



SANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY
(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)
ANGUCHETTPALAYAM, PANRUTI – 607 106.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

LAB MANUAL

CS3361 – DATA SCIENCE LABORATORY

Regulation 2021

Year / Semester : II /III

PREPARED BY

Mr. R. MANICKAVASAGAN, M.E.,
Assistant Professor / AIML

TABLE OF CONTENTS

Ex. No.	Date	Name of the Experiments	Page No.	Staff Sign
1.		Download, install and explore the features of NumPy, SciPy, Jupyter, Stats models and Pandas packages.	1	
2.		Working with Numpy arrays	10	
3.		a) Working with Pandas data frames	15	
		b) Creation of Pandas data frames to csv file	18	
		c) Creation of Pandas data frames time series	19	
4.		a) Reading data from text files	20	
		b) Reading data from excel files	21	
		c) Exploring various commands for doing descriptive analytics on the iris data set.	22	
5.		Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a) Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.	25	
		b) Bivariate analysis: linear and logistic regression modeling.	29	
		c) Multiple Regression analysis	36	
6.		Apply and explore various plotting functions on UCI data sets. a) Normal curves	39	
		b) Density and contour plots	41	
		c) Correlation and scatter plots	44	
		d) Histograms	46	
		e) Three dimensional plotting	47	
7.		Visualizing Geographic Data with Basemap	50	
Additional Experiments				
1		Normal distribution for two dimension array and to use sea born library for visualization of curve	53	
2		Correlation And Scatter Plots	55	
3		Implement Navie Bayes Models	58	
4		Computation for correlation coefficient	62	

Ex No. 1	Exploring the features of NumPy, SciPy, Jupyter, Stats models and Pandas packages.
Date:	

INSTALLING ANACONDA ON WINDOWS

Anaconda distribution of Python is the best option for problem solvers who want to use Python. Anaconda is free (although the download is large which can take time) and can be installed. Anaconda comes bundled with about 600 packages pre-installed including **NumPy**, **Matplotlib** and **SymPy**. These three packages are very useful for problem solvers and will be discussed in subsequent chapters.

Follow the steps below to install the Anaconda distribution of Python on Windows.

Steps:

1. Visit [Anaconda.com/downloads](https://anaconda.com/downloads)
2. Select Windows
3. Download the **.exe** installer
4. Open and run the **.exe** installer
5. Open the **Anaconda Prompt** and run some Python code

1. Visit the Anaconda downloads page

Go to the following link: [Anaconda.com/downloads](https://anaconda.com/downloads)

The Anaconda Downloads Page will look something like this:



2. Select Windows

Select Windows where the three operating systems are listed.

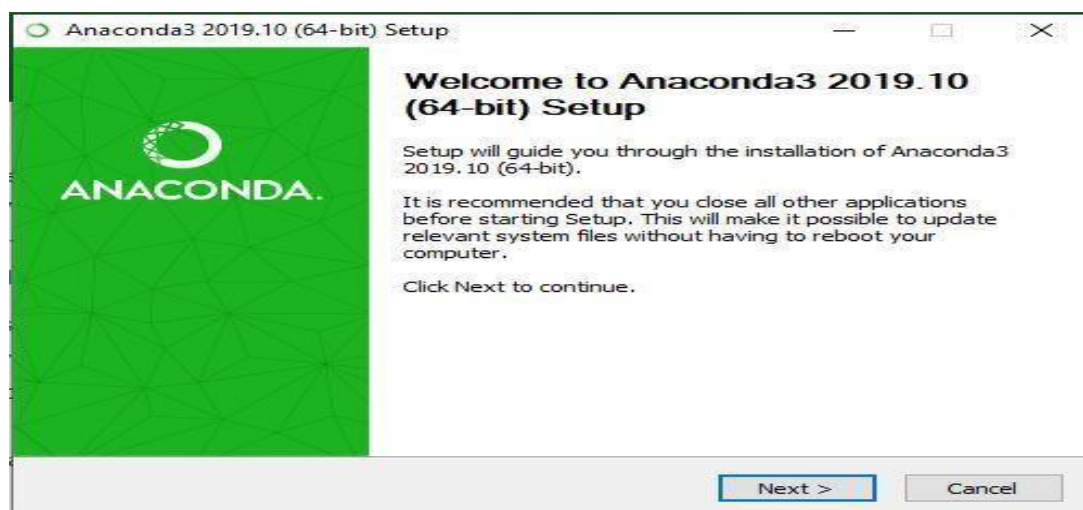


3. Download

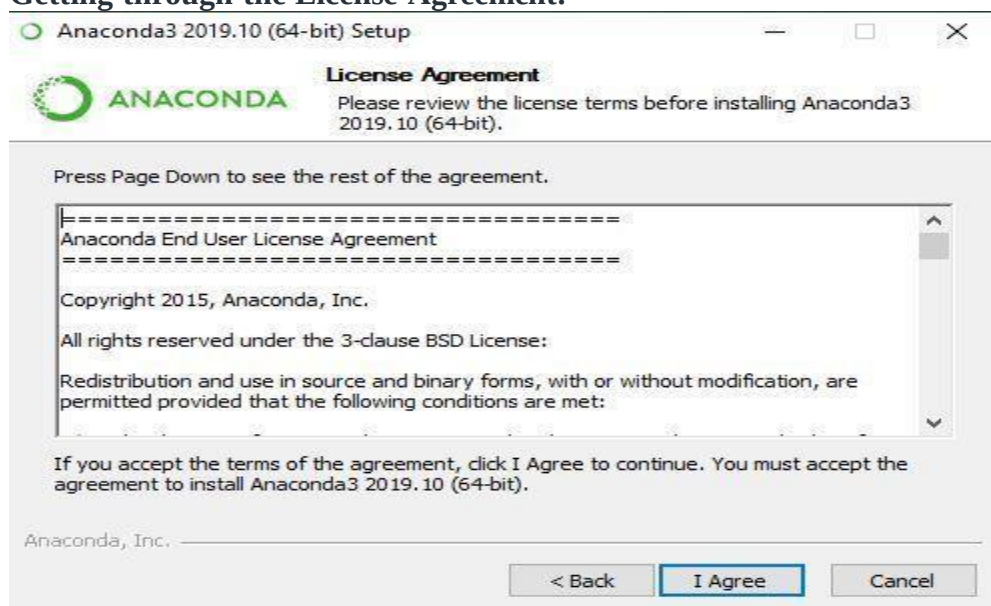
Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.



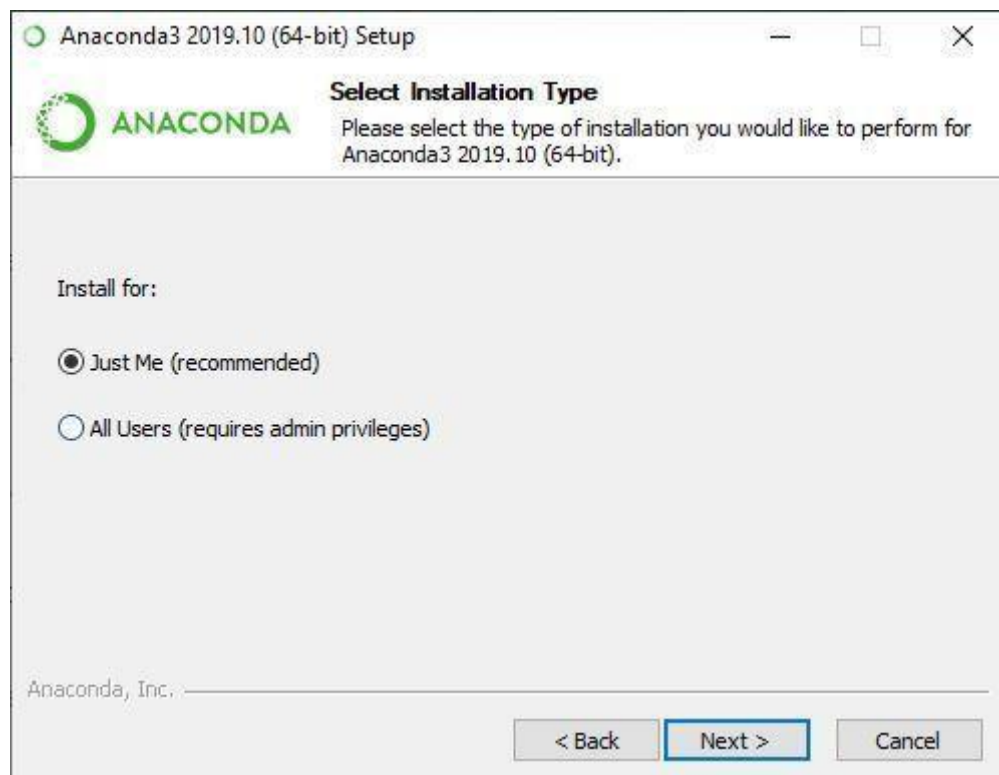
Begin with the installation process:
Getting Started:



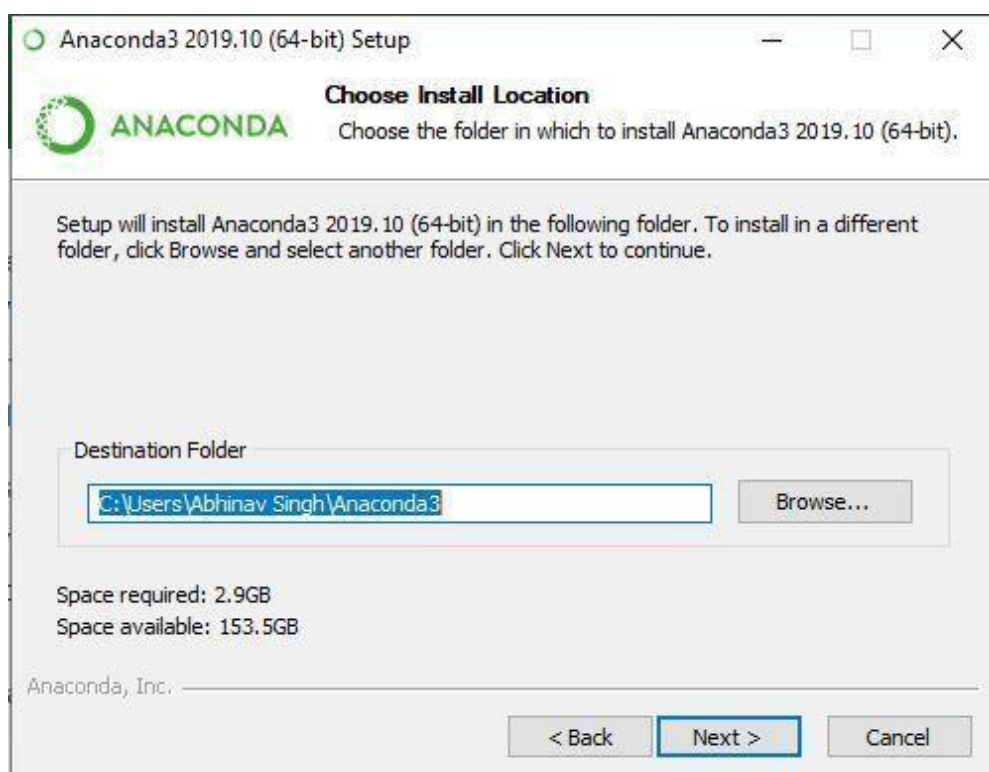
- **Getting through the License Agreement:**



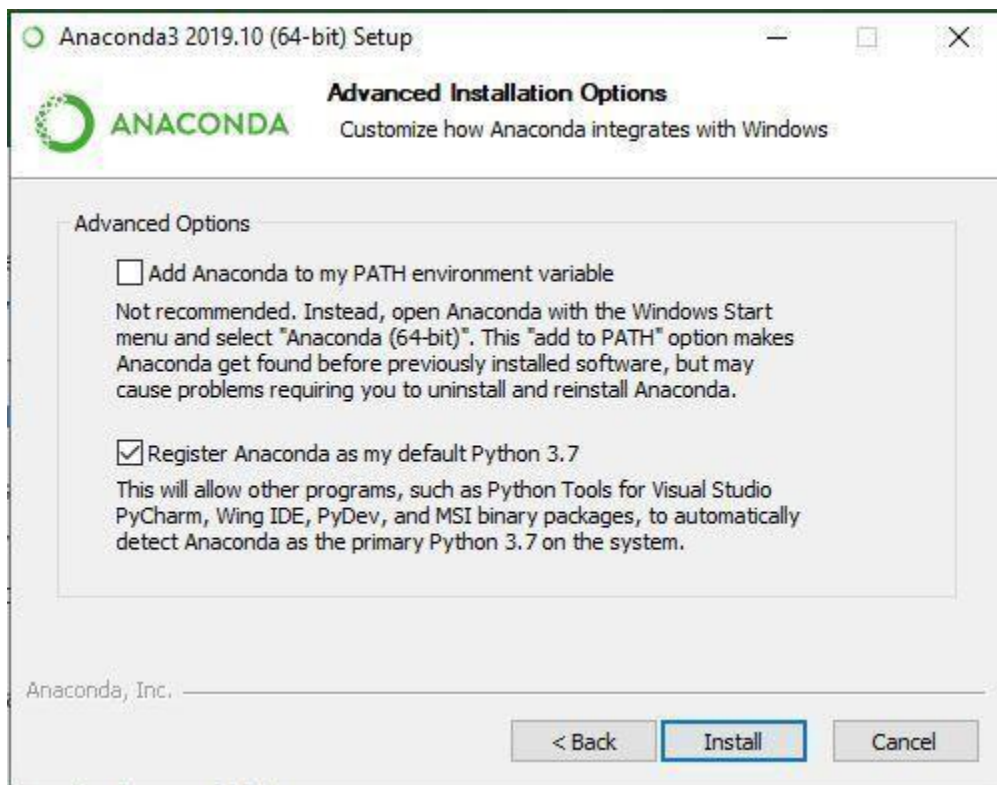
- **Select Installation Type:** Select **Just Me** if you want the software to be used by a single User



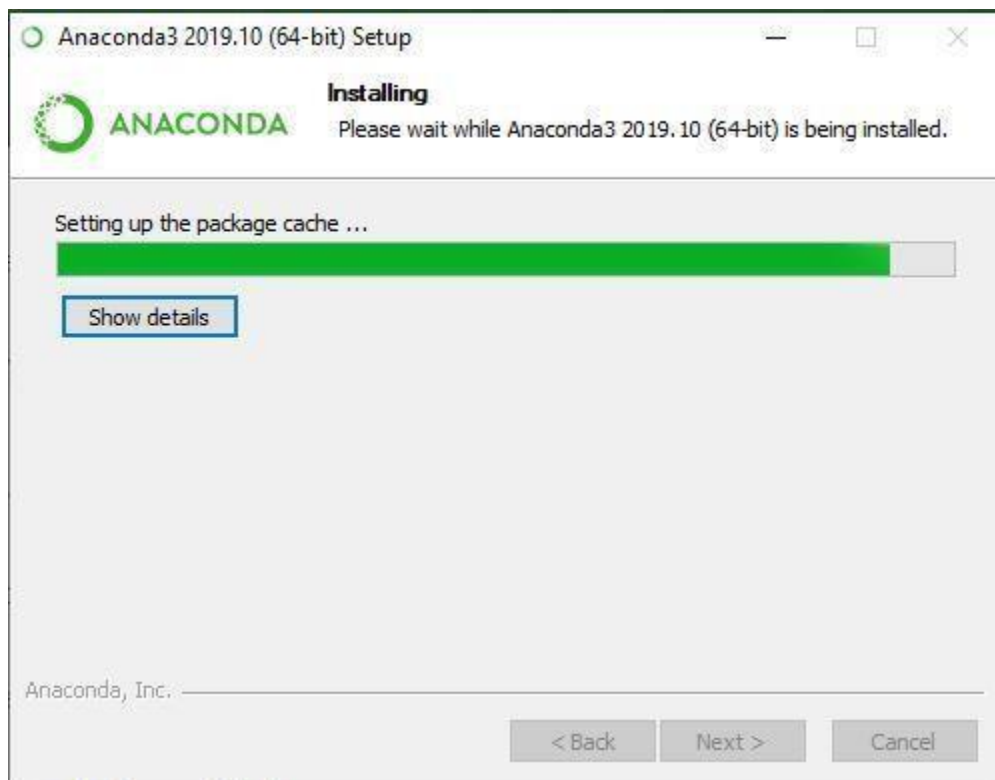
Choose Installation Location:



