

EXPERIMENT NUMBER – 1

AIM: Install, configure and run python, numPy and Pandas.

DESCRIPTION:

Python:

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

NumPy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software.

Pandas:

Pandas is an open-source library in Python that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library of Python. Pandas is fast and it has high performance & productivity for users.

PROCEDURE:

Installation of Python:

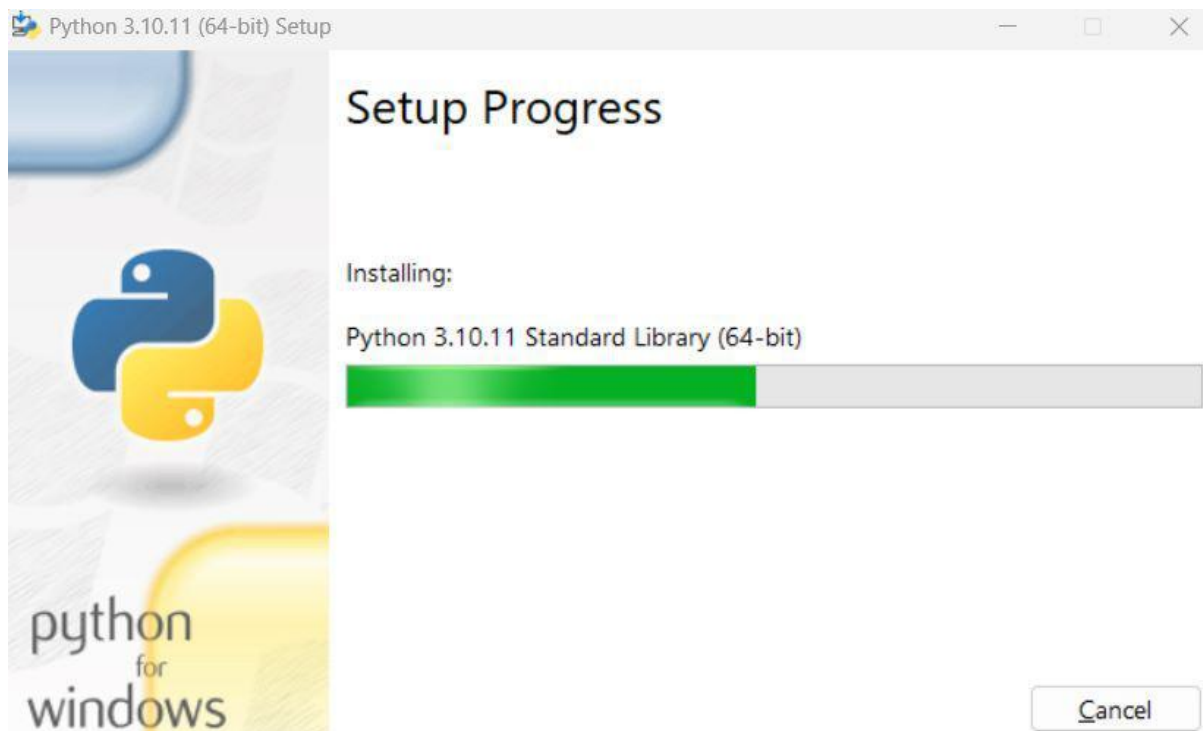
Step-1: Visit the official page for Python <https://www.python.org/downloads/> on the Windows operating system. Locate a reliable version of Python 3, preferably version 3.10.11, which was used in testing this tutorial. Choose the correct link for your device from the options provided: either Windows installer (64-bit) or Windows installer (32-bit) and proceed to download the executable file.



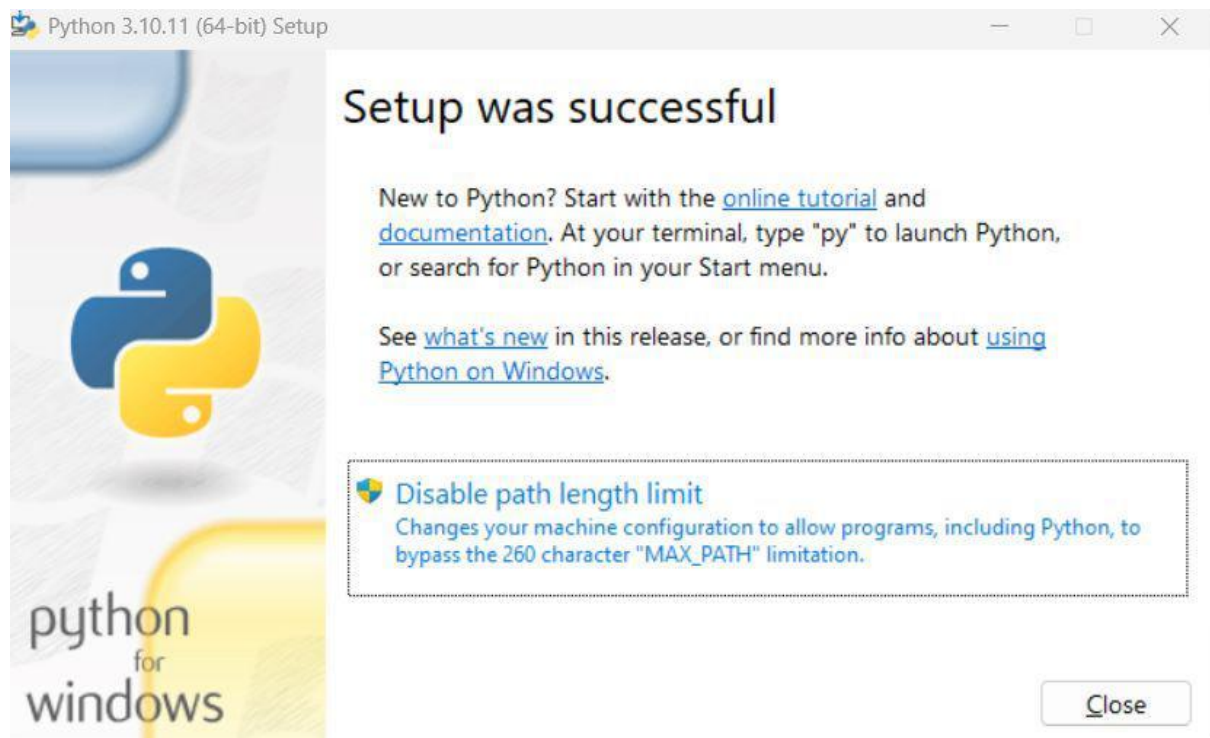
Step-2: Once you have downloaded the installer, open the .exe file, such as python-3.10.11-amd64.exe, by double-clicking it to launch the Python installer. Choose the option to Install the launcher for all users by checking the corresponding checkbox, so that all users of the computer can access the Python launcher application. Enable users to run Python from the command line by checking the Add python.exe to PATH checkbox.



After Clicking the Install Now Button the setup will start installing Python on your Windows system. You will see a window like this.



Step-3: After completing the setup. Python will be installed on your Windows system. You will see a successful message.

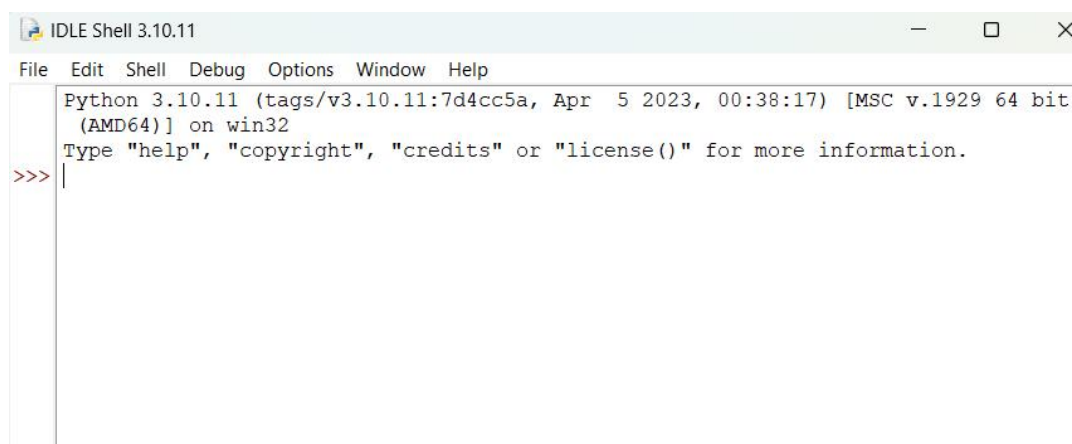


Step-4: Close the window after successful installation of Python. You can check if the installation of Python was successful by using either the command line or the Integrated Development Environment (IDLE), which you may have installed. To access the command line, click on the Start menu and type "cmd" in the search bar. Then click on Command Prompt.

python --version

```
PS C:\Users\sivasankar> python --version
Python 3.9.13
```

You can also check the version of Python by opening the IDLE application. Go to Start and enter IDLE in the search bar and then click the [IDLE](#) app, for example, IDLE (Python 3.10.11 64-bit). If you can see the Python IDLE window then you are successfully able to download and installed Python on Windows.



Installation of NumPy:

For Conda Users:

If you want the installation to be done through *conda*, you can use the below command:

```
conda install -c anaconda numpy
```

You will get a similar message once the installation is complete

```

Anaconda PowerShell Prompt (anaconda3)
The following packages will be downloaded:

package-----|-----build-----|-----
blas-1.0-----|-----mkl-----| 6 KB  anaconda
-----|-----|-----
Total: 6 KB

The following NEW packages will be INSTALLED:

blas                anaconda/win-64::blas-1.0-mkl
intel-openmp        pkgs/main/win-64::intel-openmp-2021.3.0-haa95532_3372
mkl                 pkgs/main/win-64::mkl-2021.3.0-haa95532_524
mkl-service         pkgs/main/win-64::mkl-service-2.4.0-py38h2bbff1b_0
mkl_fft             pkgs/main/win-64::mkl_fft-1.3.0-py38h277e83a_2
mkl_random          pkgs/main/win-64::mkl_random-1.2.2-py38hf11a4ad_0
numpy               pkgs/main/win-64::numpy-1.20.3-py38ha4e8547_0
numpy-base         pkgs/main/win-64::numpy-base-1.20.3-py38hc2deb75_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
blas-1.0 | 6 KB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Make sure you follow the best practices for installation using *conda* as:

- Use an environment for installation rather than in the base environment using the below command:

```
conda create -n my-env
```

```
conda activate my-env
```

For PIP Users:

Users who prefer to use *pip* can use the below command to install NumPy:

```
pip install numpy
```

You will get a similar message once the installation is complete:

```

ca. Command Prompt
Microsoft Windows [Version 10.0.19042.1202]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Geeks>pip install numpy
Collecting numpy
  Downloading numpy-1.21.2-cp39-cp39-win_amd64.whl (14.0 MB)
    |#####| 14.0 MB 6.4 MB/s
Installing collected packages: numpy
Successfully installed numpy-1.21.2
```

Now that we have installed Numpy successfully in our system

Installation of Pandas:

Using pip

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large “online repository” termed as Python Package Index (PyPI).

Install Python Pandas using Command Prompt

Pandas can be installed using PIP by use of the following command in Command Prompt.

