Practical – 1: Install configure and run python, numpy and pandas.

Step 1: Open Jupyterlite or install Anaconda Navigator and open Jupyter Notebook. Type the following code.

```
import pandas as pd
data = pd.read_csv("crime.csv")
data
```

➢

	Year	Population	Murder	Rape	Robbery	Assault	Burglary	CarTheft
0	1965	18073000	836	2320	28182	27464	183443	58452
1	1966	18258000	882	2439	30098	29142	196127	64368
2	1967	18336000	996	2665	40202	31261	219157	83775
3	1968	18113000	1185	2527	59857	34946	250918	104877
4	1969	18321000	1324	2902	64754	36890	248477	115400
5	1970	18190740	1444	2875	81149	39145	267474	125674
6	1971	18391000	1823	3225	97682	42318	273704	127658
7	1972	18366000	2026	4199	86391	45926	239886	105081
8	1973	18265000	2040	4852	80795	47781	246246	112328
9	1974	18111000	1919	5240	86814	51454	271824	104095
10	1975	18120000	1996	5099	93499	54593	301996	116274
11	1976	18084000	1969	4663	95718	54638	318919	133504
12	1977	17924000	1919	5272	84703	57193	309735	133669
13	1978	17748000	1820	5168	83785	58484	292956	119264

```
type(data)
```

➢

```
pandas.core.frame.DataFrame
```

```
data.shape
```

➢

```
(47, 8)
```

data.isnull().tail()

➢

	Year	Population	Murder	Rape	Robbery	Assault	Burglary	CarTheft
42	False	False	False	False	False	False	False	False
43	False	False	False	False	False	False	False	False
44	False	False	False	False	False	False	False	False
45	False	False	False	False	False	False	False	False
46	False	False	False	False	False	False	False	False

data.notnull().tail()

➢

	Year	Population	Murder	Rape	Robbery	Assault	Burglary	CarTheft
42	True	True	True	True	True	True	True	True
43	True	True	True	True	True	True	True	True
44	True	True	True	True	True	True	True	True
45	True	True	True	True	True	True	True	True
46	True	True	True	True	True	True	True	True

data.isnull().sum()

➢

```
Year          0
Population    0
Murder        0
Rape          0
Robbery       0
Assault       0
Burglary      0
CarTheft      0
dtype: int64
```

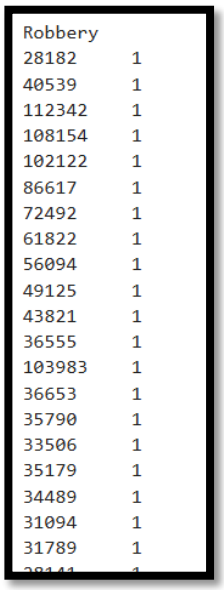
data[data.Robbery.isnull()]

➢

Year	Population	Murder	Rape	Robbery	Assault	Burglary	CarTheft

data['Robbery'].value_counts()

➢

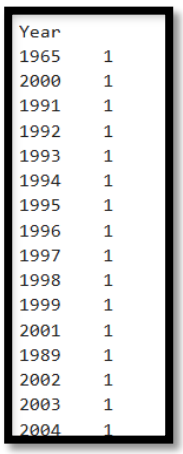


```
Robbery
28182     1
40539     1
112342    1
108154    1
102122    1
86617     1
72492     1
61822     1
56094     1
49125     1
43821     1
36555     1
103983    1
36653     1
35790     1
33506     1
35179     1
34489     1
31094     1
31789     1
```

for col in data.columns:

display(data[col].value_counts())

➢



```
Year
1965    1
2000    1
1991    1
1992    1
1993    1
1994    1
1995    1
1996    1
1997    1
1998    1
1999    1
2001    1
1989    1
2002    1
2003    1
2004    1
```

data_len = len(data)

data_len

➢

47

data_col = len(data.columns)

data_col

➢

8

data.describe()

➢

	Year	Population	Murder	Rape	Robbery	Assa
count	47.000000	4.700000e+01	47.000000	47.000000	47.000000	47.000(
mean	1988.000000	1.834426e+07	1549.978723	4200.425532	70429.297872	58022.234(
std	13.711309	6.024504e+05	590.454265	1096.569507	30204.823764	17455.534:
min	1965.000000	1.750669e+07	774.000000	2320.000000	28141.000000	27464.000(
25%	1976.500000	1.793700e+07	922.500000	3197.000000	36604.000000	45477.500(
50%	1988.000000	1.816900e+07	1683.000000	4199.000000	81149.000000	57193.000(
75%	1999.500000	1.868373e+07	2016.000000	5241.000000	94141.000000	64864.500(
max	2011.000000	1.954145e+07	2605.000000	5706.000000	120344.000000	92105.000(

data.Murder.describe()

➢

```
count      47.000000
mean     1549.978723
std       590.454265
min       774.000000
25%       922.500000
50%      1683.000000
75%      2016.000000
max      2605.000000
Name: Murder, dtype: float64
```

data.skew()

➢

```
Year          0.000000
Population    0.795669
Murder        0.059733
Rape         -0.237130
Robbery      -0.134085
Assault       0.464637
Burglary     -0.020278
CarTheft     -0.129653
dtype: float64
```

data.var()

➢

```
Year          1.880000e+02
Population    3.629465e+11
Murder        3.486362e+05
Rape          1.202465e+06
Robbery       9.123314e+08
Assault       3.046957e+08
Burglary      8.146192e+09
CarTheft      2.181550e+09
dtype: float64
```

data.kurtosis()

➢

```
Year         -1.200000
Population   -0.692220
Murder       -1.513564
Rape         -1.471445
Robbery      -1.527674
Assault      -0.482013
Burglary     -1.186281
CarTheft     -0.951036
dtype: float64
```

print(data.dtypes)

➢

```
Year          int64
Population    int64
Murder        int64
Rape          int64
Robbery       int64
Assault       int64
Burglary      int64
CarTheft      int64
dtype: object
```

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Array:", arr)

print(type(arr))

➢

Array: [1 2 3 4 5]

<class 'numpy.ndarray'>

print(np.__version__)

➢

2.0.2

ar1 = np.array(42)

print("Dimension:", ar1.ndim)

print("0-D array:", ar1)

ar2 = np.array((1, 2, 3, 4, 5))

print("Dimension:", ar2.ndim)

print("1-D array:", ar2)

ar3 = np.array([[1, 2, 3], [4, 5, 6]])

print("Dimension:", ar3.ndim)

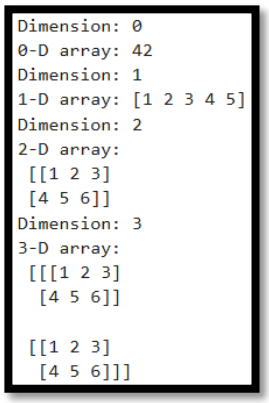
print("2-D array:\n", ar3)

ar4 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print("Dimension:", ar4.ndim)

print("3-D array:\n", ar4)

➢



ar5 = np.array([1, 2, 3, 4], ndmin = 5)

print(ar5)

```
print("Number of dimensions:", ar5.ndim)
```

➢

```
[[[[[1 2 3 4]]]]]
Number of dimensions: 5
```

```
arr = np.array([1, 2, 3, 4])
print("First element:", arr[0])
print("Last element:", arr[-1])
print("Sum of second and third element:", arr[1] + arr[2])
```

➢

```
First element: 1
Last element: 4
Sum of second and third element: 5
```

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print("Second element on 1st row:", arr[0, 1])
print('Fifth element on 2nd row:', arr[1, 4])
```

➢

```
Second element on 1st row: 2
Fifth element on 2nd row: 10
```

Practical – 2: Install, configure and run Hadoop and HDFS.

Prerequisites:

1. JRE 1.8
2. JDK 1.8

Step 1: Download and install Hadoop 2.9.2 or any other stable version. Install the Hadoop file with the extension tar.gz. Unzip the Hadoop-2.9.2.tar.gz file. Once extracted, again extract the .tar file.

Step 2: Add the environment variables for jdk. Create a new system variable named JAVA_HOME and provide its value as the installation folder of your java jdk file (C:\Program Files\Java\jdk-1.8). Add the path to bin folder (%JAVA_HOME%\bin) under the path system variable to access java in your terminal.

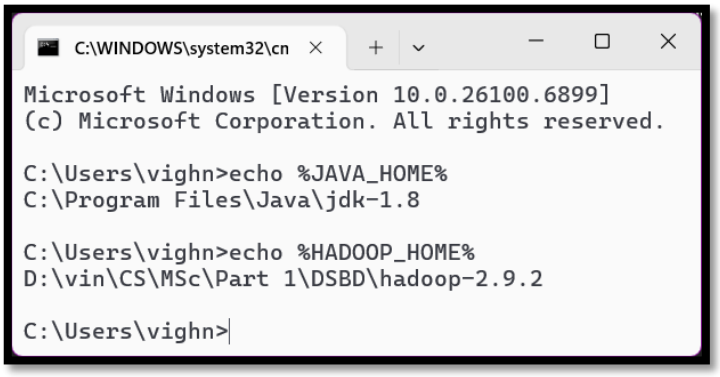
Step 3: Add another environment variable named HADOOP_HOME under system variables and add its value as the installation folder of the Hadoop folder (D:\CS\MSC\hadoop-2.9.2). Add the path to the bin and sbin folder of Hadoop installation as: %HADOOP_HOME%\sbin and %HADOOP_HOME%\bin.

Step 4: Run the following commands in your terminal to verify that the paths have been added to the system correctly.

echo %JAVA_HOME%

echo %HADOOP_HOME%

echo %PATH%



Step 5: Once we have configured the environment variables, next step is to configure Hadoop. It has 3 parts in total.

Step 5.1: We need to create a folder named data in the Hadoop directory first. Add 2 folders named namenode and datanode in the data folder.

Step 5.2: We need to edit the following config files in Hadoop for configuring it:

(These files are located in Hadoop -> etc -> hadoop)

- core-site.xml
- hdfs-site.xml
- mapred-site.xml
- yarn-site.xml

Step 5.2.1: Editing core-site.xml file (change the content inside the configuration tag):

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

Step 5.2.2: Editing hdfs-site.xml file:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
```

```
    <value>D:\vin\CS\MSc\Part 1\DSBD\hadoop-2.9.2\data\namenode</value>
  </property>
  <property>
    <name>dfs.datanode.name.dir</name>
    <value>D:\vin\CS\MSc\Part 1\DSBD\hadoop-2.9.2\data\datanode</value>
  </property>
</configuration>
```

