

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
28141	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	47.000000
mean	1988.000000	1.834426e+07	1549.978723	4200.425532	70429.297872	58022.234000
std	13.711309	6.024504e+05	590.454265	1096.569507	30204.823764	17455.534000
min	1965.000000	1.750669e+07	774.000000	2320.000000	28141.000000	27464.000000
25%	1976.500000	1.793700e+07	922.500000	3197.000000	36604.000000	45477.500000
50%	1988.000000	1.816900e+07	1683.000000	4199.000000	81149.000000	57193.000000
75%	1999.500000	1.868373e+07	2016.000000	5241.000000	94141.000000	64864.500000
max	2011.000000	1.954145e+07	2605.000000	5706.000000	120344.000000	92105.000000

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)
```



```
Dimension: 0
0-D array: 42
Dimension: 1
1-D array: [1 2 3 4 5]
Dimension: 2
2-D array:
[[1 2 3]
 [4 5 6]]
Dimension: 3
3-D array:
[[[1 2 3]
  [4 5 6]]
 [[1 2 3]
  [4 5 6]]]
```

```
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.

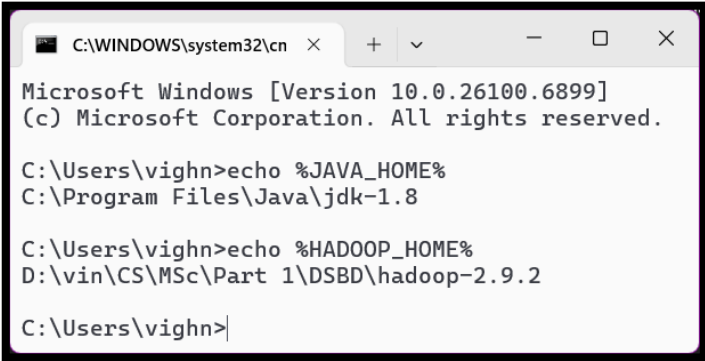
Step 3: Add another environment variable named HADOOP_HOME under system variables and add its value as the installation folder of the Hadoop folder (D:\CS\MSc\hadoop-2.9.2). Add the path to the bin and sbin folder of Hadoop installation as: %HADOOP_HOME%\sbin and %HADOOP_HOME%\bin.

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%
```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.26100.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vighn>echo %JAVA_HOME%
C:\Program Files\Java\jdk-1.8

C:\Users\vighn>echo %HADOOP_HOME%
D:\vin\CS\MSc\Part 1\DSBD\hadoop-2.9.2

C:\Users\vighn>
```


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>
```

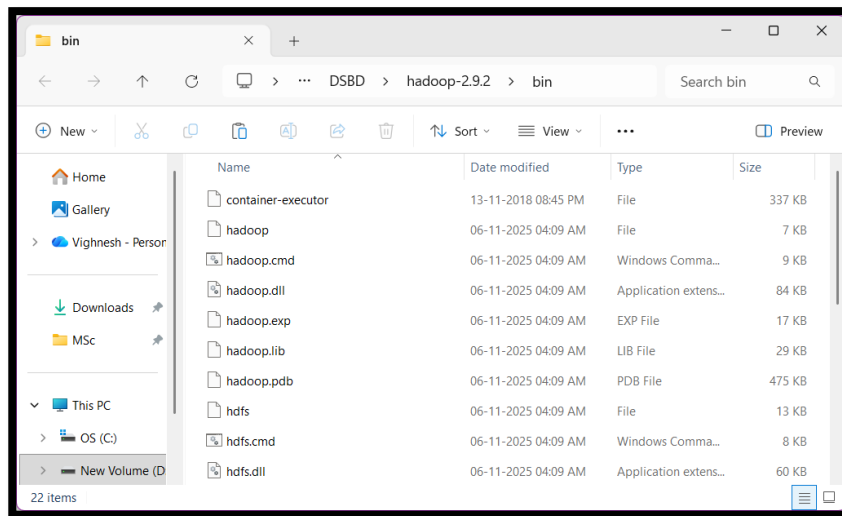
Step 5.2.3: Editing mapred-site.xml file:

```
<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`

```

Command Prompt
25/11/06 06:27:26 INFO namenode.NameNode: Caching file names occurring more than 10 times
25/11/06 06:27:26 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: falseskipCapture
AccessTimeOnlyChange: false
25/11/06 06:27:26 INFO util.GSet: Computing capacity for map cachedBlocks
25/11/06 06:27:26 INFO util.GSet: VM type = 64-bit
25/11/06 06:27:26 INFO util.GSet: 0.25% max memory 889 MB = 2.2 MB
25/11/06 06:27:26 INFO util.GSet: capacity = 2^18 = 262144 entries
25/11/06 06:27:26 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
25/11/06 06:27:26 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
25/11/06 06:27:26 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
25/11/06 06:27:26 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
25/11/06 06:27:26 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry c
ache entry expiry time is 600000 millis
25/11/06 06:27:26 INFO util.GSet: Computing capacity for map NameNodeRetryCache
25/11/06 06:27:26 INFO util.GSet: VM type = 64-bit
25/11/06 06:27:26 INFO util.GSet: 0.029999999329447746% max memory 889 MB = 273.1 KB
25/11/06 06:27:26 INFO util.GSet: capacity = 2^15 = 32768 entries
25/11/06 06:27:27 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1057410942-10.169.43.122-1
762390647025
25/11/06 06:27:27 INFO common.Storage: Storage directory D:\vin\CS\MSc\Part_1\DSBD\hadoop-2.9.2\data\namenode has been successfully formatted.
25/11/06 06:27:27 INFO namenode.FSImageFormatProtobuf: Saving image file D:\vin\CS\MSc\Part_1\DSB
D\hadoop-2.9.2\data\namenode\current\fsimage.ckpt_000000000000000000 using no compression
25/11/06 06:27:27 INFO namenode.FSImageFormatProtobuf: Image file D:\vin\CS\MSc\Part_1\DSBD\hadoo
p-2.9.2\data\namenode\current\fsimage.ckpt_000000000000000000 of size 324 bytes saved in 0 secon
ds
25/11/06 06:27:27 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >=
0
25/11/06 06:27:27 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****

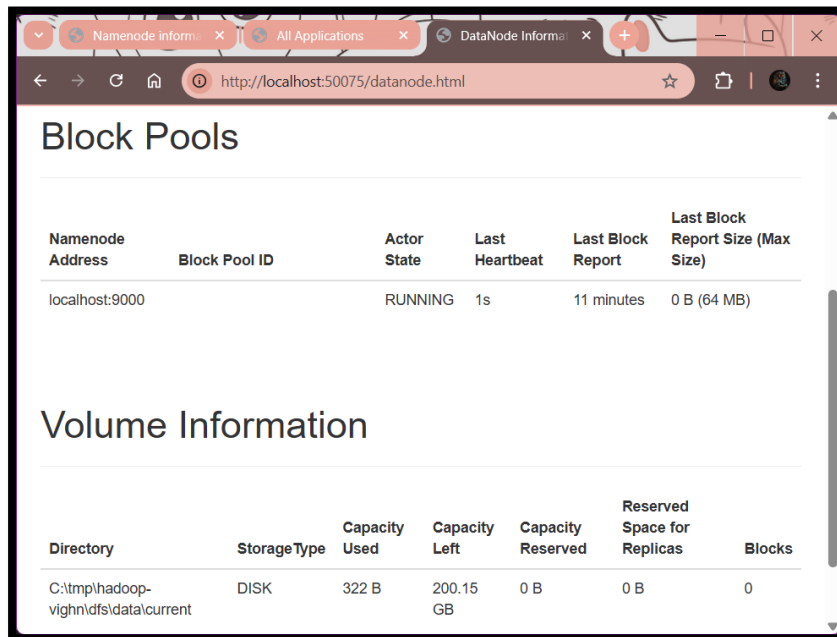
```