```
pip install pandas
```

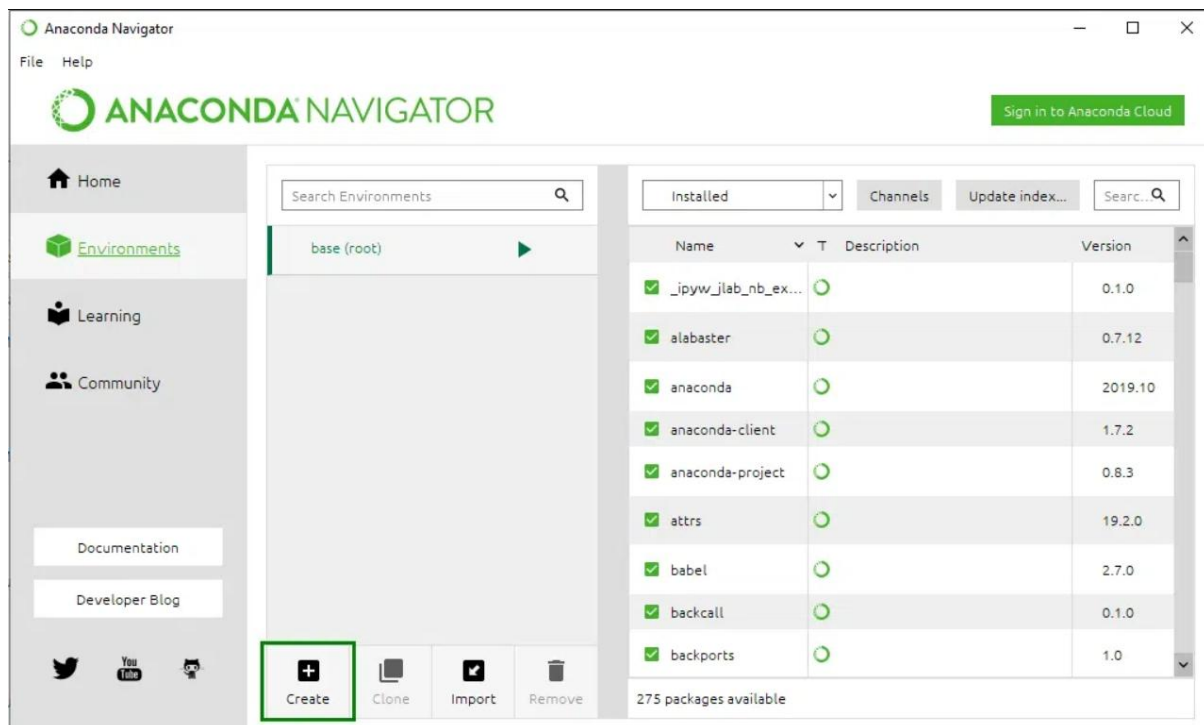
Install Pandas using Anaconda

Anaconda is open-source software that contains Jupyter, spyder, etc that is used for large data processing, Data Analytics, and heavy scientific computing. If your system is not pre-equipped with Anaconda Navigator, you can learn how to install Anaconda Navigator on Windows or Linux.

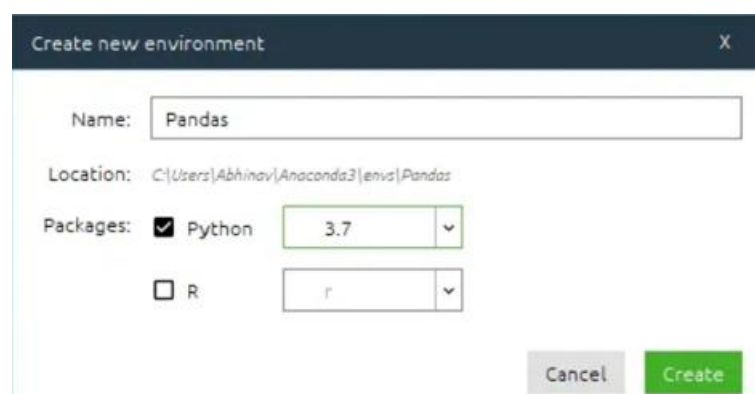
Install and Run Pandas from Anaconda Navigator

Step 1: Search for Anaconda Navigator in Start Menu and open it.

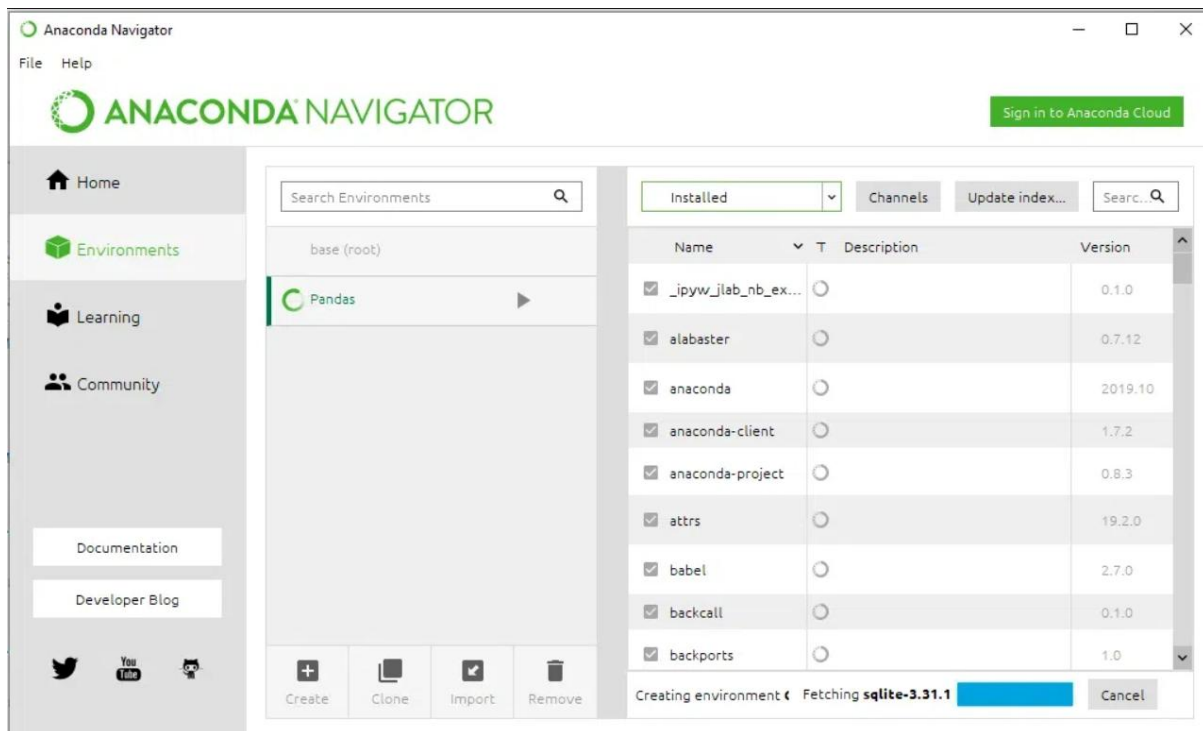
Step 2: Click on the Environment tab and then click on the Create button to create a new Pandas Environment.



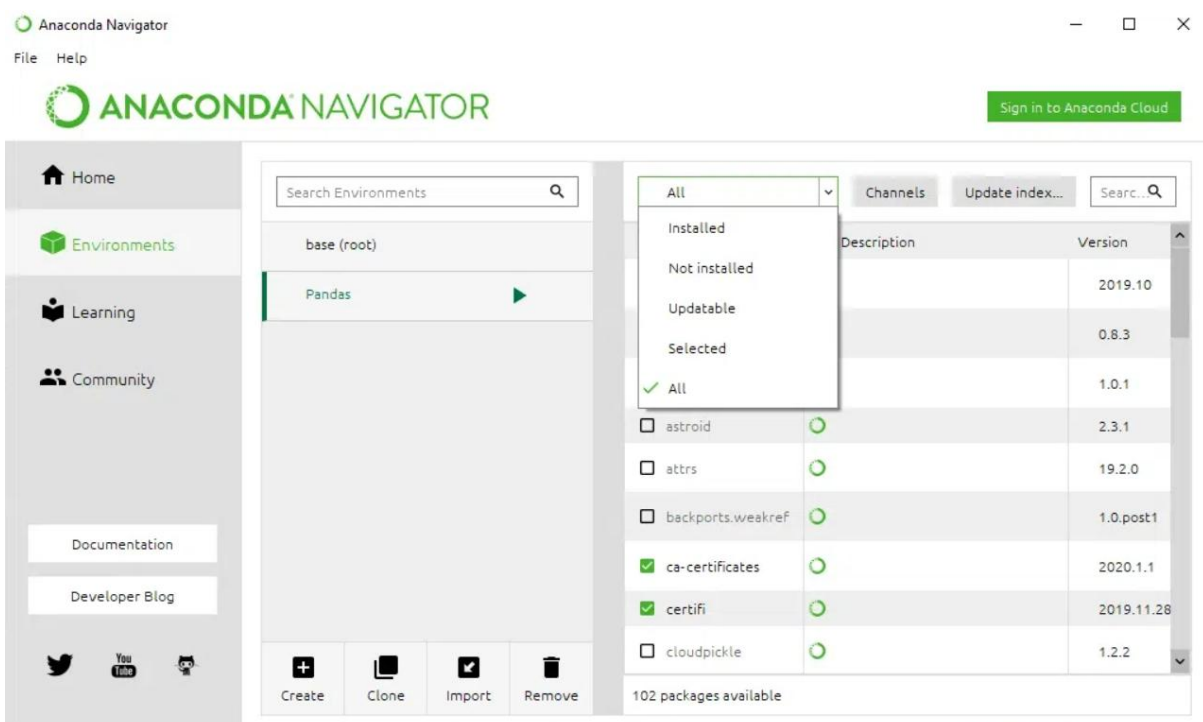
Step 3: Give a name to your Environment, e.g. Pandas, and then choose a Python and its version to run in the environment. Now click on the Create button to create Pandas Environment.



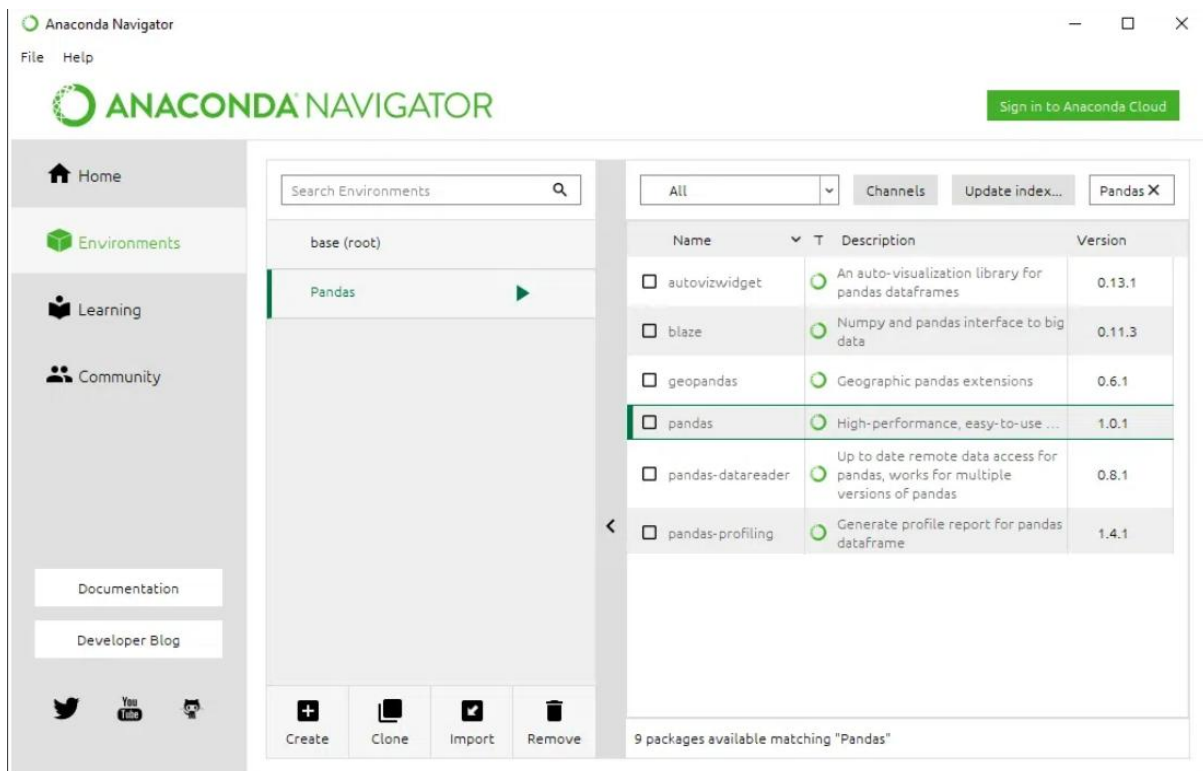
Step 4: Now click on the Pandas Environment created to activate it. Activate the environment



Step 5: In the list above package names, select All to filter all the packages.



Step 6: Now in the Search Bar, look for 'Pandas'. Select the Pandas package for Installation.



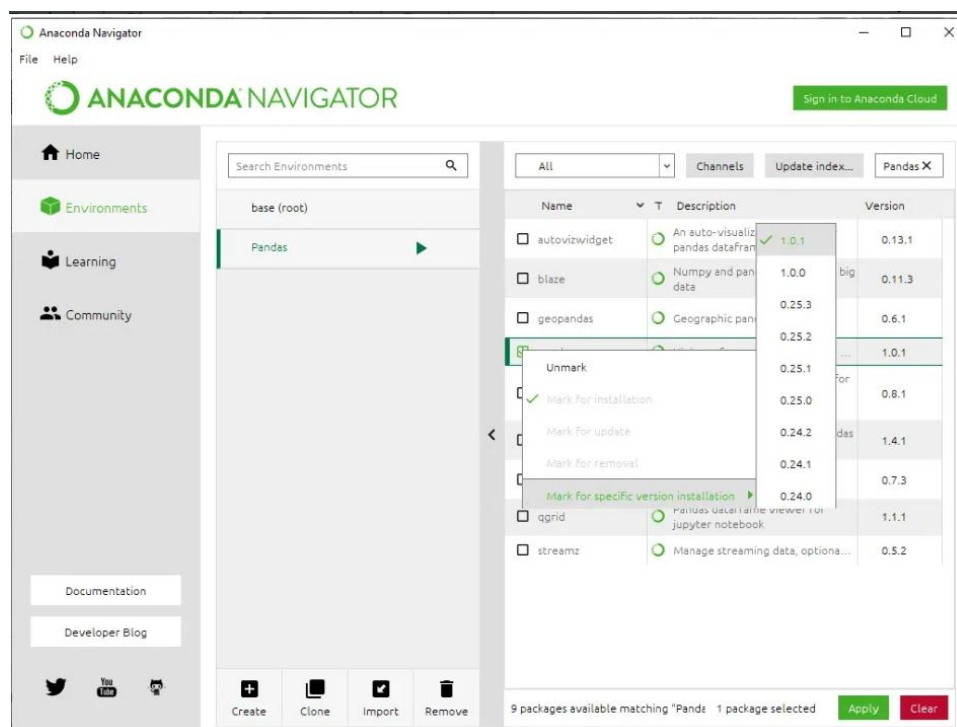
Step 7: Now Right Click on the checkbox given before the name of the package and then go to 'Mark for specific version installation'. Now select the version that you want to install.

Step 8: Click on the Apply button to install the Pandas Package.

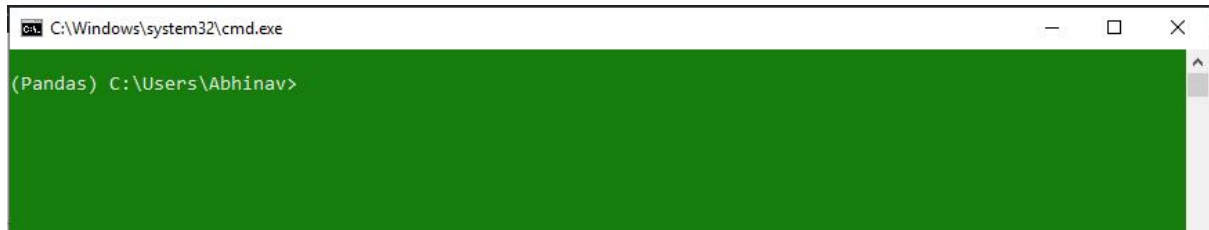
Step 9: Finish the Installation process by clicking on the Apply button.

Step 10: Now to open the Pandas Environment, click on the Green Arrow on the right of the package name and select the Console with which you want to begin your Pandas programming.

Pandas



Terminal Window:



```
C:\Windows\system32\cmd.exe
(Pandas) C:\Users\Abhinav>
```

CONCLUSION:

We have successfully installed and configured Python, Numpy and Pandas.

EXPERIMENT NUMBER – 2

AIM: To Install, configure and run Hadoop and HDFS.

DESCRIPTION:

Hadoop:

Hadoop is an open-source framework from Apache and is used to store process and analyze data which are very huge in volume. Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover, it can be scaled up just by adding nodes in the cluster.

Modules of Hadoop:

1. **HDFS:** Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
2. **Yarn:** Yet another Resource Negotiator is used for job scheduling and manage the cluster.
3. **Map Reduce:** This is a framework which helps Java programs to do the parallel computation on data using key value pair. The Map task takes input data and converts it into a data set which can be computed in Key value pair. The output of Map task is consumed by reduce task and then the out of reducer gives the desired result.
4. **Hadoop Common:** These Java libraries are used to start Hadoop and are used by other Hadoop modules.

HDFS:

Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.

It is cost effective as it uses commodity hardware. It involves the concept of blocks, data nodes and node name.

HDFS Concepts:

1. **Blocks:** A Block is the minimum amount of data that it can read or write. HDFS blocks are 128 MB by default and this is configurable. Files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a file system, if the file is in HDFS is smaller than block size, then it does not occupy full block's size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only. The HDFS block size is large just to minimize the cost of seek.
2. **Name Node:** HDFS works in master-worker pattern where the name node acts as master. Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block. The metadata are small, so it is stored in the memory of name node, allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently, so all this information is handled by a single

machine. The file system operations like opening, closing, renaming etc. are executed by it.

3. **Data Node:** They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

PROCEDURE:

Installation of Hadoop:

1) Java Installation

Step-1: Type "java -version" in prompt to find if the java is installed or not. If not then download java from <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html> . The tar filejdk-7u71-linux-x64.tar.gz will be downloaded to your system.

Step-2: Extract the file using the below command

```
#tar xzf jdk-7u71-linux-x64.tar.gz
```

Step-3: To make java available for all the users of UNIX move the file to /usr/local and set the path. In the prompt switch to root user and then type the command below to move the jdk to /usr/lib.

```
# mv jdk1.7.0_71 /usr/lib/
```

Now in ~/.bashrc file add the following commands to set up the path.

```
# export JAVA_HOME=/usr/lib/jdk1.7.0_71
```

```
# export PATH=PATH:$JAVA_HOME/bin
```

Now, you can check the installation by typing "java -version" in the prompt.

2) SSH Installation

SSH is used to interact with the master and slaves computer without any prompt for password. First of all create a Hadoop user on the master and slave systems

```
# useradd hadoop
```

```
# passwd Hadoop
```

To map the nodes open the hosts file present in /etc/ folder on all the machines and put the ip address along with their host name.

```
# vi /etc/hosts
```

Enter the lines below

```
190.12.1.114  hadoop-master
```

```
190.12.1.121  hadoop-salve-one
```

```
190.12.1.143  hadoop-slave-two
```

Set up SSH key in every node so that they can communicate among themselves without password. Commands for the same are:

```
# su hadoop
```

```
$ ssh-keygen -t rsa
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
```

```
$ chmod 0600 ~/.ssh/authorized_keys
```

```
$ exit
```

3) Hadoop Installation

Hadoop can be downloaded from

<http://developer.yahoo.com/hadoop/tutorial/module3.html>

Now extract the Hadoop and copy it to a location.

```
$ mkdir /usr/hadoop
```

```
$ sudo tar vxzf hadoop-2.2.0.tar.gz ?c /usr/hadoop
```

Change the ownership of Hadoop folder

```
$sudo chown -R hadoop usr/hadoop
```

Change the Hadoop configuration files:

All the files are present in /usr/local/Hadoop/etc/hadoop

1) In hadoop-env.sh file add

```
export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
```

2) In core-site.xml add following between configuration tabs,

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://hadoop-master:9000</value>
</property>
<property>
<name>dfs.permissions</name>
<value>>false</value>
</property>
</configuration>
```

3) In hdfs-site.xml add following between configuration tabs,

```
<configuration>
<property>
<name>dfs.data.dir</name>
<value>usr/hadoop/dfs/name/data</value>
<final>>true</final>
</property>
<property>
<name>dfs.name.dir</name>
<value>usr/hadoop/dfs/name</value>
<final>>true</final>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

4) Open the Mapred-site.xml and make the change as shown below

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>hadoop-master:9001</value>
</property>
</configuration>
```

5) Finally, update your \$HOME/.bashrc

```
cd $HOME
```

```
vi .bashrc
```

Append following lines in the end and save and exit

#Hadoop variables

```
export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
```

```
export HADOOP_INSTALL=/usr/hadoop
```

```
export PATH=$PATH:$HADOOP_INSTALL/bin
```

```
export PATH=$PATH:$HADOOP_INSTALL/sbin
```

```
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
```

```
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
```

```
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
```

```
export YARN_HOME=$HADOOP_INSTALL
```

On the slave machine install Hadoop using the command below

```
# su hadoop
```

```
$ cd /opt/hadoop
```

```
$ scp -r hadoop hadoop-slave-one:/usr/hadoop
```

```
$ scp -r hadoop hadoop-slave-two:/usr/Hadoop
```

Configure master node and slave node

```
$ vi etc/hadoop/masters
```

```
hadoop-master
```

```
$ vi etc/hadoop/slaves
```

```
hadoop-slave-one
```

```
hadoop-slave-two
```

After this format the name node and start all the daemons

```
# su hadoop
```

```
$ cd /usr/hadoop
```

```
$ bin/hadoop namenode -format
```

```
$ cd $HADOOP_HOME/sbin
```

```
$ start-all.sh
```

Starting HDFS

The HDFS should be formatted initially and then started in the distributed mode. Commands are given below.