- **Advanced Installation Option:**



- **Getting through the Installation Process:**



- **Recommendation to Install Pycharm:**

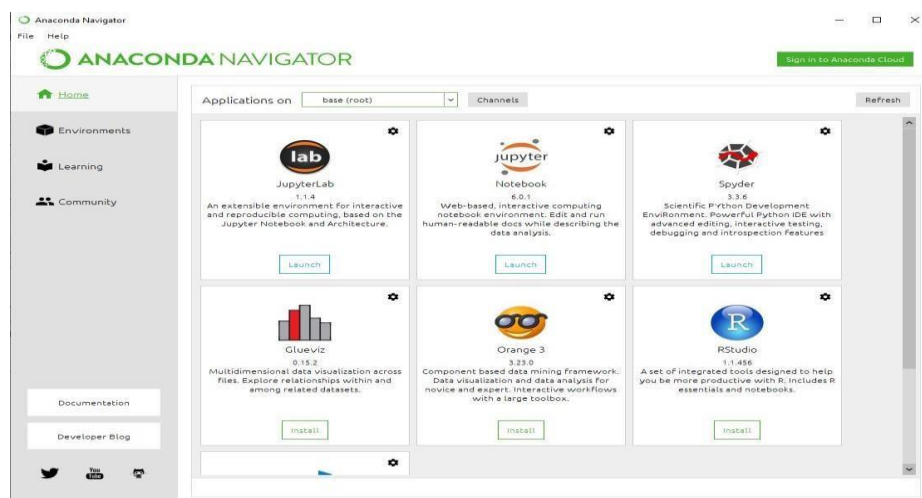
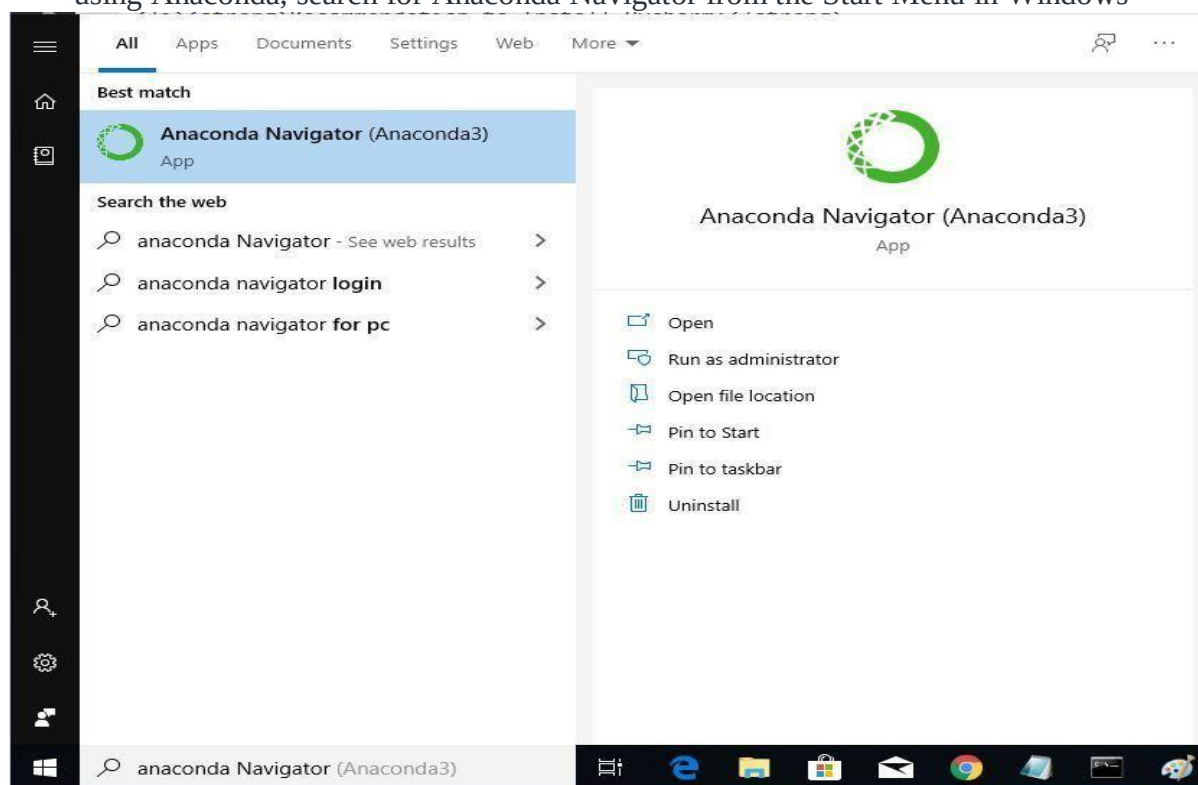


- **Finishing up the Installation:**



Working with Anaconda:

Once the installation process is done, Anaconda can be used to perform multiple operations. To begin using Anaconda, search for Anaconda Navigator from the Start Menu in Windows



Exploring NumPy Packages:

NumPy is a Python package used for numerical computation. NumPy is one of the foundational packages for scientific computing with Python. NumPy's core data type is the array and NumPy functions operate on arrays.

Installing NumPy

Before NumPy's functions and methods can be used, NumPy must be installed. Depending on which distribution of Python you use, the installation method is slightly different.

Install NumPy on Anaconda

If you installed the Anaconda distribution of Python, NumPy comes pre-installed and no further installation steps are necessary.

If you use a version of Python from python.org or a version of Python that came with your operating system, the **Anaconda Prompt** and **conda** or **pip** can be used to install NumPy.

Install NumPy with the Anaconda Prompt

To install NumPy, open the **Anaconda Prompt** and type:

```
> conda install numpy
```

Type y for yes when prompted.

Verify NumPy installation

To verify NumPy is installed, invoke NumPy's version using the Python REPL. Import NumPy and call the `__version__` attribute common to most Python packages.

In [1]:

```
import numpy as np  
np.version
```

Out[1]:'1.16.4'

A version number like '1.16.4' indicates a successful NumPy installation.

Exploring SciPy Packages:

Installing With Pip

You can install SciPy from PyPI with pip:

```
python -m pip install scipy
```

Installing Via Conda

You can install SciPy from the defaults or conda-forge channels with conda:

```
conda install scipy
```

Exploring Jupyter Packages:

Installing Jupyter

The simplest way to install **Jupyter notebooks** is to download and install the Anaconda distribution of Python. The Anaconda distribution of Python comes with Jupyter notebook included and no further installation steps are necessary.

Installing Jupyter on Windows using the Anaconda Prompt

To install Jupyter on Windows, open the **Anaconda Prompt** and type:

```
> conda install jupyter
```

Type y for yes when prompted. Once Jupyter is installed, type the command below into the **Anaconda Prompt** to open the Jupyter notebook file browser and start using Jupyter notebooks.

> jupyter notebook

Exploring Stats models Packages:

The easiest way to install stats models is to install it as part of the [Anaconda](#) distribution, a cross-platform distribution for data analysis and scientific computing. This is the recommended installation method for most users.

Instructions for installing from PyPI, source or a development version are also provided.

Python Support

Stats models supports Python 3.8, 3.9, and 3.10.

Anaconda

Stats models is available through conda provided by [Anaconda](#). The latest release can be installed using:
`conda install -c conda-forge statsmodels`

PyPI (pip)

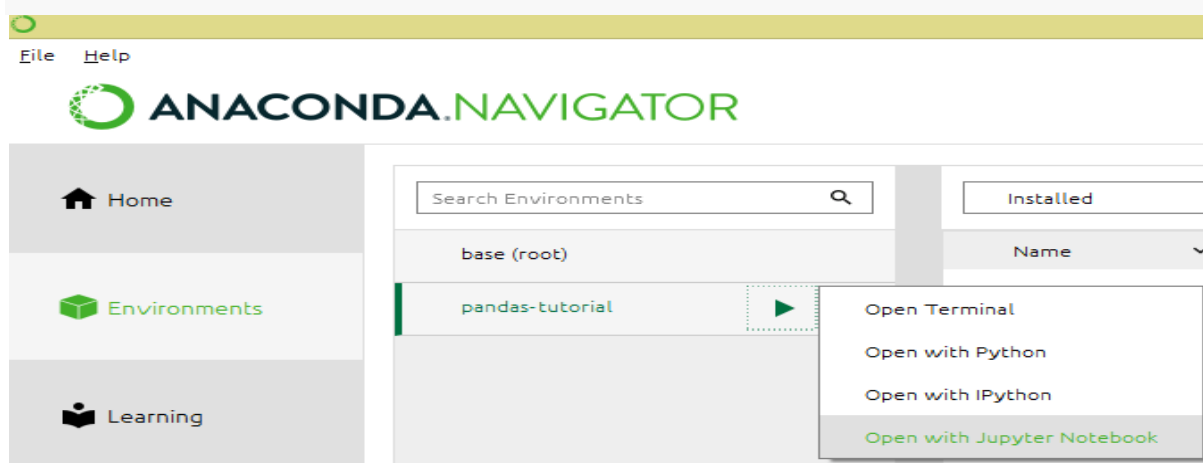
To obtain the latest released version of stats models using pip:

```
python -m pip install statsmodels
```

Follow [this link to our PyPI page](#) to directly download wheels or source.