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



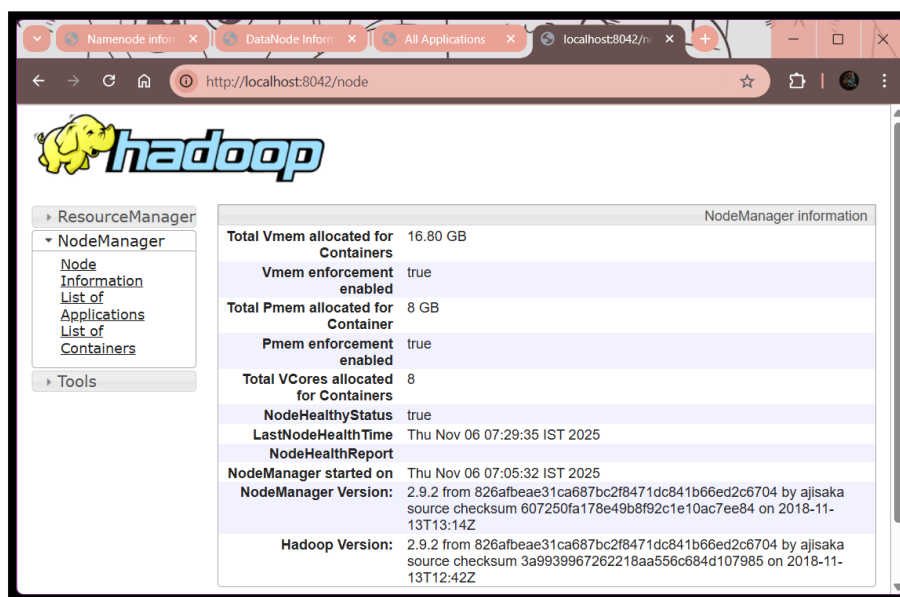
The screenshot shows the Hadoop DataNode web interface at <http://localhost:50075/datanode.html>. The page is divided into two main sections: 'Block Pools' and 'Volume Information'.

Block Pools

Namenode Address	Block Pool ID	Actor State	Last Heartbeat	Last Block Report	Last Block Report Size (Max Size)
localhost:9000		RUNNING	1s	11 minutes	0 B (64 MB)

Volume Information

Directory	Storage Type	Capacity Used	Capacity Left	Capacity Reserved	Reserved Space for Replicas	Blocks
C:\tmp\hadoop-vighn\dfs\data\current	DISK	322 B	200.15 GB	0 B	0 B	0



The screenshot shows the Hadoop NodeManager web interface at <http://localhost:8042/node>. The page features the Hadoop logo and a sidebar with navigation links. The main content area displays 'NodeManager information'.

NodeManager information

Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	true
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total VCores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Thu Nov 06 07:29:35 IST 2025
NodeHealthReport	
NodeManager started on	Thu Nov 06 07:05:32 IST 2025
NodeManager Version:	2.9.2 from 826afbeae31ca687bc2f8471dc841b66ed2c6704 by ajisaka source checksum 607250fa178e49b8f92c1e10ac7ee84 on 2018-11-13T13:14Z
Hadoop Version:	2.9.2 from 826afbeae31ca687bc2f8471dc841b66ed2c6704 by ajisaka source checksum 3a9939967262218aa556c684d107985 on 2018-11-13T12:42Z

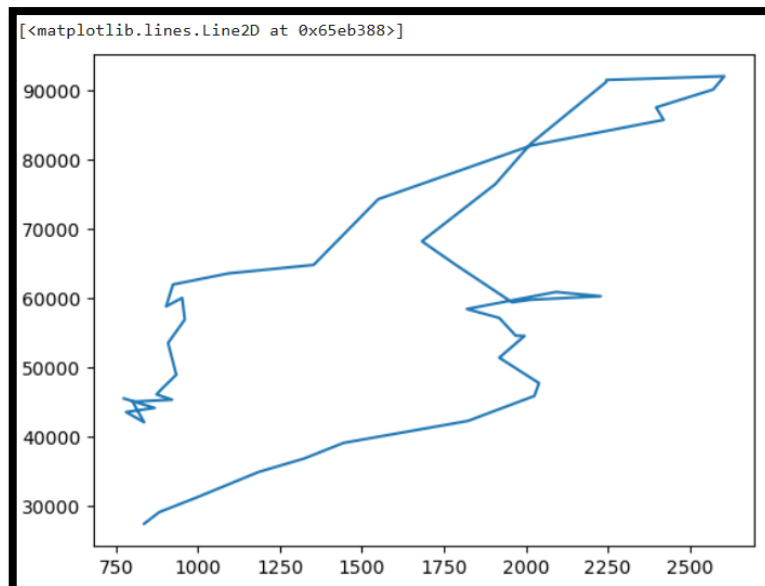
Practical – 3: Visualize data using plotting techniques in python.

Step 1: Open [Jupyterlite](#) and type the following code.

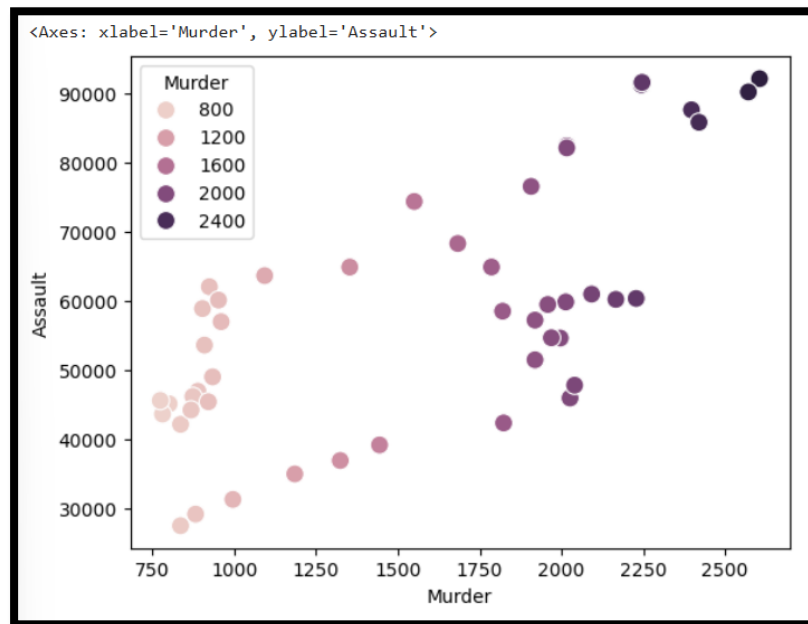
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
crime = pd.read_csv("crime.csv")
crime.head()
```

	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)
```