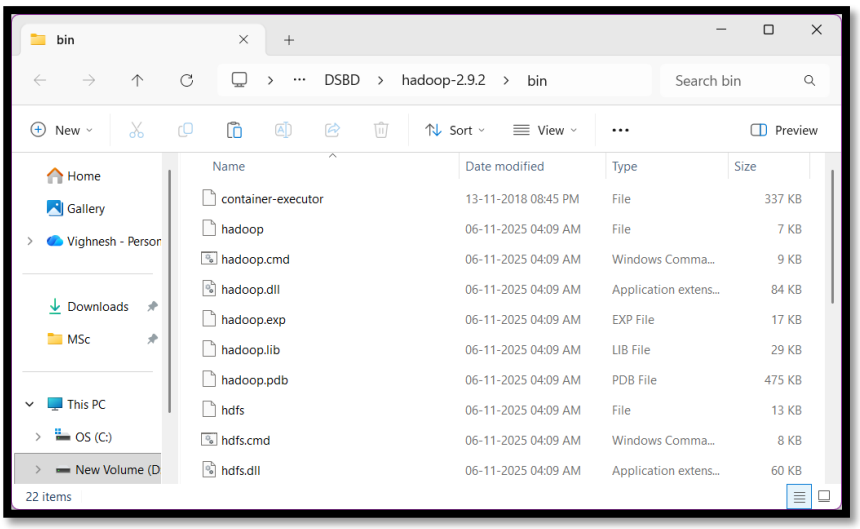
Step 5.2.3: Editing mapred-site.xml file:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Step 5.2.4: Editing yarn-site.xml file:

```
<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>
</configuration>
```

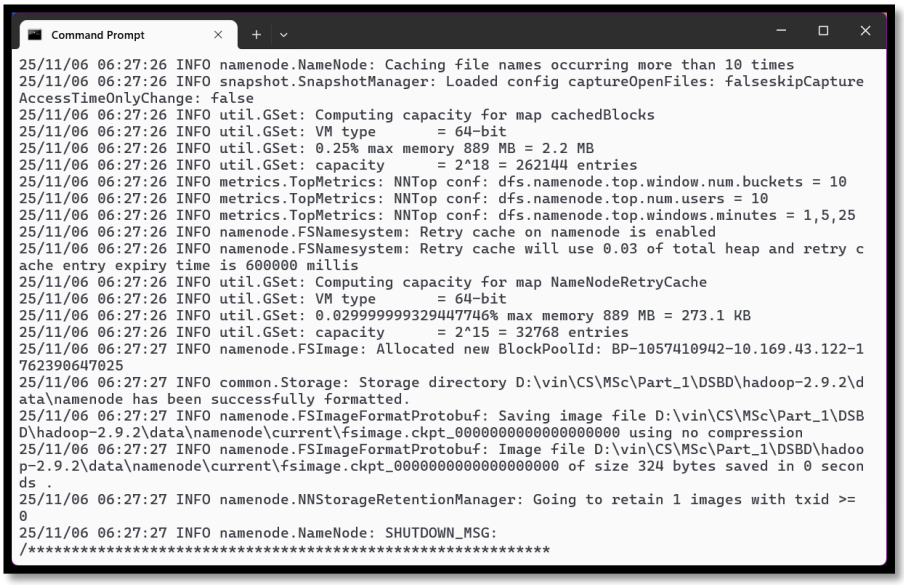
Step 5.3: Download and replace the bin folder inside the Hadoop files. Go to the GitHub repo and download the zip file. Extract the folder and then copy the bin folder of Hadoop-2.9.2 and replace the bin files inside your Hadoop installation.



Step 6: We need to test the setup now and for that we will first format the namenode and then launch Hadoop.

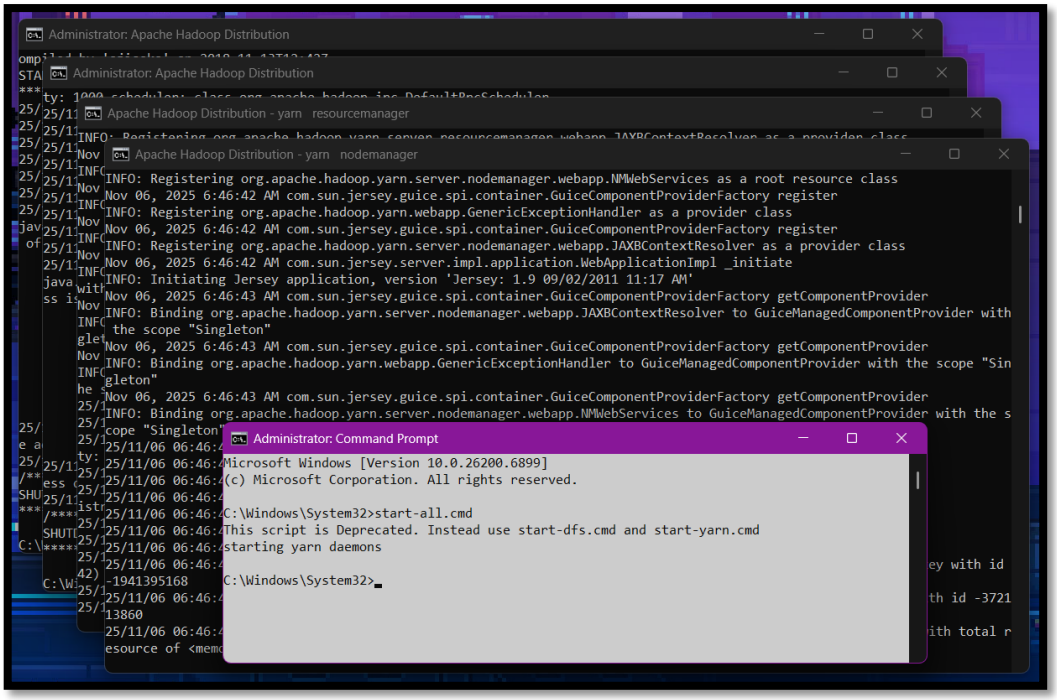
Step 6.1: Open command prompt and run the following command:

hdfs namenode -format

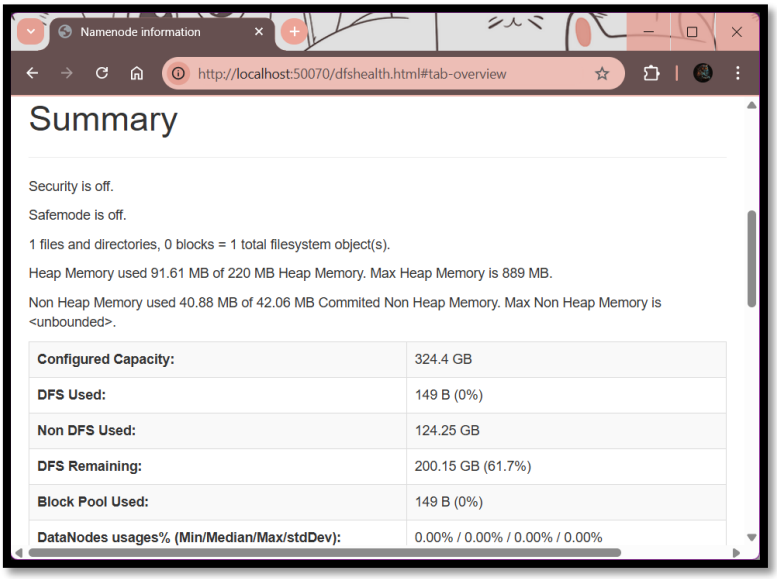


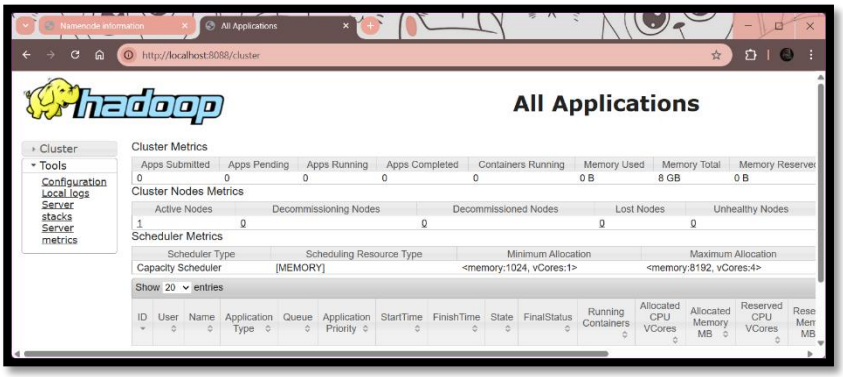
Step 6.2: Open command prompt as administrator and run the following command. This will open 4 new windows running different Daemons of Hadoop, which are namenode, datanode, resourcemanager and nodemanager.