To Format `$ hadoop namenode -format`

To Start `$ start-dfs.sh`

HDFS Basic File Operations

1. Putting data to HDFS from local file system

- First create a folder in HDFS where data can be put from local file system.

```
$ hadoop fs -mkdir /user/test
```

- Copy the file "data.txt" from a file kept in local folder /usr/home/Desktop to HDFS folder /user/ test

```
$ hadoop fs -copyFromLocal /usr/home/Desktop/data.txt /user/test
```

- Display the content of HDFS folder

```
$ Hadoop fs -ls /user/test
```

2. Copying data from HDFS to local file system

- `$ hadoop fs -copyToLocal /user/test/data.txt /usr/bin/data_copy.txt`

3. Compare the files and see that both are same
 - `$ md5 /usr/bin/data_copy.txt /usr/home/Desktop/data.txt`
- Recursive deleting
- `hadoop fs -rmr <arg>`
- Example:
- `hadoop fs -rmr /user/sonoo/`

HDFS Other commands:

The below is used in the commands

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system

- `put <localSrc><dest>`: Copies the file or directory from the local file system identified by localSrc to dest within the DFS.
- `copyFromLocal <localSrc><dest>`: Identical to -put
- `copyFromLocal <localSrc><dest>`: Identical to -put
- `moveFromLocal <localSrc><dest>`: Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.
- `get [-crc] <src><localDest>`: Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.
- `cat <filename>`: Displays the contents of filename on stdout.
- `moveToLocal <src><localDest>`: Works like -get, but deletes the HDFS copy on success.
- `setrep [-R] [-w] rep <path>`: Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)
- `touchz <path>`: Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.
- `test [-ezd] <path>`: Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.
- `stat [format] <path>`: Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).

CONCLUSION:

We have successfully installed and configured Hadoop and HDFS.

EXPERIMENT NUMBER – 3

AIM: Visualize data using basic plotting techniques in Python.

DESCRIPTION:

Data visualization is the discipline of trying to understand data by placing it in a visual context so that patterns, trends, and correlations that might not otherwise be detected can be exposed.

Python offers multiple great graphing libraries packed with lots of different features. Whether you want to create interactive or highly customized plots, Python has an excellent library for you.

To get a little overview, here are a few popular plotting libraries:

- **Matplotlib:** low level, provides lots of freedom
- **Pandas Visualization:** easy to use interface, built on Matplotlib
- **Seaborn:** high-level interface, great default styles
- **plotnine:** based on R's ggplot2, uses Grammar of Graphics
- **Plotly:** can create interactive plots

EXECUTION:

Importing Datasets

The Iris and Wine_Reviews dataset, which we can both load into memory using pandas read_csv method.

```
import pandas as pd
```

```
iris = pd.read_csv('iris.csv', names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])
```

```
print(iris.head())
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
wine_reviews = pd.read_csv('winemag-data-130k-v2.csv', index_col=0)
```

```
wine_reviews.head()
```

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	winery
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainstorm
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. Julian 2013 Reserve Late Harvest Riesling ...	Riesling	St. Julian
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Sweet Cheeks 2012 Vintner's Reserve Wild Child...	Pinot Noir	Sweet Cheeks

Matplotlib:

Matplotlib is the most popular Python plotting library. It is a low-level library with a Matlab-like interface that offers lots of freedom at the cost of having to write more code.

To install Matplotlib, pip, and conda can be used.

```
pip install matplotlib
```

or

```
conda install matplotlib
```

Matplotlib is specifically suitable for creating basic graphs like line charts, bar charts, histograms, etc. It can be imported by typing:

```
import matplotlib.pyplot as plt
```

Scatter Plot:

To create a scatter plot in Matplotlib, we can use the scatter method. We will also create a figure and an axis using plt.subplots to give our plot a title and labels.

```
# create a figure and axis
```

```
fig, ax = plt.subplots()
```

```
# scatter the sepal_length against the sepal_width
```

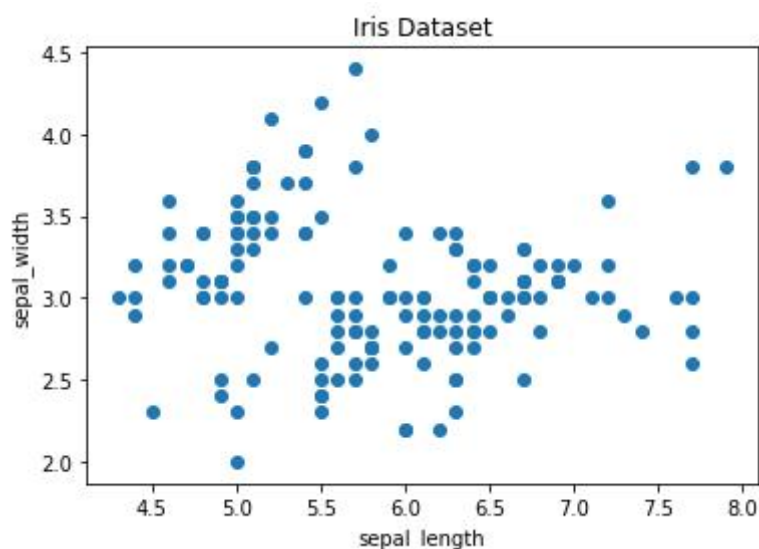
```
ax.scatter(iris['sepal_length'], iris['sepal_width'])
```

```
# set a title and labels
```

```
ax.set_title('Iris Dataset')
```

```
ax.set_xlabel('sepal_length')
```

```
ax.set_ylabel('sepal_width')
```



We can give the graph more meaning by coloring each data point by its class. This can be done by creating a dictionary that maps from class to color and then scattering each point on its own using a for-loop and passing the respective color.

```
# create color dictionary
```

```
colors = {'Iris-setosa':'r', 'Iris-versicolor':'g', 'Iris-virginica':'b'}
```

```
# create a figure and axis
```

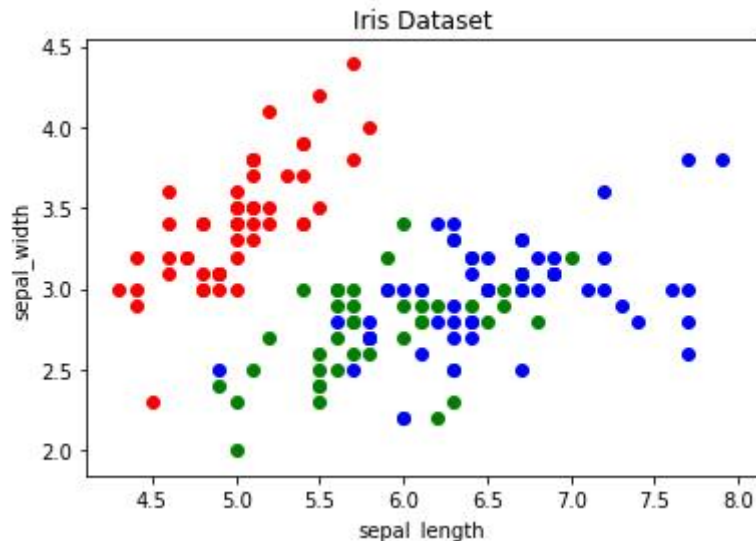
```
fig, ax = plt.subplots()
```

```
# plot each data-point
```

```
for i in range(len(iris['sepal_length'])):
```

```
    ax.scatter(iris['sepal_length'][i], iris['sepal_width'][i], color=colors[iris['class'][i]])
```

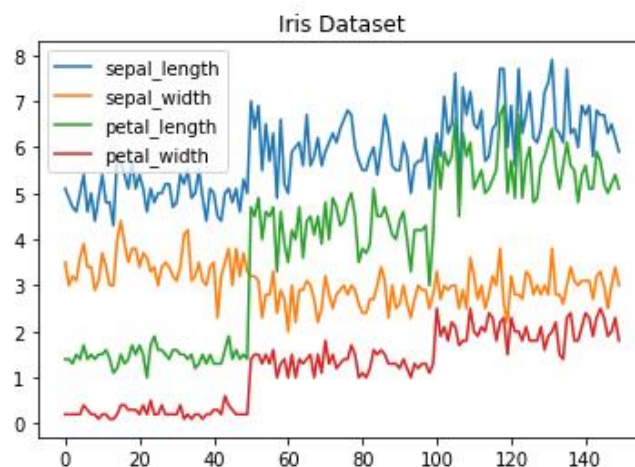
```
# set a title and labels
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```



Line Chart:

In Matplotlib, we can create a line chart by calling the plot method. We can also plot multiple columns in one graph by looping through the columns we want and plotting each column on the same axis.

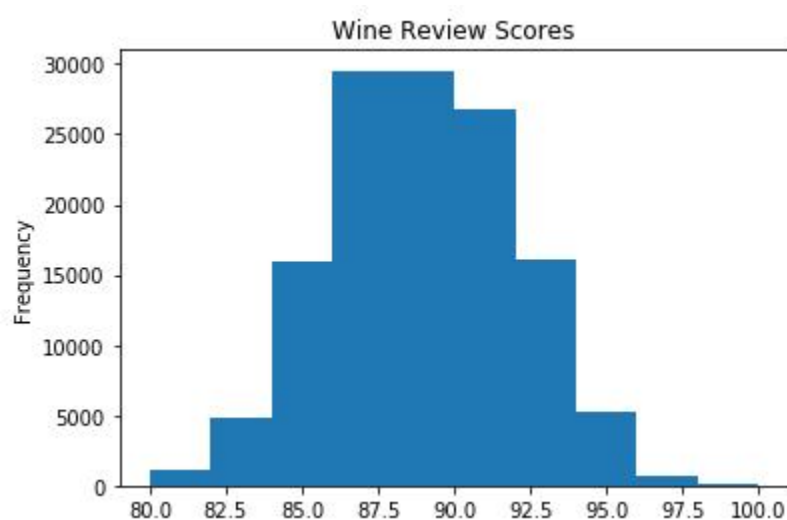
```
# get columns to plot
columns = iris.columns.drop(['class'])
# create x data
x_data = range(0, iris.shape[0])
# create figure and axis
fig, ax = plt.subplots()
# plot each column
for column in columns:
    ax.plot(x_data, iris[column])
# set title and legend
ax.set_title('Iris Dataset')
ax.legend()
```



Histogram:

In Matplotlib, we can create a Histogram using the hist method. If we pass categorical data like the points column from the wine-review dataset, it will automatically calculate how often each class occurs.

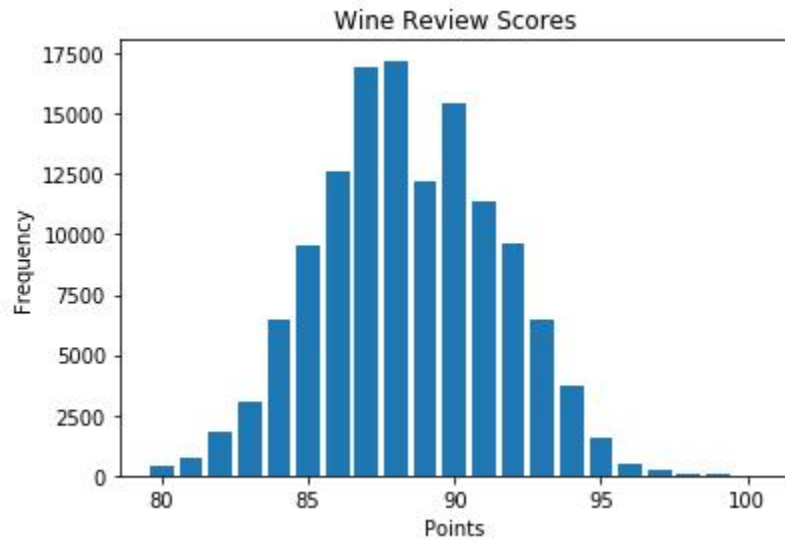
```
# create figure and axis  
fig, ax = plt.subplots()  
# plot histogram  
ax.hist(wine_reviews['points'])  
# set title and labels  
ax.set_title('Wine Review Scores')  
ax.set_xlabel('Points')  
ax.set_ylabel('Frequency')
```



Bar Chart:

A bar chart can be created using the bar method. The bar chart isn't automatically calculating the frequency of a category, so we will use pandas value_counts method to do this. The bar chart is useful for categorical data that doesn't have a lot of different categories (less than 30) because else it can get quite messy.

```
# create a figure and axis  
fig, ax = plt.subplots()  
# count the occurrence of each class  
data = wine_reviews['points'].value_counts()  
# get x and y data  
points = data.index  
frequency = data.values  
# create bar chart  
ax.bar(points, frequency)  
# set title and labels  
ax.set_title('Wine Review Scores')  
ax.set_xlabel('Points')  
ax.set_ylabel('Frequency')
```



Pandas Visualization:

Pandas is an open-source, high-performance, and easy-to-use library providing data structures, such as data frames and data analysis tools like the visualization tools we will use in this article.

Pandas Visualization makes it easy to create plots out of a pandas dataframe and series. It also has a higher-level API than Matplotlib, and therefore we need less code for the same results.

Pandas can be installed using either pip or conda.

pip install pandas

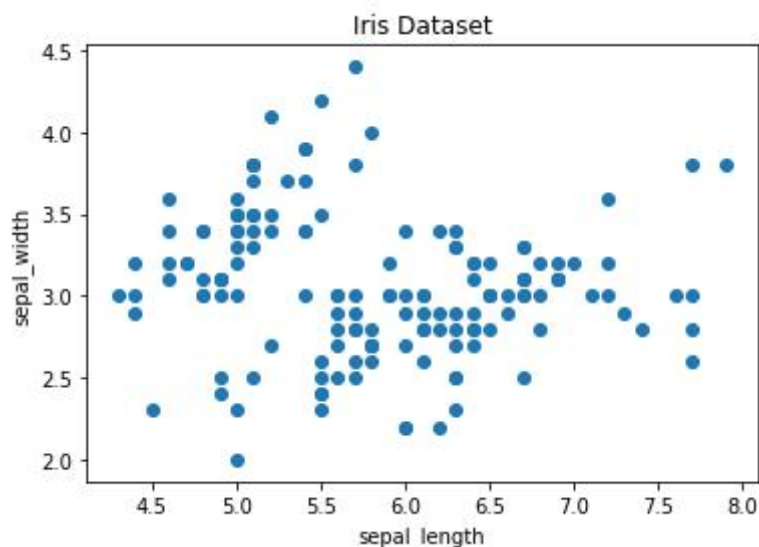
or

conda install pandas

Scatter Plot:

To create a scatter plot in Pandas, we can call `<dataset>.plot.scatter()` and pass it two arguments, the name of the x-column and the name of the y-column. Optionally we can also give it a title.

iris.plot.scatter(x='sepal_length', y='sepal_width', title='Iris Dataset')

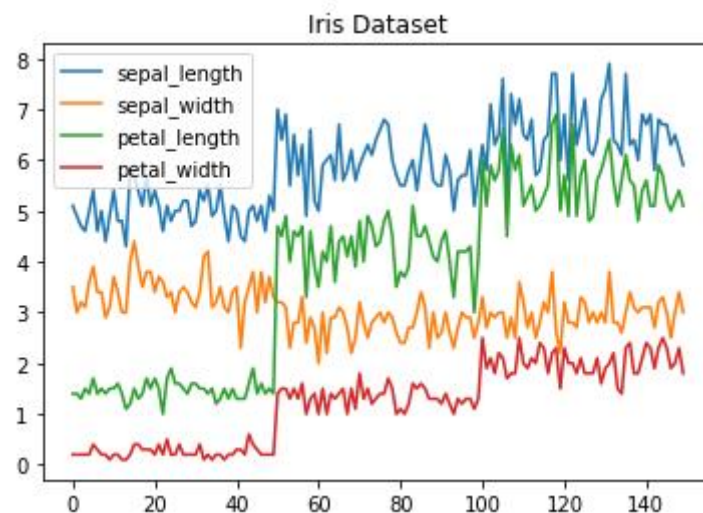


As you can see in the image, it is automatically setting the x and y label to the column names.

Line Chart:

To create a line chart in Pandas we can call `<dataframe>.plot.line()`. While in Matplotlib, we needed to loop through each column we wanted to plot, in Pandas we don't need to do this because it automatically plots all available numeric columns (at least if we don't specify a specific column/s).