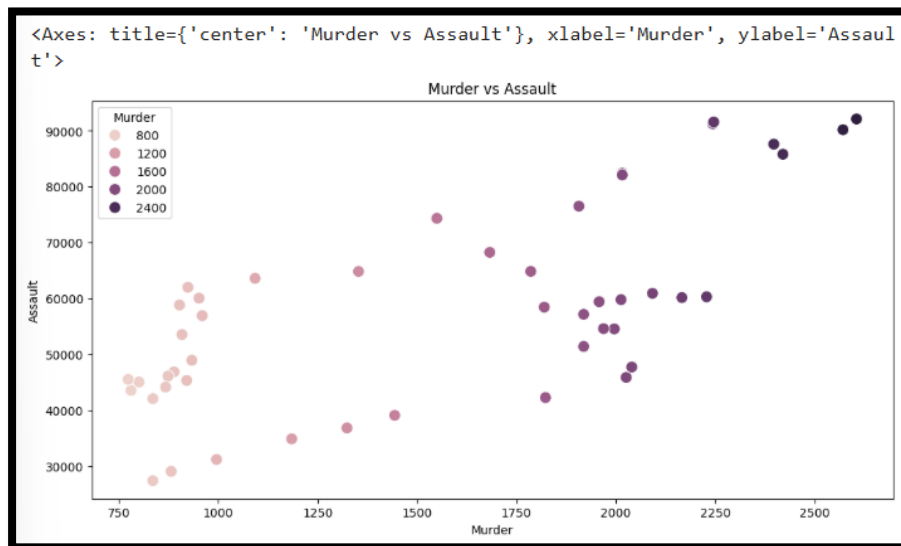
```
sns.scatterplot(x = crime.Murder, y = crime.Assault, hue = crime.Murder, s = 100)
```



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

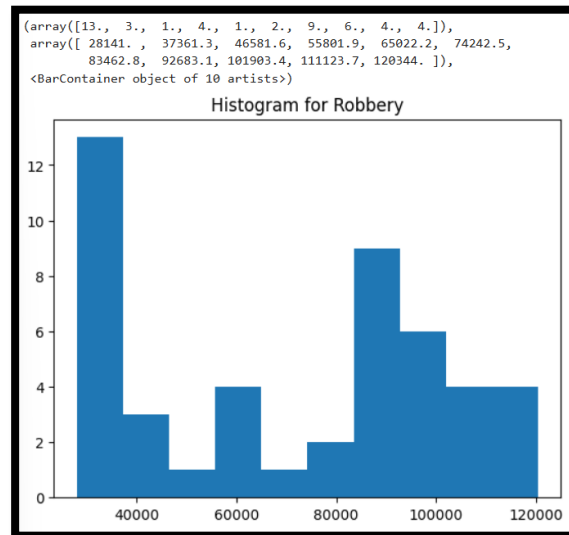
```
plt.title("Murder vs Assault")
```

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

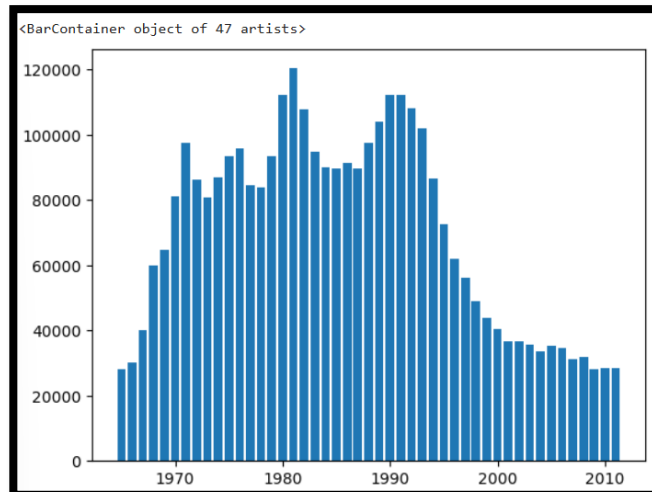


```
plt.title("Histogram for Robbery")
```

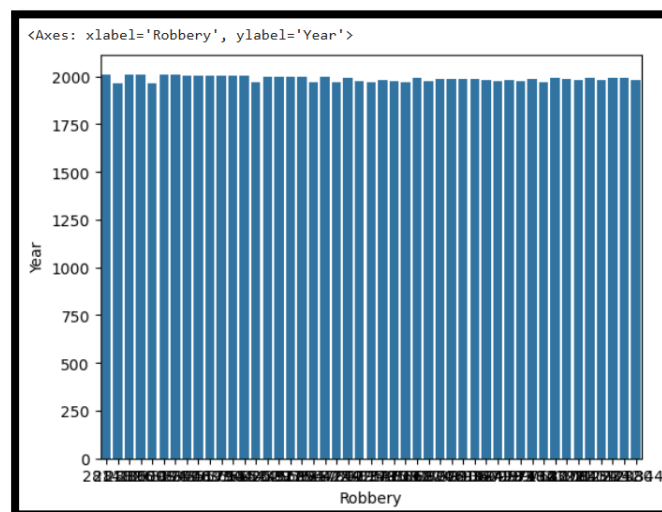
```
plt.hist(crime.Robbery)
```



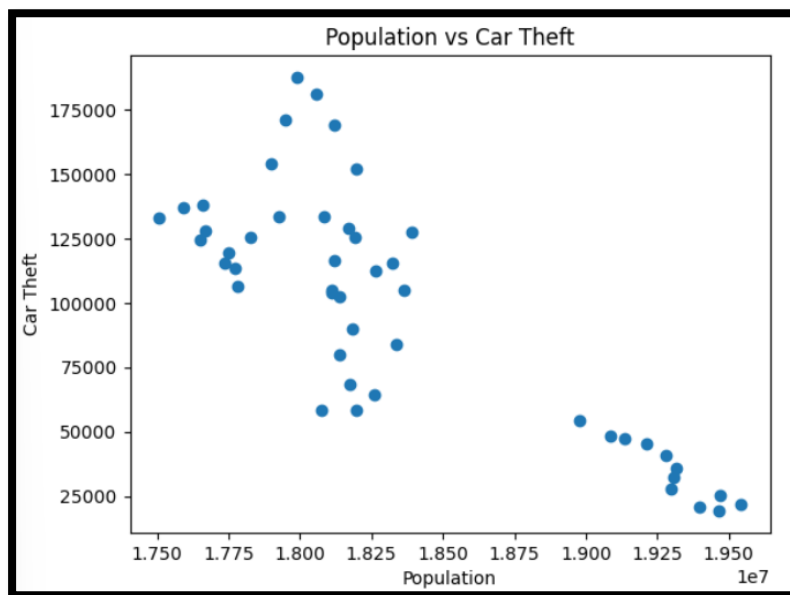
```
plt.bar(crime.Year, crime.Robbery)
```



```
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.

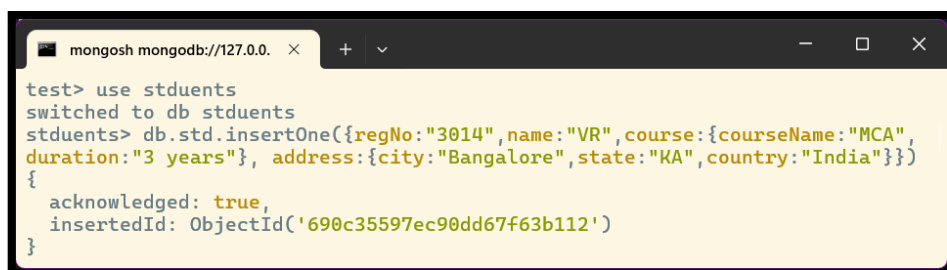
Step 2: Open command prompt and type the mongosh command to open the mongosh shell.

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"}})
```



```
mongosh mongodb://127.0.0.1
test> use stduents
switched to db stduents
stduents> db.std.insertOne({regNo:"3014",name:"VR",course:{courseName:"MCA",duration:"3 years"}, address:{city:"Bangalore",state:"KA",country:"India"}})
{
  acknowledged: true,
  insertedId: ObjectId('690c35597ec90dd67f63b112')
}
```