start-all.cmd

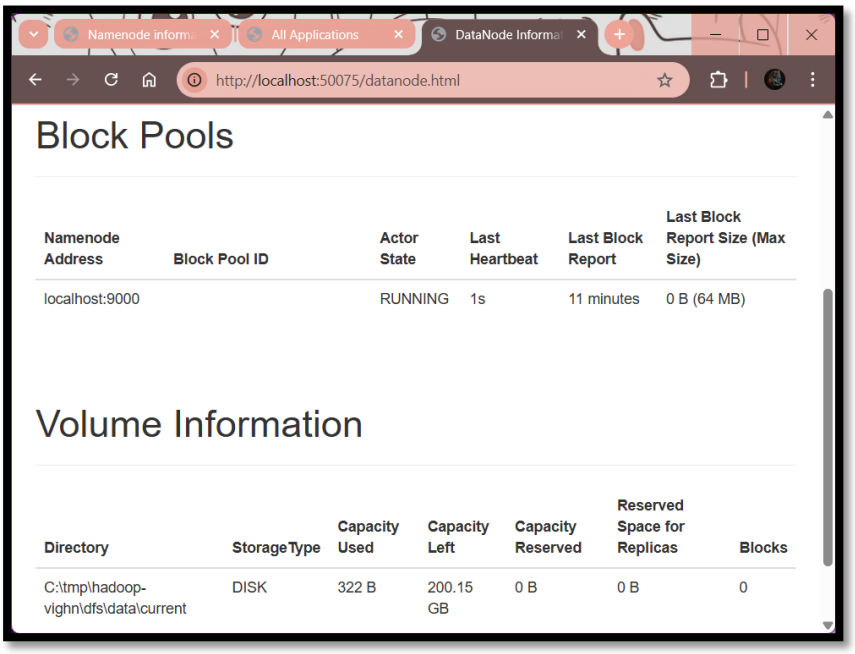


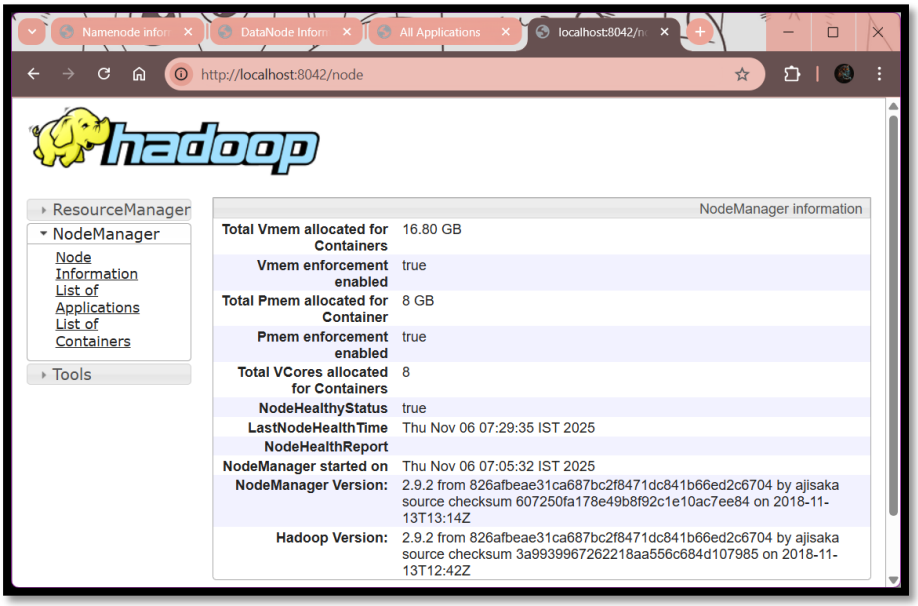
Step 7: Open localhost:50070 to verify namenode health. Open localhost:8088 to check the resourcemanager details.





Step 8: Open localhost:50075 to check the resourcemanager details. Open localhost:8042 to check nodemanager details.





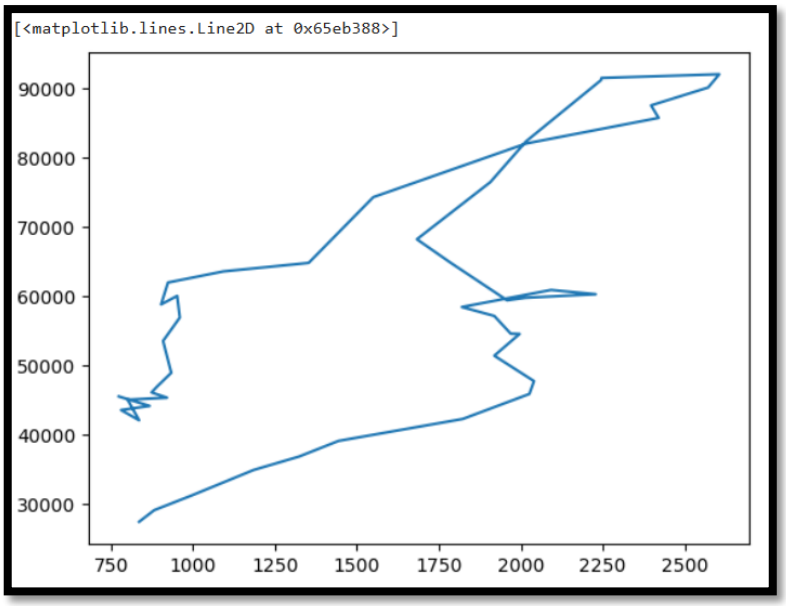
Practical – 3: Visualize data using plotting techniques in python.

Step 1: Open Jupyterlite and type the following code.

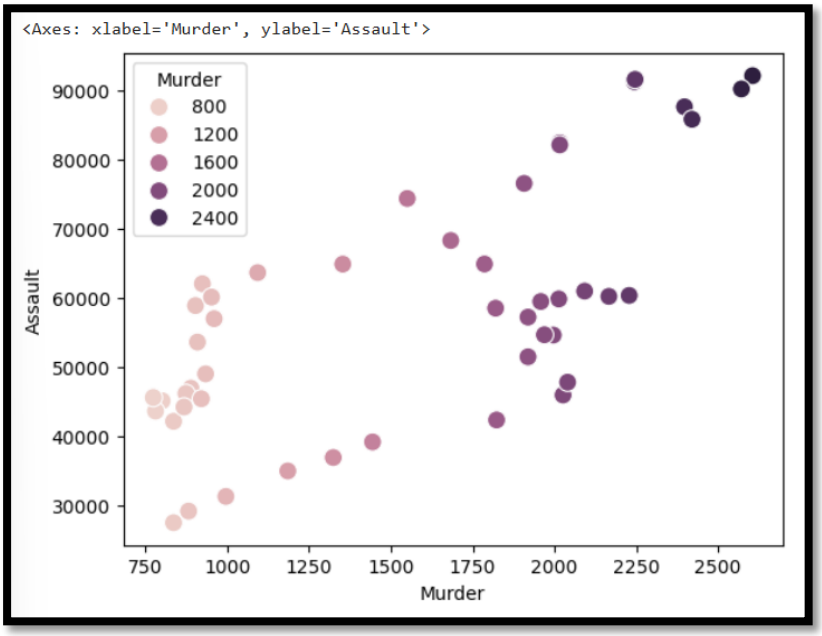
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
crime = pd.read_csv("crime.csv")
crime.head()
```

	Year	Population	Murder	Rape	Robbery	Assault	Burglary	CarTheft
0	1965	18073000	836	2320	28182	27464	183443	58452
1	1966	18258000	882	2439	30098	29142	196127	64368
2	1967	18336000	996	2665	40202	31261	219157	83775
3	1968	18113000	1185	2527	59857	34946	250918	104877
4	1969	18321000	1324	2902	64754	36890	248477	115400