Exploring Pandas packages

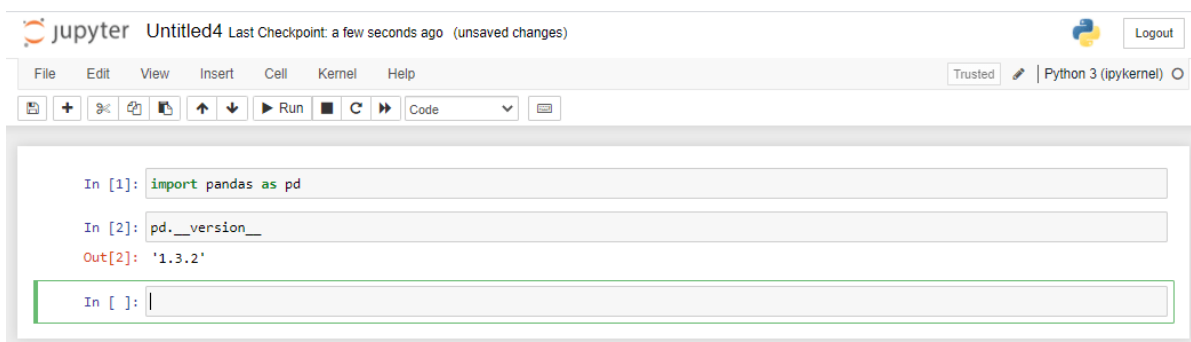
Go to **Anaconda Navigator** -> **Environments** -> **your environment** (mine pandas-tutorial) -> select **Open with Jupyter Notebook**



This opens up Jupyter Notebook in the default browser.



Now select New -> Python X and enter the below lines and select Run.



This completes installing Anaconda and running pandas on Jupyter Notebook.

Ex No. 2	WORKING WITH NUMPY ARRAY
Date:	

Aim:

To perform basic NumPy operation such as creating an array, finding the dimension and data size, reshape, slicing and arithmetic operation.

NumPy:

NumPy stands for Numerical Python and is one of the most useful scientific libraries in Python programming. It provides support for large multidimensional array objects and various tools to work with them. NumPy arrays are called **ndarray** or **N-dimensional arrays** and they store elements of the same type and size. It is known for its high-performance and provides efficient storage and data operations as arrays grow in size. NumPy comes pre-installed when you download Anaconda.

NumPy Array Attributes:

Single-dimensional Numpy Array:

```
import numpy as np
a=np.array([1,2,3])
print(a)
```

Output : [1 2 3]

Multi-dimensional Array:

```
import numpy as np
a=np.array([(1,2,3), (4, 5, 6)])
print(a)
```

Output : [[1 2 3]
[4 5 6]]

Array creation using Random numbers:

```
import numpy as np
np.random.seed(0) # seed for reproducibility
x1 = np.random.randint(10, size=6) # One-dimensional array
x2 = np.random.randint(10, size=(2, 4)) # Two-dimensional array
print (x1)
print(x2)
```

Output:

```
[5, 0, 3, 3, 7, 9])
([[3, 5, 2, 4],
 [7, 6, 8, 8]])
```

Python NumPy Operations

ndim:

To find the dimension of the array, whether it is a two-dimensional array or a single dimensional array.

```
import numpy as np
a=np.array([(1,2,3), (4, 5, 6)])
print(a.ndim)
```

Output : 2

itemsize:

To calculate the byte size of each element.

```
import numpy as np
a=np.array([(1,2,3)])
print(a.itemsize)
```

Output : 4

So every element occupies 4 byte in the above numpy array

dtype:

To find the data type of the elements that are stored in an array.

```
import numpy as np
a=np.array([(1,2,3)])
print(a.dtype)
```

Output : int32

The data type of an array is integer 32 bits. Similarly, you can also find the size and shape of the array using 'size' and 'shape' function respectively.

```
import numpy as np
a = np.array([(1,2,3,4,5,6)])
print(a.size)
print(a.shape)
```

Output : 6 (1,6)

reshape:

Reshape is when you change the number of rows and columns which gives a new view to an object.

```
import numpy as np
a = np.array([(8,9,10),(11,12,13)])
print(a)
a=a.reshape(3,2)
print(a)
```

Output : [[8 9 10] [11 12 13]]

[[8 9] [10 11] [12 13]]

Slicing:

Slicing is basically extracting particular set of elements from an array.

```
import numpy as np
a= np.array([(1,2,3,4),(3,4,5,6)])
print(a[0,2])
```

Output : 3

Here, the array(1,2,3,4) is your index 0 and (3,4,5,6) is index 1 of the python numpy array.

Slicing: Accessing subarrays

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the *slice* notation, marked by the colon (:) character.

```
x2 = np.random.randint(10, size=(2, 4))
x2[:2, :3] # two rows, three columns
array([[3, 5, 2],
       [7, 6, 8]])
```

linspace

This is another operation in python numpy which returns evenly spaced numbers over a specified interval.

```
import numpy as np
a=np.linspace(1,3,10)
print(a)
```

Output : [1. 1.22222222 1.44444444 1.66666667 1.88888889 2.11111111 2.33333333 2.55555556 2.77777778 3.]

It has printed 10 values between 1 to 3.

max/ min

To find the minimum, maximum number in numpy array.

```
import numpy as np
a=np.array([1,2,3])
print(a.min())
print(a.max())
```

Output: 1, 3

Arithmetic Operation:

To perform more operations on numpy array i.e addition, subtraction, multiplication and division of the two matrices.

```
import numpy as np
x=np.array([(1,2,3),(3,4,5)])
```

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

```
print(x+y)
```

```
print(x-y)
```

```
print(x*y)
```

```
print(x/y)
```

Output :

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

```
[[0 0 0] [0 0 0]]
```

```
[[ 1 4 9] [ 9 16 25]]
```

```
[[ 1. 1. 1.] [ 1. 1. 1.]]
```

Square Root

There are various mathematical functions that can be performed using python numpy. To find the square root of the number.

```
import numpy as np
```

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

```
print(np.sqrt(a))
```

Output –

```
[[ 1. 1.41421356 1.73205081]
 [ 1.73205081 2. 2.23606798]]
```

Trigonometric functions

NumPy provides a large number of useful function, and some of the most useful for the data scientist are the trigonometric functions.

```
theta = np.linspace(0, np.pi, 3)
```

```
print("theta = ", theta)
```

```
print("sin(theta) = ", np.sin(theta))
```

```
print("cos(theta) = ", np.cos(theta))
```

```
print("tan(theta) = ", np.tan(theta))
```

Output:

```
theta = [0.          1.57079633 3.14159265]
```

```
sin(theta) = [0.00000000e+00 1.00000000e+00 1.2246468e-16]
```

```
cos(theta) = [ 1.0000000e+00 6.123234e-17 -1.0000000e+00]
```

```
tan(theta) = [ 0.00000000e+00 1.63312394e+16 -1.22464680e-16]
```

Exponents:

Another common type of operation available in a NumPy ufunc are the exponentials

```
x = [1, 2, 3]
```

```
print("x =", x)
```

```
print("e^x =", np.exp(x))
print("2^x =", np.exp2(x))
print("3^x =", np.power(3, x))
```

Output:

```
x = [1, 2, 3]
e^x = [ 2.71828183  7.3890561 20.08553692]
2^x = [2.  4.  8.]
3^x = [ 3  9 27]
```

Logarithms:

The basic `np.log` gives the natural logarithm; if you prefer to compute the base-2 logarithm or the base-10 logarithm, these are available as well.

```
x = [1, 2, 4, 10]
print("x =", x)
print("ln(x) =", np.log(x))
print("log2(x) =", np.log2(x))
print("log10(x) =", np.log10(x))
```

Output:

```
x = [1, 2, 4, 10]
ln(x) = [0.         0.69314718 1.38629436 2.30258509]
log2(x) = [0.         1.         2.         3.32192809]
log10(x) = [0.         0.30103   0.60205999 1.         ]
```

Result:

Thus, the working of Numpy array was executed successfully.

Exp No. 3A	CREATION OF PANDAS DATA FRAMES
Date:	

Aim:

To write a Python program

- (a) To create a Data Frames of student detail that contains name, score, Attempts and qualifier.
- (b) Display a summary about a specified Data frame, its data.
- (c) Change the name 'Prasath' to 'Saran' in the name column.
- (d) Delete a column.
- (e) Inserting a new column in existing data frames.
- (f) Display three rows.

Algorithm:

Step 1: Start the program

Step 2: import the library pandas.

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: Summarize the data frame using the function info()

Step 5: Use replace() function to change the name and pop() function to delete the column.

Step 6: Print only the first three rows of the data frames use .iloc() function.

Step 7: Stop the program.