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



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



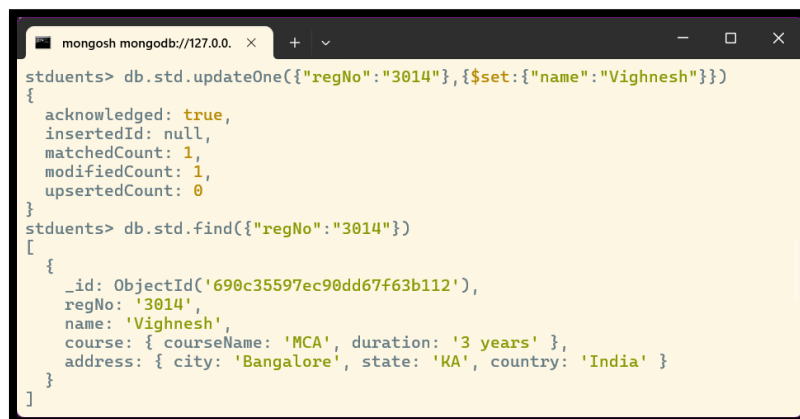
```
mongosh mongodb://127.0.0.1
stduents> db.std.find({"regNo":"3014"})
[
  {
    _id: ObjectId('690c35597ec90dd67f63b112'),
    regNo: '3014',
    name: 'VR',
    course: { courseName: 'MCA', duration: '3 years' },
    address: { city: 'Bangalore', state: 'KA', country: 'India' }
  }
]
```

- 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"})
```



```
mongosh mongodb://127.0.0.1
stduents> db.std.updateOne({"regNo":"3014"},{$set:{"name":"Vighnesh"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
stduents> db.std.find({"regNo":"3014"})
[
  {
    _id: ObjectId('690c35597ec90dd67f63b112'),
    regNo: '3014',
    name: 'Vighnesh',
    course: { courseName: 'MCA', duration: '3 years' },
    address: { city: 'Bangalore', state: 'KA', country: 'India' }
  }
]
```

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



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

```
db.std.find()
```



```
mongosh mongodb://127.0.0.1
stduents> db.std.deleteOne({"regNo":"3014"})
{ acknowledged: true, deletedCount: 1 }
stduents> 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"]}
```

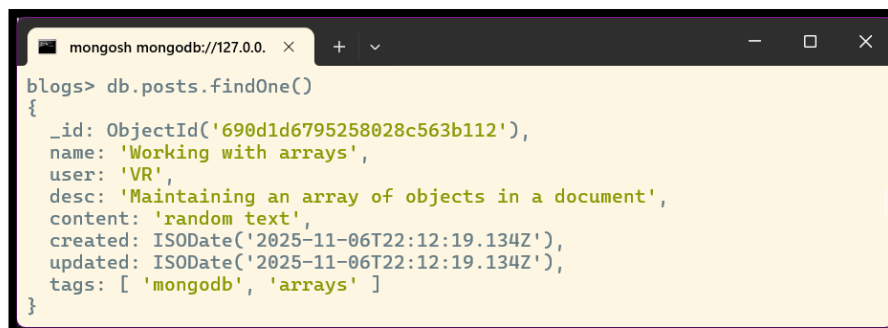
```
db.posts.insertOne(new_post)
```



```
mongosh mongodb://127.0.0.1
test> use blogs
switched to db blogs
blogs> db.createCollection("posts")
{ ok: 1 }
blogs> new_post={name:"Working with arrays",user:"VR",desc:"Maintaining a  
n array of objects in a document",content:"random text",created:ISODate()  
,updated:ISODate(),tags:[ "mongodb","arrays"]}
{
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ]
}
blogs> db.posts.insertOne(new_post)
{
  acknowledged: true,
  insertedId: ObjectId('690d1d6795258028c563b112')
}
```



```
db.posts.findOne()
```



```
mongosh mongodb://127.0.0.1
blogs> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ]
}
```

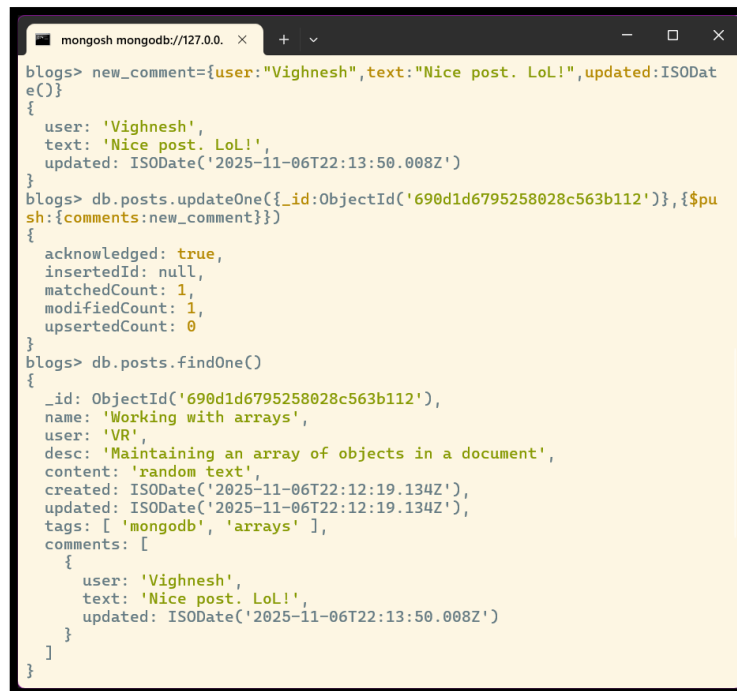


```
new_comment={user: "Vighnesh",text: "Nice post. LoL!",updated:ISODate()}
```

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

```
db.posts.findOne()
```





```

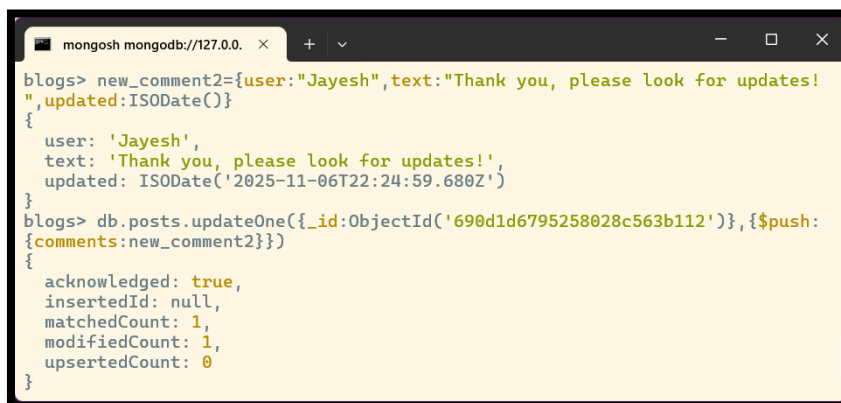
mongosh mongodb://127.0.0.1
blogs> new_comment={user:"Vighnesh",text:"Nice post. LoL!",updated:ISODate()}
{
  user: 'Vighnesh',
  text: 'Nice post. LoL!',
  updated: ISODate('2025-11-06T22:13:50.008Z')
}
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{ $push: {comments:new_comment}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
blogs> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ],
  comments: [
    {
      user: 'Vighnesh',
      text: 'Nice post. LoL!',
      updated: ISODate('2025-11-06T22:13:50.008Z')
    }
  ]
}

```



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

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