```
plt.plot(crime.Murder, crime.Assault)
```



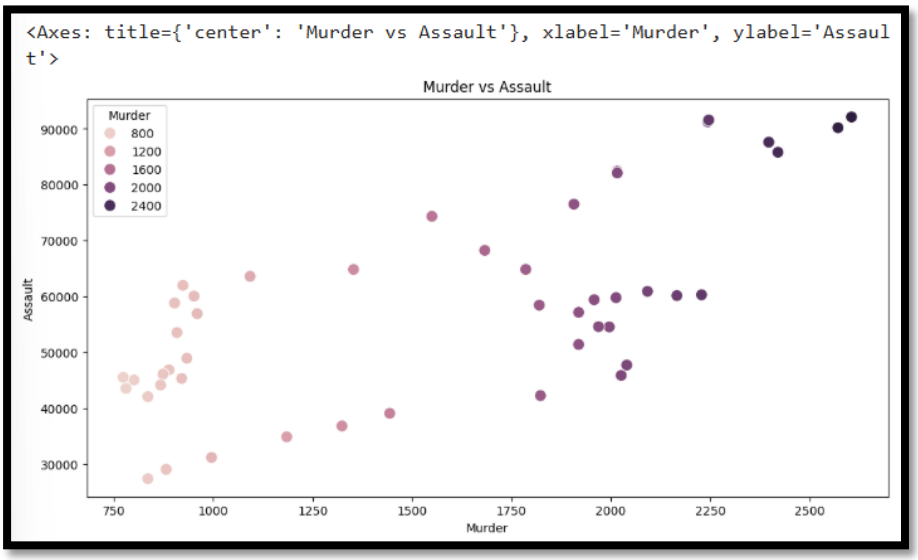
sns.scatterplot(x = crime.Murder, y = crime.Assault, hue = crime.Murder, s = 100)



plt.figure(figsize = (12, 6))

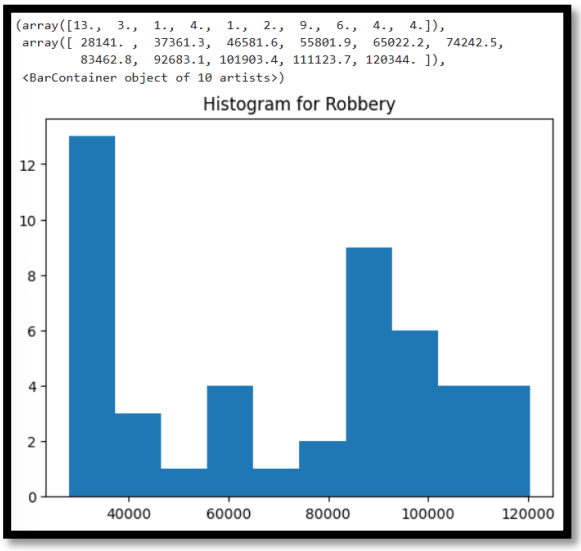
plt.title("Murder vs Assault")

sns.scatterplot(x = crime.Murder, y = crime.Assault, hue = crime.Murder, s = 100)

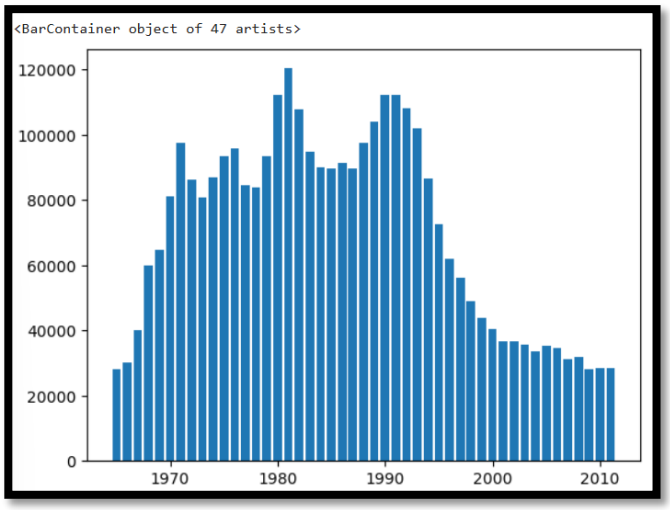


plt.title("Histogram for Robbery")

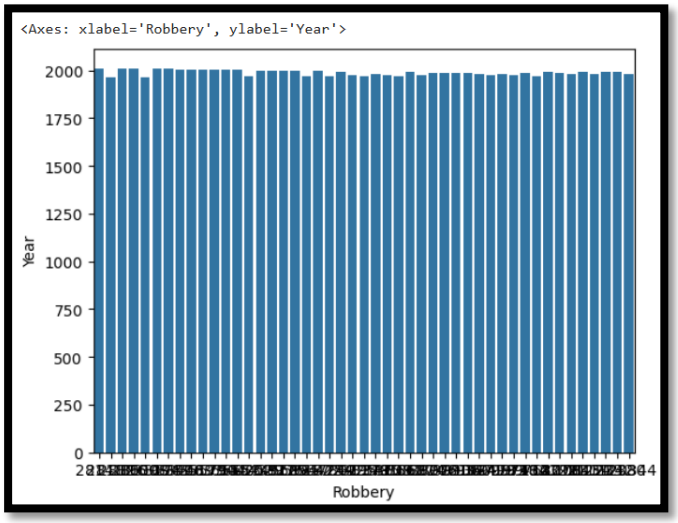
plt.hist(crime.Robbery)



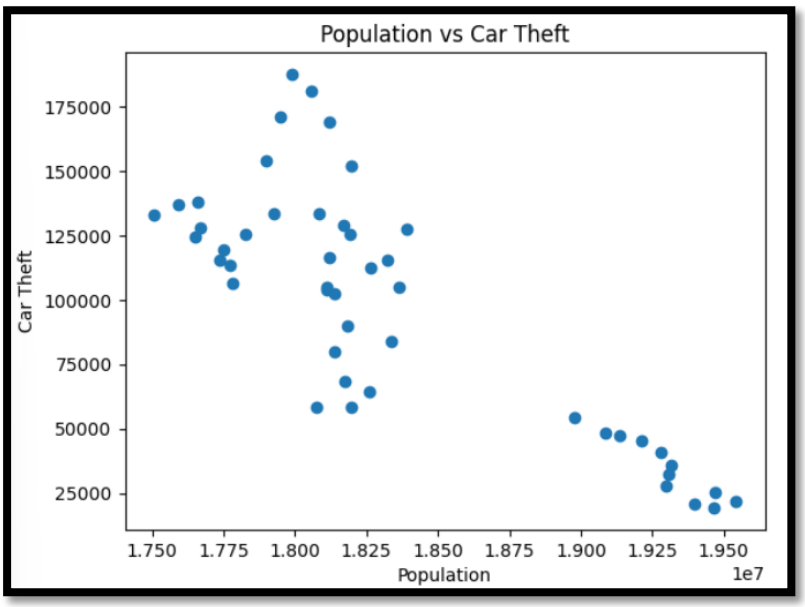
plt.bar(crime.Year, crime.Robbery)



sns.barplot(x = "Robbery", y = "Year", data = crime)



