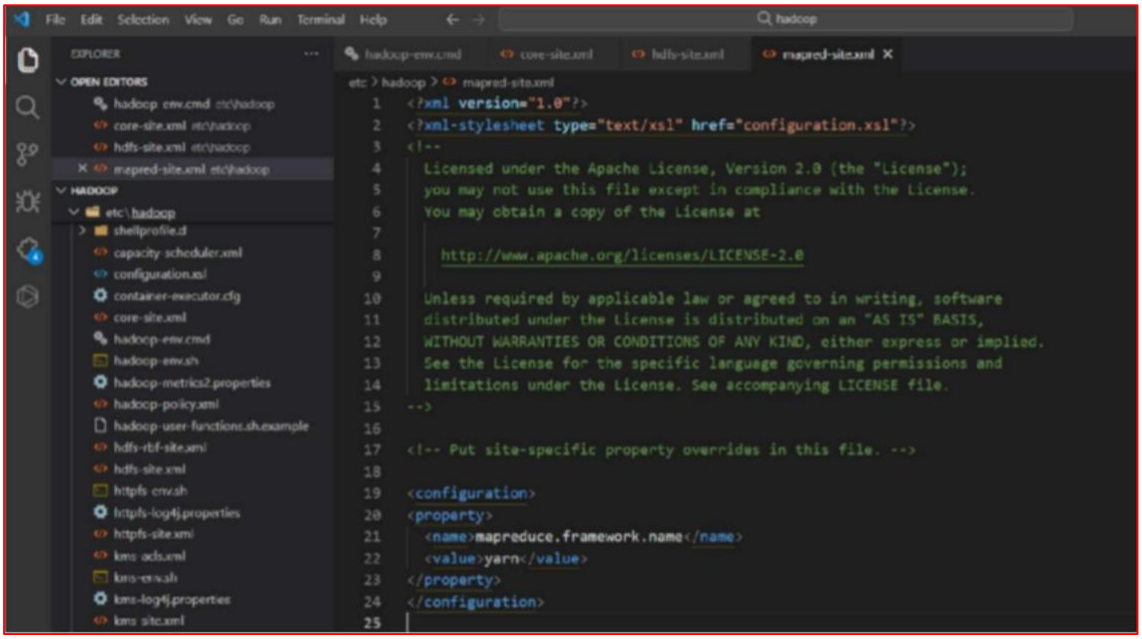
<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>



iv) For yarn-site.xml

Code:

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shuffle</value>

</property>

<property>

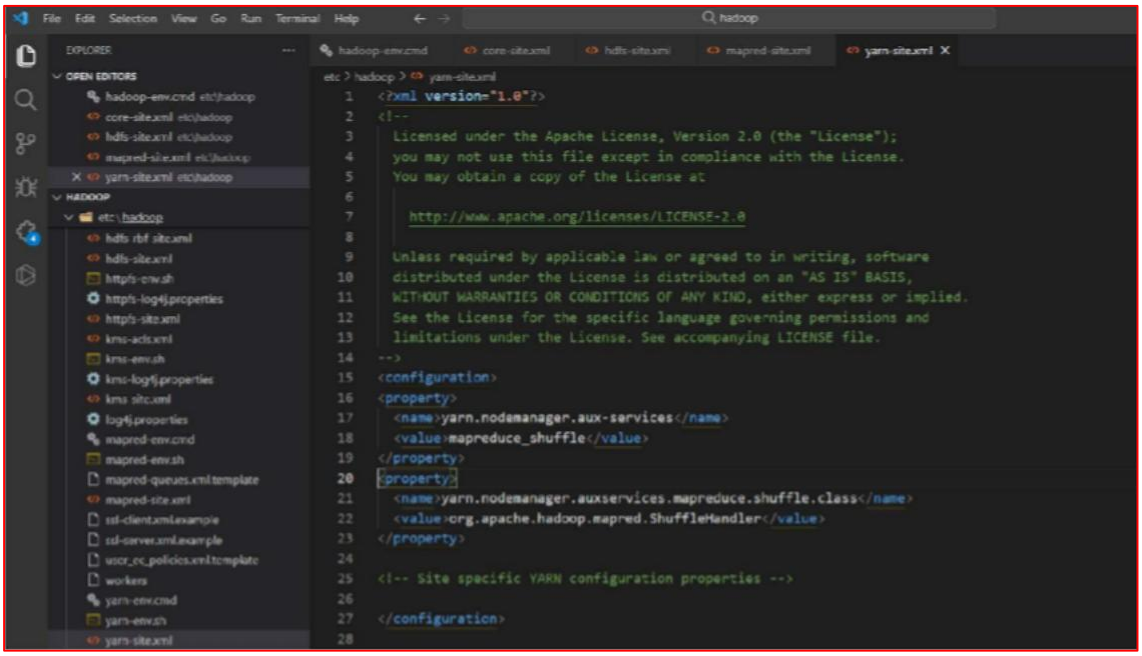
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

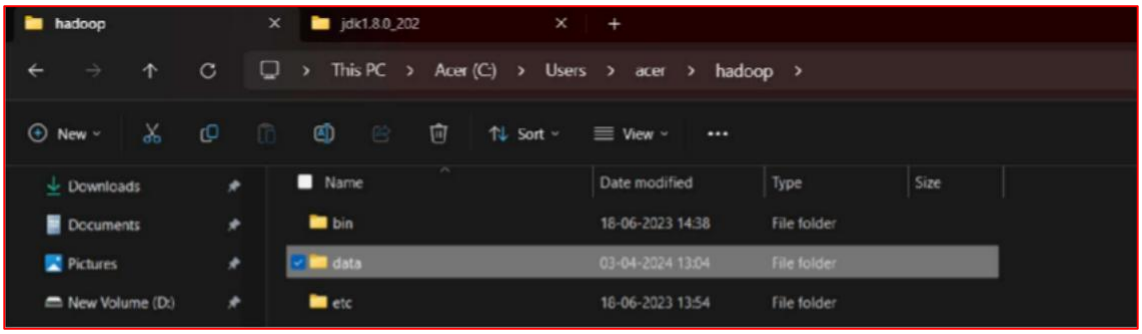
</property>

<!-- Site specific YARN configuration properties -->

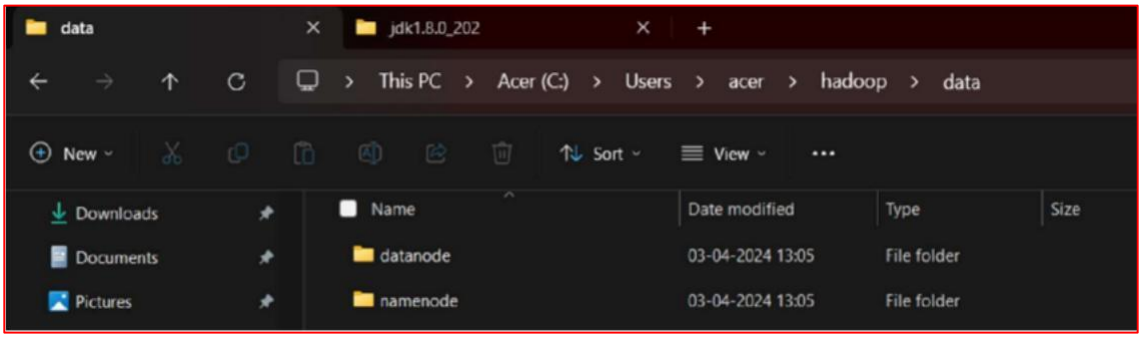
</configuration>



10) Create a folder **"data"** in hadoop directory

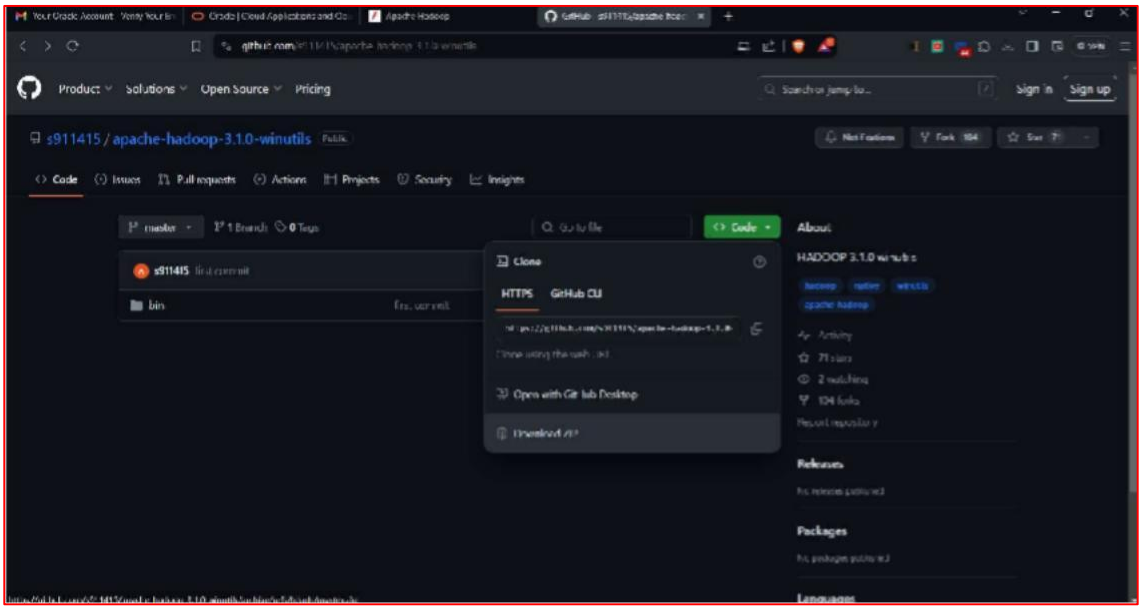


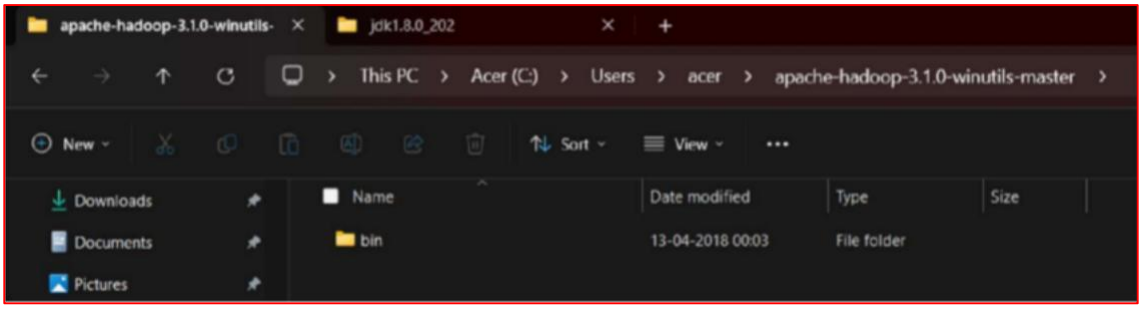
11) Create a folder with name **"datanode"** and folder **"namenode"** in the **data** folder

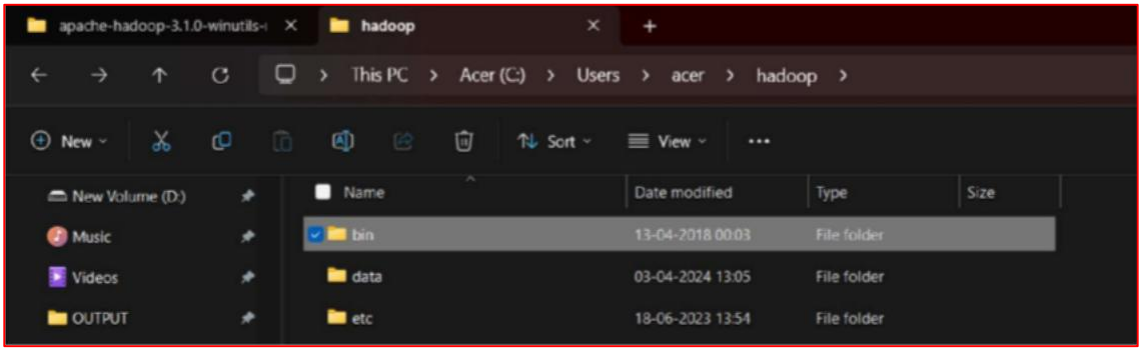


12) Download the zip file using the below link, Extract it and copy the bin folder in hadoop directory and replace it

(Link: https://github.com/s911415/apache-hadoop-3.1.0-winutils)