```

mongosh mongodb://127.0.0.1
blogs> new_comment2={user:"Jayesh",text:"Thank you, please look for updates!",updated:ISODate()}
{
  user: 'Jayesh',
  text: 'Thank you, please look for updates!',
  updated: ISODate('2025-11-06T22:24:59.680Z')
}
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{ $push: {comments:new_comment2}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```



`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()`



```

mongosh mongodb://127.0.0.1:27027
> new_content="Thank you! Please look for the updates - updated post."
> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{"comment
s.user":"Jayesh"},{$set:{"comments.$.text":new_content}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ],
  comments: [
    {
      user: 'Vighnesh',
      text: 'Nice post. LoL!',
      updated: ISODate('2025-11-06T22:13:50.008Z')
    },
    {
      user: 'Jayesh',
      text: 'Thank you! Please look for the updates - updated post.',
      updated: ISODate('2025-11-06T22:24:59.680Z')
    }
  ]
}

```



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

```
db.posts.findOne()
```



```

mongosh mongodb://127.0.0.1:27027
> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$pull:
{comments:{user:"Jayesh"}}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ],
  comments: [
    {
      user: 'Vighnesh',
      text: 'Nice post. LoL!',
      updated: ISODate('2025-11-06T22:13:50.008Z')
    }
  ]
}

```



```
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}})
```



```

mongosh mongodb://127.0.0.1
blogs> new_comment3={user:"Anriq",text:"Thank you for your comment. I have updated the post",updated:ISODate()}
{
  user: 'Anriq',
  text: 'Thank you for your comment. I have updated the post',
  updated: ISODate('2025-11-06T22:39:44.968Z')
}
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comments:new_comment3}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```



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

```
db.posts.findOne()
```



```

mongosh mongodb://127.0.0.1
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$set: {"comments.$.likes":0}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
blogs> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ],
  comments: [
    {
      user: 'Vighnesh',
      text: 'Nice post. Lol!',
      updated: ISODate('2025-11-06T22:13:50.008Z'),
      likes: 0
    },
    {
      user: 'Anriq',
      text: 'Thank you for your comment. I have updated the post',
      updated: ISODate('2025-11-06T22:39:44.968Z'),
      likes: 0
    }
  ]
}

```



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

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



```

mongosh mongodb://127.0.0.1
blogs> new_comment3={user:"Ashok",text:"Thanks for the update!",updated:ISO Date()}
{
  user: 'Ashok',
  text: 'Thanks for the update!',
  updated: ISODate('2025-11-07T00:11:15.361Z')
}
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{$push:{comments:new_comment3}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```



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

```
db.posts.findOne()
```



```
mongosh mongodb://127.0.0.1
blogs> db.posts.updateOne({_id:ObjectId('690d1d6795258028c563b112')},{ $inc: {"comments.$[ele].likes":1}}, {arrayFilters:[ {"ele.user":"Ashok","ele.likes":{"$exists:false}}]})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
blogs> db.posts.findOne()
{
  _id: ObjectId('690d1d6795258028c563b112'),
  name: 'Working with arrays',
  user: 'VR',
  desc: 'Maintaining an array of objects in a document',
  content: 'random text',
  created: ISODate('2025-11-06T22:12:19.134Z'),
  updated: ISODate('2025-11-06T22:12:19.134Z'),
  tags: [ 'mongodb', 'arrays' ],
  comments: [
    {
      user: 'Vighnesh',
      text: 'Nice post. LoL!',
      updated: ISODate('2025-11-06T22:13:50.008Z'),
      likes: 0
    },
    {
      user: 'Anriq',
      text: 'Thank you for your comment. I have updated the post',
      updated: ISODate('2025-11-06T22:39:44.968Z'),
      likes: 0
    },
    {
      user: 'Ashok',
      text: 'Thanks for the update!',
      updated: ISODate('2025-11-06T22:57:46.539Z'),
      likes: 1
    }
  ]
}
```


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

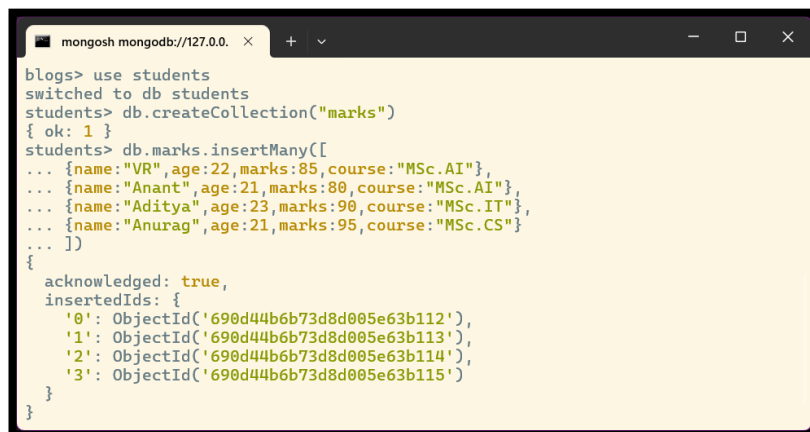
Step 1: Open command prompt and type mongosh command to open the mongosh shell.

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"}
])
```



```
mongosh mongodb://127.0.0.1:27020
blogs> use students
switched to db students
students> db.createCollection("marks")
{ ok: 1 }
students> 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"}
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('690d44b6b73d8d005e63b112'),
    '1': ObjectId('690d44b6b73d8d005e63b113'),
    '2': ObjectId('690d44b6b73d8d005e63b114'),
    '3': ObjectId('690d44b6b73d8d005e63b115')
  }
}
```

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://127.0.0.1
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://127.0.0.1
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://127.0.0.1
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.

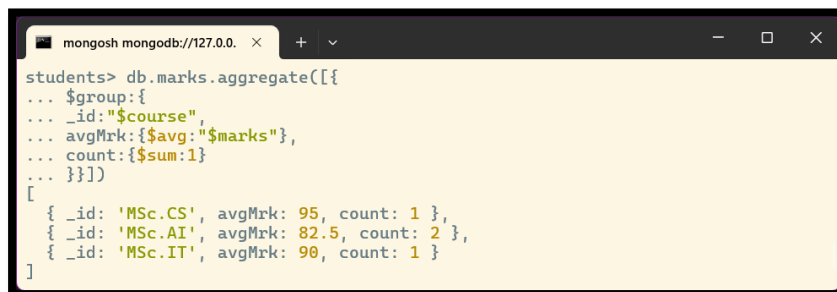
```
db.marks.find().sort({marks:-1}).skip(2).limit(4)
```



```
mongosh mongodb://127.0.0.1:27017
students> db.marks.find().sort({marks:-1}).skip(2).limit(4)
[
  {
    _id: ObjectId('690d44b6b73d8d005e63b112'),
    name: 'VR',
    age: 22,
    marks: 85,
    course: 'MSc.AI'
  },
  {
    _id: ObjectId('690d44b6b73d8d005e63b113'),
    name: 'Anant',
    age: 21,
    marks: 80,
    course: 'MSc.AI'
  }
]
```

- Aggregate

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