```
x = crime.Population
y = crime.CarTheft
plt.scatter(x, y)
plt.xlabel("Population")
plt.ylabel("Car Theft")
plt.title("Population vs Car Theft")
plt.show()
```



Practical – 4: Implement NoSQL Database Operations: CRUD operations, arrays using MongoDB).

Step 1: Download MongoDB Server to perform CRUD operations like insert, read, update and delete operations.

Step 2: Open command prompt and type the mongosh command to open the mongosh shell.

Step 3: Perform insert, read, update and delete operations in the mongosh shell.

- Insert operation: The command db.collection.insertOne() will perform an insert operation into a collection of a document.
-

db.std.insertOne({regNo:"3014",name:"VR",course:{courseName:"MCA",duration:"3 years"}, address:{city:"Bangalore",state:"KA",country:"India"}})

-

- Read operation: The command db.collection.find() will retrieve all the documents of the given collection.
-

db.std.find({"regNo":"3014"})

-

- Update operation: The command db.collection.update() method will take the field name and the new value as argument to update a document.
-

db.std.updateOne({"regNo":"3014"},{$set:{"name":"Vighnesh"}})

db.std.find({"regNo":"3014"})

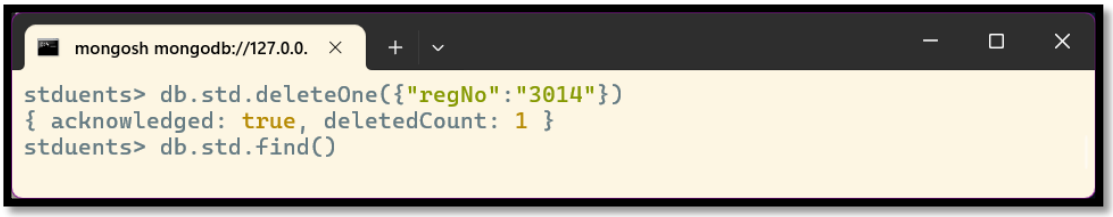
-

- Delete operation: The command db.collection.deleteOne() will delete an entry from the collection.
-

db.std.deleteOne({"regNo":"3014"})

db.std.find()

-



Step 4: Perform array operations in MongoDB.

- In a MongoDB database, data is stored in collections and a collection has documents. A document has fields and values, like in a JSON. The field types include scalar types (string, number, date, etc.) and composite types (arrays and objects).
- Type the following code:
-

use blogs

db.createCollection("posts")

new_post={name: "Working with arrays", user: "VR", desc: "Maintaining an array of objects in a document", content: "random text", created:ISODate(), updated:ISODate(), tags:["mongodb","arrays"]}

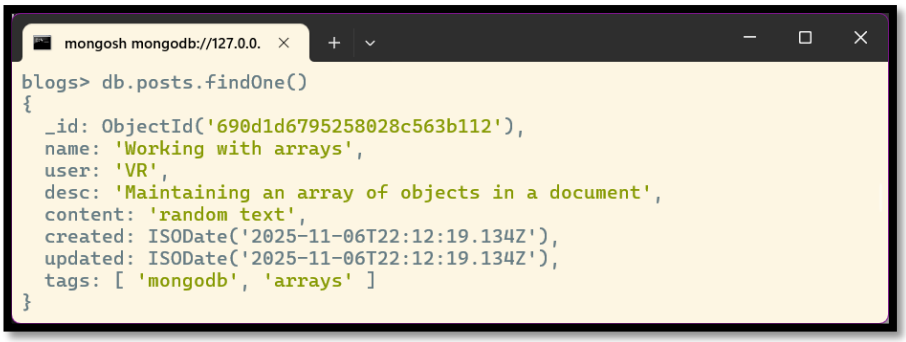
db.posts.insertOne(new_post)

➢

➢

db.posts.findOne()

➢



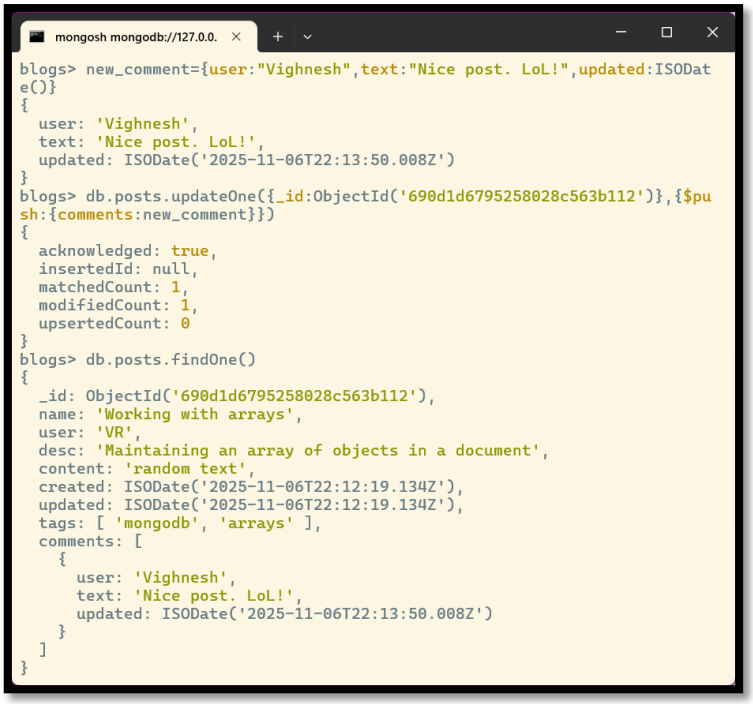
➢

new_comment={user: "Vighnesh",text: "Nice post. LoL!",updated:ISODate()}

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comments:new_comment}})

db.posts.findOne()

➢

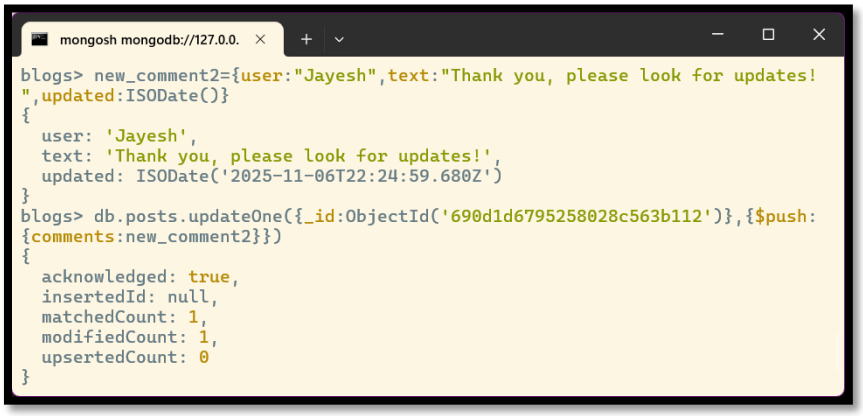


➢

new_comment2={user: "Jayesh", text: "Thank you, please look for updates!", updated: ISODate()}

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comments:new_comment2}})

➢



➢

new_content="Thank you! Please look for the updates - updated post."

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112'),"comments.user":"Jayesh"},{$set:{"comments.$.text":new_content}})

db.posts.findOne()

➢

➢

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$pull:{comment
s:{user:"Jayesh"}}})

db.posts.findOne()

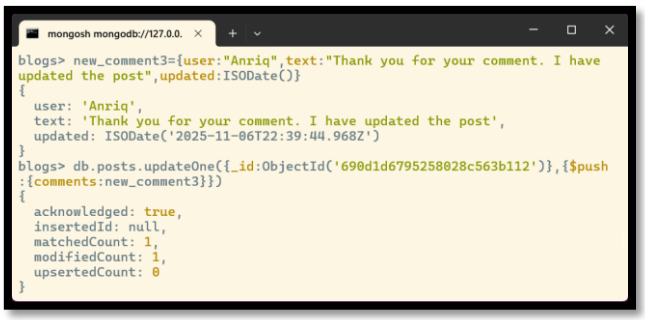
➢

➢

new_comment3={user:"Anriq",text:"Thank you for your comment. I have updated the
post",updated:ISODate()}

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comment
s:new_comment3}})

➢

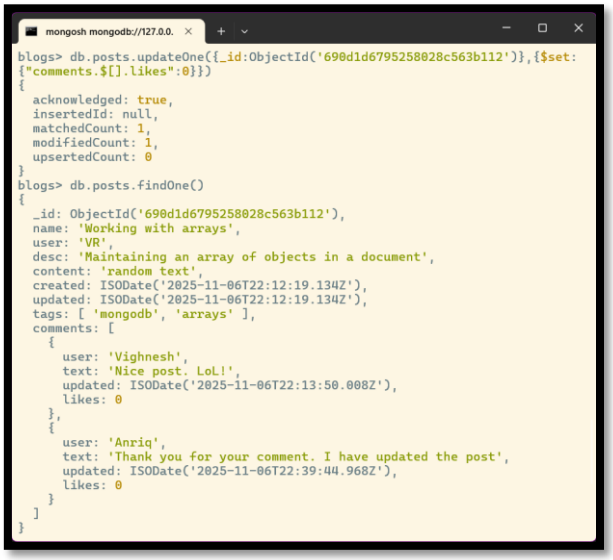


➢

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$set:{"comments.$[].likes":0}})

db.posts.findOne()

➢

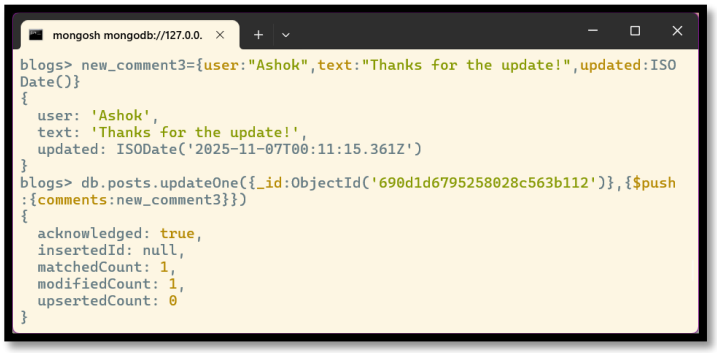


➢

new_comment3={user:"Ashok",text:"Thanks for the update!",updated:ISODate()}

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comments:new_comment3}})

➢



➢

db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$inc:{"comments.$[ele].likes":1}},{arrayFilters:[{"ele.user":"Ashok","ele.likes":{$exists:false}}]})

db.posts.findOne()

➢

Practical – 5: Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB.

Step 1: Open command prompt and type mongosh command to open the mongosh shell.

Step 2: Consider the following collection and documents in MongoDB to perform operations on it.

```
use students
db.createCollection("marks")
db.marks.insertMany([
    {name:"VR",age:22,marks:85,course:"MSc.AI"},
    {name:"Anant",age:21,marks:80,course:"MSc.AI"},
    {name:"Aditya",age:23,marks:90,course:"MSc.IT"},
    {name:"Anurag",age:21,marks:95,course:"MSc.CS"}
])
```

Step 3: Perform the following operations:

- Count: It is used to count the number of documents that match a query. The count() is deprecated but instead we use the countDocuments() function.

```
db.marks.countDocuments({marks:{$gte:85}})
db.marks.countDocuments()
```

```
mongosh mongodb://127.0.0.
students> db.marks.countDocuments({marks:{$gte:85}})
3
students> db.marks.countDocuments()
4
```

- Sort: It is used to sort the documents in either ascending or descending order. The sort() function is used along with 1 and -1 as the indicator for ascending and descending respectively along with the field tags.

db.marks.find().sort({course:1,marks:-1})

```
mongosh mongodb://127.0.0.
students> db.marks.find().sort({course:1,marks:-1})
[
  {
    _id: ObjectId('690d44b6b73d8d005e63b112'),
    name: 'VR',
    age: 22,
    marks: 85,
    course: 'MSc.AI'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b113'),
    name: 'Anant',
    age: 21,
    marks: 80,
    course: 'MSc.AI'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b115'),
    name: 'Anurag',
    age: 21,
    marks: 95,
    course: 'MSc.CS'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b114'),
    name: 'Aditya',
    age: 23,
    marks: 90,
    course: 'MSc.IT'
  }
]
```

- Limit: It restricts the number of documents returned. The limit() function is used along with a constant value which specifies the number of documents to be returned.

db.marks.find().sort({marks:-1}).limit(3)

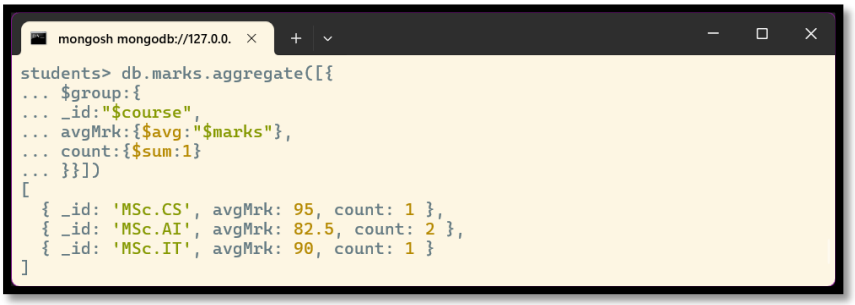
```
mongosh mongodb://127.0.0.
students> db.marks.find().sort({marks:-1}).limit(3)
[
  {
    _id: ObjectId('690d44b6b73d8d005e63b115'),
    name: 'Anurag',
    age: 21,
    marks: 95,
    course: 'MSc.CS'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b114'),
    name: 'Aditya',
    age: 23,
    marks: 90,
    course: 'MSc.IT'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b112'),
    name: 'VR',
    age: 22,
    marks: 85,
    course: 'MSc.AI'
  }
]
```

- Skip: It skips a specified number of documents. The skip() function is used along with a constant value which specifies the number of documents to be skipped.