Program:

```
import pandas as pd
import numpy as np

data={ 'name' : [ 'Abishek' , 'Abirami' , 'Akila' , 'Balaji', 'Dinesh' , 'Apoorva' , 'Ragunath' , 'Prasath' ,
'Pradeep', 'Mohan' ],
'Scores' : [ 12.5, 9, 16.5,15, 9, 20, 14.5, 8,12, 19],
'attempts' : [ 1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
'qualify' : [ 'yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes'] }

labels = [ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j' ]

df = pd.DataFrame(data, index = labels)

# Display summary of data

print("Summary of the basic info. About the data frames and its data:")

print(df.info())

# Change the name

print("Original rows:")

print(df)
```

```

print("\n Change the name 'Prasath' to 'Saran':")
df['name'] = df['name'].replace('Prasath', 'Saran')
print(df)

# Delete the attempt column from the Data Frames
print("\n Delete the 'attempts' column from the data frames:")
df.pop('attempts')
print(df)

#insert the new column
color = ['Red', 'Blue', 'Orange', 'Red', 'White', 'White', 'Blue', 'Red', 'Green', 'Green']
df['color'] = color
print("\n New Dataframes after inserting the 'color' column")
print(df)

# Displaying three rows of the dataframe
print("First three rows of data frames")
print(df.iloc[:3])

```

OUTPUT:

```

Summary of the basic info. About the data frames and its data:
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
-----
0   name        10 non-null     object
1   Scores      10 non-null     float64
2   attempts    10 non-null     int64
3   qualify     10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
None

```

Original rows:

	name	Scores	attempts	qualify
a	Abishek	12.5	1	yes
b	Abirami	9.0	3	no
c	Akila	16.5	2	yes
d	Balaji	15.0	3	no
e	Dinesh	9.0	2	no
f	Apoorva	20.0	3	yes
g	Ragunath	14.5	1	yes
h	Prasath	8.0	1	no
I	Pradeep	12.0	2	no
j	Mohan	19.0	1	yes

Change the name 'Prasath' to 'Saran':

	name	Scores	attempts	qualify
a	Abishek	12.5	1	yes
b	Abirami	9.0	3	no
c	Akila	16.5	2	yes
d	Balaji	15.0	3	no
e	Dinesh	9.0	2	no
f	Apoorva	20.0	3	yes
g	Ragunath	14.5	1	yes
h	Saran	8.0	1	no
i	Pradeep	12.0	2	no
j	Mohan	19.0	1	yes

Delete the 'attempts' column from the data frames:

	name	Scores	qualify
a	Abishek	12.5	yes
b	Abirami	9.0	no
c	Akila	16.5	yes
d	Balaji	15.0	no
e	Dinesh	9.0	no
f	Apoorva	20.0	yes
g	Ragunath	14.5	yes
h	Saran	8.0	no
i	Pradeep	12.0	no
j	Mohan	19.0	yes

New Dataframes after inserting the 'color' column

	name	Scores	qualify	color
a	Abishek	12.5	yes	Red
b	Abirami	9.0	no	Blue
c	Akila	16.5	yes	Orange
d	Balaji	15.0	no	Red
e	Dinesh	9.0	no	White
f	Apoorva	20.0	yes	White
g	Ragunath	14.5	yes	Blue
h	Saran	8.0	no	Red
i	Pradeep	12.0	no	Green
j	Mohan	19.0	yes	Green

First three rows of data frames

	name	Scores	qualify	color
a	Abishek	12.5	yes	Red
b	Abirami	9.0	no	Blue
c	Akila	16.5	yes	Orange

Result:

Thus, the python program for creation of data frame was executed successfully.

Exp No. 3B	CREATION OF PANDAS DATA FRAMES TO CSV FILE
Date:	

Aim: To write a Pandas program to create a Data frames to CSV file using tab separator.

Algorithm:

Step 1: Start the program

Step 2: Import the library numpy and pandas.

Step 3: Create a data frames using the `pd.DataFrame()`.

Step 4: Convert the data frames to CSV files using the function `to_csv()`.

Step 5: Read the CSV files using the fnction `pd.read_csv()`.

Step 6: Print the CSV file.

Program:

```
import pandas as pd
import numpy as np
d = { 'Col1' : [ 1, 4, 3, 4, 5], 'Col2' : [ 4, 5, 6, 7, 8 ], 'Col3' : [ 7, 8, 9, 5, 1] }
df1 = pd.DataFrame(data = d )
print("original DataFrames")
print(df1)
print('Data from new_file.csv file:')
df1.to_csv('new_file.csv', sep = '\t', index = False)
new_df = pd.read_csv('new_file.csv')
print (new_df)
```

Output:

original DataFrames

	Col1	Col2	Col3
0	1	4	7
1	4	5	8
2	3	6	9
3	4	7	5
4	5	8	1

Data from new_file.csv file:

	Col1\tCol2\tCol3
0	1\t4\t7
1	4\t5\t8
2	3\t6\t9
3	4\t7\t5
4	5\t8\t1

Result:

Thus, the python program for creation of pandas data frame to csv file was executed successfully.

Exp No. 3C	CREATION OF PANDAS DATA FRAMES TIME SERIES
Date:	

Aim: To write the pandas program to create

- (a) Datetime object for Jan 15 2021.
- (b) Specific date and time of 9.20 pm
- (c) Local date and time.
- (d) A date without time.
- (e) Current date.
- (f) Time from a datetime
- (g) Current local time.

Algorithm:

Step 1: Import the library such as pandas and datetime.

Step 2: Print the specific date and time use the function datetime(). And mention the date on it.

Step 3: Print the current time and date, use the function datetime.now().

Program:

```
import pandas as pd
import datetime
from datetime import datetime
print("datetime object for Jan 15 2021:")
print(datetime(2021, 1, 15))
print("\nSpecific date and time of 9:20")
print(datetime(2021,1,15,21,20))
print(datetime.now())
print(datetime.now().date())
print(datetime.time(datetime(2021,2,1,21,8,12)))
print(datetime.now().time())
```

Output:

```
datetime object for Jan 15 2021:
2021-01-15 00:00:00
```

```
Specific date and time of 9:20
2021-01-15 21:20:00
2022-09-11 11:50:17.393674
2022-09-11
21:08:12
11:50:17.393674
```

Result:

Thus, the python program for creation of pandas data frame time series was executed successfully.

Exp No. 4A	READING DATA FROM TEXT FILES
Date:	

Aim:

To write a Python program to read data from text files.

Algorithm:

Step 1: Start the program

Step 2: import the library pandas.

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: filename.txt: As the name suggests it is the name of the text file from which we want to read data.

Step 5: sep: It is a separator field. In the text file, we use the space character(' ') as the separator.

Step 6: header: This is an optional field. By default, it will take the first line of the text file as a header. If we use header=None then it will create the header.

Step 7: names: We can assign column names while importing the text file by using the names argument.

Step 8: Stop the program.

Program:

```
#Read Text Files with Pandas using read_csv()
# importing pandas
import pandas as pd
# read text file into pandas DataFrame
df = pd.read_csv("india.txt", sep=" ")
# display DataFrame
print(df)
```

india.txt

Batsman	Bowler
Virat	Jadeja
Dhoni	Bumrah
Sachin	Siraj
Raina	Ashwin
Rohit	Shami

OUTPUT:

	Batsman	Bowler
0	Virat	Jadeja
1	Dhoni	Bumrah
2	Sachin	Siraj
3	Raina	Ashwin
4	Rohit	Shami

Result:

Thus, the python program for reading of data from text file was executed successfully.

Exp No. 4B	READING DATA FROM EXCEL FILES
Date:	

Aim:

To write a Python program to read data from Excel files.

Algorithm:

Step 1: Start the program

Step 2: import the library pandas.

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: Capture the file path

Step 5: Apply the Python code

Step 6: Run the Python code to import the Excel file

Step 7: Stop the program.

Program:

```
import pandas as pd
df = pd.read_excel ("C:\\Users\\ASUS\\OneDrive\\Desktop\\DATA SCIENCE\\DATA SCIENCE
LAB\\ex4.xlsx", sheet_name='ex4b')
print (df)
```

Output:

Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	
0	S.No	Register Number	Year	Gender	Student Name
1	1	422721104001	II	MALE	ABINASH R
2	2	422721104002	II	FEMALE	ABISHALINI P
3	3	422721104003	II	MALE	AKILAN G
4	4	422721104004	II	FEMALE	AKILANDESHWARI K
5	5	422721104005	II	MALE	ANAS AHAMED A
6	6	422721104006	II	MALE	ARAVINDH M
7	7	422721104007	II	MALE	BARATH S
8	8	422721104008	II	MALE	BARATH KUMAR R
9	9	422721104009	II	MALE	BHUVANACHANDHIRAN P
10	10	422721104011	II	MALE	CHRISTOFOR JOHN S

Result:

Thus, the python program for reading of data from excel file was executed successfully.

Exp No. 4C	Exploring Various Commands for Doing Descriptive Analytics on The Iris Data Set.
Date:	

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a technique to analyze data using some visual Techniques. With this technique, we can get detailed information about the statistical summary of the data. We will also be able to deal with the duplicate's values, outliers, and also see some trends or patterns present in the dataset.

Iris Dataset

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

You can download the Iris.csv file from the above link. Now we will use the Pandas library to load this CSV file, and we will convert it into the [dataframe](#). [read_csv\(\)](#) method is used to read CSV files.