```
mongosh mongodb://127.0.0.1:27017
students> db.marks.aggregate([ {
... $group: {
... _id:"$course",
... avgMrk: {$avg:"$marks"},
... count: {$sum:1}
... } })
[
  { _id: 'MSc.CS', avgMrk: 95, count: 1 },
  { _id: 'MSc.AI', avgMrk: 82.5, count: 2 },
  { _id: 'MSc.IT', avgMrk: 90, count: 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:

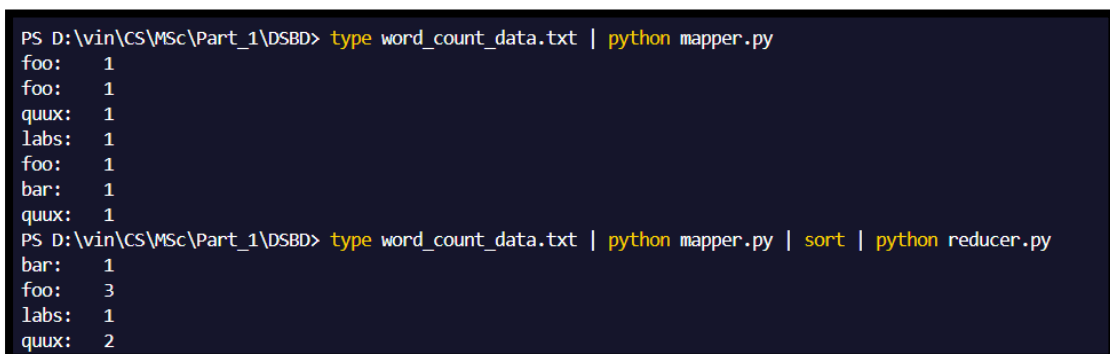
```
#!/usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
```

```
        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



```
PS D:\vin\CS\MSc\Part_1\DSBD> type word_count_data.txt | python mapper.py
foo: 1
foo: 1
quux: 1
labs: 1
foo: 1
bar: 1
quux: 1
PS D:\vin\CS\MSc\Part_1\DSBD> type word_count_data.txt | python mapper.py | sort | python reducer.py
bar: 1
foo: 3
labs: 1
quux: 2
```

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.

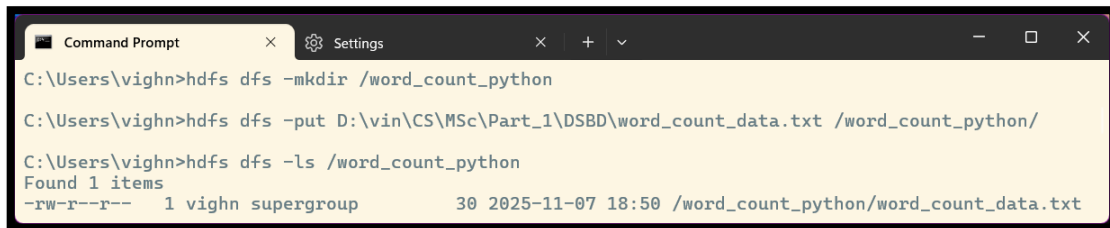
Step 4: Download [hadoop-streaming.jar](#) file and then place it in the same place as the mapper and reducer python files.

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
```



```

C:\Users\vighn>hdfs dfs -mkdir /word_count_python

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

C:\Users\vighn>hdfs dfs -ls /word_count_python
Found 1 items
-rw-r--r-- 1 vighn supergroup          30 2025-11-07 18:50 /word_count_python/word_count_data.txt

```

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

```
-input /word_count_python/word_count_data.txt ^
```

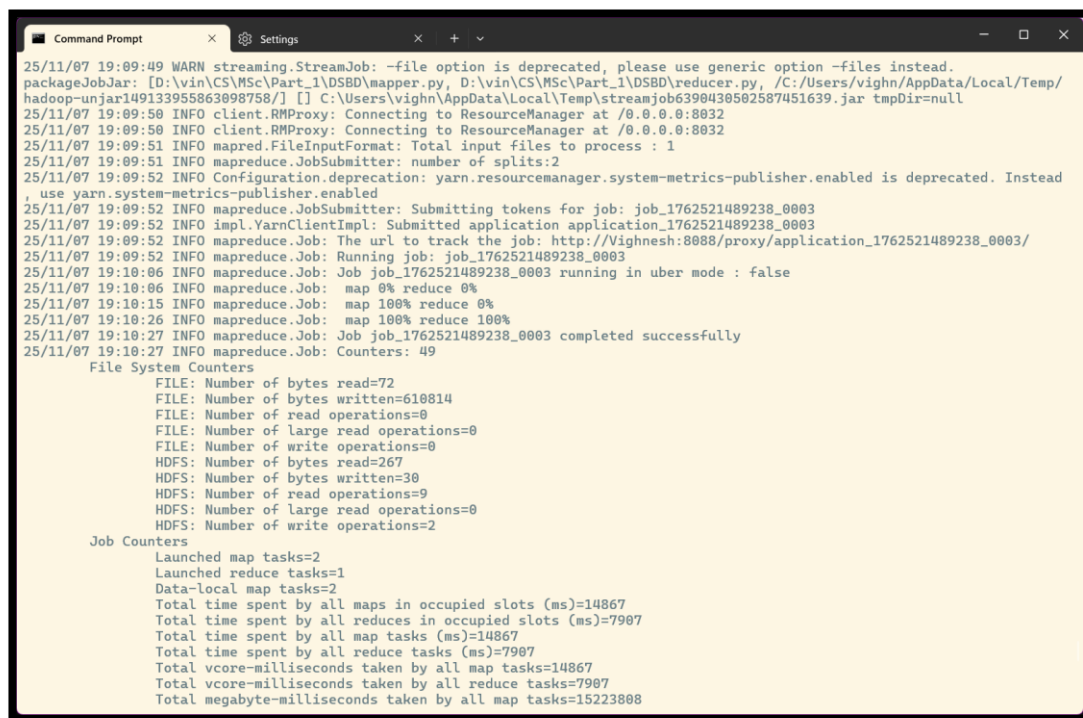
```
-output /word_count_python/output ^
```

```
-mapper "python mapper.py" ^
```

```
-reducer "python reducer.py" ^
```