```
iris.drop(['class'], axis=1).plot.line(title='Iris Dataset')
```

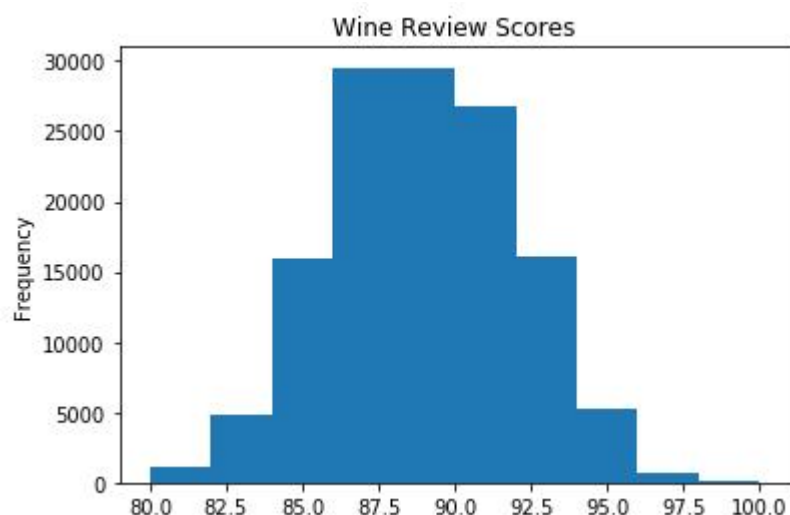


If we have more than one feature, Pandas automatically creates a legend for us, as seen in the image above.

Histogram:

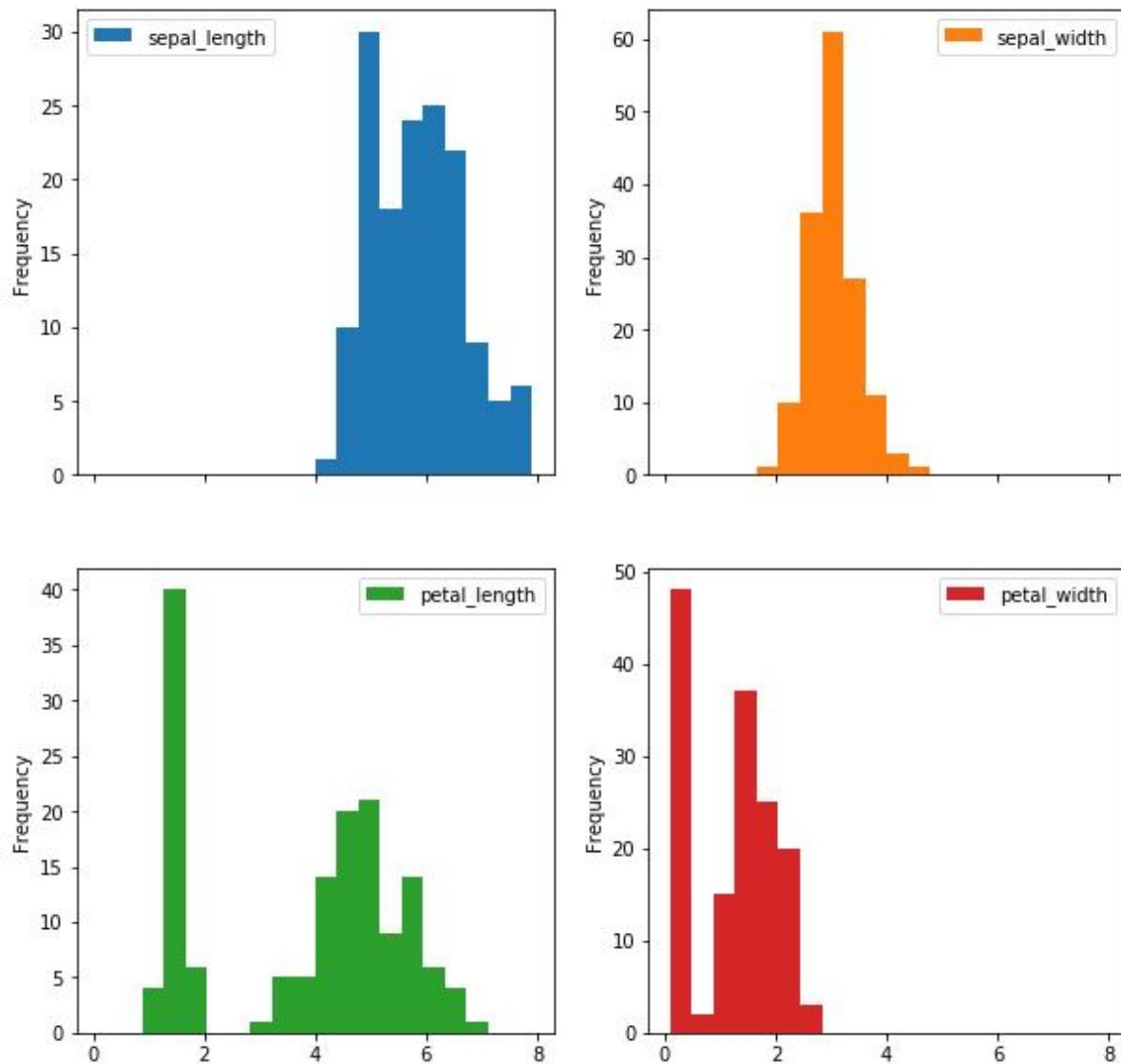
In Pandas, we can create a Histogram with the `plot.hist` method. There aren't any required arguments, but we can optionally pass some like the bin size.

```
wine_reviews['points'].plot.hist()
```



It's also straightforward to create multiple histograms.

```
iris.plot.hist(subplots=True, layout=(2,2), figsize=(10, 10), bins=20)
```

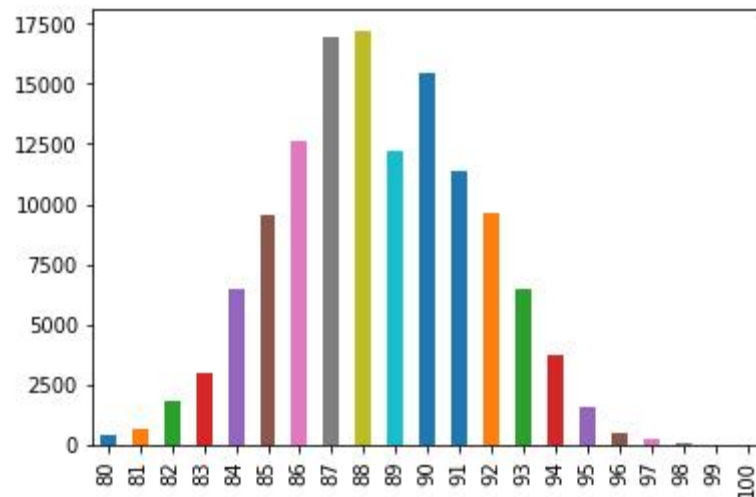


The subplots argument specifies that we want a separate plot for each feature, and the layout specifies the number of plots per row and column.

Bar Chart

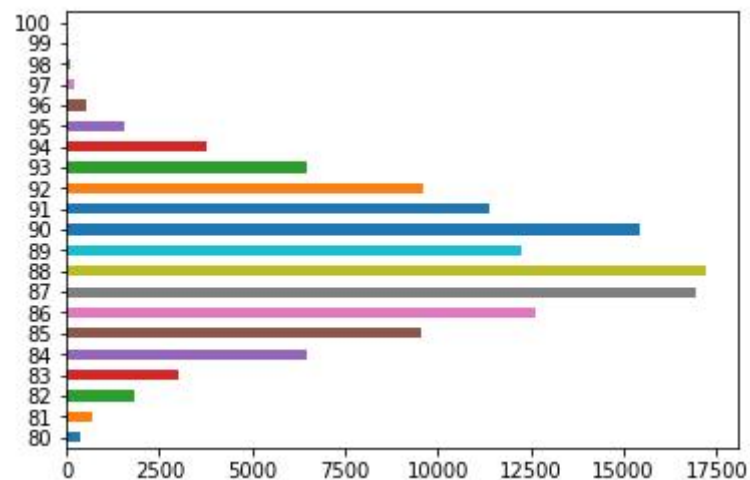
To plot a bar chart, we can use the `plot.bar()` method, but before calling this, we need to get our data. We will first count the occurrences using the `value_count()` method and then sort the occurrences from smallest to largest using the `sort_index()` method.

```
wine_reviews['points'].value_counts().sort_index().plot.bar()
```



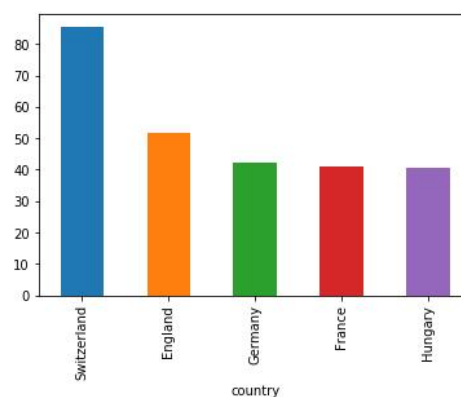
It's also really simple to make a horizontal bar chart using the `plot.barh()` method.

```
wine_reviews['points'].value_counts().sort_index().plot.barh()
```



We can also plot other data than the number of occurrences.

```
wine_reviews.groupby("country").price.mean().sort_values(ascending=False)[:5].plot.bar()
```



In the example above, we grouped the data by country, took the mean of the wine prices, ordered it, and plotted the five countries with the highest average wine price.

Seaborn

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for creating attractive graphs.

Seaborn has a lot to offer. For example, you can create graphs in one line that would take multiple tens of lines in Matplotlib. Its standard designs are awesome, and it also has a nice interface for working with Pandas dataframes.

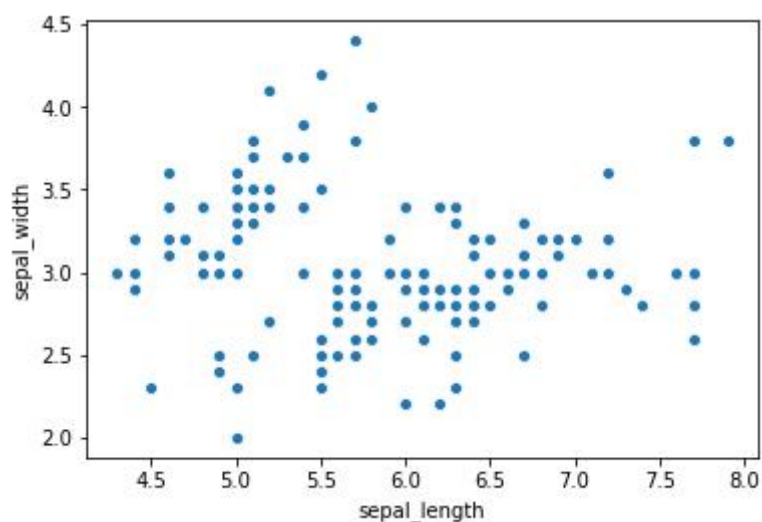
It can be imported by typing:

```
import seaborn as sns
```

Scatter plot:

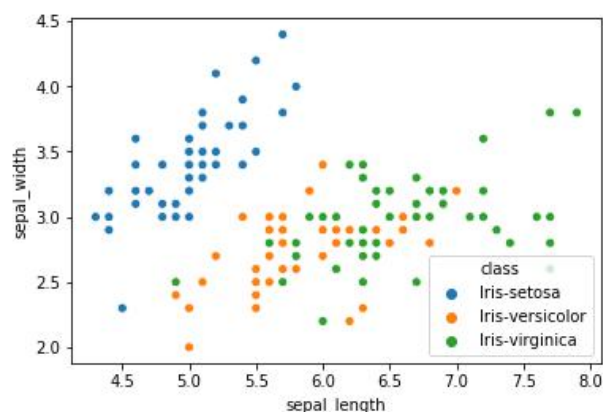
We can use the scatterplot method for creating a scatterplot, and just as in Pandas, we need to pass it the column names of the x and y data, but now we also need to pass the data as an additional argument because we aren't calling the function on the data directly as we did in Pandas.

```
sns.scatterplot(x='sepal_length', y='sepal_width', data=iris)
```



We can also highlight the points by class using the hue argument, which is a lot easier than in Matplotlib.

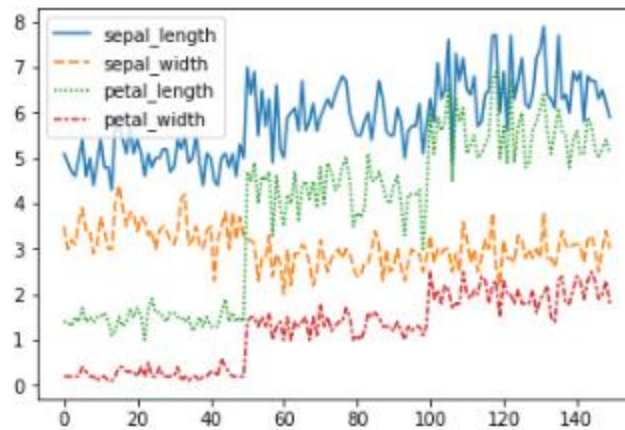
```
sns.scatterplot(x='sepal_length', y='sepal_width', hue='class', data=iris)
```



Line chart:

To create a line chart, the `sns.lineplot` method can be used. The only required argument is the data, which in our case are the four numeric columns from the Iris dataset. We could also use the `sns.kdeplot` method, which smoothes the edges of the curves and therefore is cleaner if you have a lot of outliers in your dataset.

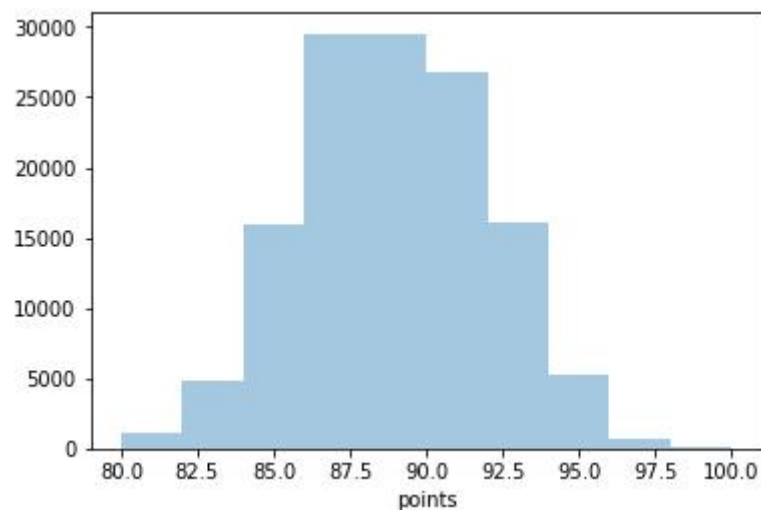
```
sns.lineplot(data=iris.drop(['class'], axis=1))
```



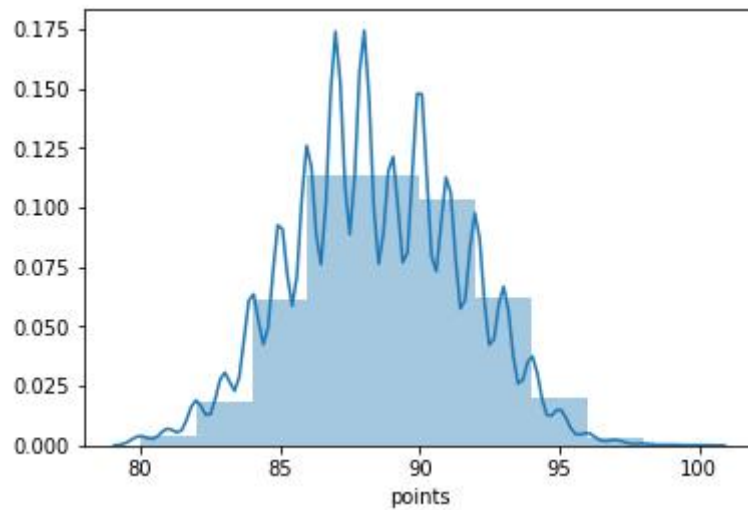
Histogram:

To create a histogram in Seaborn, we use the `sns.distplot` method. We need to pass it the column we want to plot, and it will calculate the occurrences itself. We can also pass it the number of bins and if we want to plot a gaussian kernel density estimate inside the graph.

```
sns.distplot(wine_reviews['points'], bins=10, kde=False)
```



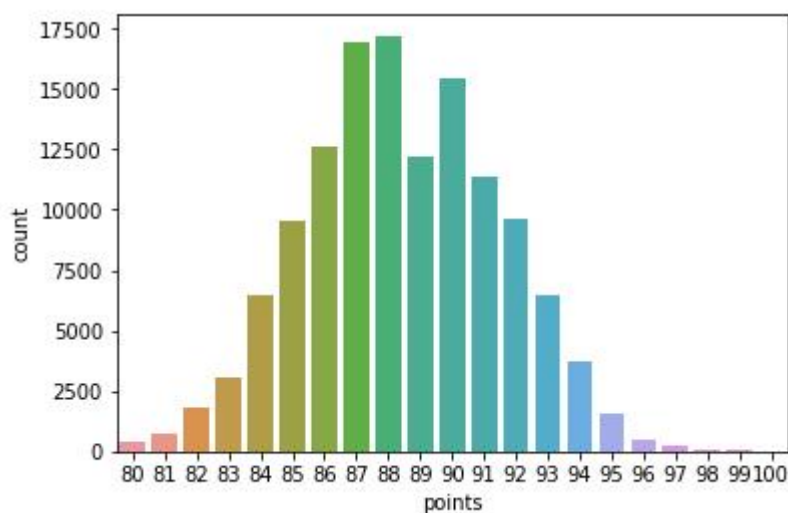
```
sns.distplot(wine_reviews['points'], bins=10, kde=True)
```



Bar chart:

In Seaborn, a bar chart can be created using the `sns.countplot` method and passing it the data.