Reading the CSV File:

```
import pandas as pd
# Reading the CSV file
df = pd.read_csv("Iris.csv")
# Printing top 5 rows
df.head()
```

Output:

	sepalength	sepalwidth	petallength	petalwidth	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

Shape of the dataset

```
df.shape
```

Output

```
(150, 5)
df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
--  --
0   sepallength  150 non-null    float64
1   sepalwidth   150 non-null    float64
2   petallength  150 non-null    float64
3   petalwidth   150 non-null    float64
4   class        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Summary of the dataset

```
df.describe()
```

Output

	sepallength	sepalwidth	petallength	petalwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Checking Missing Values

```
df.isnull().sum()
```

Output

```
sepallength    0
sepalwidth     0
petallength    0
petalwidth     0
class          0
dtype: int64
```

Counts of unique values

```
df.value_counts("class")
```

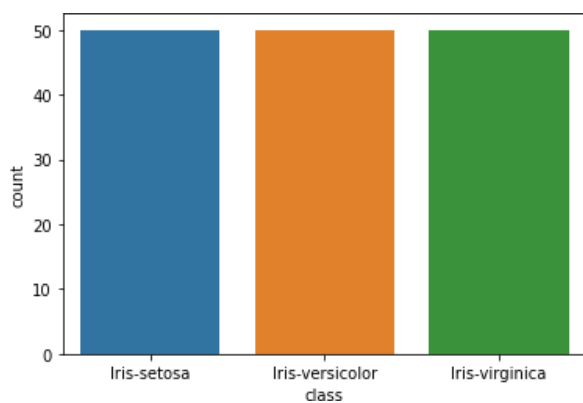
Output

```
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
```

Data Visualization

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='class', data=df, )
plt.show()
```

Output



Result:

Thus, the python program for exploring various commands for doing descriptive analytics on the iris data set was executed successfully.

Exp No. 5A	Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.
Date:	

Aim:

To write a Python program to Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.

Algorithm:

Step 1: Start the program

Step 2: import the library pandas, Numpy and Statistics

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: Accessing the dataset

Step 5: Identifying the variable that needs to be analyzed

Step 6: Identifying the questions that need to be answered through the analysis

Step 7: Determining the appropriate type of univariate analysis techniques to answer the above-identified questions

Step 8: **Calculating Central Tendency, Dispersion, Skewness and Kurtosis**

- **Central tendency** — mean, median, mode
- *Mean* — is average value of given numeric values
- **Median** — is middle most value of given values
- **Mode** — is most frequently occurring value of given numeric variables
- **Dispersion** — variance, standard deviation, range, interquartile range (IQR)
- **Variance** — its gives average deviation from mean value
- **Standard Deviation** — it is square root of variance
- **Range** — it gives difference between max and min value
- **Interquartile Range (IQR)** — it gives difference between Q3 and Q1, where Q3 is 3rd Quartile value and Q1 is 1st Quartile value.
- **Skewness** — symmetry of data along with mean value
- **Kurtosis** — peakedness of data at mean value

Step 9: Stop the program.

1. Input Data

```
import pandas as pd
```

```
import numpy as np
```

```
import statistics as st
```

```
# Load the data
```

```
df = pd.read_csv("diabetes.csv")
```

```
print(df.shape)
```

```
print(df.info())
```

Output

```
(768, 9)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#      Column                                Non-Null Count  Dtype
-----
0      Pregnancies                             768 non-null    int64
1      Glucose                                  768 non-null    int64
2      BloodPressure                           768 non-null    int64
3      SkinThickness                           768 non-null    int64
4      Insulin                                 768 non-null    int64
5      BMI                                     768 non-null    float64
6      DiabetesPedigreeFunction                768 non-null    float64
7      Age                                     768 non-null    int64
8      Outcome                                768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

2. Frequency

```
df['BMI'].value_counts()
```

Output

```
32.0    13
31.2     12
31.6     12
0.0      11
33.3     10
..
19.3      1
49.3      1
19.4      1
20.0      1
40.1      1
Name: BMI, Length: 248, dtype: int64
```

3. Mean

```
df.mean()
```

Output

```
Pregnancies    3.845052
Glucose        120.894531
BloodPressure   69.105469
SkinThickness   20.536458
Insulin         79.799479
BMI             31.992578
DiabetesPedigreeFunction  0.471876
Age            33.240885
Outcome         0.348958
dtype: float64
```

Rowwise Mean

```
df.mean(axis = 1)[0:5]
```

Output

```
0    38.469667
1    26.550111
2    34.663556
3    35.807444
4    51.043111
dtype: float64
```

4.Median

```
df.median()
```

Output

```
Pregnancies      3.0000
Glucose          117.0000
BloodPressure     72.0000
SkinThickness     23.0000
Insulin           30.5000
BMI              32.0000
DiabetesPedigreeFunction  0.3725
Age              29.0000
Outcome          0.0000
dtype: float64
```

5.Mode

```
df.mode()
```

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes PedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

6. Standard Deviation

```
df.std()
```

output

```
Pregnancies      3.369578
Glucose          31.972618
BloodPressure     19.355807
SkinThickness     15.952218
```

Insulin	115.244002
BMI	7.884160
DiabetesPedigreeFunction	0.331329
Age	11.760232
Outcome	0.476951

dtype: float64

7. Variance

df.var()

output

Pregnancies	11.354056
Glucose	1022.248314
BloodPressure	374.647271
SkinThickness	254.473245
Insulin	13281.180078
BMI	62.159984
DiabetesPedigreeFunction	0.109779
Age	138.303046
Outcome	0.227483

dtype: float64

8. Skewness

print(df.skew())

Output

Pregnancies	0.901674
Glucose	0.173754
BloodPressure	-1.843608
SkinThickness	0.109372
Insulin	2.272251
BMI	-0.428982
DiabetesPedigreeFunction	1.919911
Age	1.129597
Outcome	0.635017

dtype: float64

9. kurtosis

df.BMI.kurtosis()

output

3.290442900816981

Result:

Thus, the python program for Univariate analysis: frequency, mean, median, mode, variance, standard deviation, skewness and kurtosis was executed successfully.

Exp No. 5B	BIVARIATE ANALYSIS: LINEAR AND LOGISTIC REGRESSION MODELING
Date:	

i) BIVARIATE ANALYSIS: LINEAR REGRESSION MODELING

Aim:

To write a python program to find bivariate analysis: linear regression modeling.

Algorithm:

Step 1: Start the program

Step 2: import the library

- *NumPy* — used to perform mathematical operations mainly using multi-dimensional arrays.
- *pandas* — used for data manipulation and analysis.
- *matplotlib* — it is a plotting library as a component of NumPy
- *statsmodels* — it is used to explore data, estimate statistical models and perform statistical tests.

Step 3: Create a data frame using the function `pd.DataFrame()`

Step 4: Import the dataset and load the data.

Step 5: We can perform descriptive statistics using the following command: `Dataframe.describe()`

Step 6: Create Your First Linear Regression and Declare the Dependent and Independent Variables.

Step 7: To plot the regression line on the graph, simply define the linear regression equation,

$$\text{i.e., } y_{\text{hat}} = b_0 + (b_1 * x_1)$$

Step 8: Finally, plot the regression line using `matplotlib.pyplot()`

Step 9: Stop the program.

1. Import the Relevant Libraries:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
sns.set()
```

2. Import the Dataset

```
data = pd.read_csv("diabetes.csv")
data
```

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

3. Descriptive Statistics

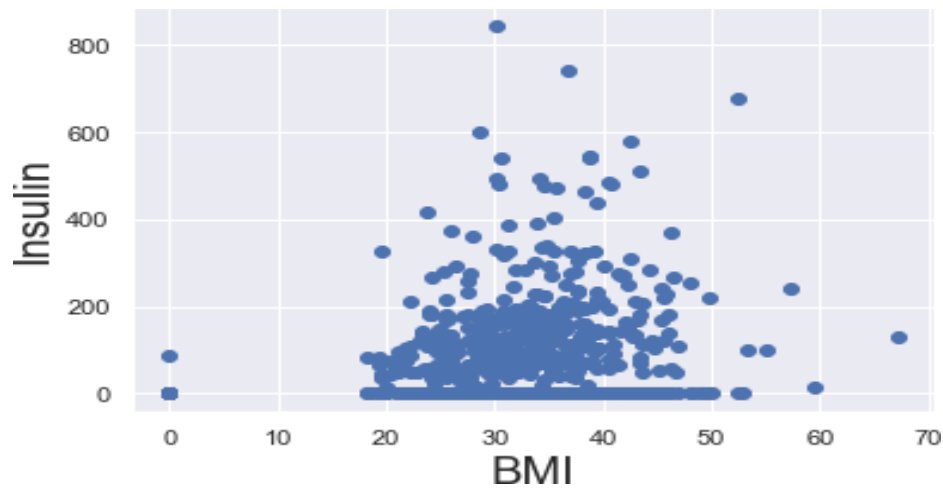
data.describe()