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

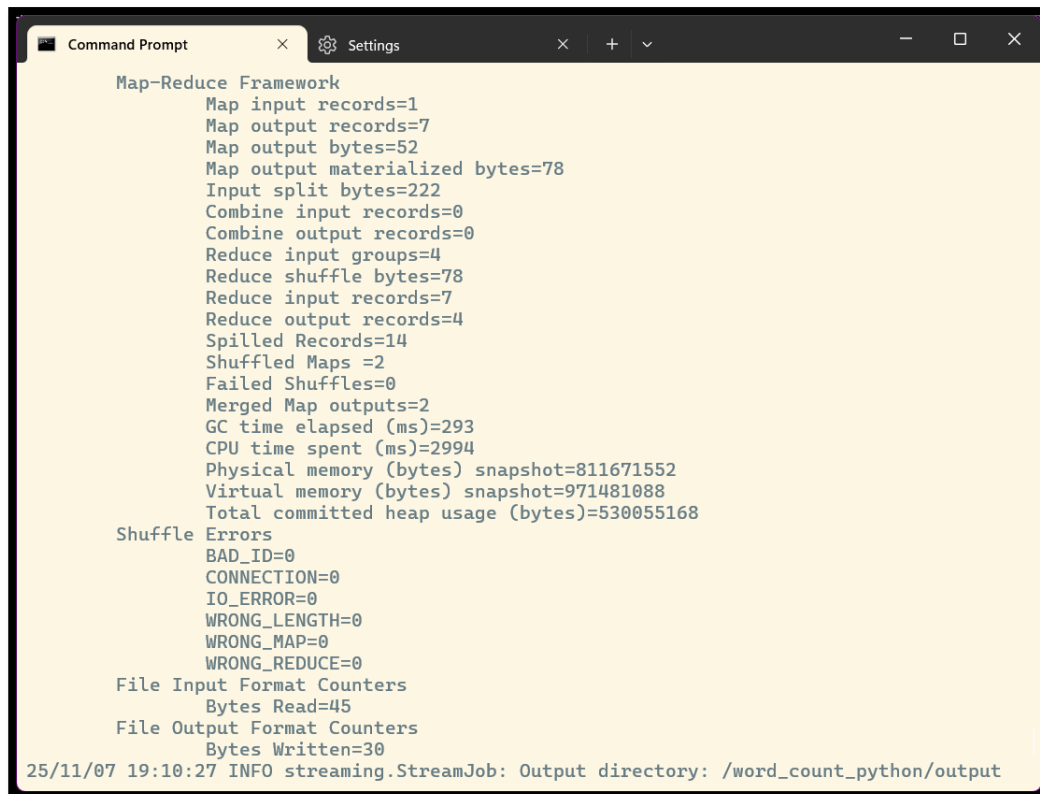
```
-file D:\vin\CS\MSc\Part_1\DSBD\reducer.py
```



```

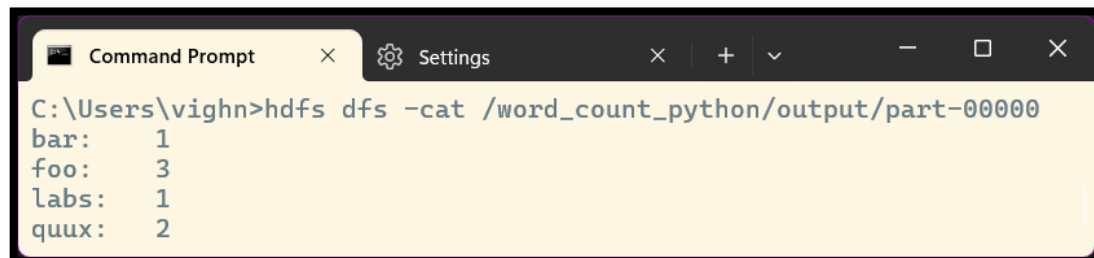
25/11/07 19:09:49 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [D:\vin\CS\MSc\Part_1\DSBD\mapper.py, D:\vin\CS\MSc\Part_1\DSBD\reducer.py, /C:/Users/vighn/AppData/Local/Temp/
hadoop-unjar149133955863898758/] [] C:\Users\vighn\AppData\Local\Temp\streamjob6390438502587451639.jar tmpDir=null
25/11/07 19:09:50 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/07 19:09:50 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/07 19:09:51 INFO mapred.FileInputFormat: Total input files to process : 1
25/11/07 19:09:51 INFO mapreduce.JobSubmitter: number of splits:2
25/11/07 19:09:52 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead
, use yarn.system-metrics-publisher.enabled
25/11/07 19:09:52 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1762521489238_0003
25/11/07 19:09:52 INFO impl.YarnClientImpl: Submitted application application_1762521489238_0003
25/11/07 19:09:52 INFO mapreduce.Job: The url to track the job: http://vighnesh:8088/proxy/application_1762521489238_0003/
25/11/07 19:09:52 INFO mapreduce.Job: Running job: job_1762521489238_0003
25/11/07 19:10:06 INFO mapreduce.Job: Job job_1762521489238_0003 running in uber mode : false
25/11/07 19:10:06 INFO mapreduce.Job: map 0% reduce 0%
25/11/07 19:10:15 INFO mapreduce.Job: map 100% reduce 0%
25/11/07 19:10:26 INFO mapreduce.Job: map 100% reduce 100%
25/11/07 19:10:27 INFO mapreduce.Job: Job job_1762521489238_0003 completed successfully
25/11/07 19:10:27 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=72
    FILE: Number of bytes written=610814
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=267
    HDFS: Number of bytes written=30
    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)=14867
    Total time spent by all reduces in occupied slots (ms)=7907
    Total time spent by all map tasks (ms)=14867
    Total time spent by all reduce tasks (ms)=7907
    Total vcore-milliseconds taken by all map tasks=14867
    Total vcore-milliseconds taken by all reduce tasks=7907
    Total megabyte-milliseconds taken by all map tasks=15223808

```



```
Map-Reduce Framework
  Map input records=1
  Map output records=7
  Map output bytes=52
  Map output materialized bytes=78
  Input split bytes=222
  Combine input records=0
  Combine output records=0
  Reduce input groups=4
  Reduce shuffle bytes=78
  Reduce input records=7
  Reduce output records=4
  Spilled Records=14
  Shuffled Maps =2
  Failed Shuffles=0
  Merged Map outputs=2
  GC time elapsed (ms)=293
  CPU time spent (ms)=2994
  Physical memory (bytes) snapshot=811671552
  Virtual memory (bytes) snapshot=971481088
  Total committed heap usage (bytes)=530055168
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=45
File Output Format Counters
  Bytes Written=30
25/11/07 19:10:27 INFO streaming.StreamJob: Output directory: /word_count_python/output
```

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



```
C:\Users\vighn>hdfs dfs -cat /word_count_python/output/part-00000
bar:      1
foo:      3
labs:     1
quux:     2
```

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}')
```

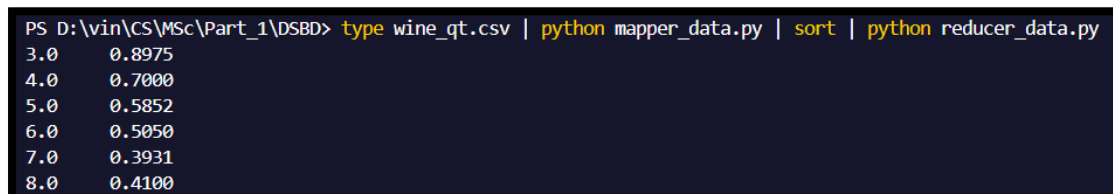
Step 3: Type the following commands in the terminal to test the created python files.