```
db.marks.find().sort({marks:-1}).skip(2).limit(4)
```

- Aggregate

```
db.marks.aggregate([{
   $group:{
   _id:"$course",
   avgMrk:{$avg:"$marks"},
   count:{$sum:1}
}}])
```



Practical – 6: Implement word count / frequency programs using MapReduce.

Step 1: Create a text file named “word_count_data” and add the following content. Create two python files named “mapper.py” and “reducer.py”.

word_count_data.txt file:

foo foo quux labs foo bar quux

mapper.py file:

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}:\t1")
```

reducer.py file:

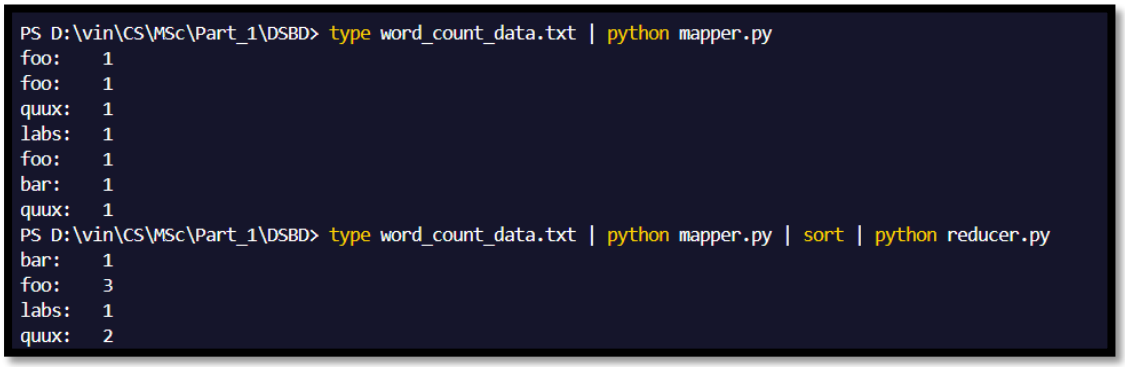
```
#!/usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
```

```
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_count = count
        current_word = word
if current_word == word:
    print(f"{current_word}\t{current_count}")
```

Step 2: Type the following commands in the terminal to test the created python files.

type word_count_data.txt | python mapper.py

type word_count_data.txt | python mapper.py | sort | python reducer.py



Step 3: Open command prompt and type “hdfs namenode -format”. Then open command prompt as administrator and type “start-all.cmd” to start all the Hadoop daemons.

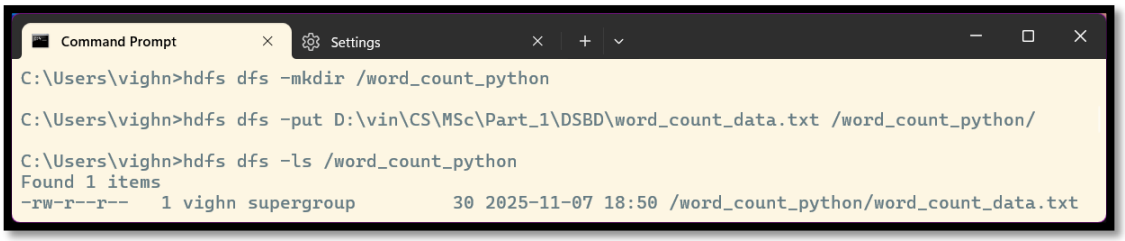
Step 4: Download hadoop-streaming.jar file and then place it in the same place as the mapper and reducer python files.

Step 5: Open command prompt and type the following commands.

hdfs dfs -mkdir /word_count_python

hdfs dfs -put D:\vin\CS\MSc\Part_1\DSBD\word_count_data.txt /word_count_python/

hdfs dfs -ls /word_count_python



hadoop jar D:\vin\CS\MSc\Part_1\DSBD\hadoop-streaming-2.7.3.jar ^

-input /word_count_python/word_count_data.txt ^

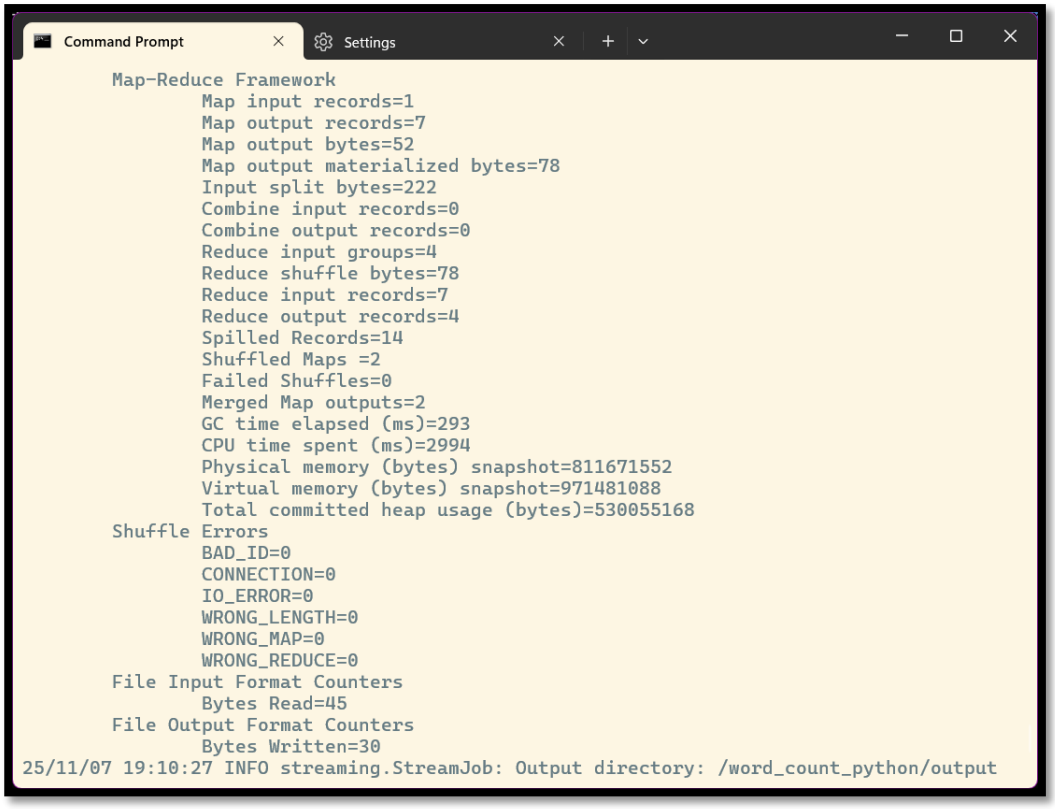
-output /word_count_python/output ^

-mapper "python mapper.py" ^

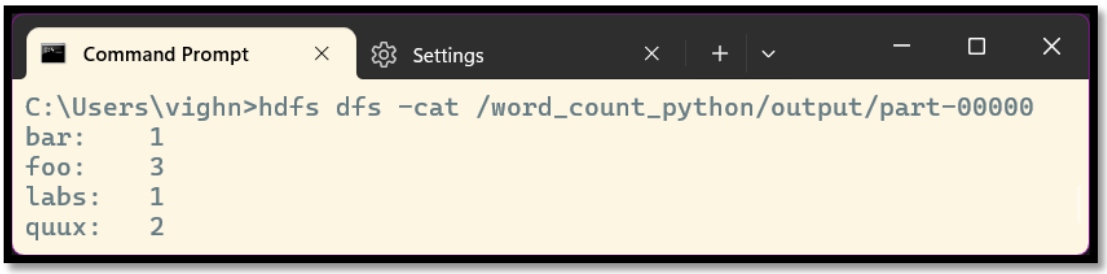
-reducer "python reducer.py" ^

-file D:\vin\CS\MSc\Part_1\DSBD\mapper.py ^

-file D:\vin\CS\MSc\Part_1\DSBD\reducer.py



hdfs dfs -cat /word_count_python/output/part-00000



Practical – 7: Implement a MapReduce program that processes a dataset.

Step 1: Download the wine quality dataset from Kaggle and store it in a folder where your mapper and reducer python programs are going to be stored.