```
sns.countplot(wine_reviews['points'])
```



Other graphs:

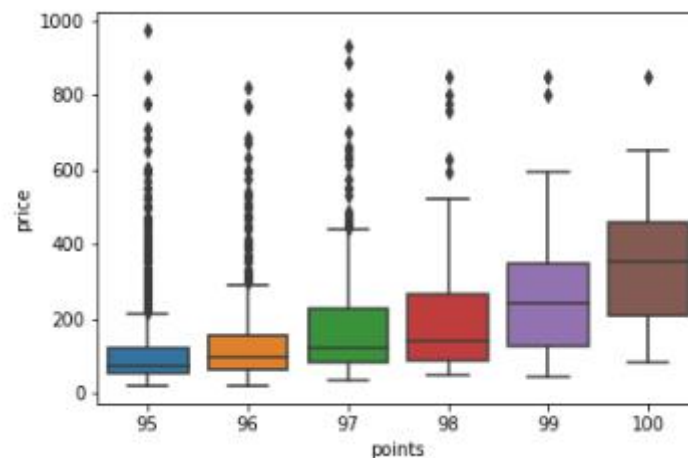
Now that you have a basic understanding of the Matplotlib, Pandas Visualization, and Seaborn syntax, I want to show you a few other graph types that are useful for extracting insides.

For most of them, Seaborn is the go-to library because of its high-level interface that allows for the creation of beautiful graphs in just a few lines of code.

Box plots:

A Box Plot is a graphical method of displaying the five-number summary. We can create box plots using seaborn's `sns.boxplot` method and passing it the data as well as the x and y column names.

```
df = wine_reviews[(wine_reviews['points']>=95) & (wine_reviews['price']<1000)]
sns.boxplot('points', 'price', data=df)
```



Box Plots, just like bar charts, are great for data with only a few categories but can get messy quickly.

Heatmap:

A Heatmap is a graphical representation of data where the individual values contained in a matrix are represented as colors. Heatmaps are perfect for exploring the correlation of features in a dataset.

To get the correlation of the features inside a dataset, we can call `<dataset>.corr()`, which is a Pandas dataframe method. This will give us the correlation matrix.

We can now use either Matplotlib or Seaborn to create the heatmap.

Matplotlib:

```
# get correlation matrix
```

```
corr = iris.corr()
```

```
fig, ax = plt.subplots()
```

```
# create heatmap
```

```
im = ax.imshow(corr.values)
```

```
# set labels
```

```
ax.set_xticks(np.arange(len(corr.columns)))
```

```
ax.set_yticks(np.arange(len(corr.columns)))
```

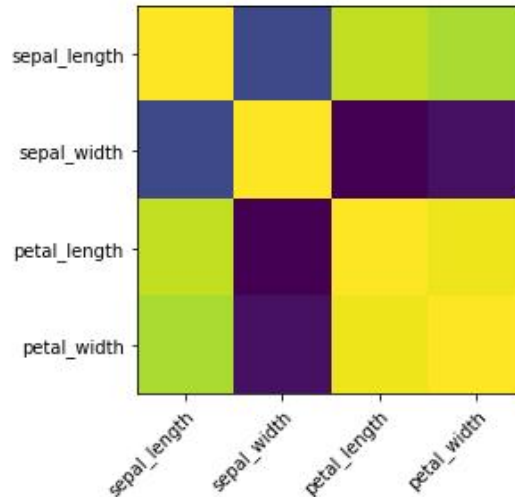
```
ax.set_xticklabels(corr.columns)
```

```
ax.set_yticklabels(corr.columns)
```

```
# Rotate the tick labels and set their alignment.
```

```
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
```

```
rotation_mode="anchor")
```



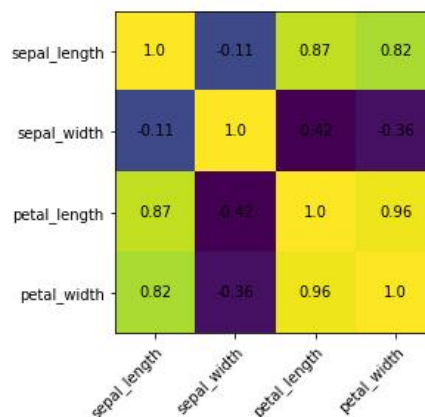
To add annotations to the heatmap, we need to add two for loops:

```
# get correlation matrix
corr = iris.corr()
fig, ax = plt.subplots()
# create heatmap
im = ax.imshow(corr.values)

# set labels
ax.set_xticks(np.arange(len(corr.columns)))
ax.set_yticks(np.arange(len(corr.columns)))
ax.set_xticklabels(corr.columns)
ax.set_yticklabels(corr.columns)

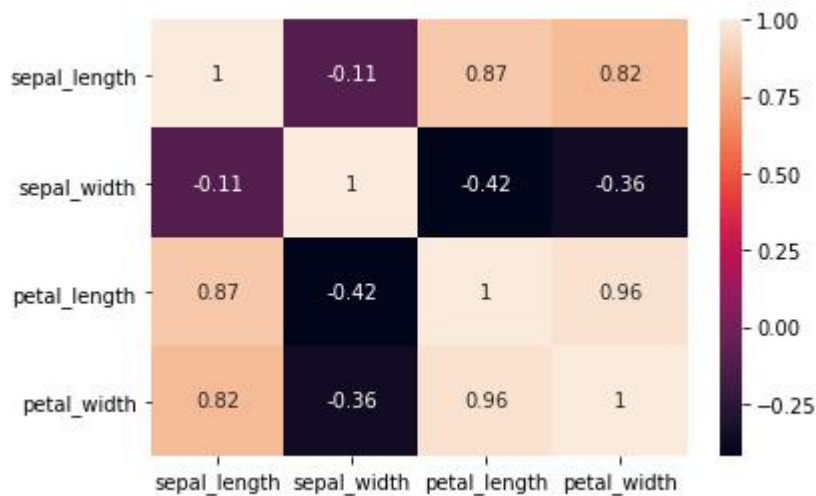
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
          rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(corr.columns)):
    for j in range(len(corr.columns)):
        text = ax.text(j, i, np.around(corr.iloc[i, j], decimals=2),
                        ha="center", va="center", color="black")
```



Seaborn makes it way easier to create a heatmap and add annotations:

```
sns.heatmap(iris.corr(), annot=True)
```



Faceting:

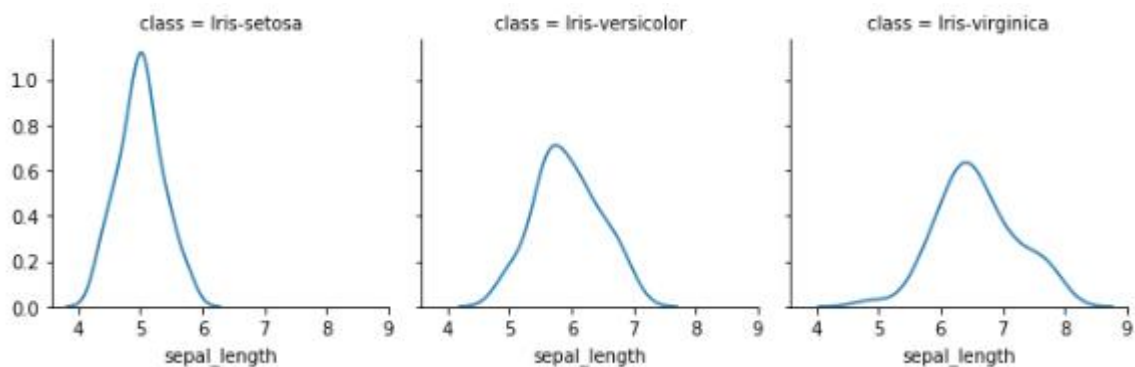
Faceting is the act of breaking data variables up across multiple subplots and combining those subplots into a single figure.

Faceting is helpful if you want to explore your dataset quickly.

To use one kind of faceting in Seaborn, we can use the FacetGrid. First of all, we need to define the FacetGrid and pass it our data as well as a row or column, which will be used to split the data. Then we need to call the map function on our FacetGrid object and define the plot type we want to use and the column we want to graph.

```
g = sns.FacetGrid(iris, col='class')
```

```
g = g.map(sns.kdeplot, 'sepal_length')
```

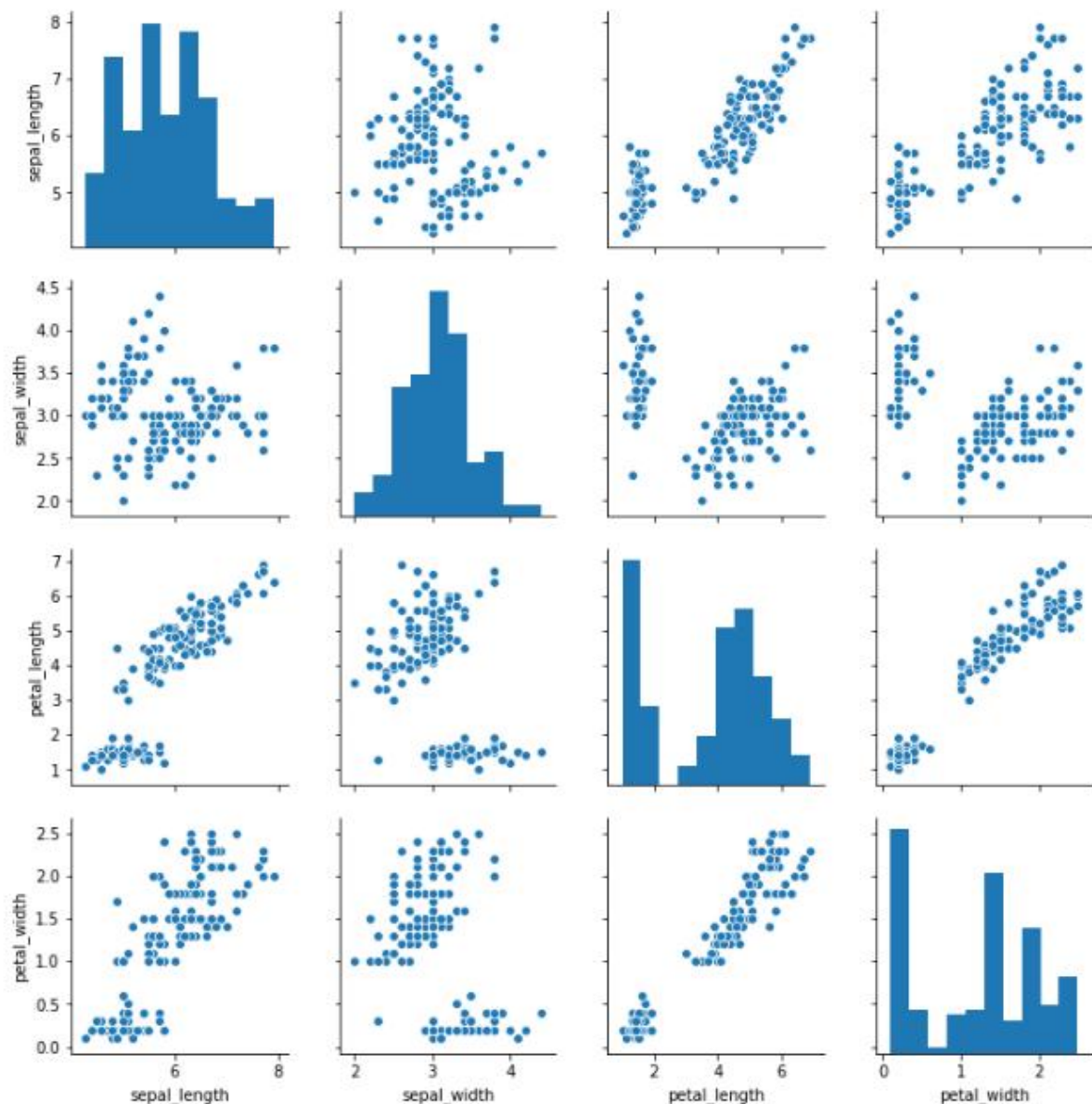


You can make plots bigger and more complicated than the example above. You can find a few examples [here](#).

Pairplot:

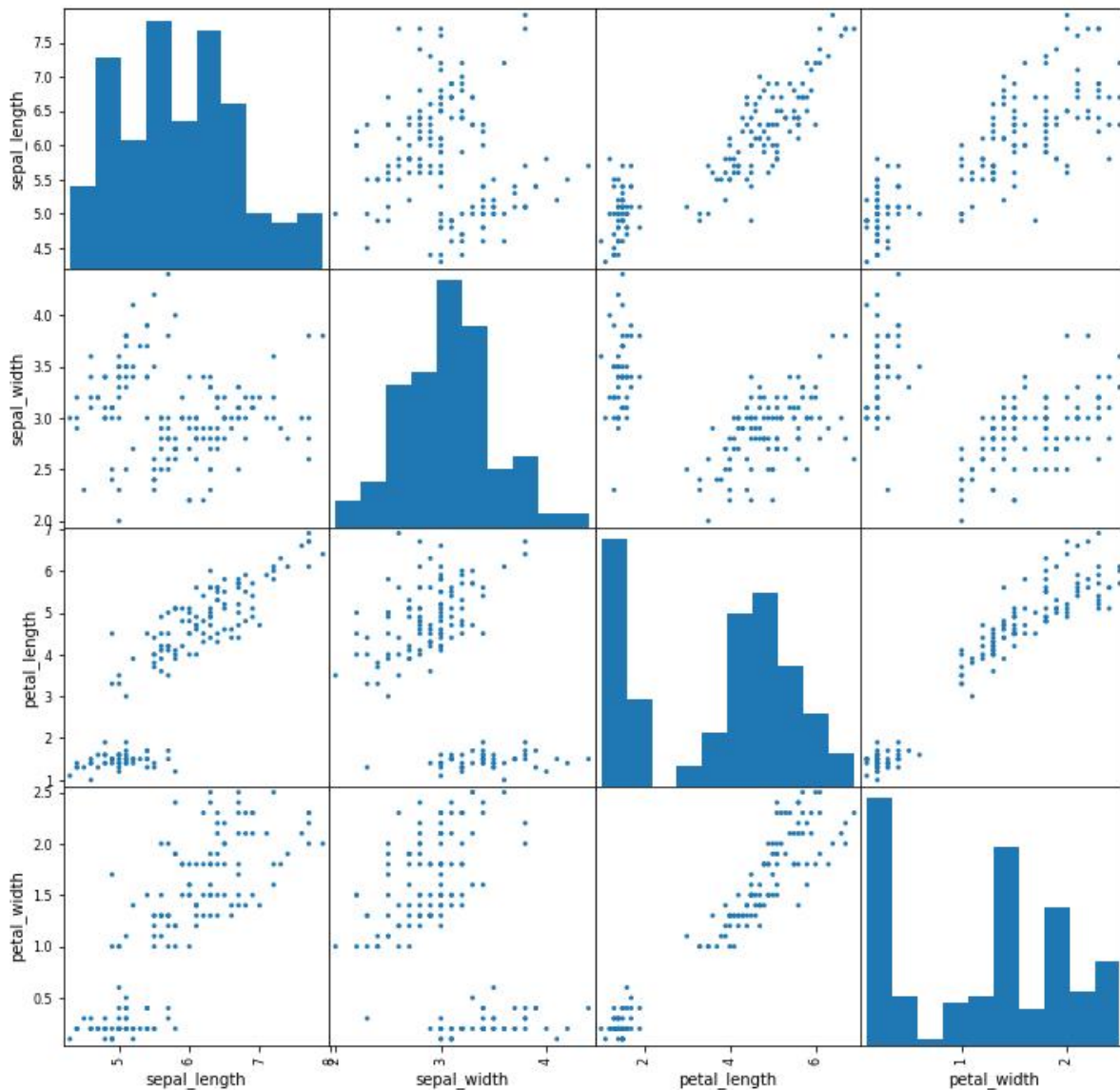
Lastly, I will show you Seaborns pairplot and Pandas scatter_matrix, which enable you to plot a grid of pairwise relationships in a dataset.

sns.pairplot(iris)