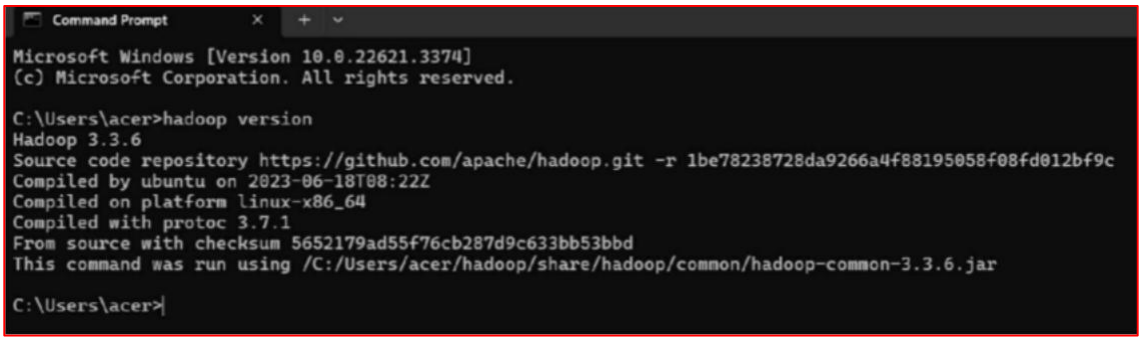


13) Check whether hadoop is successfully installed by running the below command

Command:

hadoop version

Output:



14) Format the NameNode

Formatting the NameNode is done once when hadoop is installed and not for running hadoop filesystem, else it will delete all the data inside HDFS. Run this

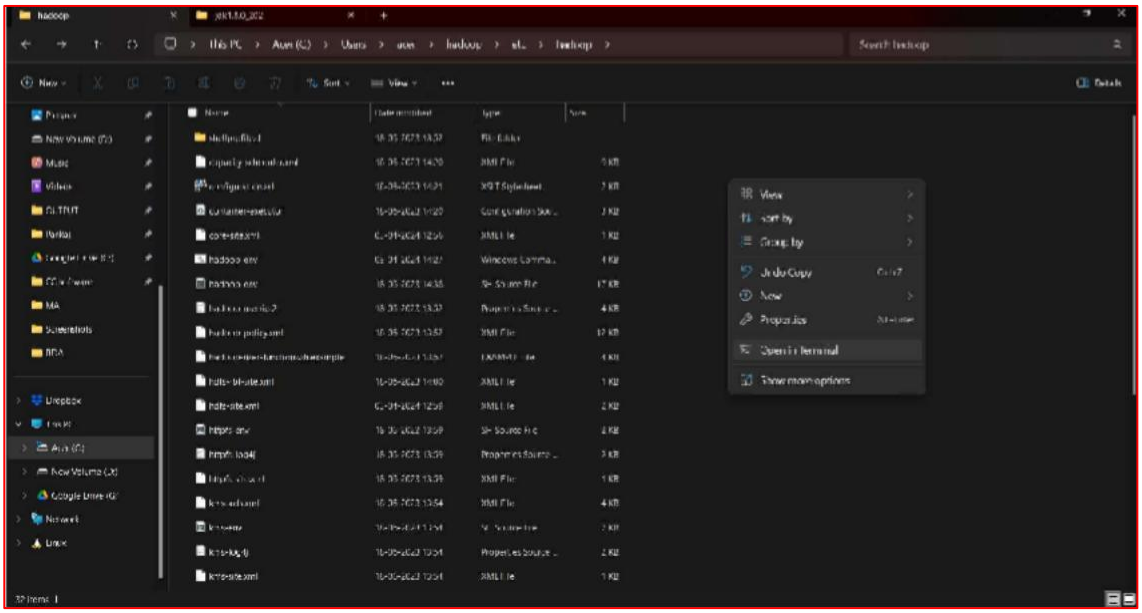
Command:

hdfs namenode -format

Output:

15) Go to hadoop inside that go to etc folder and then open hadoop directory and right click choose the option open in terminal.

C:\Users\acer\hadoop\etc\hadoop

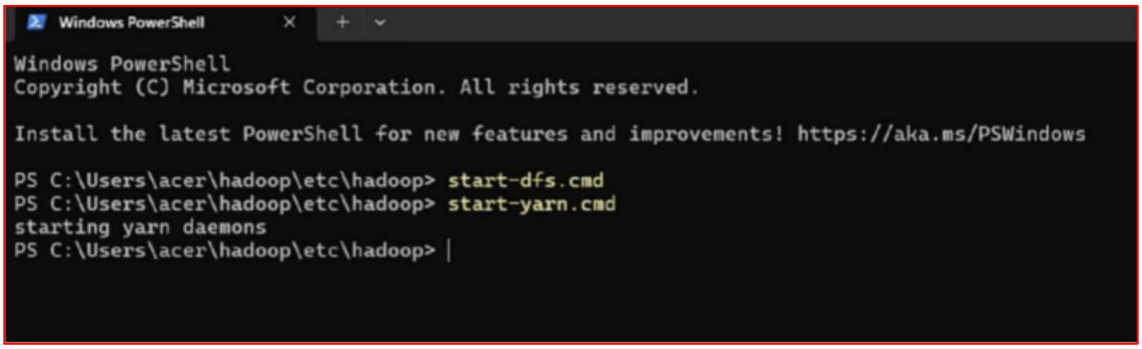


Command:

start-dfs.cmd

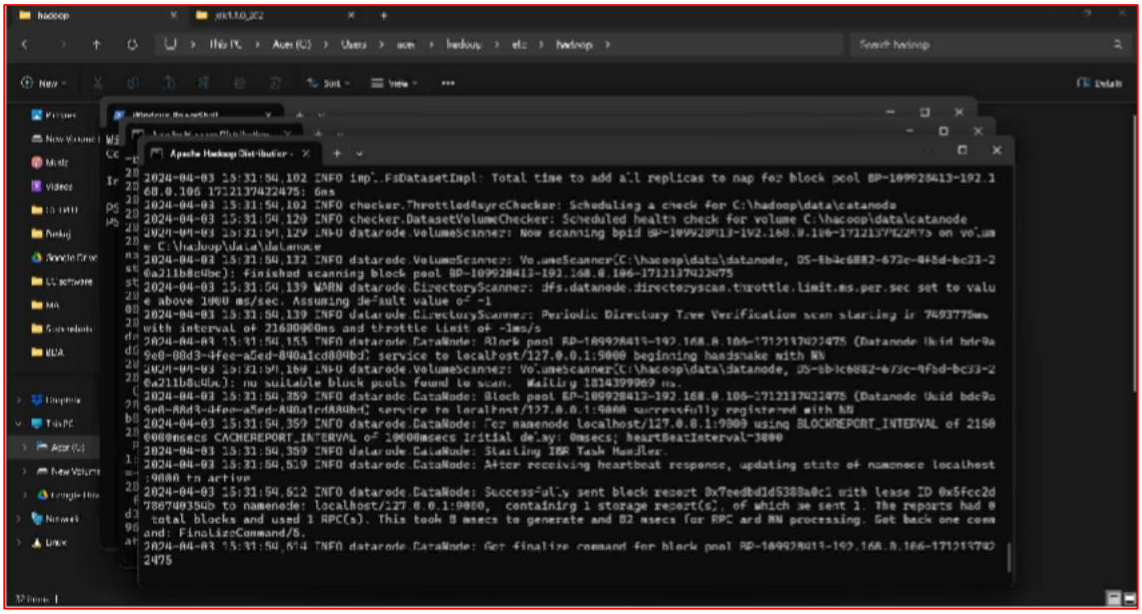
start-yarn cmd

Output:

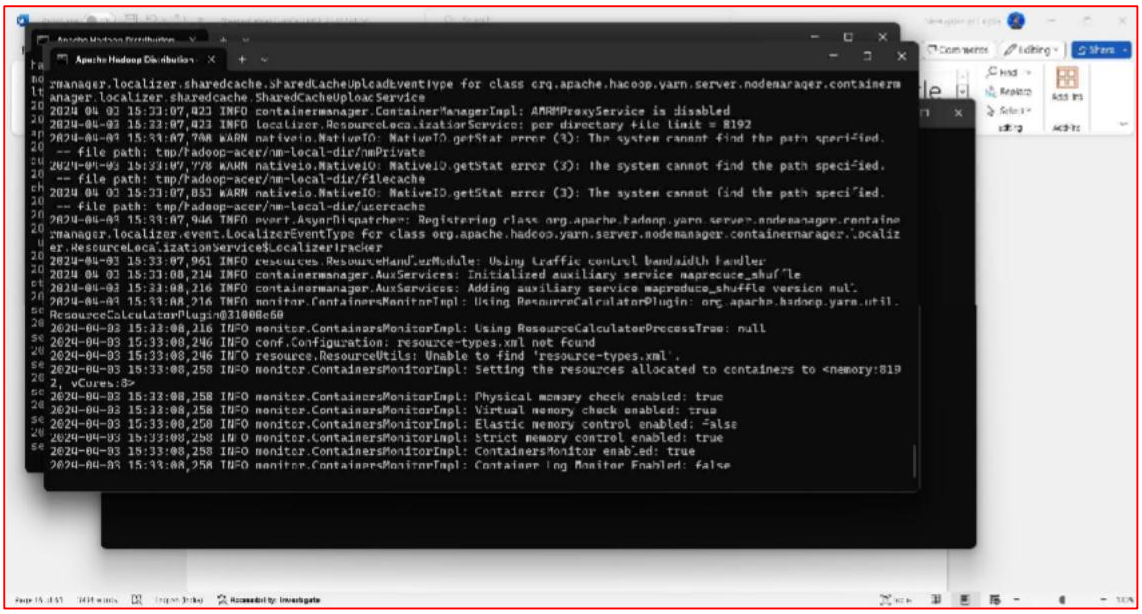


Note:- ensure that the **sbin** directory is included in your system's **PATH** variable

start-dfs.cmd - output



start-yarn.cmd - output

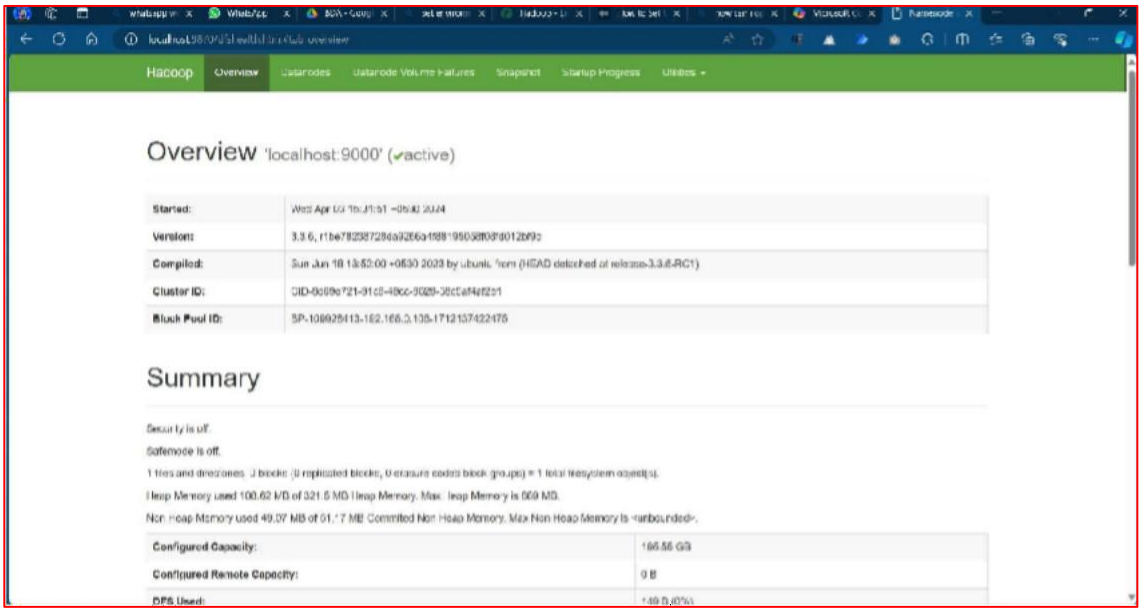


16) Two more windows will open, one for yarn resource manager and one for yarn node manager

Note: Make sure all the 4 Apache Hadoop Distribution windows are up n running. If they are not running, you will see an error or a shutdown message. In that case, you need to debug the error.

To access information about resource manager current jobs, successful and failed jobs, go to this link in browser-

http://localhost:9870



PRACTICAL NO 2

Aim: To implement file management tasks in Hadoop System (HDFS)

Description:

Hadoop HDFS is a distributed file system that provides redundant storage space for files having huge sizes. It is used for storing files that are in the range of terabytes to petabytes.