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
count	768.0000	768.00000	768.00000	768.00000	768.0000	768.00000	768.00000	768.00000	768.00000
mean	3.845052	120.89453	69.105469	20.536458	79.79947	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.2440	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.00000	72.000000	23.000000	30.50000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.25000	80.000000	32.000000	127.2500	36.600000	0.626250	41.000000	1.000000
max	17.00000	199.00000	122.00000	99.000000	846.0000	67.100000	2.420000	81.000000	1.000000

4. Create Your First Linear Regression

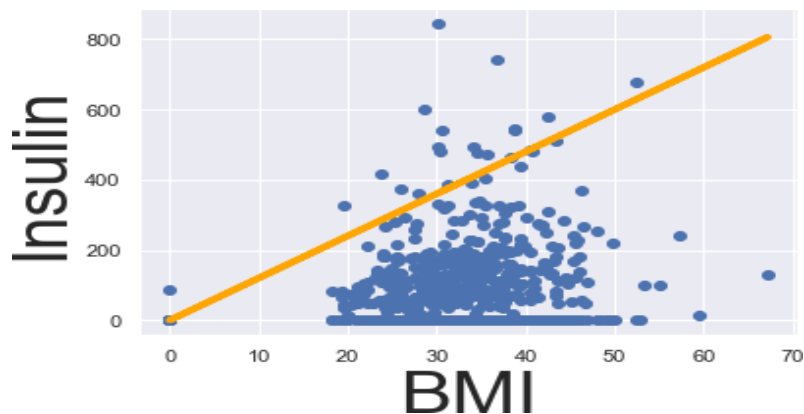
y = data['Insulin']

```
x1 = data['BMI']
plt.scatter(x1,y)
plt.xlabel('BMI', fontsize=20)
plt.ylabel('Insulin', fontsize=20)
plt.show()
```



5. Plot the Regression Line

```
plt.scatter(x1,y)
yhat = 12*x1 + 0.275
fig = plt.plot(x1, yhat, lw=4,c='orange', label= 'regression line')
plt.xlabel('BMI', fontsize=40)
plt.ylabel('Insulin', fontsize=40)
plt.show()
```



Result: Thus, the python program for bivariate analysis: linear regression modeling was executed successfully.

ii) BIVARIATE ANALYSIS: LOGISTIC REGRESSION MODELING

Aim:

To write a python program to find bivariate analysis: logistic regression modeling.

Algorithm:

Step 1: Start the program

Step 2: Import the library

- NumPy — used to perform mathematical operations mainly using multi-dimensional arrays.
- Pandas — used for data manipulation and analysis.
- Matplotlib — it is a plotting library as a component of NumPy
- Statsmodels — it is used to explore data, estimate statistical models and perform statistical tests.
- Seaborn and Matplotlib for visualizations.
- Import logistic regression algorithm from sklearn. This algorithm will help us build our classification model. joblib available in sklearn to save our model for future use.

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: Import the dataset and load the data.

Step 5 : Data Description

Step 6: Data Exploration

Step 7: Data Preparation

Step 8: Training and Evaluating the Machine Learning Model

Step 9: Interpreting the ML Model

Step 10: Saving the Model

Step 11: Making Predictions with the Model

Step 12: Stop the program.

PROGRAM:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib
```

Data Description

```
diabetesDF = pd.read_csv('diabetes.csv')
print(diabetesDF.head())
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

diabetesDF.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Pregnancies                          768 non-null    int64
1   Glucose                             768 non-null    int64
2   BloodPressure                       768 non-null    int64
3   SkinThickness                       768 non-null    int64
4   Insulin                             768 non-null    int64
5   BMI                                 768 non-null    float64
6   DiabetesPedigreeFunction            768 non-null    float64
7   Age                                 768 non-null    int64
8   Outcome                             768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
corr = diabetesDF.corr()
print(corr)
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns)
```

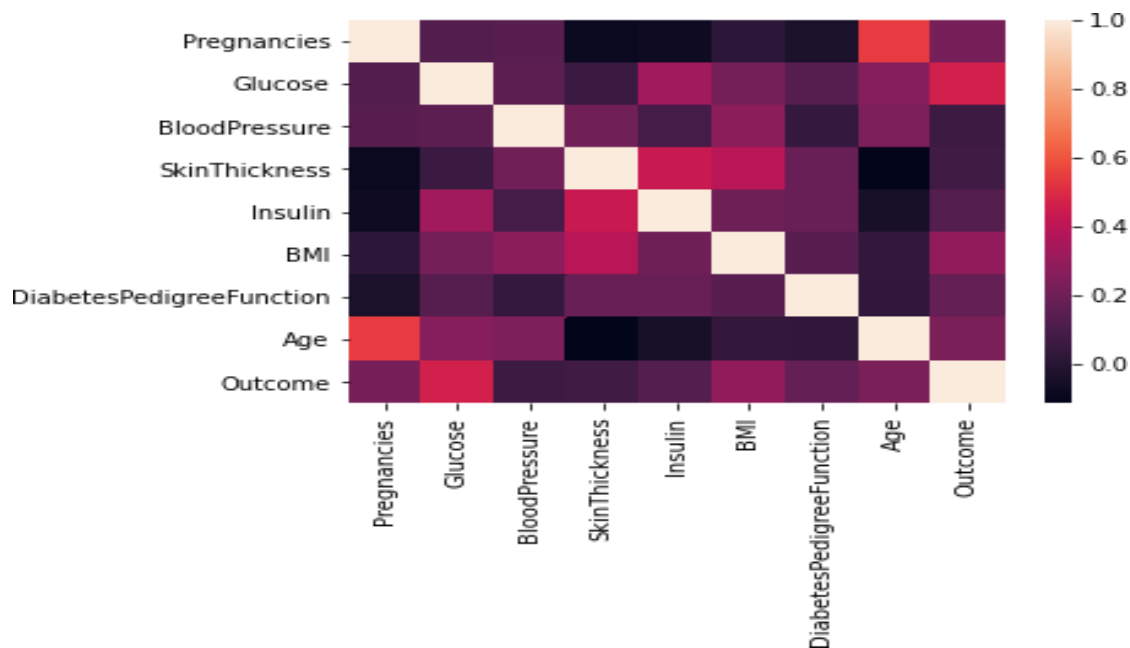
	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

	Insulin	BMI	DiabetesPedigreeFunction \
Pregnancies	-0.073535	0.017683	-0.033523
Glucose	0.331357	0.221071	0.137337
BloodPressure	0.088933	0.281805	0.041265
SkinThickness	0.436783	0.392573	0.183928
Insulin	1.000000	0.197859	0.185071
BMI	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	0.185071	0.140647	1.000000
Age	-0.042163	0.036242	0.033561
Outcome	0.130548	0.292695	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.239528	0.065068
SkinThickness	-0.113970	0.074752
Insulin	-0.042163	0.130548
BMI	0.036242	0.292695

DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

<AxesSubplot:>



```
dfTrain = diabetesDF[:650]
dfTest = diabetesDF[650:750]
dfCheck = diabetesDF[750:]
trainLabel = np.asarray(dfTrain['Outcome'])
trainData = np.asarray(dfTrain.drop('Outcome',1))
testLabel = np.asarray(dfTest['Outcome'])
testData = np.asarray(dfTest.drop('Outcome',1))
means = np.mean(trainData, axis=0)
stds = np.std(trainData, axis=0)
trainData = (trainData - means)/stds
testData = (testData - means)/stds
# np.mean(trainData, axis=0) => check that new means equal 0
# np.std(trainData, axis=0) => check that new stds equal 1
diabetesCheck = LogisticRegression()
diabetesCheck.fit(trainData, trainLabel)
```

O/P: LogisticRegression()

```
accuracy = diabetesCheck.score(testData, testLabel)
print("accuracy = ", accuracy * 100, "%")
```

accuracy = 78.0 %

Result: Thus, the python program for bivariate analysis: logistic regression modeling was executed successfully.

Exp No. 5C	Multiple Regression analysis
Date:	

Aim:

To write a python program to find Multiple Regression analysis.

Algorithm:

Step 1: Start the program

Step 2: import the library

- *NumPy* — used to perform mathematical operations mainly using multi-dimensional arrays.
- *pandas* — used for data manipulation and analysis.
- *matplotlib* — it is a plotting library as a component of NumPy
- *statsmodels* — it is used to explore data, estimate statistical models and perform statistical tests.

Step 3: Create a data frame using the function `pd.DataFrame()`

Step 4: Import the dataset and load the data.

Step 5: We can perform descriptive statistics using the following command: `Dataframe.describe()`

Step 6: Create Your First multiple Regression and Declare the Dependent and Independent Variables.

Step 7: To plot the regression line on the graph, use Pearson method of correlation in Jupyter.

Step 8: Finally, plot the regression line using `matplotlib.pyplot()`

Step 9: Stop the program.

Program