```
from pandas.plotting import scatter_matrix
```

```
fig, ax = plt.subplots(figsize=(12,12))  
scatter_matrix(iris, alpha=1, ax=ax)
```



As you can see in the images above, these techniques are always plotting two features with each other. The diagonal of the graph is filled with histograms, and the other plots are scatter plots.

CONCLUSION:

We have successfully visualized the data using basic plotting techniques in Python.

EXPERIMENT NUMBER – 4

AIM: To implement NoSQL Database Operations: CRUD operations using MongoDB.

DESCRIPTION:

1. Create Operations:

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

- i. **db.createCollection():** It is used to create an empty collection
- ii. **db.collection.insertOne():** It is used to insert a single document in the collection
- iii. **db.collection.insertMany():** It is used to insert a multiple documents in the collection

2. Read Operations:

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

- i. **db.collection.find():** It is used to retrieve documents from the collection.

3. Update Operations:

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

- i. **db.collection.updateOne():** It is used to update a single document in the collection that satisfy the given criteria.
- ii. **db.collection.updateMany():** It is used to update multiple documents in the collection that satisfy the given criteria.

4. Delete Operations:

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

- i. **db.collection.deleteOne():** It is used to delete a single document from the collection that satisfy the given criteria
- ii. **db.collection.deleteMany():** It is used to delete multiple documents from the collection that satisfy the given criteria.

QUERIES AND OUTPUTS:

1. Use:

Description: This command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax: use DATABASE_NAME

Query: use mongo_practice012

Output:

```
test> use mongo_practice012
switched to db mongo_practice012
```

2. db.createCollection():

Description: This command creates a new collection. This method is used primarily for creating new collections that use specific options

Syntax: db.createCollection(name, option)

Query: db.createCollection("movies")

Output:

```
mongo_practice012> db.createCollection("movies")
{ ok: 1 }
```

3. db.collection.insertOne():

Description: This command inserts a single document into a collection

Syntax: db.collection.insertOne(
 <document>,
 {
 writeConcern: <document>
 }
)

Query: db.movies.insertOne({
 title : "Fight Club",
 writer : "Chuck Palahniuk",
 year : "1999",
 actors : [
 "Brad Pitt",
 "Edward Norton"],
})

Output:

```
mongo_practice012> db.movies.insertOne({ title : "Fight Club", writer : "Chuck Palahniuk", year : "1999", actors : [ "Brad Pitt", "Edward Norton" ] })
{ acknowledged: true, insertedId: ObjectId("63e9c4ee0809224e32b88d81") }

mongo_practice012> db.movies.insertOne({title : "Pee Wee Herman's Big Adventure", titttitle : "Avatar"})
{ acknowledged: true, insertedId: ObjectId("63e9c9510809224e32b88d87") }
```

4. db.collection.insertMany():

Description: This command inserts multiple documents into a collection

Syntax: db.collection.insertMany(
[<document 1> , <document 2>, ...],
{
 writeConcern: <document>,
 ordered: <boolean>
}
)

Query: db.mongo_practice.insertMany([
{
 title : "The Hobbit: The Desolation of Smaug",
 writer : "J.R.R. Tolkein",
 year : 2013,
 franchise : "The Hobbit",
},
{
 title : "The Hobbit: The Battle of the Five Armies",
 writer : "J.R.R. Tolkein",
 year : 2012,
 franchise : "The Hobbit",
 synopsis : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness.",
},
],
)

Output:

```
mongo_practice012> db.movies.insertMany([ { title : "Pulp Fiction", writer : "Quentin Tarantino", year : 1994, actors : [ "John Travolta", "Uma Thurman" ] }, { title : "Inglorious Basterds", writer : "Quentin Tarantino", year : 2009, actors : [ "Brad Pitt", "Diane Kruger", "Eli Roth" ] }, { title : "The Hobbit: An Unexpected Journey", writer : "J.R.R. Tolkein", year : 2012, Franchise : "The Hobbit" } ] )
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63e9c7480809224e32b88d82"),
    '1': ObjectId("63e9c7480809224e32b88d83"),
    '2': ObjectId("63e9c7480809224e32b88d84")
  }
}
mongo_practice012> db.movies.insertMany([ { title : "The Hobbit: The Desolation of Smaug", writer : "J.R.R. Tolkein", year : 2013, franchise : "The Hobbit" }, { title : "The Hobbit: The Battle of the Five Armies", writer : "J.R.R. Tolkein", year : 2012, franchise : "The Hobbit", synopsis : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness." } ] )
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63e9c90f0809224e32b88d85"),
    '1': ObjectId("63e9c90f0809224e32b88d86")
  }
}
```


5. Write a query to display all the documents

Description: In this query, `getCollectionInfos()` command can be used. This command returns an array of documents with collection or view information, such as name and options, for the current database

Syntax: `db.getCollectionInfos(filter, nameOnly, authorizedCollections)`

Query: `db.getCollectionInfos()`

Output:

```
mongo_practice012> db.getCollectionInfos()
[
  {
    name: 'movies',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
      uuid: new UUID("55001607-7560-42f9-8417-d58dbc24fa6d")
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_' }
  }
]
```

6. Write a query to get all documents with writer set to "Quentin Tarantino" and restrict the result set to one object

Description: In this query, `db.collection.find()` command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: `db.collection.find(query, projection, options)`

Query: `db.movies.find({writer:"Quentin Tarantino"}).pretty().limit(1)`

Output:

```
mongo_practice012> db.movies.find({writer: "Quentin Tarantino"}).pretty().limit(1)
[
  {
    _id: ObjectId("63e9c7480809224e32b88d82"),
    title: 'Pulp Fiction',
    writer: 'Quentin Tarantino',
    year: 1994,
    actors: [ 'John Travolta', 'Uma Thurman' ]
  }
]
```

7. Write a query to get all movies released in the 1990s

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({year:{\$gt:"1990", \$lt:"2000"}})

Output:

```
mongo_practice012> db.movies.find({year: {$gt: 1990, $lt:2000}})
[
  {
    _id: ObjectId("63e9c4ee0809224e32b88d81"),
    title: 'Fight Club',
    writer: 'Chuck Palahniuk',
    year: 1999,
    actors: [ 'Brad Pitt', 'Edward Norton' ]
  },
  {
    _id: ObjectId("63e9c7480809224e32b88d82"),
    title: 'Pulp Fiction',
    writer: 'Quentin Tarantino',
    year: 1994,
    actors: [ 'John Travolta', 'Uma Thurman' ]
  }
]
```

8. Write a query to get all movies released before this year 2000 or after 2010

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({\$or:[{year:{\$gt:"2010"}},{year: {\$lt:"2000"}}]})

Output:

```
mongo_practice012> db.movies.find({$or: [{year:{$gt: 2010}}, {year: {$lt: 2000}}]})
[
  {
    _id: ObjectId("63e9c4ee0809224e32b88d81"),
    title: 'Fight Club',
    writer: 'Chuck Palahniuk',
    year: 1999,
    actors: [ 'Brad Pitt', 'Edward Norton' ]
  },
  {
    _id: ObjectId("63e9c7480809224e32b88d82"),
    title: 'Pulp Fiction',
    writer: 'Quentin Tarantino',
    year: 1994,
    actors: [ 'John Travolta', 'Uma Thurman' ]
  },
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    franchise: 'The Hobbit'
  },
  {
    _id: ObjectId("63e9c90f0809224e32b88d85"),
    title: 'The Hobbit: The Desolation of Smaug',
    writer: 'J.R.R. Tolkein',
    year: 2013,
    franchise: 'The Hobbit'
  },
  {
    _id: ObjectId("63e9c90f0809224e32b88d86"),
    title: 'The Hobbit: The Battle of the Five Armies',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    franchise: 'The Hobbit',
    synopsis: 'Bilbo and Company are forced to engage in a war against an array of co
mbatants and keep the Lonely Mountain from falling into the hands of a rising darkne
ss.'
  }
]
```

9. Write a query to find the objected for movie “Pulp Fiction”, add an actor named “Samuel L. Jackson” and display the result.

Description: In this query db.collection.update() command can be used. This command modifies an existing fields of an existing document or documents or replace an existing document entirely, depending on the update parameter

Syntax: db.collection.update(query, update, options)

Query:

```
db.movies.find({field:"value"}).pretty()
```

```
db.movies.update({_id:ObjectId(" ")}, {$push:{actors:"Samuel L. Jackson"}})
```

Output:

```
mongo_practice012> db.movies.find({title : "Pulp Fiction"}).pretty()
[
  {
    _id: ObjectId("63e9c7480809224e32b88d82"),
    title: 'Pulp Fiction',
    writer: 'Quentin Tarantino',
    year: 1994,
    actors: [ 'John Travolta', 'Uma Thurman' ]
  }
]
mongo_practice012> db.movies.update({_id:ObjectId("63e9c7480809224e32b88d82")}, {$push:{actors: "Samuel L. Jackson"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mongo_practice012> db.movies.find({title : "Pulp Fiction"}).pretty()
[
  {
    _id: ObjectId("63e9c7480809224e32b88d82"),
    title: 'Pulp Fiction',
    writer: 'Quentin Tarantino',
    year: 1994,
    actors: [ 'John Travolta', 'Uma Thurman', 'Samuel L. Jackson' ]
  }
]
```

10. Write a query add a synopsis to "The Hobbit: An Unexpected Journey" : "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."

Description: In this query db.collection.update() command can be used. This command modifies an existing fields of an existing document or documents or replace an existing document entirely, depending on the update parameter

Syntax: db.collection.update(query, update, options)

Query: db.movies.update({_id:ObjectId("5c9f98e5e5c2dfe9b3729bfe")}, {\$set:{synopsis:"A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})

Output:

```
mongo_practice012> db.movies.find({title : "The Hobbit: An Unexpected Journey"}).pretty()
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit'
  }
]
mongo_practice012> db.movies.update({_id: ObjectId("63e9c7480809224e32b88d84")}, {$set:{synopsis: "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mongo_practice012> db.movies.find({title : "The Hobbit: An Unexpected Journey"}).pretty()
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit',
    synopsis: 'A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug.'
  }
]
```

11. Write a query to find all movies that have a synopsis that contains the word “Bilbo”

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({synopsis:{\$regex: “Bilbo”}})

Output:

```
mongo_practice012> db.movies.find({synopsis:{$regex:"Bilbo"}})
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit',
    synopsis: 'A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug.'
  },
  {
    _id: ObjectId("63e9c90f0809224e32b88d86"),
    title: 'The Hobbit: The Battle of the Five Armies',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    franchise: 'The Hobbit',
    synopsis: 'Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness.'
  }
]
```

12. Write a query to find all movies that have a synopsis that contains the word “Gandalf”

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({synopsis: {\$regex: “Gandalf”}})

Output:

```
mongo_practice012> db.movies.find({synopsis:{$regex:"Gandalf"}})
```


13. Write a query to find all movies that have a synopsis that contains the word “Bilbo” and not the word “Gandalf”

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({\$and:[{synopsis:{\$regex:"Bilbo"}}, {synopsis:{\$not:/Gandalf/}}]})

Output:

```
mongo_practice012> db.movies.find({$and:[{synopsis:{$regex:"Bilbo"}}, {synopsis:{$not
$not:/Gandalf/}}]})
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit',
    synopsis: 'A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain wit
h a spirited group of dwarves to reclaim their mounntain home - and the gold within i
t - from the dragon Smaug.'
  },
  {
    _id: ObjectId("63e9c90f0809224e32b88d86"),
    title: 'The Hobbit: The Battle of the Five Armies',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    franchise: 'The Hobbit',
    synopsis: 'Bilbo and Company are forced to engage in a war against an array of co
mbatants and keep the Lonely Mountain from falling into the hands of a rising darkne
ss.'
  }
]
```

14. Write a query to find all movies that a synopsis that contains the word “dwarves” or “hobbit”

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Queries: db.movies.find({\$or:[{synopsis:{\$regex:"dwarves"}}, {synopsis:{\$regex:"hobbit"}}]})

Output:

```
mongo_practice012> db.movies.find({$or:[{synopsis:{$regex:"dwarves"}}, {synopsis:{$reg
ex:"hobbit"}}]})
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit',
    synopsis: 'A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain wit
h a spirited group of dwarves to reclaim their mounntain home - and the gold within i
t - from the dragon Smaug.'
  }
]
```

15. Write a query to find all movies that have a synopsis that contains the word “gold” and “dragon”

Description: In this query, db.collection.find() command can be used. This command selects documents in a collection or view and returns a cursor selected documents

Syntax: db.collection.find(query, projection, options)

Query: db.movies.find({\$and:[{synopsis:{\$regex:"gold"}}, {synopsis:{\$regex:"dragon"}}])

Output:

```
mongo_practice012> db.movies.find({$and:[{synopsis:{$regex:"gold"}},{synopsis:{$regex:"dragon"}}])
[
  {
    _id: ObjectId("63e9c7480809224e32b88d84"),
    title: 'The Hobbit: An Unexpected Journey',
    writer: 'J.R.R. Tolkein',
    year: 2012,
    Franchise: 'The Hobbit',
    synopsis: 'A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug.'
  }
]
```

16. Write a query to delete the movie “Fee Wee Herman’s Big Adventure”

Description: In this query, db.collection.deleteOne() command is used. This command is used to delete at most a single document that matches a specified filter.

Syntax:

```
db.collection.deleteOne(
  <filter>,
  {
    writeConcern: <document>,
    collation: <document>,
    hint: <document|string> // Available starting in MongoDB 4.4
  }
)
```

Query: db.movies.deleteOne({_id: ObjectId("")})

Output:

```
mongo_practice012> db.movies.deleteOne({_id: ObjectId("63e9c90f0809224e32b88d84")})
{ acknowledged: true, deletedCount: 0 }
```

17. Write a query to delete the movie "Avatar"

Description: In this query, `db.collection.deleteOne()` command is used. This command is used to delete at most a single document that matches a specified filter.

Syntax:

```
db.collection.deleteOne(  
<filter>,  
{  
  writeConcern: <document>,  
  collation: <document>,  
  hint: <document|string> // Available starting in MongoDB 4.4  
}  
)
```

Query: `db.movies.deleteOne({_id: ObjectId(" ")})`

Output:

```
mongo_practice012> db.movies.deleteOne({_id: ObjectId("63e9c9  
510809224e32b88d87")})  
{ acknowledged: true, deletedCount: 1 }
```

CONCLUSION:

We have successfully understood and executed the CRUD Operations on MongoDB

EXPERIMENT NUMBER – 5

AIM: To implement Functions Aggregation Operations in MongoDB

DESCRIPTION:

Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages.

The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

One of the most common use cases of Aggregation is to calculate aggregate values for groups of documents.

This is similar to the basic aggregation available in SQL with the GROUP BY clause and COUNT, SUM and AVG functions.

To perform aggregation operations, we can use:

- Aggregation pipelines, which are the preferred method for performing aggregations.

- Single purpose aggregation methods, which are simple but lack the capabilities of an aggregation pipeline.

MongoDB Aggregation Pipelines: The pipeline consists of the following stages

- Input-->\$match-->\$group-->\$sort-->output

- \$match stage – filters those documents we need to work with, those that fit our needs

- \$group stage – does the aggregation job

- \$sort stage – sorts the resulting documents the way we require (ascending or descending)

Syntax:

```
db.collectionName.aggregate(pipeline, options)
```

collectionName – is the name of a collection,

pipeline – is an array that contains the aggregation stages,

options – optional parameters for the aggregation

Example:

```
pipeline = [  
  { $match : { ... } },  
  { $group : { ... } },  
  { $sort : { ... } }  
]
```

QUERIES AND OUTPUT:

1. Create the Database Universities:

```
> use universities  
< 'switched to db universities'
```

```

> db.universities.insert([
  {
    country : 'Spain',
    city : 'Salamanca',
    name : 'USAL',
    location : {
      type : 'Point',
      coordinates : [ -5.6722512, 17, 40.9607792 ]
    },
    students : [
      { year : 2014, number : 24774 },
      { year : 2015, number : 23166 },
      { year : 2016, number : 21913 },
      { year : 2017, number : 21715 }
    ]
  },
  {
    country : 'Spain',
    city : 'Salamanca',
    name : 'UPSA',
    location : {
      type : 'Point',
      coordinates : [ -5.6691191, 17, 40.9631732 ]
    },
    students : [
      { year : 2014, number : 4788 },
      { year : 2015, number : 4821 },
      { year : 2016, number : 6550 },
      { year : 2017, number : 6125 }
    ]
  }
])

```

```

]
}
])
< 'DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.'
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63f30902edb08840727517ea"),
    '1': ObjectId("63f30902edb08840727517eb")
  }
}
> db.courses.insert([
  {
    university : 'USAL',
    name : 'Computer Science',
    level : 'Excellent'
  },
  {
    university : 'USAL',
    name : 'Electronics',
    level : 'Intermediate'
  },
  {
    university : 'USAL',
    name : 'Communication',
    level : 'Excellent'
  }
])
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63f30a46edb08840727517ed"),
    '1': ObjectId("63f30a46edb08840727517ee"),
    '2': ObjectId("63f30a46edb08840727517ef")
  }
}

```

2. MongoDB \$match:

Description: This command filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

Syntax: { \$match: { <query> } }

Query: db.universities.aggregate([{ \$match : { country : 'Spain', city : 'Salamanca' } }]).pretty()

Output:

```
> db.universities.aggregate([
  { $match : { country : 'Spain', city : 'Salamanca' } }
]).pretty()
< {
  _id: ObjectId("63f30902edb08840727517ea"),
  country: 'Spain',
  city: 'Salamanca',
  name: 'USAL',
  location: {
    type: 'Point',
    coordinates: [
      -5.6722512,
      17,
      40.9607792
    ]
  },
  students: [
    {
      year: 2014,
      number: 24774
    },
    {
      year: 2015,
      number: 23166
    },
    {
      year: 2016,
      number: 21913
    },
  ],
}
```

```
{
  _id: ObjectId("63f30902edb08840727517eb"),
  country: 'Spain',
  city: 'Salamanca',
  name: 'UPSA',
  location: {
    type: 'Point',
    coordinates: [
      -5.6691191,
      17,
      40.9631732
    ]
  },
  students: [
    {
      year: 2014,
      number: 4788
    },
    {
      year: 2015,
      number: 4821
    },
  ],
}
```

3. MongoDB \$project:

Description: This command passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

Syntax: { \$project: { <specification(s)> } }

Query: db.universities.aggregate([{ \$project : { _id : 0, country : 1, city : 1, name : 1 } }]).pretty()

Output:

```
> db.universities.aggregate([
  { $project : { _id : 0, country : 1, city : 1, name : 1 } }
]).pretty()
< {
  country: 'Spain',
  city: 'Salamanca',
  name: 'USAL'
}
{
  country: 'Spain',
  city: 'Salamanca',
  name: 'UPSA'
}
```

4. MongoDB \$group:

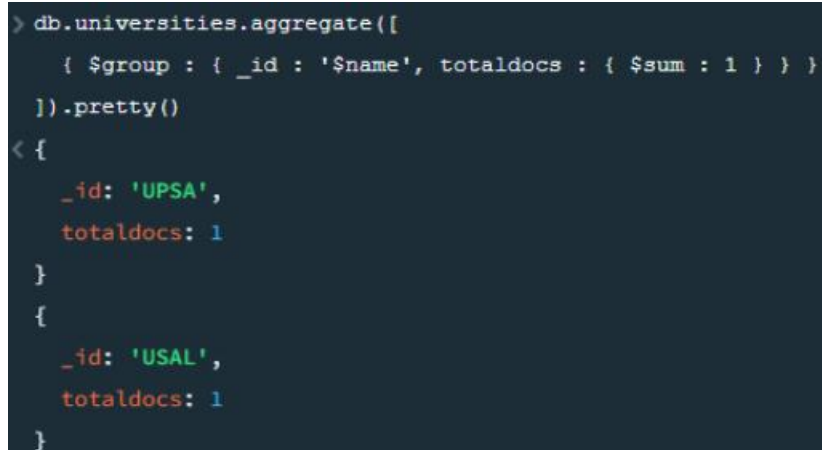
Description: The \$group stage separates documents into groups according to a “group key”. The output is one document for each unique group key

Syntax:

```
{
  $group:
  {
    _id: <expression>, // Group key
    <field1>: { <accumulator1> : <expression1> },
    ...
  }
}
```

Query: db.universities.aggregate([{ \$group : { _id : '\$name', totaldocs : { \$sum : 1 } } }]).pretty()

Output:



```
> db.universities.aggregate([
  { $group : { _id : '$name', totaldocs : { $sum : 1 } } }
]).pretty()
< {
  _id: 'UPSA',
  totaldocs: 1
}
{
  _id: 'USAL',
  totaldocs: 1
}
```

CONCLUSION:

We have successfully understood and executed the Aggregation Operations on MongoDB

EXPERIMENT NUMBER – 6

AIM: To implement word count / frequency programs using MapReduce.

DESCRIPTION:

In MapReduce word count example, we find out the frequency of each word. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

PROGRAM:

mapper.py:

```
#!/usr/bin/python
```

```
# import sys because we need to read and write data to STDIN and STDOUT
import sys
```

```
# reading entire line from STDIN (standard input)
for line in sys.stdin:
    # to remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()

    # we are looping over the words array and printing the word
    # with the count of 1 to the STDOUT
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        print('%s\t%s' % (word, 1))
```

reducer.py:

```
#!/usr/bin/python
```

```
from operator import itemgetter
import sys
```

```
current_word = None
current_count = 0
word = None
```

```
# read the entire line from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # splitting the data on the basis of tab we have provided in mapper.py
    word, count = line.split('\t', 1)
```

```

# convert count (currently a string) to int
try:
    count = int(count)
except ValueError:
    # count was not a number, so silently
    # ignore/discard this line
    continue

# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
    current_count += count
else:
    if current_word:
        # write result to STDOUT
        print('%s\t%s' % (current_word, current_count))
    current_count = count
    current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))

```

word_count_data.txt:

```

hello world tensorflow
mapreduce wordcount mapreduce tensorflow

```

OUTPUT:

```

nikki@nikki-VirtualBox:~$ chmod +x mapper.py
nikki@nikki-VirtualBox:~$ chmod +x reducer.py
nikki@nikki-VirtualBox:~$ cat word_count_data.txt | python3 mapper.py
hello      1
world      1
tensorflow      1
mapreduce      1
wordcount      1
mapreduce      1
tensorflow      1
nikki@nikki-VirtualBox:~$ cat word_count_data.txt | python3 mapper.py | sort -k1
,1 | python3 reducer.py
hello      1
mapreduce      2
tensorflow      2
wordcount      1
world      1

```

CONCLUSION:

We have successfully implemented word count / frequency programs using MapReduce.

EXPERIMENT NUMBER – 7

AIM: To implement a program that processes a dataset using Pyspark.

DESCRIPTION:

Processing a dataset using PySpark involves using the PySpark library, which is an open-source framework for distributed data processing, to perform various data manipulation and analysis tasks. PySpark is particularly well-suited for handling large-scale datasets and can process data in a distributed, parallelized manner across a cluster of machines.

CODE AND EXECUTION:

pip install pyspark

```
collecting pyspark
Downloading pyspark-3.5.0.tar.gz (316.9 MB)
316.9/316.9 MB 4.4 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
Building wheel for pyspark (setup.py) ... done
Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=ced9af2a93530fef27b65d55b9a8d129478d7a75d1f636ab14dae67186cb2739
Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('Dataprocessing').getOrCreate()
df=spark.read.csv('cruise_ship_info.csv',inferSchema=True,header=True)
df.show(5)
```

Ship_name	Cruise_line	Age	Tonnage	passengers	length	cabins	passenger_density	crew
Journey	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55
Quest	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55
Celebration	Carnival	26	47.262	14.86	7.22	7.43	31.8	6.7
Conquest	Carnival	11	110.0	29.74	9.53	14.88	36.99	19.1
Destiny	Carnival	17	101.353	26.42	8.92	13.21	38.36	10.0

only showing top 5 rows

df.printSchema()

```
root
|-- Ship_name: string (nullable = true)
|-- Cruise_line: string (nullable = true)
|-- Age: integer (nullable = true)
|-- Tonnage: double (nullable = true)
|-- passengers: double (nullable = true)
|-- length: double (nullable = true)
|-- cabins: double (nullable = true)
|-- passenger_density: double (nullable = true)
|-- crew: double (nullable = true)
```



```
#label Encoding
indexers=StringIndexer(inputCol='Cruise_line',outputCol='id')
output=indexers.fit(df).transform(df)
output.show(5)
```

Ship_name	Cruise_line	Age	Tonnage	passengers	length	cabins	passenger_density	crew	id
Journey	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0
Quest	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0
Celebration	Carnival	26	47.262	14.86	7.22	7.43	31.8	6.7	1.0
Conquest	Carnival	11	110.0	29.74	9.53	14.88	36.99	19.1	1.0
Destiny	Carnival	17	101.353	26.42	8.92	13.21	38.36	10.0	1.0

only showing top 5 rows

```
#Feature Processing
assembler=VectorAssembler(inputCols=['Age',
'Tonnage',
'passengers',
'length',
'cabins',
'passenger_density'],outputCol='features')
output=assembler.transform(output)
output.show(5)
```

Ship_name	Cruise_line	Age	Tonnage	passengers	length	cabins	passenger_density	crew	id	features
Journey	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0	[6.0,30.276999999...
Quest	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0	[6.0,30.276999999...
Celebration	Carnival	26	47.262	14.86	7.22	7.43	31.8	6.7	1.0	[26.0,47.262,14.8...
Conquest	Carnival	11	110.0	29.74	9.53	14.88	36.99	19.1	1.0	[11.0,110.0,29.74...
Destiny	Carnival	17	101.353	26.42	8.92	13.21	38.36	10.0	1.0	[17.0,101.353,26...

only showing top 5 rows

```
#normalization based on mean and standard deviation
scaler = StandardScaler(inputCol="features", outputCol="scaled_feature", withStd=True,
withMean=True)
output=scaler.fit(output).transform(output)
#final preprocessed output
output.show(5)
```

Ship_name	Cruise_line	Age	Tonnage	passengers	length	cabins	passenger_density	crew	id	features	scaled_feature
Journey	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0	[6.0,30.276999999...	[-1.2723564208380...
Quest	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55	16.0	[6.0,30.276999999...	[-1.2723564208380...
Celebration	Carnival	26	47.262	14.86	7.22	7.43	31.8	6.7	1.0	[26.0,47.262,14.8...	[1.35380052876893...
Conquest	Carnival	11	110.0	29.74	9.53	14.88	36.99	19.1	1.0	[11.0,110.0,29.74...	[-0.6158171834363...
Destiny	Carnival	17	101.353	26.42	8.92	13.21	38.36	10.0	1.0	[17.0,101.353,26...	[0.17202990144577...

only showing top 5 rows

CONCLUSION:

We have successfully processed the dataset using PySpark.

EXPERIMENT NUMBER – 8

AIM: To implement Linear Regression using SPARK

DESCRIPTION:

Linear regression is also a type of machine-learning algorithm more specifically a supervised machine-learning algorithm that learns from the labeled datasets and maps the data points to the most optimized linear functions. which can be used for prediction on new datasets.

Problem Statement: Build a predictive Model for the shipping company, to find an estimate of how many Crew members a ship requires. The dataset contains 159 instances with 9 features.

The Description of dataset is as below:

- Ship Name
- Cruise Line
- Age (as of 2013)
- Tonnage (1000s of tons)
- passengers (100s)
- Length (100s of feet)
- Cabins (100s)
- Passenger Density
- Crew (100s)

CODE AND EXECUTION:

pip install pyspark

```
collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
  316.9/316.9 MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=a8c0a7209ff91c7fc546c80d68e45b217be8ce6d1dbe761ffff44a8af89734f79
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('housing_price_model').getOrCreate()
df=spark.read.csv('cruise_ship_info.csv',inferSchema=True,header=True)
df.show(10)
```

Ship_name	Cruise_line	Age	Tonnage	passengers	length	cabins	passenger_density	crew
Journey	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55
Quest	Azamara	6	30.276999999999997	6.94	5.94	3.55	42.64	3.55
Celebration	Carnival	26	47.262	14.86	7.22	7.43	31.8	6.7
Conquest	Carnival	11	110.0	29.74	9.53	14.88	36.99	19.1
Destiny	Carnival	17	101.353	26.42	8.92	13.21	38.36	10.0
Ecstasy	Carnival	22	70.367	20.52	8.55	10.2	34.29	9.2
Elation	Carnival	15	70.367	20.52	8.55	10.2	34.29	9.2
Fantasy	Carnival	23	70.367	20.56	8.55	10.22	34.23	9.2
Fascination	Carnival	19	70.367	20.52	8.55	10.2	34.29	9.2
Freedom	Carnival	6	110.23899999999999	37.0	9.51	14.87	29.79	11.5

only showing top 10 rows

```
df.printSchema()
```

```
root
|-- Ship_name: string (nullable = true)
|-- Cruise_line: string (nullable = true)
|-- Age: integer (nullable = true)
|-- Tonnage: double (nullable = true)
|-- passengers: double (nullable = true)
|-- length: double (nullable = true)
|-- cabins: double (nullable = true)
|-- passenger_density: double (nullable = true)
|-- crew: double (nullable = true)
```

```
df.columns
```

```
['Ship_name',
 'Cruise_line',
 'Age',
 'Tonnage',
 'passengers',
 'length',
 'cabins',
 'passenger_density',
 'crew']
```

```

from pyspark.ml.feature import StringIndexer
indexer=StringIndexer(inputCol='Cruise_line',outputCol='cruise_cat')
indexed=indexer.fit(df).transform(df)
for item in indexed.head(5):
    print(item)
    print("\n")

```

```
Row(Ship_name='Journey', Cruise_line='Azamara', Age=6, Tonnage=30.276999999999997, passengers=6.94, length=5.94, cabins=3.55, passenger_density=42.64, crew=3.55, cruise_cat=16.0)
```

```
Row(Ship_name='Quest', Cruise_line='Azamara', Age=6, Tonnage=30.276999999999997, passengers=6.94, length=5.94, cabins=3.55, passenger_density=42.64, crew=3.55, cruise_cat=16.0)
```

```
Row(Ship_name='Celebration', Cruise_line='Carnival', Age=26, Tonnage=47.262, passengers=14.86, length=7.22, cabins=7.43, passenger_density=31.8, crew=6.7, cruise_cat=1.0)
```

```
Row(Ship_name='Conquest', Cruise_line='Carnival', Age=11, Tonnage=110.0, passengers=29.74, length=9.53, cabins=14.88, passenger_density=36.99, crew=19.1, cruise_cat=1.0)
```

```
Row(Ship_name='Destiny', Cruise_line='Carnival', Age=17, Tonnage=101.353, passengers=26.42, length=8.92, cabins=13.21, passenger_density=38.36, crew=10.0, cruise_cat=1.0)
```

```

from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
assembler=VectorAssembler(inputCols=['Age',
    'Tonnage',
    'passengers',
    'length',
    'cabins',
    'passenger_density',
    'cruise_cat'],outputCol='features')
output=assembler.transform(indexed)
output.select('features','crew').show(5)

```

```

+-----+-----+
|          features | crew |
+-----+-----+
|[6.0,30.276999999... | 3.55 |
|[6.0,30.276999999... | 3.55 |
|[26.0,47.262,14.8... | 6.7 |
|[11.0,110.0,29.74... | 19.1 |
|[17.0,101.353,26.... | 10.0 |
+-----+-----+
only showing top 5 rows

```

```

final_data=output.select('features','crew')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()

```

```

+-----+-----+
|summary|          crew |
+-----+-----+
|  count|             105 |
|   mean|7.950761904761916 |
| stddev|3.504040441250561 |
|    min|              0.59 |
|    max|             21.0 |
+-----+-----+

```

```
test_data.describe().show()
```

```
+-----+-----+
|summary|          crew|
+-----+-----+
|  count|           53|
|   mean|  7.483962264150944|
| stddev| 3.5149834990435767|
|    min|           0.59|
|    max|           19.1|
+-----+-----+
```

```
from pyspark.ml.regression import LinearRegression
ship_lr=LinearRegression(featuresCol='features',labelCol='crew')
trained_ship_model=ship_lr.fit(train_data)
ship_results=trained_ship_model.evaluate(train_data)
print('Rsquared Error :',ship_results.r2)
```

```
Rsquared Error : 0.9481847866542444
```

```
unlabeled_data=test_data.select('features')
unlabeled_data.show(5)
```

```
+-----+
|          features|
+-----+
|[5.0,86.0,21.04,9...|
|[5.0,115.0,35.74,...|
|[5.0,122.0,28.5,1...|
|[5.0,160.0,36.34,...|
|[6.0,30.276999999...|
+-----+
only showing top 5 rows
```

```
predictions=trained_ship_model.transform(unlabeled_data)
predictions.show()
```