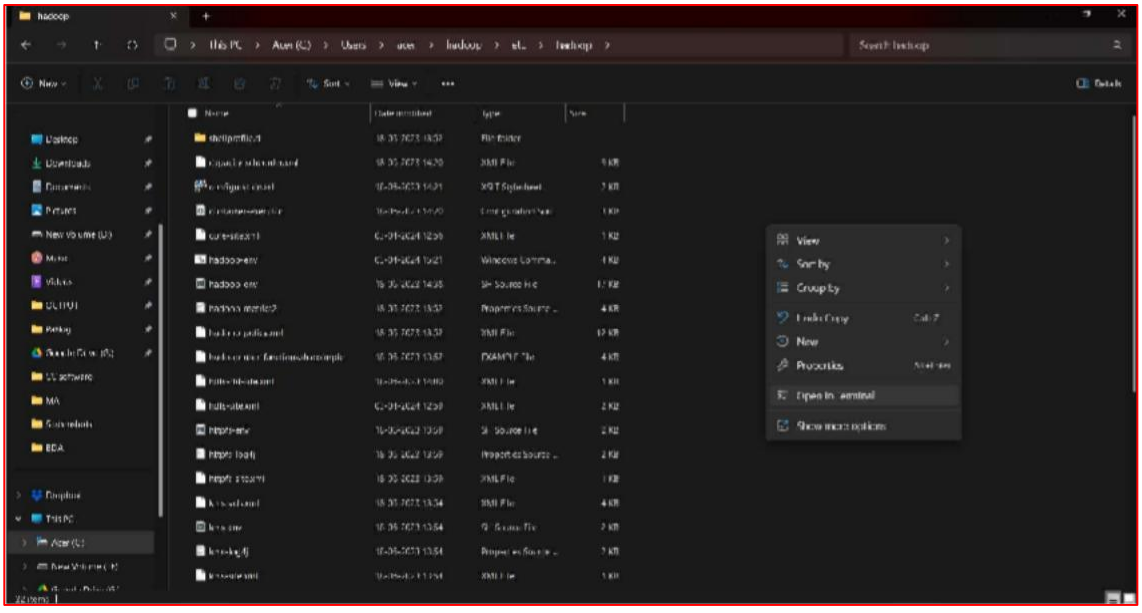
Go to hadoop inside that go to etc folder and then open hadoop directory and right click choose the option open in terminal and run below commands.

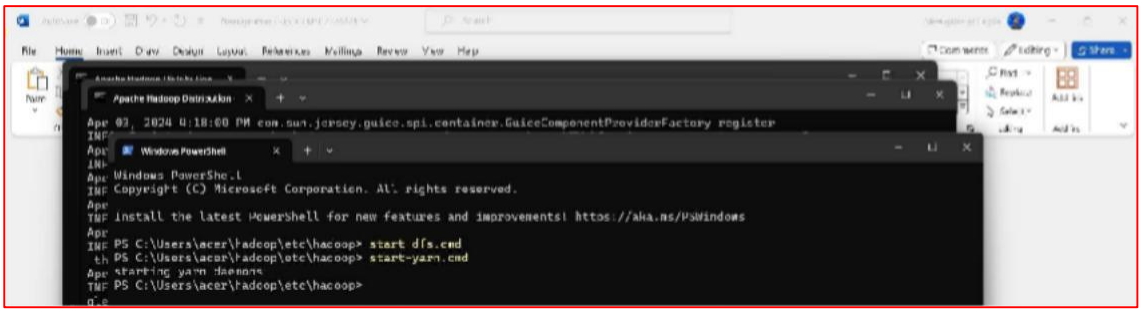
hadoop > etc > hadoop > right click > open in terminal (here it will open in powershell)

Commands:-

start-dfs.cmd

start-yarn.cmd



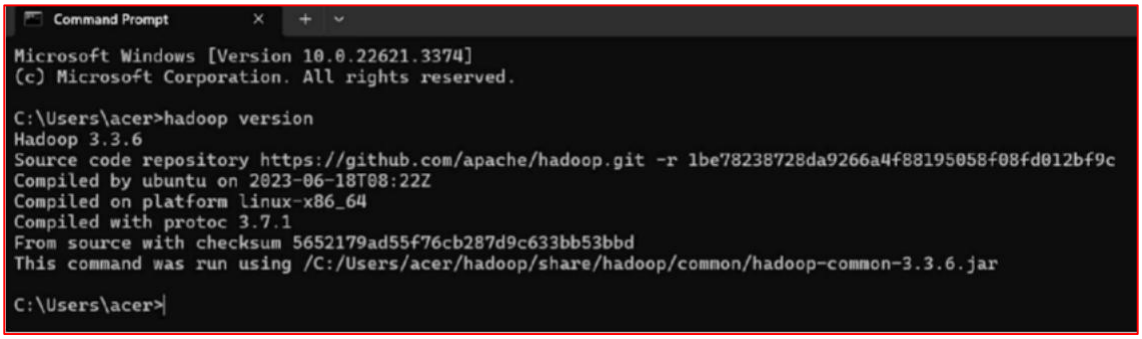


1) Open new command prompt and check the version

Command:

hadoop version

Output:



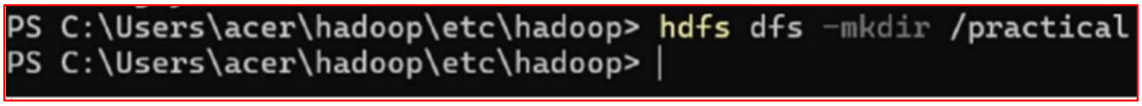
2) mkdir

Create a directory in Hadoop

Command:

hdfs dfs -mkdir /practical

Output:



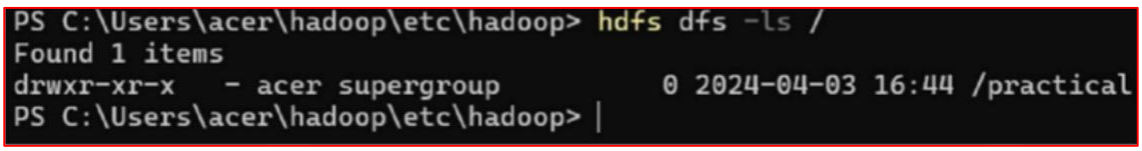
3) ls

This command is used to enlist the files and directories present in HDFS.

Command:

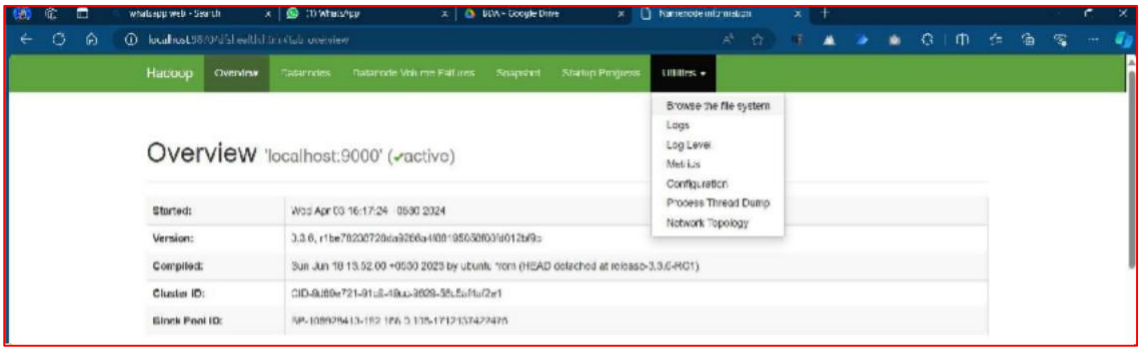
hdfs dfs -ls /

Output:

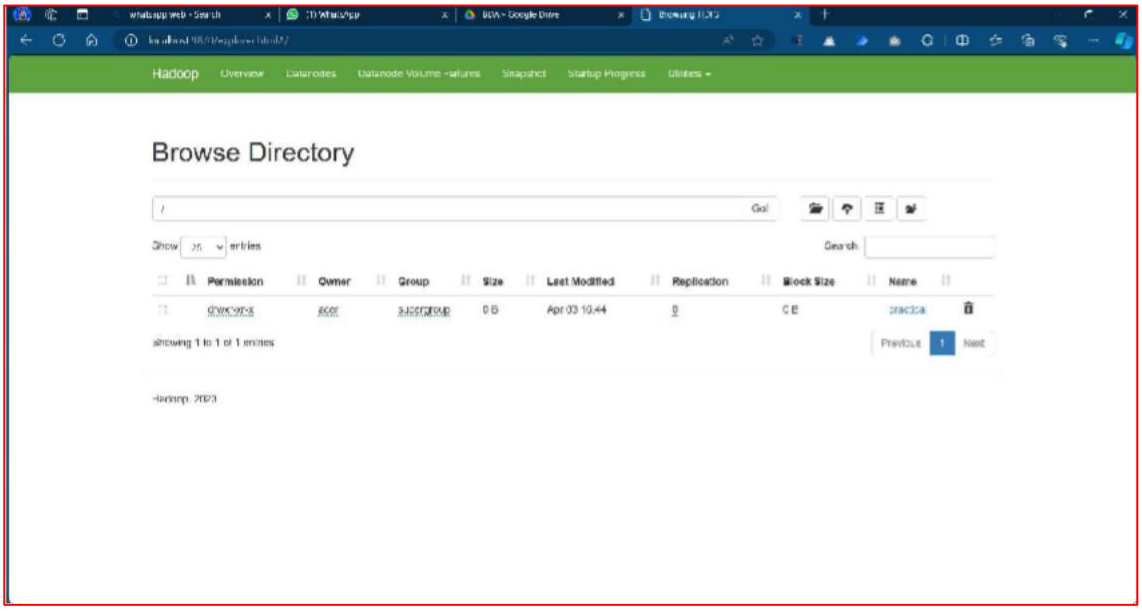


Now check the website whether directory is created or not by going to localhost:9870 through your browser

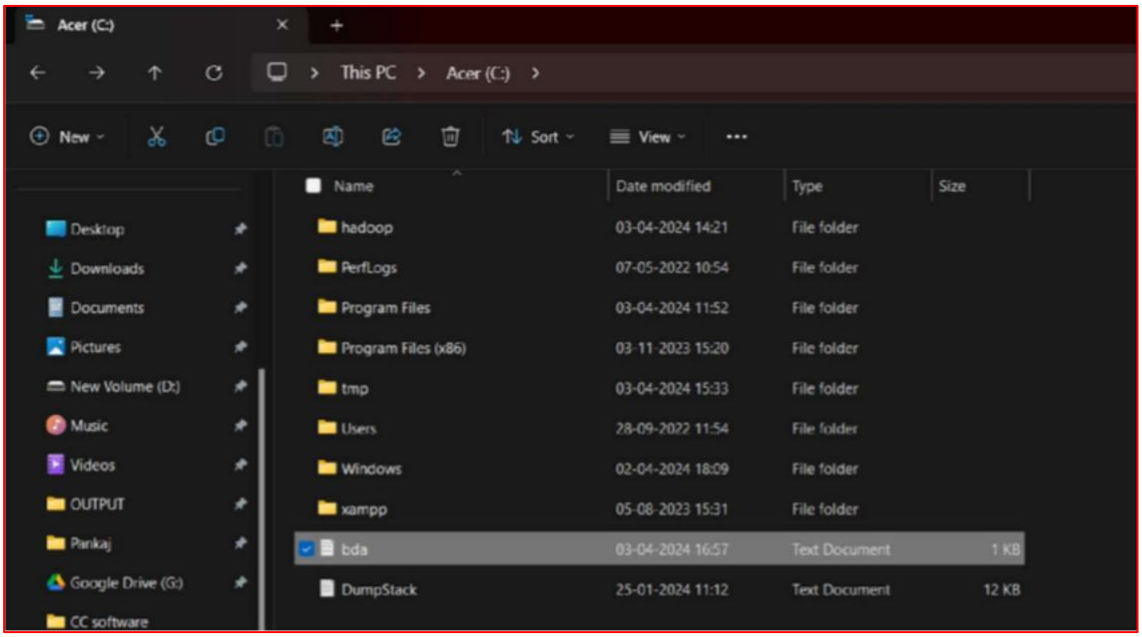
Once the page is open click on '**utilities**' and select '**Browse the file system**'



New window will appear named browse directory



4) Create a txt file in desktop or local drive and write 50 words (bda.txt)



5) put

This command is used to copy file from local system to Hadoop file system

Command:

hdfs dfs -put C:/bda.txt /practical

hdfs dfs -ls /practical

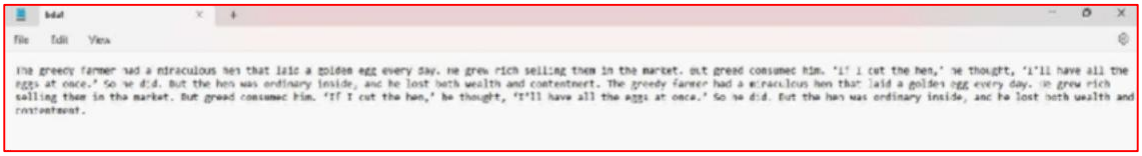
Output:

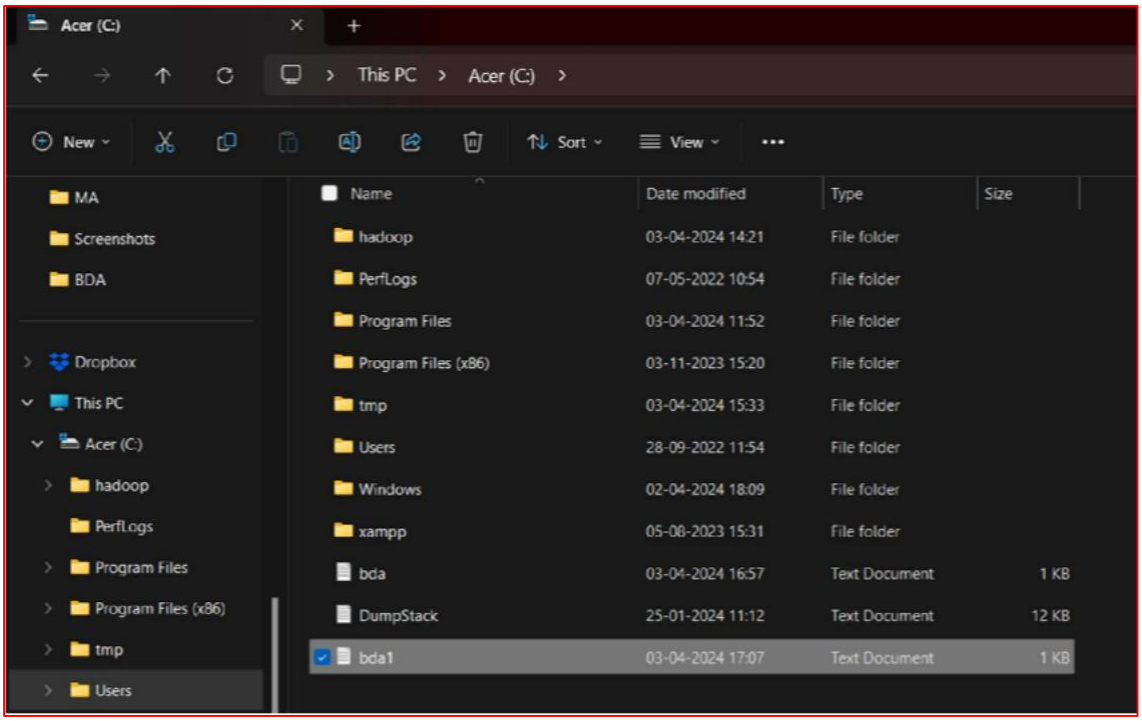
```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -put C:/bda.txt /practical
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical
Found 1 items
-rw-r--r--   1 acer supergroup        298 2024-04-03 17:05 /practical/bda.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

6) copyFromLocal

This command is similar to put command which is used to copy the file from local system to Hadoop system.

Create one more file in Desktop or local drive and save it (bda1.txt)





Command:

hdfs dfs -copyFromLocal C:/bda1.txt /practical

hdfs dfs -ls /practical

Output:

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -copyFromLocal C:/bda1.txt /practical
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical
Found 2 items
-rw-r--r--   1 acer supergroup        298 2024-04-03 17:05 /practical/bda.txt
-rw-r--r--   1 acer supergroup        597 2024-04-03 17:11 /practical/bda1.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

7) get

This command is used to copy the file from Hadoop System to local system

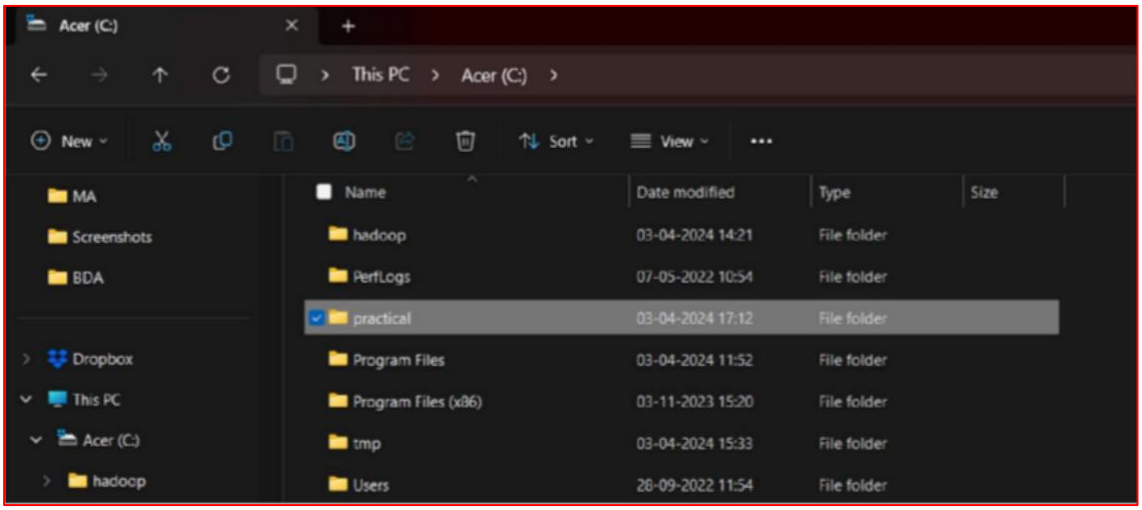
Command:

hdfs dfs -get /practical C:/

Output:

It will be automatically created on desktop or local drive (i.e. acer (C:))

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -get /practical C:/
PS C:\Users\acer\hadoop\etc\hadoop>
```



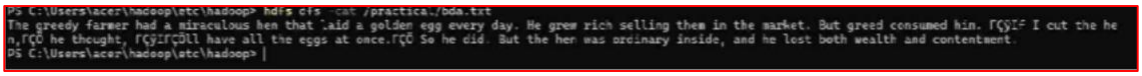
8) cat

This command is used to display the content of the file.

Command:

hdfs dfs -cat /practical/bda.txt

Output:



9) mkdir

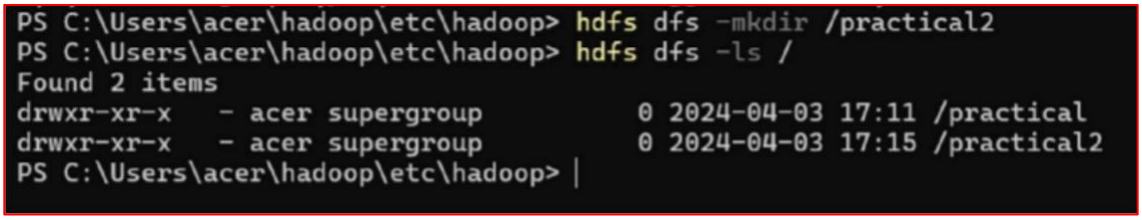
Create a directory in Hadoop

Command:

hdfs dfs -mkdir /practical2

hdfs dfs -ls /

Output:



10) mv

This command is used to move the file from one directory to another directory within HDFS system.

Command:

hdfs dfs -mv /practical/bda.txt /practical2

hdfs dfs -ls /practical2

Output:

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -mv /practical/bda.txt /practical2
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical2
Found 1 items
-rw-r--r--   1 acer supergroup        298 2024-04-03 17:05 /practical2/bda.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

11) cp

This command is used to copy the file from one directory to another directory within HDFS system.

Command:

hdfs dfs -cp /practical/bda1.txt /practical2

hdfs dfs -ls /practical2

Output:

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -cp /practical/bda1.txt /practical2
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical2
Found 2 items
-rw-r--r--   1 acer supergroup        298 2024-04-03 17:05 /practical2/bda.txt
-rw-r--r--   1 acer supergroup        597 2024-04-03 17:19 /practical2/bda1.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

12) moveFromLocal

This command is used to move the file from the local filesystem to the Hadoop filesystem.

Command:

hdfs dfs -moveFromLocal C:/bda.txt /practical

hdfs dfs -ls /practical

Output:

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -moveFromLocal C:/bda.txt /practical
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical
Found 2 items
-rw-r--r--   1 acer supergroup        298 2024-04-03 17:20 /practical/bda.txt
-rw-r--r--   1 acer supergroup        597 2024-04-03 17:11 /practical/bda1.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

13) rm

This command is used to remove the files from the directory

Command:

hdfs dfs -rm /practical/bda.txt

hdfs dfs -ls /practical

Output:

```
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -rm /practical/bda.txt
Deleted /practical/bda.txt
PS C:\Users\acer\hadoop\etc\hadoop> hdfs dfs -ls /practical
Found 1 items
-rw-r--r--   1 acer supergroup        597 2024-04-03 17:11 /practical/bda1.txt
PS C:\Users\acer\hadoop\etc\hadoop>
```

PRACTICAL NO 3

Aim: To implement CRUD operation in mongo db

Software: MongoDB Server, Command line, Mongo shell service

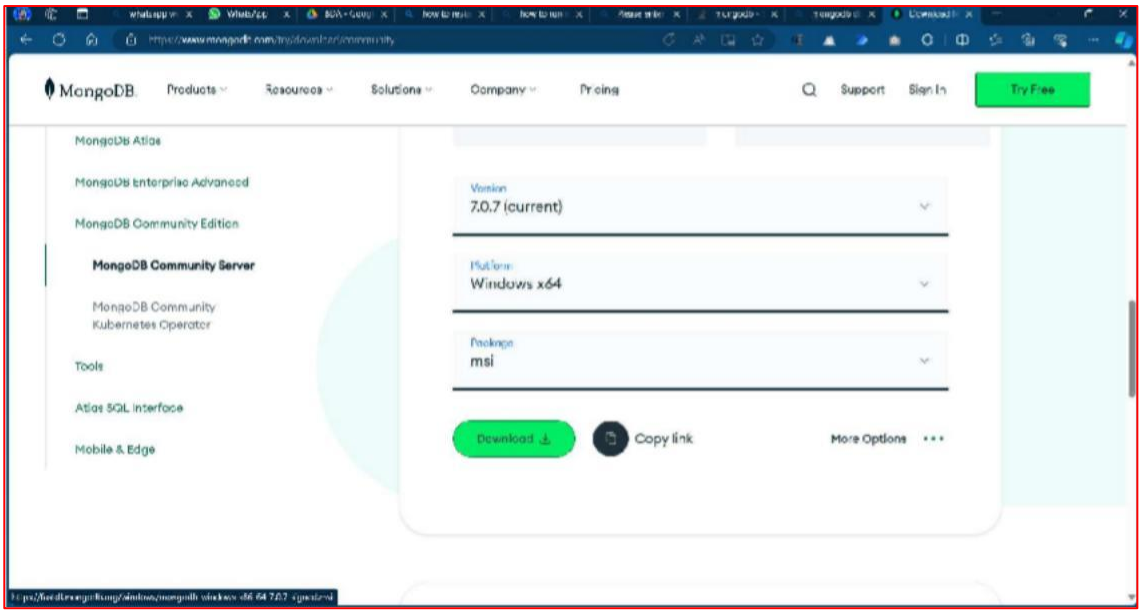
Description:

MongoDB for various things like building an application (including web and mobile), or analysis of data, or an administrator of a MongoDB database, in all these cases we need to interact with the MongoDB server to perform certain operations like entering new data into the application, updating data into the application, deleting data from the application, and reading the data of the application. MongoDB provides a set of some basic but most essential operations that will help you to easily interact with the MongoDB server and these operations are known as CRUD operations.

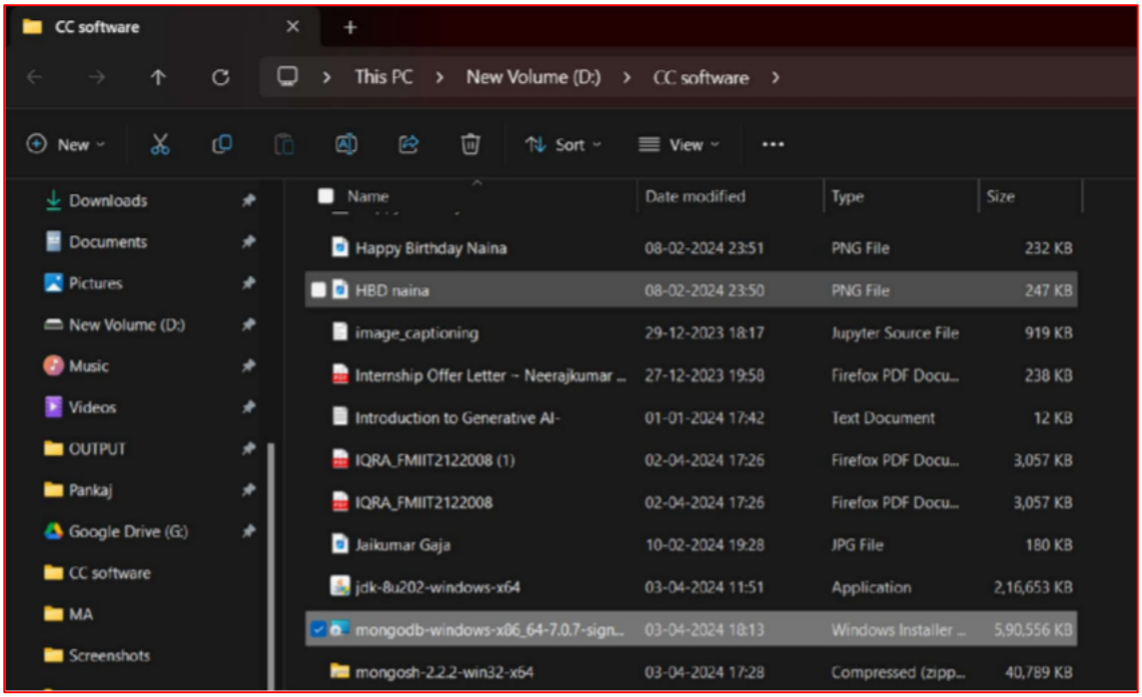
Steps:

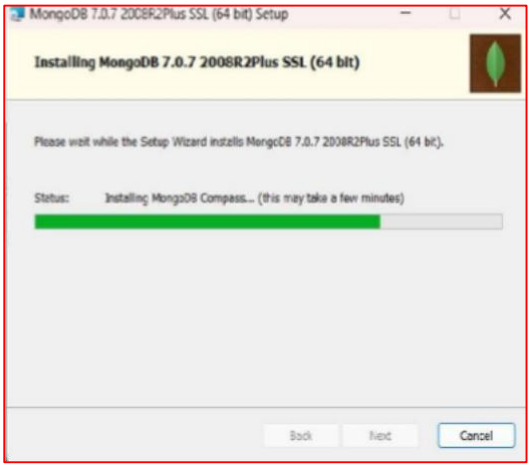
Download MongoDB Server from the below given link as it is necessary in order to run the MongoDB Shell.