```
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
```



```
Command Prompt
C:\Users\vighn>hdfs dfs -mkdir /data_read

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

C:\Users\vighn>hdfs dfs -ls /data_read
Found 1 items
-rw-r--r-- 1 vighn supergroup 78057 2025-11-08 07:16 /data_read/wine_qt.csv
```

```

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

```

```

packageJobJar: [D:\vin\CS\MSc\Part_1\DSBD\mapper_data.py, D:\vin\CS\MSc\Part_1\DSBD\reducer_data.py, /C:/Users/vighn/AppData/Local/Temp/hadoop-unjar5181667229286955133/] [ C:\Users\vighn\AppData\Local\Temp\streamjob5687192327346171899.jar tmpDir=null
25/11/08 07:19:19 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/08 07:19:19 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/08 07:19:20 INFO mapred.FileInputFormat: Total input files to process : 1
25/11/08 07:19:20 INFO mapreduce.JobSubmitter: number of splits:2
25/11/08 07:19:20 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
25/11/08 07:19:20 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1762564284924_0001
25/11/08 07:19:21 INFO impl.YarnClientImpl: Submitted application application_1762564284924_0001
25/11/08 07:19:21 INFO mapreduce.Job: The url to track the job: http://Vighnesh:8088/proxy/application_1762564284924_0001/
25/11/08 07:19:21 INFO mapreduce.Job: Running job: job_1762564284924_0001
25/11/08 07:19:38 INFO mapreduce.Job: Job job_1762564284924_0001 running in uber mode : false
25/11/08 07:19:38 INFO mapreduce.Job: map 0% reduce 0%
25/11/08 07:19:51 INFO mapreduce.Job: map 100% reduce 0%
25/11/08 07:20:00 INFO mapreduce.Job: map 100% reduce 100%
25/11/08 07:20:01 INFO mapreduce.Job: Job job_1762564284924_0001 completed successfully
25/11/08 07:20:01 INFO mapreduce.Job: Counters: 49

File System Counters
  FILE: Number of bytes read=6540
  FILE: Number of bytes written=623798
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=82343
  HDFS: Number of bytes written=66
  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=1
  Total time spent by all maps in occupied slots (ms)=19351
  Total time spent by all reduces in occupied slots (ms)=7541
  Total time spent by all map tasks (ms)=19351
  Total time spent by all reduce tasks (ms)=7541
  Total vcore-milliseconds taken by all map tasks=19351
  Total vcore-milliseconds taken by all reduce tasks=7541
  Total megabyte-milliseconds taken by all map tasks=19815424
  Total megabyte-milliseconds taken by all reduce tasks=7721984

```

```

Map-Reduce Framework
  Map input records=1184
  Map output records=592
  Map output bytes=53358
  Map output materialized bytes=6546
  Input split bytes=190
  Combine input records=0
  Combine output records=0
  Reduce input groups=6
  Reduce shuffle bytes=6546
  Reduce input records=592
  Reduce output records=6
  Spilled Records=1184
  Shuffled Maps =2
  Failed Shuffles=0
  Merged Map outputs=2
  GC time elapsed (ms)=285
  CPU time spent (ms)=3261
  Physical memory (bytes) snapshot=824635392
  Virtual memory (bytes) snapshot=981536768
  Total committed heap usage (bytes)=528482384

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=82153
  File Output Format Counters
  Bytes Written=66
25/11/08 07:20:01 INFO streaming.StreamJob: Output directory: /data_read/output

```

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

```

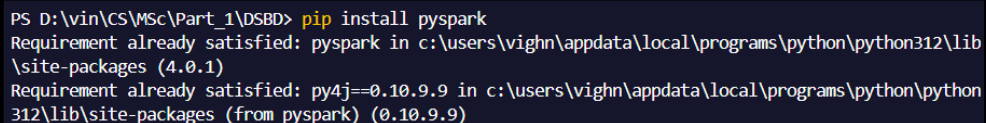
C:\Users\vighn>hdfs dfs -cat /data_read/output/part-00000
3.0      0.5950
4.0      0.6997
5.0      0.5866
6.0      0.4992
7.0      0.4304
8.0      0.4300

```

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\python312\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)

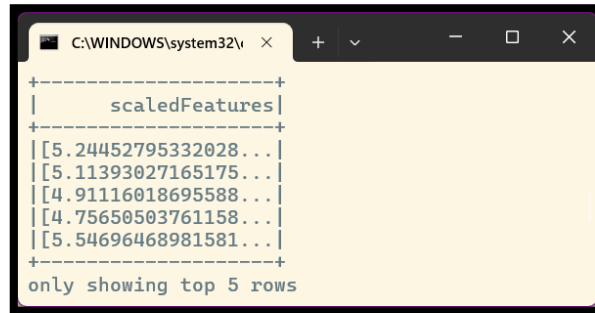
scalerModel = scaler.fit(final_data)

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)

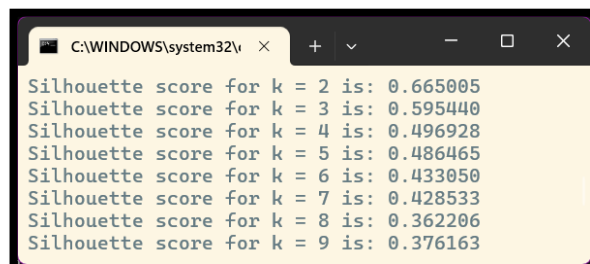
    score = evaluator.evaluate(pred)

    silhouette_score.append(score)

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

print()

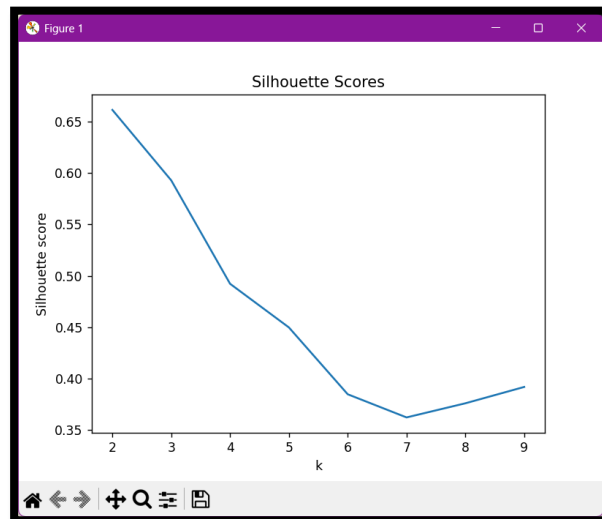
```



```

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()

```

```

C:\WINDOWS\system32\cmd. x + v
Cluster Centers:
[ 6.32636687 12.38115343 37.39222755 13.9206997  9.75485787  2.41428438
 12.28078861]
[ 4.0648023  10.14242485 35.82143905 11.81918014  7.51855717  3.19362266
 10.40520609]
[ 4.91309043 10.92012526 37.32658724 12.37724251  8.59393872  1.82261116
 10.35389957]

```

```

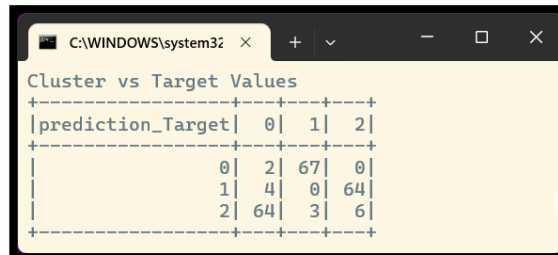
from pyspark.sql import functions as F

```

```

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

```

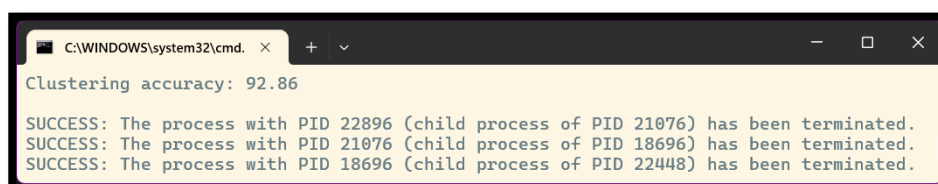


prediction_Target	0	1	2
0	2	67	0
1	4	0	64
2	64	3	6

```

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()

```



```

Clustering accuracy: 92.86
SUCCESS: The process with PID 22896 (child process of PID 21076) has been terminated.
SUCCESS: The process with PID 21076 (child process of PID 18696) has been terminated.
SUCCESS: The process with PID 18696 (child process of PID 22448) has been terminated.

```