Step 2: Create two python files named “mapper_data.py” and “reducer_data.py”.

mapper_data.py file:

```
#!/usr/bin/env python3
import sys
import pandas as pd
data = pd.read_csv(sys.stdin)
chunks = [data.iloc[0:399, :], data.iloc[400:800, :], data.iloc[801:1200, :], data.iloc[1201:, :]]
def mapper(df):
    mapped = []
    for _, row in df.iterrows():
        try:
            q = row["quality"]
            a = float(row["volatile acidity"])
            mapped.append((q, a))
        except Exception:
            continue
    return mapped
for c in chunks:
    mapped_results = mapper(c)
    for quality, acidity in mapped_results:
        print(f"{quality}\t{acidity}")
```

reducer_data.py file:

```
#!/usr/bin/env python3
import sys
current_quality = None
total_acidity = 0.0
count = 0
for line in sys.stdin:
    parts = line.strip().split("\t")
    if len(parts) != 2:
        continue
    quality, acidity = parts
    try:
        acidity = float(acidity)
    except ValueError:
        continue
    if current_quality == quality:
        total_acidity += acidity
        count += 1
    else:
        if current_quality is not None:
            avg = total_acidity / count if count else 0
            print(f"{current_quality}\t{avg:.4f}")
        current_quality = quality
        total_acidity = acidity
        count = 1
if current_quality is not None:
    avg = total_acidity / count if count else 0
```

```
print(f"{current_quality}\t{avg:.4f}")
```

Step 3: Type the following commands in the terminal to test the created python files.

type wine_qt.csv | python mapper_data.py

type wine_qt.csv | python mapper_data.py | sort | python reducer_data.py

```
PS D:\vin\CS\MSc\Part_1\DSBD> type wine_qt.csv | python mapper_data.py
5.0     0.7
5.0     0.88
5.0     0.76
6.0     0.28
5.0     0.7
5.0     0.66
5.0     0.6
7.0     0.65
7.0     0.58
5.0     0.58
5.0     0.615
5.0     0.61
7.0     0.28
6.0     0.32
```

```
PS D:\vin\CS\MSc\Part_1\DSBD> type wine_qt.csv | python mapper_data.py | sort | python reducer_data.py
3.0     0.8975
4.0     0.7000
5.0     0.5852
6.0     0.5050
7.0     0.3931
8.0     0.4100
```

Step 4: Open command prompt and type “hdfs namenode -format”. Then open command prompt as administrator and type “start-all.cmd” to start all the Hadoop daemons.

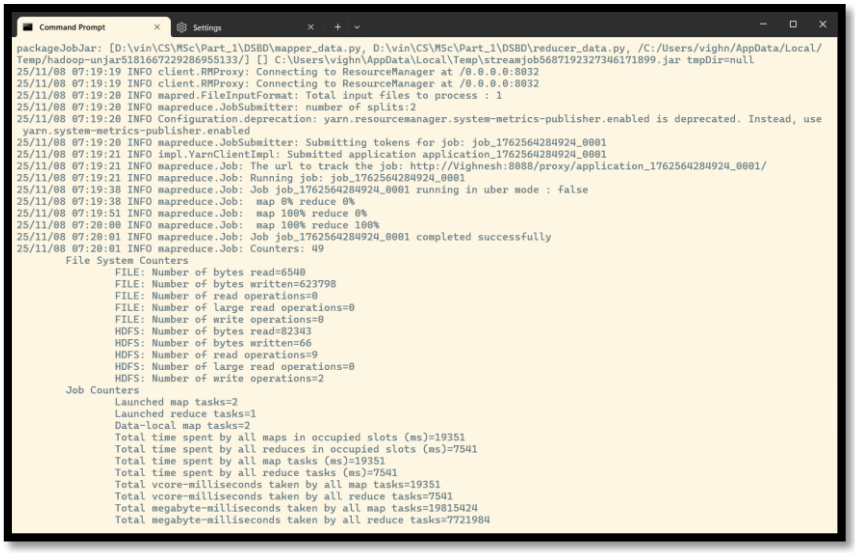
Step 5: Open command prompt and type the following commands.

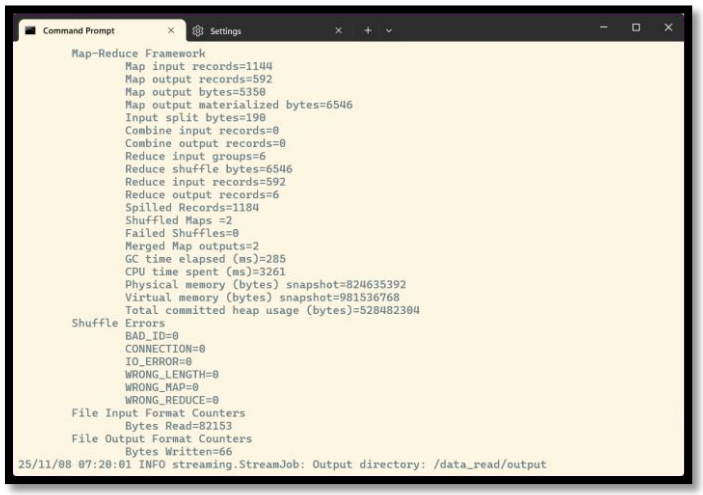
hdfs dfs -mkdir /data_read

hdfs dfs -put D:\vin\CS\MSc\Part_1\DSBD\wine_qt.csv /data_read/

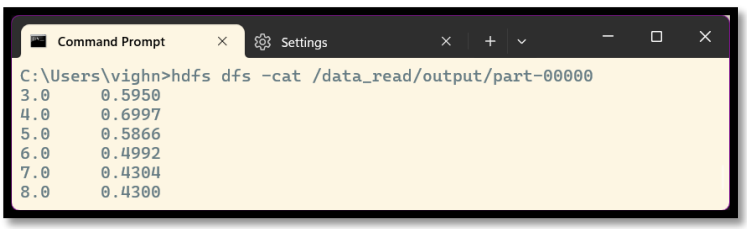
hdfs dfs -ls /data_read

```
hadoop jar D:\vin\CS\MSc\Part_1\DSBD\hadoop-streaming-2.7.3.jar ^
-input /data_read/wine_qt.csv ^
-output /data_read/output ^
-mapper "python mapper_data.py" ^
-reducer "python reducer_data.py" ^
-file D:\vin\CS\MSc\Part_1\DSBD\mapper_data.py ^
-file D:\vin\CS\MSc\Part_1\DSBD\reducer_data.py
```






```
hdfs dfs -cat /data_read/output/part-00000
```



Practical – 8: Implement clustering techniques using SPARK.

- Apache Spark is better for clustering because it processes large datasets in parallel across multiple nodes, drastically speeding up the computation process. Unlike Hadoop, Spark keeps most data in memory (RAM) which makes iterative algorithms like K-Means much faster than disk-based systems. Spark's MLlib provides an optimal, scalable K-Means implementation that is quite easier to use. If a node fails during computation, Spark automatically recomputes lost data using its RDD (Resilient Distributed Dataset) Lineage, ensuring reliability.

Step 1: Open VS code and in its terminal type the following command to install the module named pyspark: pip install pyspark

```
PS D:\vin\CS\MSc\Part_1\DSBD> pip install pyspark
Requirement already satisfied: pyspark in c:\users\vighn\appdata\local\programs\python\python312\lib
\site-packages (4.0.1)
Requirement already satisfied: py4j==0.10.9.9 in c:\users\vighn\appdata\local\programs\python\python
312\lib\site-packages (from pyspark) (0.10.9.9)
```

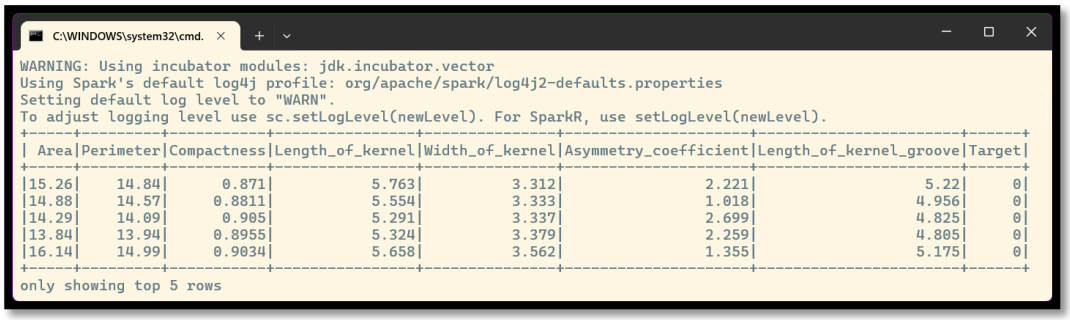
Step 2: Install Java JDK 17 and set the JAVA_HOME environment variable value as the location of the jdk-17 folder.

Step 3: Download the seeds dataset from Kaggle which contains information about kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment.

Step 4: Create a python file and type the following code. Open command prompt and type the command "python spark_cluster.py" to execute the file.