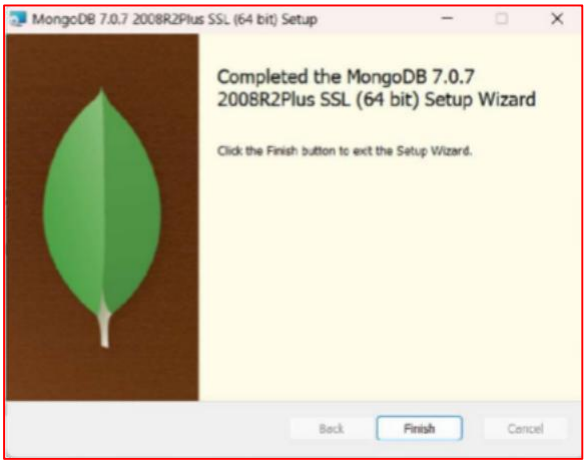
(Link: https://www.mongodb.com/try/download/community)



Once downloaded open the location and click on the downloaded file. Give all the necessary permissions and install the complete version of mongodb server.





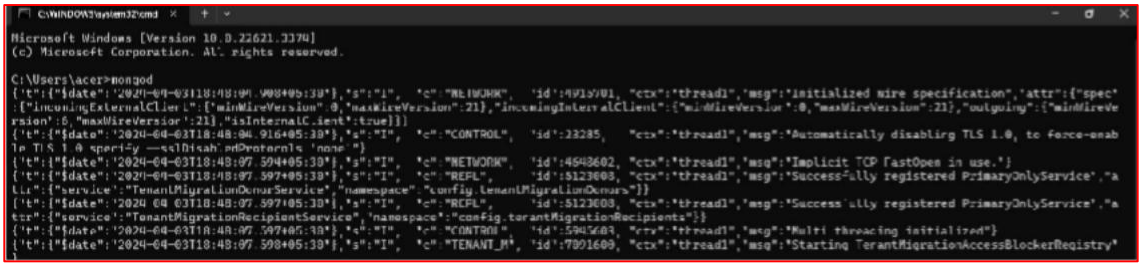


After complete installation add the directory to environment variable path by copying the mongodb bin path (C:\Program Files\MongoDB\Server\7.0\bin).

Now open the command prompt and type '**mongod –version'** to check if mongodb server is been setup properly or not.

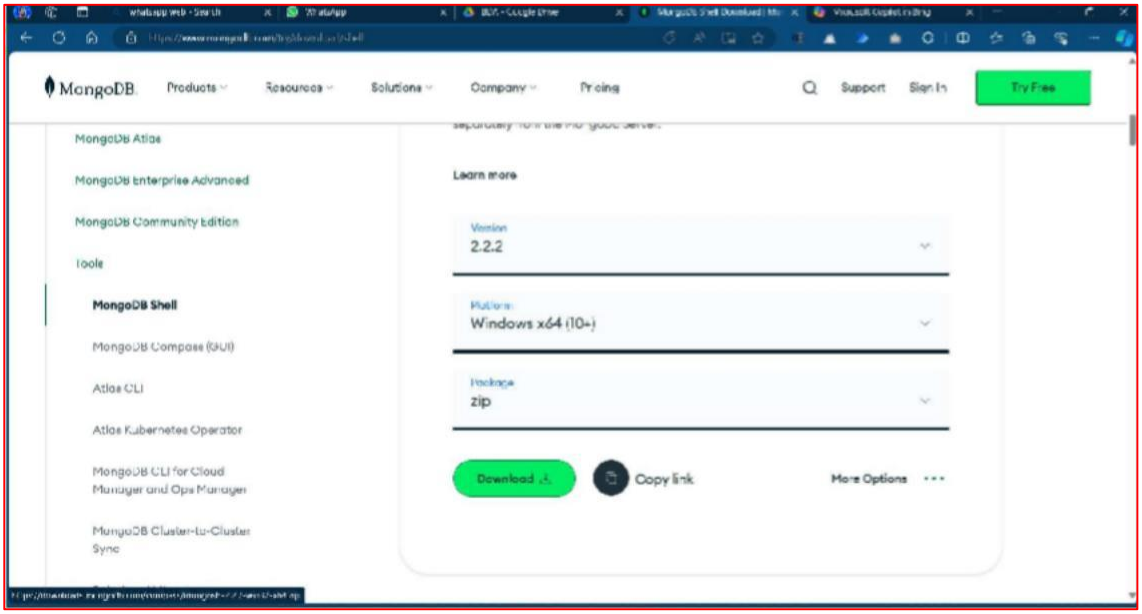
Note:- Before running mongodb shell always run the **'mongod'** command in the command prompt.

```
C:\Users\acer>mongod --version
db version v7.0.7
Build Info: {
    "version": "7.0.7",
    "gitVersion": "cfb08e1ab7ef741b4abdd0638351b322514c45bd",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

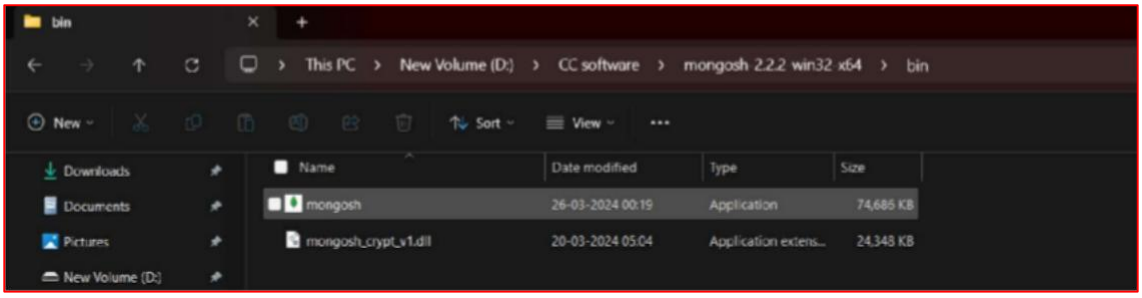


1) Download MongoDB Shell

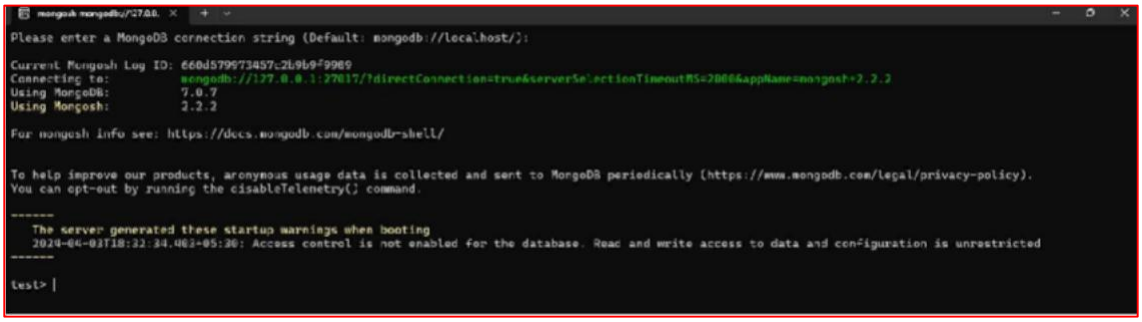
(Link: https://www.mongodb.com/try/download/shell)



2) Extract it where it is been downloaded and install the application. Click and open the shell application



MongoDB Shell……



Note:- Sometime it will ask you to create database to store the data in that situation you need to create a data folder in the C: drive and inside data folder create a db folder.

4) **Create Operations –**

The create or insert operations is used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. You can perform, create operations using the following methods provided by the MongoDB:

It is used to insert a single document in the collection.

Code:-

```
use BDA

db.collection.insertOne()
```

Example 1: In this example, we are inserting details of a single student in the form of document in the student collection using db.collection.insertOne() method.

Output:

```
test> use BDA
switched to db BDA
BDA> db.student.insertOne({
... name : "Neeraj",
... age : 22,
... branch : "IT",
... course : "MSC",
... mode : "offline",
... paid : true,
... amount : 52000
... })
{
  acknowledged: true,
  insertedId: ObjectId('660d5cc431728aa2899f990a')
}
BDA>
```

5) **Insert Many -**

It is used to insert multiple documents in the collection

Code:

db.collection.insertMany().

Example 2:

In this example, we are inserting details of the multiple students in the form of documents in the student collection using db.collection.insertMany() method.

```
BDA> db.student.insertMany([
... {
... name : "Gautam",
... age : 21,
... branch : "IT",
... course : "MSC",
... mode : "offline",
... paid : true,
... amount : 52000
... },
...
... {
... name : "Kiran",
... age : 22,
... branch : "IT",
... course : "MSC",
... mode : "offline",
... paid : true,
... amount : 52000
... }
...
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('660d5de931728aa2899f990b'),
    '1': ObjectId('660d5de931728aa2899f990c')
  }
}
BDA>
```

6) Read Operations –

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. You can perform read operation using the following method provided by the MongoDB:

It is used to retrieve documents from the collection.

Code:-

```
db.collection.find().pretty()
```

Example :

In this example, we are retrieving the details of students from the student collection using db.collection.find() method.

7) Update Operations –

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

It is used to update a single document in the collection that satisfy the given criteria.

Code:

```
db.collection.updateOne()
```

Example 1:

In this example, we are updating the age of Sumit in the student collection using db.collection.updateOne() method.

```
BDA> db.student.updateOne({name: "Neeraj"},{$set:{age:30}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('660d5cc431728aa2899f990a'),
    name: 'Neeraj',
    age: 30,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000
  },
  {
    _id: ObjectId('660d5de931728aa2899f990b'),
    name: 'Gautam',
    age: 21,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000
  },
  {
    _id: ObjectId('660d5de931728aa2899f990c'),
    name: 'Kiran',
    age: 22,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000
  }
]
```

It is used to update multiple documents in the collection that satisfy the given criteria.

Code:

db.collection.updateMany()

Example 2:

In this example, we are updating the year of course in all the documents in the student collection using db.collection.updateMany() method.

```
BDA> db.student.updateMany({},{$set:{year:2024}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('660d5cc431728aa2899f990a'),
    name: 'Neeraj',
    age: 30,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990b'),
    name: 'Gautam',
    age: 21,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990c'),
    name: 'Kiran',
    age: 22,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  }
]
BDA>
```

8) Delete Operations –

The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB:

It is used to delete a single document from the collection that satisfy the given criteria.

Code:

db.collection.deleteOne()

Example 1:

In this example, we are deleting a document from the student collection using db.collection.deleteOne() method.

```
BDA> db.student.deleteOne({name : "Neeraj"})
{ acknowledged: true, deletedCount: 1 }
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('660d5de931728aa2899f990b'),
    name: 'Gautam',
    age: 21,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990c'),
    name: 'Kiran',
    age: 22,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  }
]
BDA>
```

It is used to delete multiple documents from the collection that satisfy the given criteria.

Code:

db.collection.deleteMany()

Example 2:

In this example, we are deleting all the documents from the student collection using db.collection.deleteMany() method.