```
+-----+-----+
|          features          | prediction |
+-----+-----+
|[5.0,86.0,21.04,9...| 9.278710202707135|
|[5.0,115.0,35.74,...| 11.76297202430732|
|[5.0,122.0,28.5,1...| 6.097929011890256|
|[5.0,160.0,36.34,...| 14.999343148777454|
|[6.0,30.276999999...| 4.502605773133349|
|[6.0,90.0,20.0,9....| 10.25952060536163|
|[6.0,110.23899999...| 10.872193239369155|
|[9.0,105.0,27.2,8...| 11.238882153319183|
|[9.0,110.0,29.74,...| 12.042083055746394|
|[10.0,46.0,7.0,6....| 2.8333565182617333|
|[10.0,68.0,10.8,7...| 6.716896911843758|
|[10.0,90.09,25.01...| 8.775263533366306|
|[10.0,105.0,27.2,...| 11.226614045747292|
|[11.0,91.0,20.32,...| 9.274239553427998|
|[11.0,110.0,29.74...| 12.03114238409525|
|[12.0,42.0,14.8,7...| 6.792692274315869|
|[12.0,88.5,21.24,...| 9.46980994116172|
|[12.0,88.5,21.24,...| 10.407256080350553|
|[12.0,91.0,20.32,...| 9.261971445856107|
|[12.0,91.0,20.32,...| 9.261971445856107|
+-----+-----+
only showing top 20 rows
```

CONCLUSION:

We have successfully implemented Linear Regression using SPARK.

EXPERIMENT NUMBER – 9

AIM: To implement Naïve Bayes Classification techniques using SPARK

DESCRIPTION:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.

CODE AND EXECUTION:

pip install pyspark

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
    316.9/316.9 MB 4.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=c9559224f356dc0116f6841bcd9f00a6afc1561be90879fe5c83e946b08e41d2
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

#naive bayes

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Read data from the vehicle_stolen_dataset.csv
spark=SparkSession.builder.appName("bayesclass").getOrCreate()
data=spark.read.csv('vehicle_stolen_dataset.csv',inferSchema=True)
data.show()
```

```
+-----+-----+-----+-----+-----+
|_c0|_c1|_c2|_c3|_c4|
+-----+-----+-----+-----+-----+
|N001|BMW|black|night|yes|
|N002|Audi|black|night|no|
|N003|NISSAN|black|night|yes|
|N004|VEGA|red|day|yes|
|N005|BMW|blue|day|no|
|N006|Audi|black|day|yes|
|N007|VEGA|red|night|no|
|N008|Audi|blue|day|yes|
|N009|VEGA|black|day|yes|
|N010|NISSAN|blue|day|no|
|N011|BMW|black|night|yes|
|N012|NISSAN|red|day|no|
|N013|VEGA|black|night|yes|
|N014|BMW|red|day|no|
|N015|Audi|black|day|yes|
|N016|Audi|blue|night|yes|
|N017|Audi|red|day|no|
|N018|NISSAN|black|day|yes|
|N019|BMW|blue|day|yes|
|N020|BMW|red|night|yes|
+-----+-----+-----+-----+-----+
```

```
data.columns
```

```
['_c0', '_c1', '_c2', '_c3', '_c4']
```

```
vehicle_df = data.select(col("_c0").alias("number_plate"), col("_c1").alias("brand"),
col("_c2").alias("color"),
col("_c3").alias("time"),
col("_c4").alias("stoled"))
indexers = [
StringIndexer(inputCol="brand", outputCol = "brand_index"),
StringIndexer(inputCol="color", outputCol = "color_index"), StringIndexer(inputCol="time",
outputCol = "time_index"), StringIndexer(inputCol="stoled", outputCol = "label")]
pipeline = Pipeline(stages=indexers)
#Fitting a model to the input dataset.
indexed_vehicle_df = pipeline.fit(vehicle_df).transform(vehicle_df)
indexed_vehicle_df.show(5,False)
```

number_plate	brand	color	time	stoled	brand_index	color_index	time_index	label
N001	BMW	black	night	yes	1.0	0.0	1.0	0.0
N002	Audi	black	night	no	0.0	0.0	1.0	1.0
N003	NISSAN	black	night	yes	2.0	0.0	1.0	0.0
N004	VEGA	red	day	yes	3.0	1.0	0.0	0.0
N005	BMW	blue	day	no	1.0	2.0	0.0	1.0

only showing top 5 rows

```
vectorAssembler = VectorAssembler(inputCols = ["brand_index", "color_index",
"time_index"],outputCol = "features")
vindexed_vehicle_df = vectorAssembler.transform(indexed_vehicle_df)
vindexed_vehicle_df.show(5, False)
```

number_plate	brand	color	time	stoled	brand_index	color_index	time_index	label	features
N001	BMW	black	night	yes	1.0	0.0	1.0	0.0	[1.0,0.0,1.0]
N002	Audi	black	night	no	0.0	0.0	1.0	1.0	[0.0,0.0,1.0]
N003	NISSAN	black	night	yes	2.0	0.0	1.0	0.0	[2.0,0.0,1.0]
N004	VEGA	red	day	yes	3.0	1.0	0.0	0.0	[3.0,1.0,0.0]
N005	BMW	blue	day	no	1.0	2.0	0.0	1.0	[1.0,2.0,0.0]

only showing top 5 rows

```
indexed_vehicle_df.show(3)
```

number_plate	brand	color	time	stoled	brand_index	color_index	time_index	label
N001	BMW	black	night	yes	1.0	0.0	1.0	0.0
N002	Audi	black	night	no	0.0	0.0	1.0	1.0
N003	NISSAN	black	night	yes	2.0	0.0	1.0	0.0

only showing top 3 rows

```

splits = vindexed_vehicle_df.randomSplit([0.6,0.4], 42)
# optional value 42 is seed for sampling
train_df = splits[0]
test_df = splits[1]
nb = NaiveBayes(modelType="multinomial")
nbmodel = nb.fit(train_df)
predictions_df = nbmodel.transform(test_df)
predictions_df.show(5, True)

```

number_plate	brand	color	time	stolen	brand_index	color_index	time_index	label	features	rawPrediction	probability	prediction
N001	BMW	black	night	yes	1.0	0.0	1.0	0.0	[1.0,0.0,1.0]	[-2.8415815937267...	[0.70850202429149...	0.0
N003	NISSAN	black	night	yes	2.0	0.0	1.0	0.0	[2.0,0.0,1.0]	[-3.5347287742866...	[0.85868498527968...	0.0
N005	BMW	blue	day	no	1.0	2.0	0.0	1.0	[1.0,2.0,0.0]	[-3.2470467018348...	[0.80201649862511...	0.0
N007	VEGA	red	night	no	3.0	1.0	1.0	1.0	[3.0,1.0,1.0]	[-5.3264882435147...	[0.92678896750413...	0.0
N009	VEGA	black	day	yes	3.0	0.0	0.0	0.0	[3.0,0.0,0.0]	[-2.4361164856185...	[0.97330367074527...	0.0

only showing top 5 rows

```

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
metricName="accuracy")
nbaccuracy = evaluator.evaluate(predictions_df)
print("Test accuracy = " + str(nbaccuracy))

```

Test accuracy = 0.5

CONCLUSION:

We have successfully implemented Naïve Bayes Classification using PySpark.

EXPERIMENT NUMBER – 10

AIM: To implement clustering techniques using Spark

DESCRIPTION:

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Here, we will be implementing two different types of clustering techniques:

1. **Density-Based Spatial Clustering Of Applications With Noise (DBSCAN):** Clusters are dense regions in the data space, separated by regions of the lower density of points. The *DBSCAN algorithm* is based on this intuitive notion of “clusters” and “noise”. The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.
2. **K-means Clustering:** K-means is a clustering algorithm that groups data points into K distinct clusters based on their similarity. It is an unsupervised learning technique that is widely used in data mining, machine learning, and pattern recognition. The algorithm works by iteratively assigning data points to a cluster based on their distance from the cluster’s centroid and then recomputing the centroid of each cluster. The process continues until the clusters’ centroids converge or a maximum number of iterations is reached.

CODE AND EXECUTION:

pip install pyspark

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
    316.9/316.9 MB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=0d082275199321030f85ad035adfab0a04dd39900257fc84d75a2af64f514b28
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

1. DBSCAN Clustering Technique:

#DBSCAN

import pyspark

from pyspark.sql import SparkSession

spark=SparkSession.builder.appName('testmodel').getOrCreate()

df=spark.read.csv('irisC.csv',inferSchema=True,header=True)

df.show(10)

```
+-----+-----+-----+
|Sepal Length|Sepal Width|Species|
+-----+-----+-----+
|          5.1|          3.5|Iris-setosa|
|          4.9|          3.0|Iris-setosa|
|          4.7|          3.2|Iris-setosa|
|          4.6|          3.1|Iris-setosa|
|          5.0|          3.6|Iris-setosa|
|          5.4|          3.9|Iris-setosa|
|          4.6|          3.4|Iris-setosa|
|          5.0|          3.4|Iris-setosa|
|          4.4|          2.9|Iris-setosa|
|          4.9|          3.1|Iris-setosa|
+-----+-----+-----+
only showing top 10 rows
```

df.printSchema()

```
root
 |-- Sepal Length: double (nullable = true)
 |-- Sepal Width: double (nullable = true)
 |-- Species: string (nullable = true)
```

from pyspark.ml.linalg import Vectors

from pyspark.ml.feature import VectorAssembler

assembler=VectorAssembler(inputCols=['Sepal Length','Sepal Width'],outputCol='features')

output=assembler.transform(df)

output.select('features').show(5)

```
+-----+
| features|
+-----+
|[5.1,3.5]|
|[4.9,3.0]|
|[4.7,3.2]|
|[4.6,3.1]|
|[5.0,3.6]|
+-----+
only showing top 5 rows
```

```
final_data=output.select('features','Species')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()
```

summary	Species
count	104
mean	NULL
stddev	NULL
min	Iris-setosa
max	Iris-virginica

```
import numpy as np
np.array(final_data.select('features'))

array(DataFrame[features: vector], dtype=object)

pandas_df = final_data.toPandas()
pandas_df.head()
```

	features	Species
0	[5.1, 3.5]	Iris-setosa
1	[4.9, 3.0]	Iris-setosa
2	[4.7, 3.2]	Iris-setosa
3	[4.6, 3.1]	Iris-setosa
4	[5.0, 3.6]	Iris-setosa

```

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.6, min_samples=3)
dbscan_labels = dbscan.fit_predict(pandas_df['features'].tolist())
print(dbscan_labels)

```

```

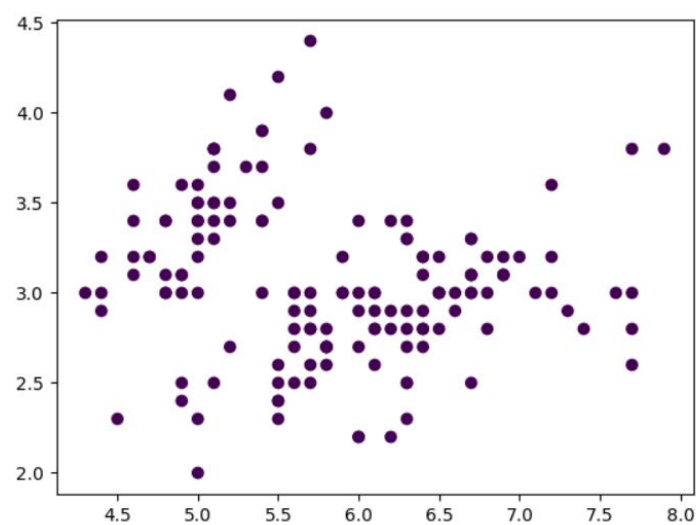
array([[5.1],
       [4.9],
       [4.7],
       [4.6],
       [5. ],
       [5.4],
       [4.6],
       [5. ],
       [4.4],
       [4.9],
       [5.4],
       [4.8],
       [4.8],
       [4.3],
       [5.8],
       [5.7],
       [5.4],
       [5.1],
       [5.7],
       [5.1],
       [5.4],
       [5.1],
       [4.6],
       [5.1],
       [4.8],
       [5. ],
       [5. ],
       [5.2],
       [5.2],

```

```

import matplotlib.pyplot as plt
plt.scatter(np.array(X),np.array(Y),c=dbscan_labels)
plt.show()

```



```
spark.stop()
```

1. K-Means Clustering Technique:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('cluster').getOrCreate()
print('Spark Version: {}'.format(spark.version))
```

```
Spark Version: 3.5.0
```

#Loading the data

```
dataset = spark.read.csv("seeds_dataset.csv",header=True,inferSchema=True)
```

#show the data in the above file using the below command

```
dataset.show(5)
```

```
+-----+-----+-----+-----+-----+-----+-----+
| Area|Perimeter|Compactness|Length_of_kernel|Width_of_kernel|Asymmetry_coefficient|Length_of_kernel_grove|
+-----+-----+-----+-----+-----+-----+-----+
|15.26| 14.84| 0.871| 5.763| 3.312| 2.221| 5.22|
|14.88| 14.57| 0.8811| 5.554| 3.333| 1.018| 4.956|
|14.29| 14.09| 0.905| 5.291| 3.337| 2.699| 4.825|
|13.84| 13.94| 0.8955| 5.324| 3.379| 2.259| 4.805|
|16.14| 14.99| 0.9034| 5.658| 3.562| 1.355| 5.175|
+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 5 rows

#Print schema

```
dataset.printSchema()
```

```
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_grove: double (nullable = true)
```

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

```
vec_assembler = VectorAssembler(inputCols = dataset.columns, outputCol='features')
```

```
final_data = vec_assembler.transform(dataset)
```

```
final_data.select('features').show(5)
```

```
+-----+
|          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)
# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
# Normalize each feature to have unit standard deviation.
final_data = scalerModel.transform(final_data)
final_data.select('scaledFeatures').show(5)

```

```

+-----+
| scaledFeatures |
+-----+
|[ 5.24452795332028... |
|[ 5.11393027165175... |
|[ 4.91116018695588... |
|[ 4.75650503761158... |
|[ 5.54696468981581... |
+-----+
only showing top 5 rows

```

```

#Importing the model
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)
    predictions=model.transform(final_data)
    score=evaluator.evaluate(predictions)
    silhouette_score.append(score)
    print('Silhouette Score for k =',i,'is',score)

```

```

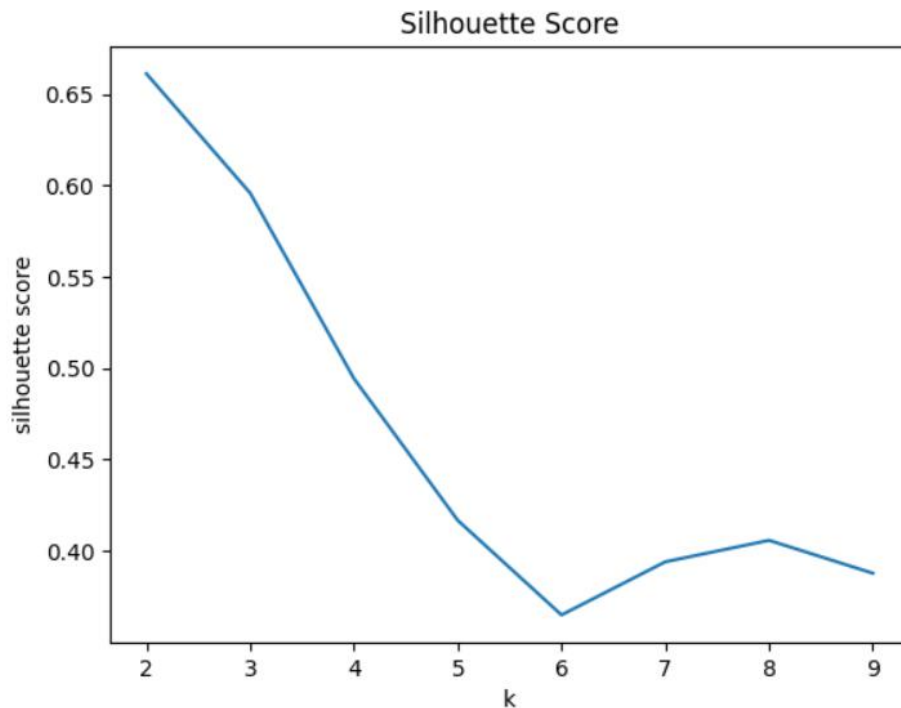
Silhouette Score for k = 2 is 0.6613125038335929
Silhouette Score for k = 3 is 0.5959078263451633
Silhouette Score for k = 4 is 0.4943210687863144
Silhouette Score for k = 5 is 0.4166976682907412
Silhouette Score for k = 6 is 0.3648649810130078
Silhouette Score for k = 7 is 0.39397743262544
Silhouette Score for k = 8 is 0.40573744412356627
Silhouette Score for k = 9 is 0.3877256432563701

```

```

#Visualizing the silhouette scores in a plot
import matplotlib.pyplot as plt
plt.plot(range(2,10),silhouette_score)
plt.xlabel('k')
plt.ylabel('silhouette score')
plt.title('Silhouette Score')
plt.show()

```



```

# Trains a k-means model.
kmeans = KMeans(featuresCol='scaledFeatures',k=3)
model = kmeans.fit(final_data)
predictions = model.transform(final_data)
# Printing cluster centers
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)

Cluster Centers:
[ 4.91589737 10.9321157  37.2641905  12.39722305  8.58688868  1.77370551
 10.37323607]
[ 6.3407095  12.39263108  37.41143125  13.92892299  9.77251635  2.42396744
 12.28547936]
[ 4.06818854 10.13938448  35.87110297  11.81191124  7.52564313  3.24586152
 10.40780927]

```

```
predictions.select('prediction').show(5)
```

```
+-----+  
|prediction|  
+-----+  
|         0|  
|         0|  
|         0|  
|         0|  
|         0|  
+-----+  
only showing top 5 rows
```

```
#End Session  
spark.stop()
```

CONCLUSION:

We have successfully implemented 2 different clustering techniques i.e., DBSCAN Clustering Technique and K-means Clustering technique