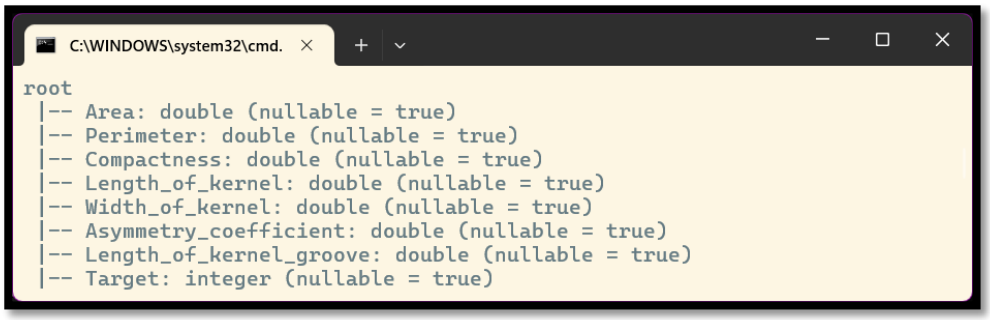
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("cluster").getOrCreate()

spark.sparkContext.setLogLevel("OFF")

data = spark.read.csv("seed_data.csv", header = True, inferSchema = True)

col_names = ["Area", "Perimeter", "Compactness", "Length_of_kernel",
"Width_of_kernel", "Asymmetry_coefficient", "Length_of_kernel_groove", "Target"]
```

```
data = data.toDF(*col_names)
data.show(5)
print()
```



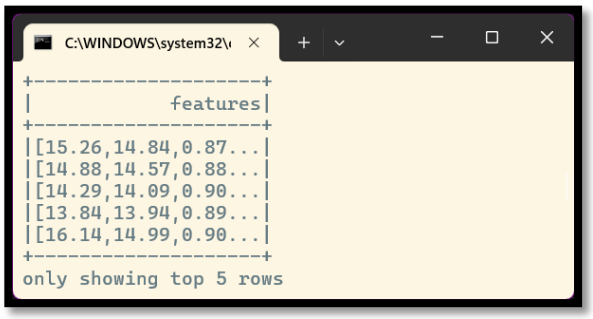
```
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
+-----+---------+-----------+----------------+---------------+--------------------+----------------------+------+
| Area|Perimeter|Compactness|Length_of_kernel|Width_of_kernel|Asymmetry_coefficient|Length_of_kernel_groove|Target|
+-----+---------+-----------+----------------+---------------+--------------------+----------------------+------+
|15.26|    14.84|      0.871|           5.763|          3.312|               2.221|                  5.22|     0|
|14.88|    14.57|     0.8811|           5.554|          3.333|               1.018|                 4.956|     0|
|14.29|    14.09|      0.905|           5.291|          3.337|               2.699|                 4.825|     0|
|13.84|    13.94|     0.8955|           5.324|          3.379|               2.259|                 4.805|     0|
|16.14|    14.99|     0.9034|           5.658|          3.562|               1.355|                 5.175|     0|
+-----+---------+-----------+----------------+---------------+--------------------+----------------------+------+
only showing top 5 rows
```

```
data.printSchema()
print()
```



```
root
 |-- Area: double (nullable = true)
 |-- Perimeter: double (nullable = true)
 |-- Compactness: double (nullable = true)
 |-- Length_of_kernel: double (nullable = true)
 |-- Width_of_kernel: double (nullable = true)
 |-- Asymmetry_coefficient: double (nullable = true)
 |-- Length_of_kernel_groove: double (nullable = true)
 |-- Target: integer (nullable = true)
```

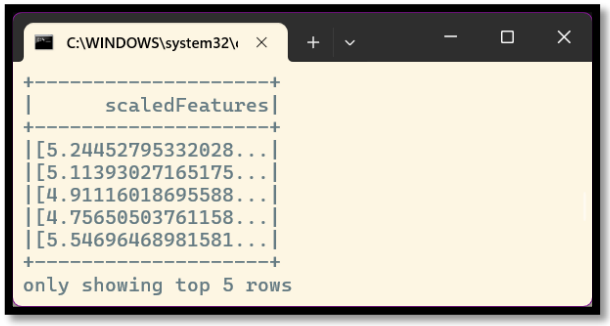
```
from pyspark.ml.feature import VectorAssembler
feature_cols = [c for c in data.columns if c != "Target"]
vec_assembler = VectorAssembler(inputCols = feature_cols, outputCol = "features")
final_data = vec_assembler.transform(data)
final_data.select("features").show(5)
print()
```



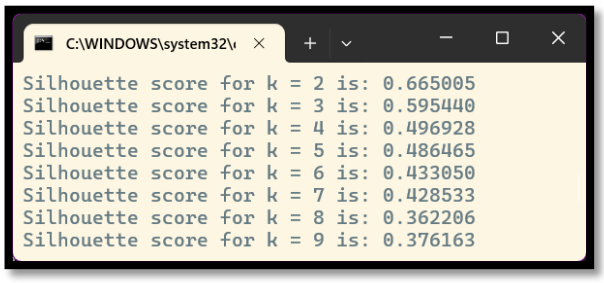
```
+--------------------+
|            features|
+--------------------+
|[15.26,14.84,0.87...|
|[14.88,14.57,0.88...|
|[14.29,14.09,0.90...|
|[13.84,13.94,0.89...|
|[16.14,14.99,0.90...|
+--------------------+
only showing top 5 rows
```

```
from pyspark.ml.feature import StandardScaler
```

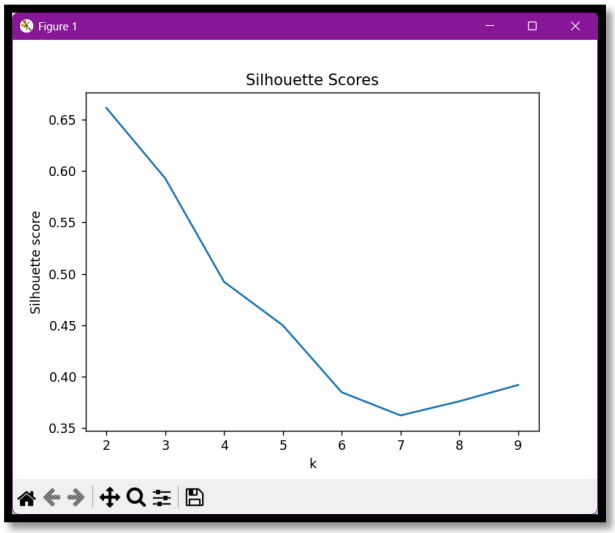
```
scaler = StandardScaler(inputCol = "features", outputCol = "scaledFeatures", withStd = True, withMean = False)

scalerModel = scaler.fit(final_data)

final_data = scalerModel.transform(final_data)

final_data.select("scaledFeatures").show(5)

print()
```



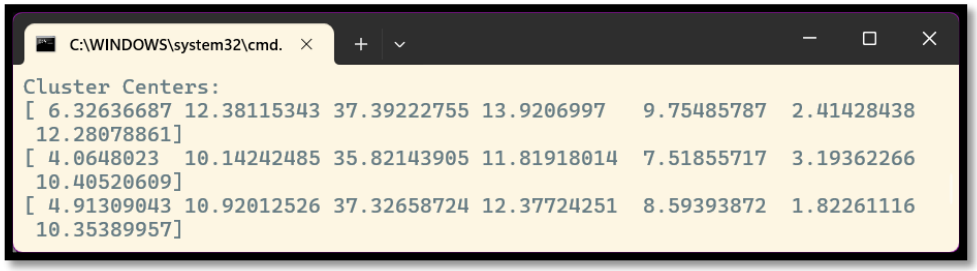

```
from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

silhouette_score = []

evaluator = ClusteringEvaluator(predictionCol = "prediction", featuresCol = "scaledFeatures", metricName = "silhouette", distanceMeasure = "squaredEuclidean")

for i in range(2, 10):

    kmeans = KMeans(featuresCol = "scaledFeatures", k = i)

    model = kmeans.fit(final_data)

    pred = model.transform(final_data)

    score = evaluator.evaluate(pred)

    silhouette_score.append(score)

    print(f"Silhouette score for k = {i} is: {score:.6f}")

print()
```



```
import matplotlib.pyplot as plt
plt.plot(range(2, 10), silhouette_score)
plt.xlabel("k")
plt.ylabel("Silhouette score")
plt.title("Silhouette Scores")
plt.show()
```

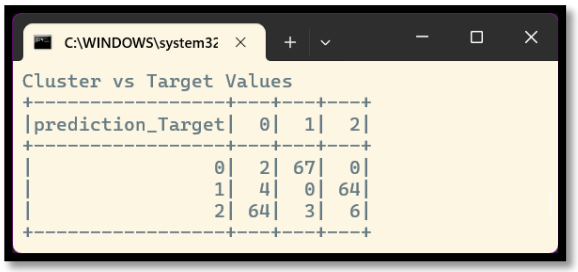



```
kmeans = KMeans(featuresCol = "scaledFeatures", k = 3)
model = kmeans.fit(final_data)
pred = model.transform(final_data)
centers = model.clusterCenters()
print("Cluster Centers:")
for center in centers:
    print(center)
print()
```




```
from pyspark.sql import functions as F
```

```
from pyspark.sql.functions import col
print("Cluster vs Target Values")
pred.crosstab("prediction", "Target").show()
print()
```




```
mapping = (pred.groupby("prediction", "Target")
        .count()
        .groupby("prediction")
        .agg(F.max(F.struct("count", "Target")).alias("max"))
        .select(col("prediction"), col("max.Target").alias("dominant_label")))
combined_data = pred.join(mapping, on = "prediction")
correct = combined_data.filter(col("Target") == col("dominant_label")).count()
total = combined_data.count()
acc = correct / total
print(f"Clustering accuracy: {acc * 100:.2f}")
print()
spark.stop()
```