```
BDA> db.student.deleteMany({})
{ acknowledged: true, deletedCount: 2 }
BDA> db.student.find().pretty()

BDA>
```

PRACTICAL NO 4

Aim: To implement programs related to MapReduce.

Software: Hadoop 3.3.0, JDK 8

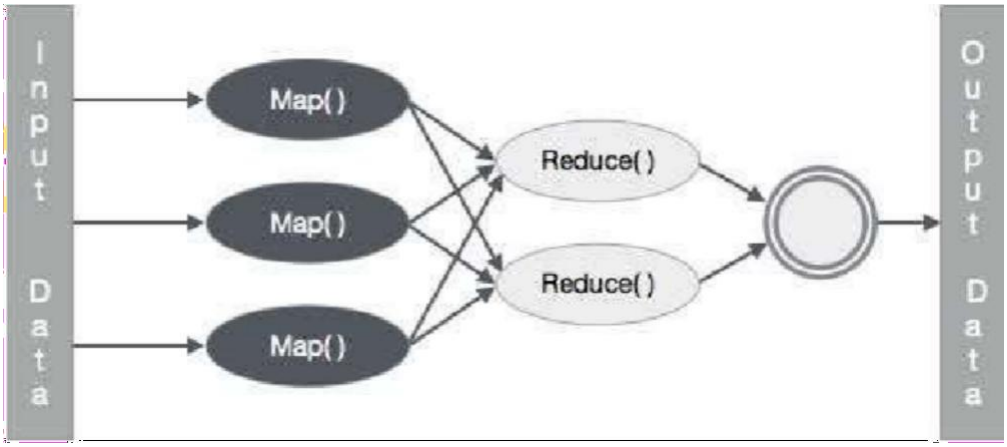
Description:

MapReduce is a framework that is used for writing applications to process huge volumes of data on large clusters of commodity hardware in a reliable manner.

MapReduce Algorithm

Generally, MapReduce paradigm is based on sending map-reduce programs to computers where the actual data resides.

- During a MapReduce job, Hadoop sends Map and Reduce tasks to appropriate servers in the cluster.
- The framework manages all the details of data-passing like issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on the nodes with data on local disks that reduces the network traffic.
- After completing a given task, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



MapReduce Implementation

The following table shows the data regarding the electrical consumption of an organization. The table includes the monthly electrical consumption and the annual average for five consecutive years

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

Input Data

1979	23	23	2	43	24	25	26	26	26	26	25	26	**25**
1980	26	27	28	28	28	30	31	31	31	30	30	30	**29**
1981	31	32	32	32	33	34	35	36	36	34	34	34	**34**
1984	39	38	39	39	39	41	42	43	40	39	38	38	**40**
1985	38	39	39	39	39	41	41	41	00	40	39	39	**45**

Source Code: package
hadoop;

import java.util.*; import
java.io.IOException; import
java.io.IOException;

import org.apache.hadoop.fs.Path; import
org.apache.hadoop.conf.*; import
org.apache.hadoop.io.*; import

```
org.apache.hadoop.mapred.*;  import
org.apache.hadoop.util.*; public class
ProcessUnits

{
   //Mapper class
   public static class E_EMapper extends MapReduceBase implements
   Mapper<LongWritable, /*Input key Type */
    Text,                    /*Input value Type*/
    Text,                    /*Output key Type*/
     IntWritable>              /*Output value Type*/
   {
      //Map function
      public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException
      {
         String line = value.toString();
         String lasttoken = null;
         StringTokenizer s = new StringTokenizer(line,"\t"); String
         year = s.nextToken();

         while(s.hasMoreTokens()){ lastt
            oken=s.nextToken();
         }

         int avgprice = Integer.parseInt(lasttoken); output.collect(new
      Text(year), new IntWritable(avgprice)); }
   }
```

```
//Reducer class


public static class E_EReduce extends MapReduceBase implements
Reducer< Text, IntWritable, Text, IntWritable >
{
   //Reduce function
   public void reduce(Text key, Iterator <IntWritable> values,
OutputCollector>Text, IntWritable> output, Reporter reporter) throws
IOException
   { int maxavg=30; int
      val=Integer.MIN_VALUE;
      while (values.hasNext())
      { if((val=values.next().get())>maxavg)
         { output.collect(key, new IntWritable(val));
         }
      }
   }
}


//Main function public static void main(String
args[])throws Exception
{
   JobConf conf = new JobConf(Eleunits.class);
   conf.setJobName("max_eletricityunits");
   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);
   conf.setMapperClass(E_EMapper.class);
   conf.setCombinerClass(E_EReduce.class);
   conf.setReducerClass(E_EReduce.class);
```

```
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

Steps to compile and run code:

Follow the steps given below to compile and execute the above program.

Step 1 − Use the following command to create a directory to store the compiled java classes. $ mkdir units

Step 2 − Download Hadoop-core-1.2.1.jar, which is used to compile and execute the MapReduce program. Download the jar from mvnrepository.com. Let us assume the download folder is /home/hadoop/.

Step 3 − The following commands are used to compile the ProcessUnits.java program and to create a jar for the program.

$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java

$ jar -cvf units.jar -C units/ .

Step 4 − The following command is used to create an input directory in HDFS.

$HADOOP_HOME/bin/hadoop fs -mkdir input_dir

Step 5 − The following command is used to copy the input file named sample.txt in the input directory of HDFS.

$HADOOP_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input_dir

Step 6 − The following command is used to verify the files in the input directory

$HADOOP_HOME/bin/hadoop fs -ls input_dir/

Step 7 − The following command is used to run the Eleunit_max application by taking input files from the input directory.

$HADOOP_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input_dir output_dir

Wait for a while till the file gets executed. After execution, the output

contains a number of input splits, Map tasks, Reducer tasks, etc. INFO

mapreduce.Job: Job job_1414748220717_0002 completed successfully

14/10/31 06:02:52

INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=61

FILE: Number of bytes written=279400

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=546

HDFS: Number of bytes written=40

HDFS: Number of read operations=9

HDFS: Number of large read operations=0

HDFS: Number of write operations=2 Job Counters

Launched map tasks=2

Launched reduce tasks=1

Data-local map tasks=2

Total time spent by all maps in occupied slots (ms)=146137

Total time spent by all reduces in occupied slots (ms)=441

Total time spent by all map tasks (ms)=14613

Total time spent by all reduce tasks (ms)=44120

Total vcore-seconds taken by all map tasks=146137

Total vcore-seconds taken by all reduce tasks=44120

Total megabyte-seconds taken by all map tasks=149644288

Total megabyte-seconds taken by all reduce tasks=45178880

Map-Reduce Framework

Map input records=5

Map output records=5

Map output bytes=45

Map output materialized bytes=67

Input split bytes=208

Combine input records=5

Combine output records=5

Reduce input groups=5

Reduce shuffle bytes=6

Reduce input records=5

Reduce output records=5

Spilled Records=10

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=948

CPU time spent (ms)=5160

Physical memory (bytes) snapshot=47749120

Virtual memory (bytes) snapshot=2899349504

Total committed heap usage (bytes)=277684224

File Output Format Counters

Bytes Written=40

Step 8 − The following command is used to verify the resultant files in the output folder.

$HADOOP_HOME/bin/hadoop fs -ls output_dir/

Step 9 − The following command is used to see the output in Part00000 file. This file is generated by HDFS.

$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000

Following is the output generated by the MapReduce program −

1981 34

1984 40

1985 45

Step 10 − The following command is used to copy the output folder from HDFS to the local file system.

$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000/bin/hadoop dfs -get output_dir /home/Hadoop

PRACTICAL NO 5

Aim: Implement Clustering and Associated algorithms

Software: R Studio

Download R Studio 4.3.0 from the official website (https://posit.co/download/rstudio-desktop/ Desktop - Posit) and install it

Description:

1. Data preparation

2. Assessing clustering tendency (i.e., the cluster ability of the data)

3. Defining the optimal number of clusters

4. Computing partitioning cluster analyses (e.g.: K-means, pam) or hierarchical clustering

5. Validating Clustering analyses

Code:

> idx<-sample(1:dim(iris)[1],40)

> irisSample<-iris[idx,]

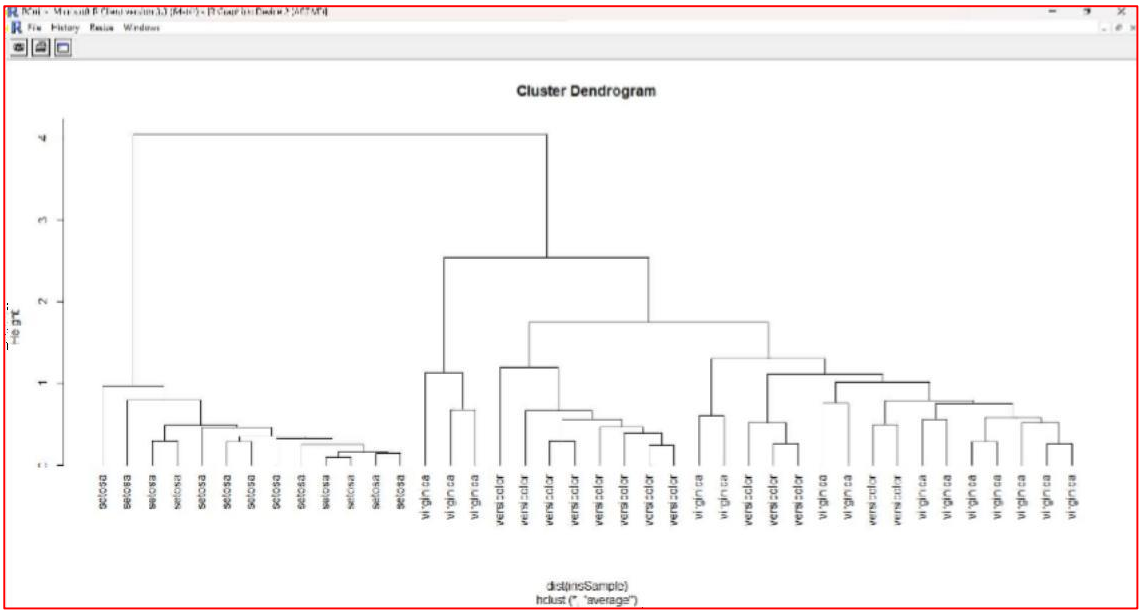
> irisSample$Species<-NULL

> hc<-hclust(dist(irisSample),method="ave")

> plot(hc,hang=-1,labels=iris$Species[idx])

```
> idx<-sample(1:dim(iris)[1],40)
> irisSample<-iris[idx,]
> irisSample$Species<-NULL
> hc<-hclust(dist(irisSample),method="ave")
> plot(hc,hang=-1,labels=iris$Species[idx])
>
```

Output:



Association Rule Mining:

Code:

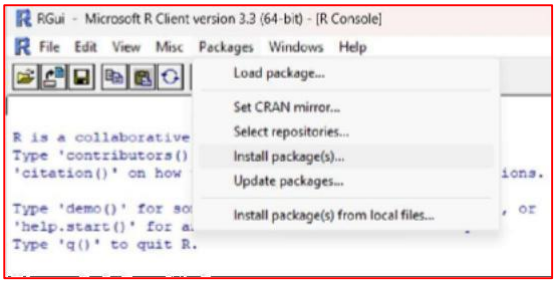
>#load data

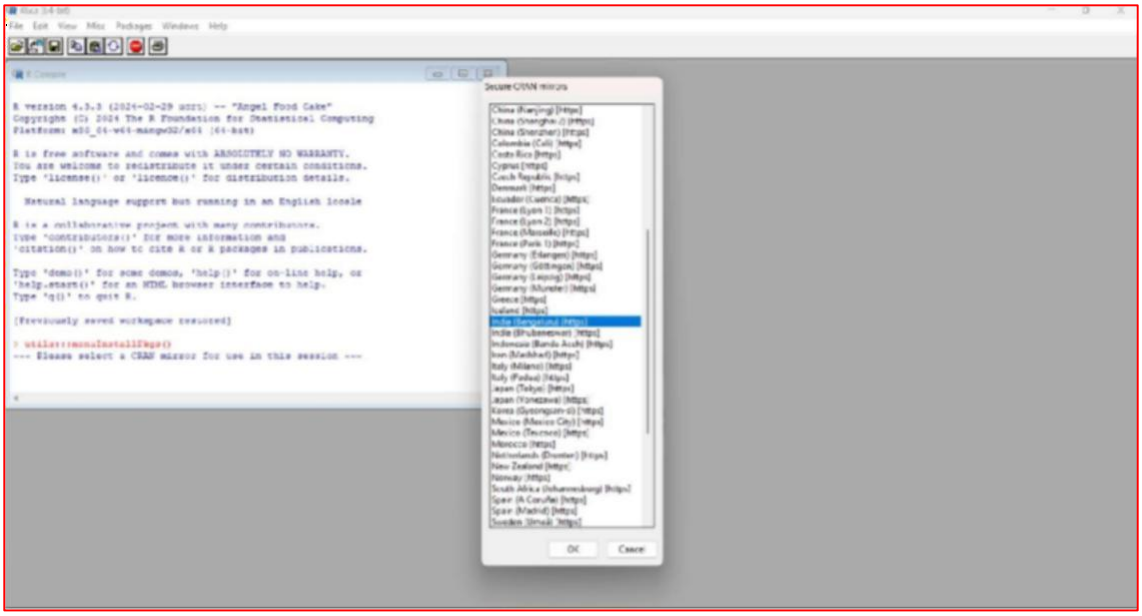
> load("D:/titanic.raw.rdata")

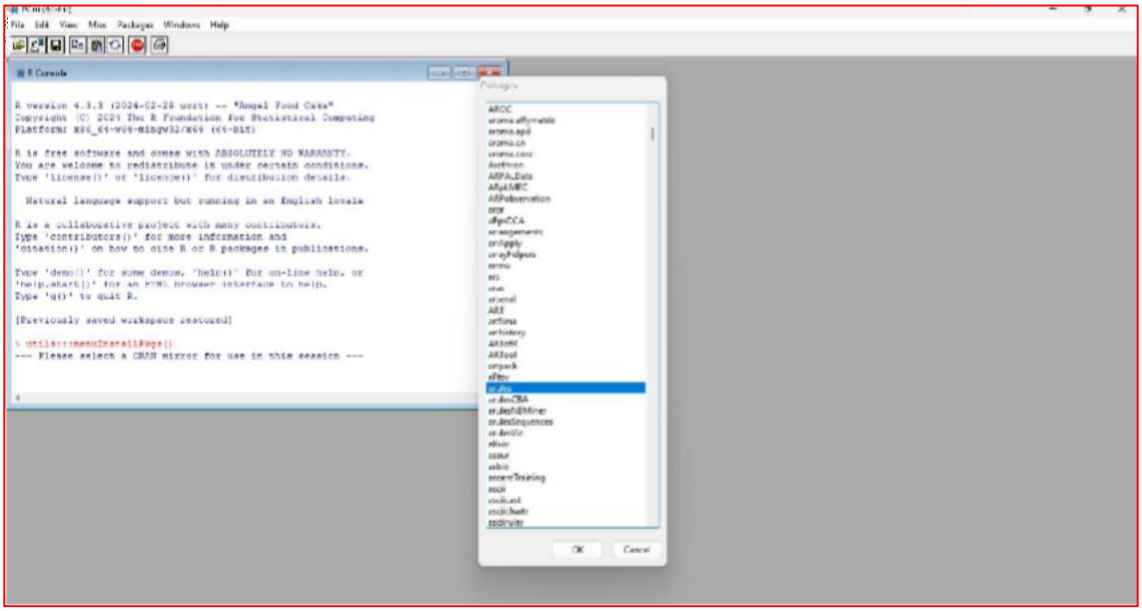
> str(titanic.raw)

```
> load("D:/titanic.raw.rdata")
> str(titanic.raw)
'data.frame':   2201 obs. of  4 variables:
 $ Class    : Factor w/ 4 levels "1st","2nd","3rd",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Sex      : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ Age      : Factor w/ 2 levels "Adult","Child": 2 2 2 2 2 2 2 2 2 2 ...
 $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Go to "**Packages**" select "**install packages**" select **India (Bengaluru) [https]** and select **"arules".** Click on yes and then it will install the packages.







Code:

> library(arules)

> # find association rules with default settings

> rules <- apriori(titanic.raw)

> inspect(rules)

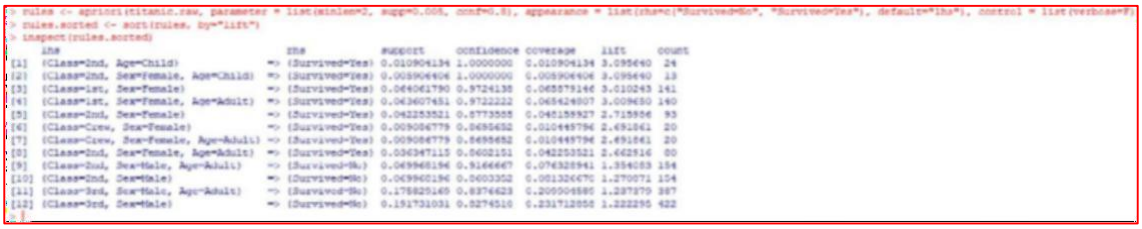
```
> library(arules)
Loading required package: Matrix

Attaching package: 'arules'

The following objects are masked from 'package:base':

    abbreviate, write
```

```
> rules <- apriori(titanic.raw)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 220

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
> inspect(rules)
     lhs                                    rhs          support    confidence coverage  lift      count
[1]  {}                                  => {Age=Adult}  0.9504771 0.9504771  1.0000000 1.0000000 2092
[2]  {Class=2nd}                         => {Age=Adult}  0.1185825 0.9157895  0.1294866 0.9635051  261
[3]  {Class=1st}                         => {Age=Adult}  0.1449341 0.9815385  0.1476602 1.0326798  319
[4]  {Sex=Female}                        => {Age=Adult}  0.1930940 0.9042553  0.2135393 0.9513700  425
[5]  {Class=3rd}                         => {Age=Adult}  0.2848705 0.8881020  0.3207633 0.9343750  627
[6]  {Survived=Yes}                      => {Age=Adult}  0.2971377 0.9198312  0.3230350 0.9677574  654
[7]  {Class=Crew}                        => {Sex=Male}   0.3916402 0.9740113  0.4020900 1.2384742  862
[8]  {Class=Crew}                        => {Age=Adult}  0.4020900 1.0000000  0.4020900 1.0521033  885
[9]  {Survived=No}                       => {Sex=Male}   0.6197183 0.9154362  0.6769650 1.1639949 1364
[10] {Survived=No}                       => {Age=Adult}  0.6533394 0.9651007  0.6769650 1.0153856 1438
[11] {Sex=Male}                          => {Age=Adult}  0.7573830 0.9630272  0.7864607 1.0132040 1667
[12] {Sex=Female, Survived=Yes}          => {Age=Adult}  0.1435711 0.9186047  0.1562926 0.9664669  316
[13] {Class=3rd, Sex=Male}               => {Survived=No} 0.1917310 0.8274510  0.2317129 1.2222950  422
[14] {Class=3rd, Survived=No}            => {Age=Adult}  0.2162653 0.9015152  0.2398910 0.9484870  476
[15] {Class=3rd, Sex=Male}               => {Age=Adult}  0.2099046 0.9058824  0.2317129 0.9530818  462
[16] {Sex=Male, Survived=Yes}            => {Age=Adult}  0.1535666 0.9209809  0.1667424 0.9689670  338
[17] {Class=Crew, Survived=No}           => {Sex=Male}   0.3044071 0.9955423  0.3057701 1.2658514  670
[18] {Class=Crew, Survived=No}           => {Age=Adult}  0.3057701 1.0000000  0.3057701 1.0521033  673
[19] {Class=Crew, Sex=Male}              => {Age=Adult}  0.3916402 1.0000000  0.3916402 1.0521033  862
[20] {Class=Crew, Age=Adult}             => {Sex=Male}   0.3916402 0.9740113  0.4020900 1.2384742  862
[21] {Sex=Male, Survived=No}             => {Age=Adult}  0.6038164 0.9743402  0.6197183 1.0251065 1329
[22] {Age=Adult, Survived=No}            => {Sex=Male}   0.6038164 0.9242003  0.6533394 1.1751385 1329
[23] {Class=3rd, Sex=Male, Survived=No}  => {Age=Adult}  0.1758292 0.9170616  0.1917310 0.9648435  387
[24] {Class=3rd, Age=Adult, Survived=No} => {Sex=Male}   0.1758292 0.8130252  0.2162653 1.0337773  387
[25] {Class=3rd, Sex=Male, Age=Adult}    => {Survived=No} 0.1758292 0.8376623  0.2099046 1.2373791  387
[26] {Class=Crew, Sex=Male, Survived=No} => {Age=Adult}  0.3044071 1.0000000  0.3044071 1.0521033  670
[27] {Class=Crew, Age=Adult, Survived=No} => {Sex=Male}  0.3044071 0.9955423  0.3057701 1.2658514  670
>
```

Code:

```
> # rules with rhs containing "Survived" only
> rules <- apriori(titanic.raw, parameter = list(minlen=2, supp=0.005, conf=0.8),
appearance = list(rhs=c("Survived=No", "Survived=Yes"), default="lhs"),
control = list(verbose=F))
> rules.sorted <- sort(rules, by="lift")
> inspect(rules.sorted)
```



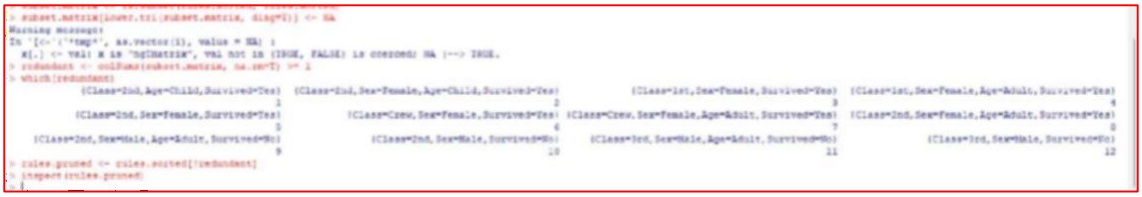
Pruning Redundant Rules

In the above result, rule 2 provides no extra knowledge in addition to rule 1, since rules 1 tells us that all 2nd-class children survived. Generally speaking, when a rule (such as rule 2) is a super rule of another rule (such as rule 1) and the former has the same or a lower lift, the former rule (rule 2) is considered to be redundant. Below we prune redundant rules.

Code:

```
> # find redundant rules
> subset.matrix <- is.subset(rules.sorted, rules.sorted)
> subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
> redundant <- colSums(subset.matrix, na.rm=T) >= 1
> which(redundant)
> # remove redundant rules
> rules.pruned <- rules.sorted[!redundant]
> inspect(rules.pruned)
```

Output:


> subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
Warning message:
In `[<-`(`*tmp*`, as.vector(i), value = NA) :
x[i,] <- val: x is "ngCMatrix", val not in {TRUE, FALSE} is coerced: NA --> TRUE.
> redundant <- colSums(subset.matrix, na.rm=T) >= 1
> which(redundant)
{Class=2nd,Age=Child,Survived=Yes} {Class=2nd,Sex=Female,Age=Child,Survived=Yes} {Class=1st,Sex=Female,Survived=Yes} {Class=1st,Sex=Female,Age=Adult,Survived=Yes}
1 2 3 4
{Class=2nd,Sex=Female,Survived=Yes} {Class=Crew,Sex=Female,Survived=Yes} {Class=Crew,Sex=Female,Age=Adult,Survived=Yes} {Class=2nd,Sex=Female,Age=Adult,Survived=Yes}
5 6 7 8
{Class=2nd,Sex=Male,Age=Adult,Survived=No} {Class=2nd,Sex=Male,Survived=No} {Class=3rd,Sex=Male,Age=Adult,Survived=No} {Class=3rd,Sex=Male,Survived=No}
9 10 11 12
> rules.pruned <- rules.sorted[!redundant]
> inspect(rules.pruned)

PRACTICAL NO 6

Aim: To implement Linear Regression.

Software: Jupyter

Datasets: data.csv

Step 1: Import libraries and dataset.

Import the important libraries and the dataset we are using to perform Polynomial Regression.

Step 2: Dividing the dataset into 2 components.

Divide dataset into two components that is X and y. X will contain the Column between 1 and 2. y will contain the 2 column.

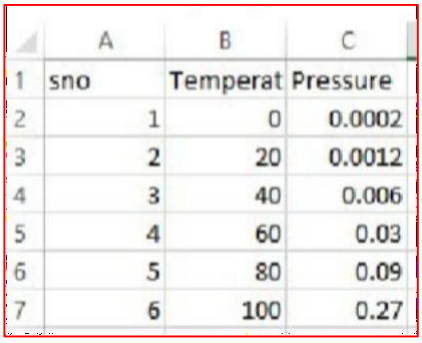
Step 3: Fitting Linear Regression to the dataset

Fitting the linear Regression model On two components.

Step 4: Fitting Polynomial Regression to the dataset

Fitting the Polynomial Regression model on two components X and y.

Step 5: In this step we are Visualising the Linear Regression results using scatter plot.



	A	B	C
1	sno	Temperat	Pressure
2	1	0	0.0002
3	2	20	0.0012
4	3	40	0.006
5	4	60	0.03
6	5	80	0.09
7	6	100	0.27

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Step 1 :Import libraries and dataset

datas = pd.read_csv('data.csv')
```

print(datas)

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd

In [5]: datas = pd.read_csv('datacsv.csv')
        print(datas)

           srno  Temperature  Pressure
        0     1            0    0.0002
        1     2           20    0.0012
        2     3           40    0.0060
        3     4           60    0.0300
        4     5           80    0.0900
        5     6          100    0.2700
```

Step 2: Dividing the dataset into 2 components

X = datas.iloc[:, 1:2].values
y = datas.iloc[:, 2].values

Step 3: Fitting Linear Regression to the dataset

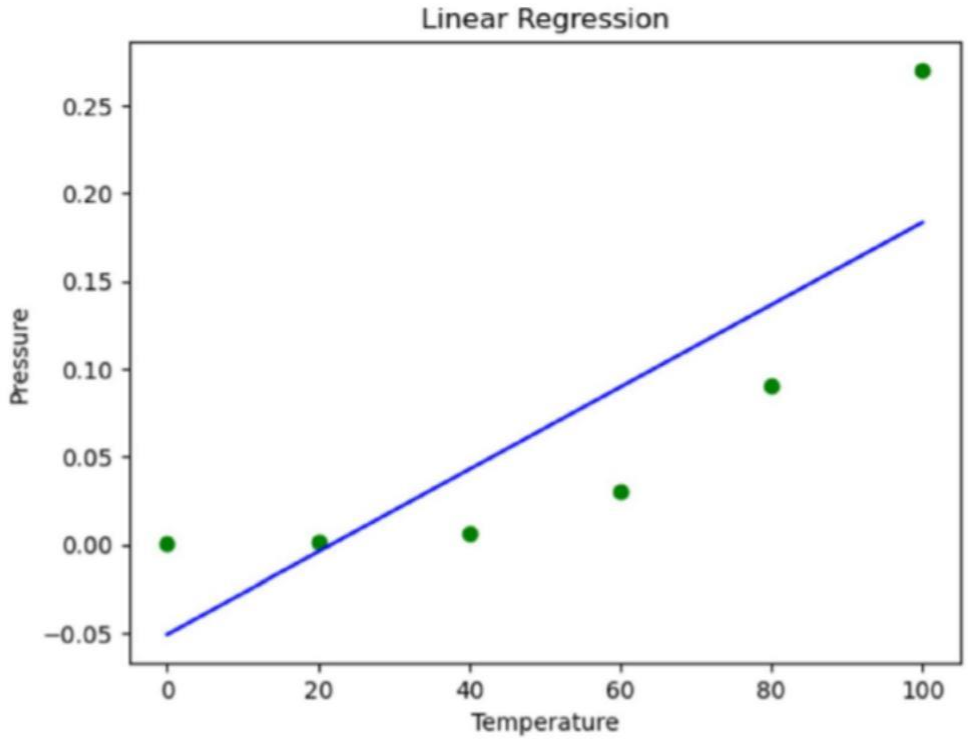
from sklearn.linear_model import LinearRegression
lin = LinearRegression()
lin.fit(X, y)

```
In [6]: X = datas.iloc[:, 1:2].values
        y = datas.iloc[:, 2].values

In [7]: from sklearn.linear_model import LinearRegression
        lin = LinearRegression()
        lin.fit(X, y)

Out[7]: LinearRegression
        LinearRegression()
```

Step 4: Visualising the Linear Regression results

plt.scatter(X, y, color = 'blue')
plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')
plt.show()

```
In [9]: plt.scatter(X, y, color = 'green')
        plt.plot(X, lin.predict(X), color = 'blue')
        plt.title('Linear Regression')
        plt.xlabel('Temperature')
        plt.ylabel('Pressure')
        plt.show()
```



PRACTICAL NO 7

Aim: To Implement Bloom Filters for Filter Stream Data

Software: Python Idle

Steps:

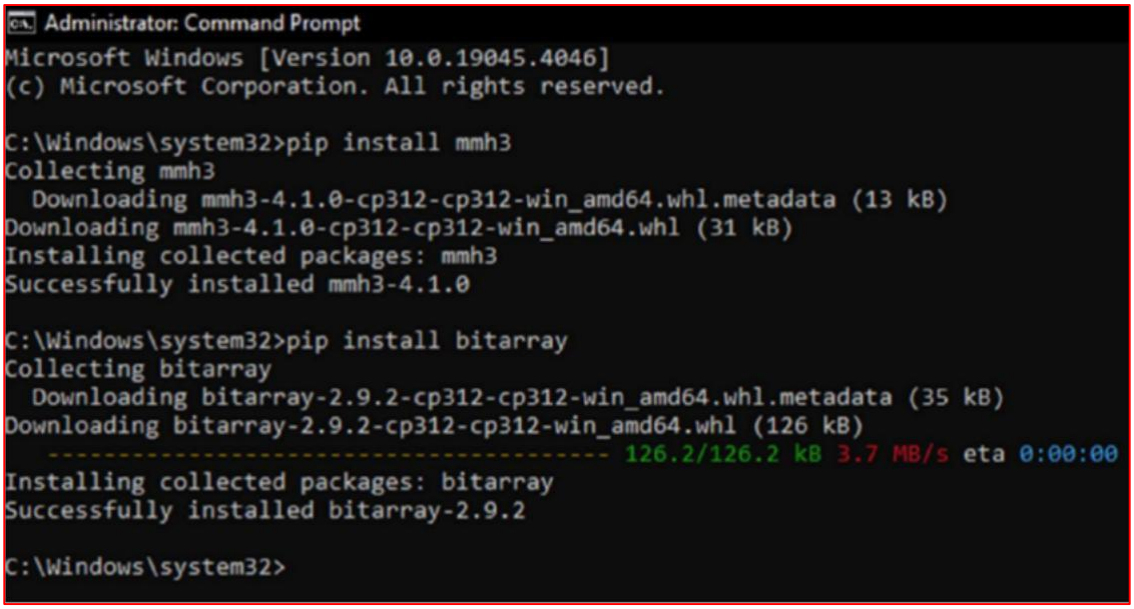
1) Open command prompt as administrator and perform below commands:

Code:

pip install mmh3

pip install bitarray

Output:



2) Open IDLE Python

Note:-

set environment variable
C:\Users\sies\AppData\Local\Programs\Python\Python312\Scripts

Paste the above defined directory as a new path in environment variable

Here we need to create two python files **"bloomfilter.py"** and **"bloom_test.py"**

Now, open idle python and create file 1 as bloomfilter.py paste the below code

Code:

"bloomfilter.py"

```
import math
import mmh3
from bitarray import bitarray


class BloomFilter(object):

'''
Class for Bloom filter, using murmur3 hash function
'''

def __init__(self, items_count, fp_prob):
'''
items_count : int
Number of items expected to be stored in bloom filter
fp_prob : float
False Positive probability in decimal
'''
# False possible probability in decimal
self.fp_prob = fp_prob

# Size of bit array to use
self.size = self.get_size(items_count, fp_prob)

# number of hash functions to use
self.hash_count = self.get_hash_count(self.size, items_count)

# Bit array of given size
self.bit_array = bitarray(self.size)

# initialize all bits as 0
self.bit_array.setall(0)

def add(self, item):
'''
Add an item in the filter
'''
digests = []
for i in range(self.hash_count):

# create digest for given item.
```

```
# i work as seed to mmh3.hash() function
# With different seed, digest created is different
digest = mmh3.hash(item, i) % self.size
digests.append(digest)

# set the bit True in bit_array
self.bit_array[digest] = True

def check(self, item):
'''
Check for existence of an item in filter
'''
for i in range(self.hash_count):
digest = mmh3.hash(item, i) % self.size
if self.bit_array[digest] == False:

# if any of bit is False then,its not present
# in filter
# else there is probability that it exist
return False
return True

@classmethod
def get_size(self, n, p):
'''
Return the size of bit array(m) to used using
following formula
m = -(n * lg(p)) / (lg(2)^2)
n : int
number of items expected to be stored in filter
p : float
False Positive probability in decimal
'''
m = -(n * math.log(p))/(math.log(2)**2)
return int(m)

@classmethod
def get_hash_count(self, m, n):
'''
Return the hash function(k) to be used using
following formula
k = (m/n) * lg(2)

m : int
size of bit array
n : int
```

```
number of items expected to be stored in filter
'''
k = (m/n) * math.log(2)
return int(k)
```

Now, open idle python and create file 2 as bloom_test.py paste the below code

"bloom_test.py"

Code:

```
from bloomfilter import BloomFilter
from random import shuffle

n = 20 #no of items to add
p = 0.05 #false positive probability

bloomf = BloomFilter(n,p)
print("Size of bit array:{}".format(bloomf.size))
print("False positive Probability:{}".format(bloomf.fp_prob))
print("Number of hash functions:{}".format(bloomf.hash_count))

# words to be added
word_present = ['abound','abounds','abundance','abundant','accessible',
'bloom','blossom','bolster','bonny','bonus','bonuses',
'coherent','cohesive','colorful','comely','comfort',
'gems','generosity','generous','generously','genial']

# word not added
word_absent = ['bluff','cheater','hate','war','humanity',
'racism','hurt','nuke','gloomy','facebook',
'geeksforgeeks','twitter']

for item in word_present:
bloomf.add(item)

shuffle(word_present)
shuffle(word_absent)

test_words = word_present[:10] + word_absent
shuffle(test_words)
for word in test_words:
if bloomf.check(word):
if word in word_absent:
print("'{}' is a false positive!".format(word))
else:
```

```
print("'{}' is probably present!".format(word))
else:
print("'{}' is definitely not present!".format(word))
```

2) Now run the “bloom_test.py” file

Output:

PRACTICAL NO 8

Aim: To Implement Time Series

Software: R Studio

Description:

Time Series

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame. The time series object is created by using the ts () function.

Syntax

The basic syntax for ts() function in time series analysis is − timeseries.object.name <- ts(data, start, end, frequency) Following is the description of the parameters used −

- data is a vector or matrix containing the values used in the time series.
- start specifies the start time for the first observation in time series.
- end specifies the end time for the last observation in time series.
- frequency specifies the number of observations per unit time. Except the parameter "data" all other parameters are optional.

Example:

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

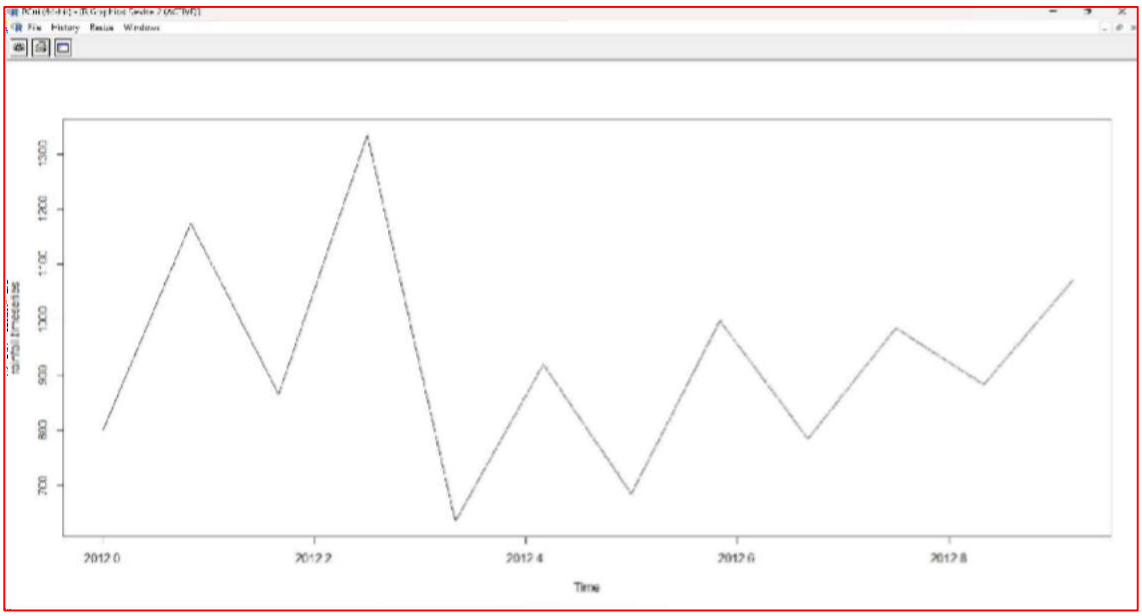
Code:

```
# Get the data points in form of a R vector.

rainfall <-
c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

# Convert it to a time series object.

        rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
```

```
# Print the timeseries data.
        print(rainfall.timeseries)
# Give the chart file a name.
        png(file = "rainfall.png")
# Plot a graph of the time series.
        plot(rainfall.timeseries)
        dev.off()
        plot(rainfall.timeseries)
```

Output:

```
> rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
> rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
> print(rainfall.timeseries)
         Jan     Feb     Mar     Apr     May     Jun     Jul     Aug     Sep     Oct     Nov     Dec
2012  799.0 1174.8  865.1 1334.6  635.4  918.5  685.5  998.6  784.2  985.0  882.8 1071.0
> png(file = "rainfall.png")
> plot(rainfall.timeseries)
> dev.off()
null device
          1
> plot(rainfall.timeseries)
>
```



Different Time Intervals

The value of the frequency parameter in the ts() function decides the time intervals at which the data points are measured. A value of 12 indicates that the time series is for 12 months. Other values and its meaning is as below −

- frequency = 12 pegs the data points for every month of a year.
- frequency = 4 pegs the data points for every quarter of a year.
- frequency = 6 pegs the data points for every 10 minutes of an hour.
- frequency = 24*6 pegs the data points for every 10 minutes of a day.

Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.

Code:

```
# Get the data points in form of a R vector.

rainfall1 <-
c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

rainfall2 <-
c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.
8)

# Convert them to a matrix.

    combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)

# Convert it to a time series object.

rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.

    print(rainfall.timeseries)

# Give the chart file a name.

    png(file = "rainfall_combined.png")

# Plot a graph of the time series.

    plot(rainfall.timeseries, main = "Multiple Time Series")

# Save the file.

    dev.off()
```

plot(rainfall.timeseries, main = "Multiple Time Series")

Output:

```
> rainfall1 <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
> rainfall2 <- c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)
> combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)
> rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)
> print(rainfall.timeseries)
         Series 1 Series 2
Jan 2012    799.0    655.0
Feb 2012   1174.8   1306.9
Mar 2012    865.1   1323.4
Apr 2012   1334.6   1172.2
May 2012    635.4    562.2
Jun 2012    918.5    824.0
Jul 2012    685.5    822.4
Aug 2012    998.6   1265.5
Sep 2012    784.2    799.6
Oct 2012    985.0   1105.6
Nov 2012    882.8   1106.7
Dec 2012   1071.0   1337.8
> png(file = "rainfall_combined.png")
> plot(rainfall.timeseries, main = "Multiple Time Series")
> dev.off()
windows
      2
> plot(rainfall.timeseries, main = "Multiple Time Series")
>
```