```
import pandas as pd
```

```
data = pd.read_csv("diabetes.csv")
```

```
df = pd.DataFrame(data)
```

```
print(df)
```

```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0             6       148             72           35     0  33.6
1             1        85             66           29     0  26.6
2             8       183             64            0     0  23.3
3             1        89             66           23    94  28.1
4             0       137             40           35   168  43.1
..          ...      ...          ...          ...      ...  ...
763          10       101             76           48   180  32.9
764           2       122             70           27     0  36.8
765           5       121             72           23   112  26.2
766           1       126             60            0     0  30.1
767           1        93             70           31     0  30.4
```

```
DiabetesPedigreeFunction  Age  Outcome
0             0.627      50         1
1             0.351      31         0
2             0.672      32         1
3             0.167      21         0
4             2.288      33         1
```

```

...
763          0.171    63          0
764          0.340    27          0
765          0.245    30          0
766          0.349    47          1
767          0.315    23          0

[768 rows x 9 columns]
y = df['Pregnancies'].values

print(y)

x = df.drop(['Pregnancies'],axis = 1).values

print(x)

[ 6  1  8  1  0  5  3 10  2  8  4 10 10  1  5  7  0  7  1  1  3  8  7  9
11 10  7  1 13  5  5  3  3  6 10  4 11  9  2  4  3  7  7  9  7  0  1  2
 7  7  1  1  5  8  7  1  7  0  0  0  2  8  5  2  7  5  0  2  1  4  2  5
13  4  1  1  7  5  0  2  3  2  7  0  5  2 13  2 15  1  1  4  7  4  2  6
 2  1  6  1  1  1  0  1  2  1  1  4  3  0  3  8  1  4  7  4  5  5  4  4
 0  6  2  5  0  1  3  1  1  0  4  9  3  8  2  2  0  0  0  5  3  5  2 10
 4  0  9  2  5  2  1  4  9  1  8  7  2  1  2 17  4  7  0  2  0  6  3  4
 4  3  6  6  2  1  2  8  6  0  5  5  6  0  1  5  4  7  8  1  8  5  3  9
 7 11  8  5  1  3  4  4  0  1  0  2  6  5  8  5  1  7  2  0  7  0  9 12
 5  6  5  5  0  2  7  7  1  1  0  3  4  0  4  6  1  4  3  4  7  0  9  0
 1  4  3  6  2  9 10  0  9  1  9  2  2  0 12  1  3  2  1 11  3  3  4  3
 4  5  0  2  0  2 10  2  3  1 13  2  7  0  5  2  0 10  7  7  2  7  5  1
 4  5  0  0  2  1  0  6  2  0 14  8  0  2  5  5  3  2 10  0  0  2  6  0
 2  3  7  2  3  3  3  6  4  3  0 13  2  1  1 10  2  6  8  2  1 12  1  0
 0  5  9  7  1  1  1  5  8  8  1  3  3  5  4  4  3  1  3  9  1 13 12  1
 5  5  5  4  4  5  6  0  3  1  3  0  0  2  2 12  0  1  4  0  1  0  1  1
 1  1  5  8  5  3  1  5  1  4  4  2  3  0  3  3  4  6  5  9  5  2  4  0
 8  1  6  1  1  1  0  3  1  4  1  3  1  2  0  2  8  4  0  1  0  1  2  3
 1  2  1  0 12  5  1  6  0  2  4  8  4  0  1  0  0  0  1  2  0  2  2 14
 1  5 10  9  9  1  8  5 10  0  0  0  8  6  1  0  0  7  4  0  2  7  8  4
 3  0  4  0  0  0  1  0  4  8  2  2  4  4  3  6  5  2  7  6  2  3  6  7
 3 10  0  1  2  8 12  0  9  2  3  3  9  7 13  6  2  3  6  9  3  3  1  3
 0  0  2  0  1  6  1  4  0  0  0  3  8  3 10  4  1  8  5  4  1  4  1  3
 6  1  1  7  1  8 11 11  6  0  1  6  0  2  1  6  4  0  3  2  3  2  1  1
 6  2 10  2  0  6 12  8  8  1  8  6  3  0 11  2  3  2  6  0  0  1  1  1
 1  6  1  7  4  1  1  1  0  1  3  3  7  6 11  3  6  2  9  0  2  2  6  0
 2  4  0  0  5  4  7  0  2  1 10 13  5  2  7  1  0  4  6  4  3  2  1  0
11  0  1  1  5  2  1  2  2  1 11  3 10  1  8  9  6  1  4 10  6  9  6  1
10  3  8  6  9  0  3  2  2  0  0  4  5  2  3  1  1  1  8 13  2  7  2  7
 3  0  4  4  2  6  1  2  4  6 10  2  9  2  3  5 10  0  3  7  3 10  1  5
 4  1  1  5  1  4  1  0  2  2  3  8  2  2  2  4  0  8  2  1 11  3  1  9
13 12  1  1  3  6  4  1  3  0  8  1  7  0  1  6  2  9  9 10  2  5  1 11]

[[148.    72.    35.    ...    0.627    50.    1.    ]
 [ 85.    66.    29.    ...    0.351    31.    0.    ]
 [183.    64.     0.    ...    0.672    32.    1.    ]
 ...
 [121.    72.    23.    ...    0.245    30.    0.    ]
 [126.    60.     0.    ...    0.349    47.    1.    ]
 [ 93.    70.    31.    ...    0.315    23.    0.    ]]

```

```

from scipy import stats

corr = stats.pearsonr(df['Age'], df['BloodPressure'])

corr2 = stats.pearsonr(df['Age'], df['Outcome'])

```

```
import numpy as np
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x, y)
coefIndep = model.coef_
coefDep = model.intercept_
from sklearn.metrics import r2_score
r2_score(y,model.predict(x))
```

Output:

```
0.3156531599136111
```

Result:

Thus, the python program for multiple regression was executed successfully.

Exp No. 6A	Normal curves
Date:	

Aim:

To write a Python Program to plot a normal curve using UCI dataset.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib.pyplot to plot the graph.

Step 3: Create a data frame using the function pd.DataFrame()

Step 4: Import the dataset and load the data.

Step 5: Create the two arrays x1 and y1. To plot the line graph use the function plt.plot().

Step 6: Create another arrays such as x2 and y2.

Step 7: Print the graph using plt.show() function.

Step 8: Stop

Program:

```
import pandas as pd
```

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

```
df = pd.read_csv('iris.csv')
```

```
type(df)
```

```
pd.set_option("display.max.columns", None)
```

```
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
%matplotlib
```

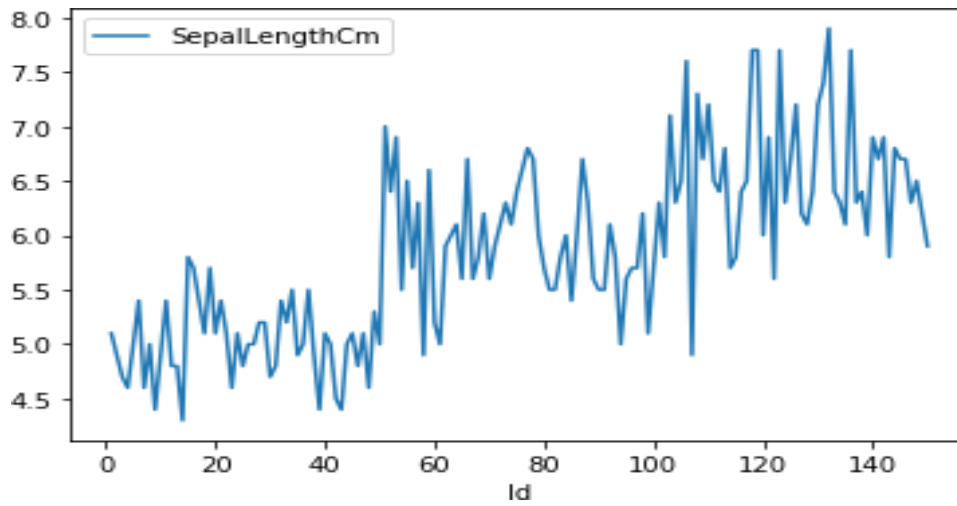
```
Out: Using matplotlib backend: Qt5Agg
```

```
%matplotlib inline
```

```
df.plot(x="Id", y="SepalLengthCm")
```

```
plt.show()
```

Output:



Result:

Thus, the python program for normal curve was executed successfully.

Exp No. 6B	Density and contour plots
Date:	

Aim:

To write a Python Program to plot density and contour plots using UCI dataset.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib.pyplot to plot the graph.

Step 3: Import the library numpy, pandas, scipy and seaborn

Step 4: Create the two random arrays X and Y.

Step 5: Scatter plot the array X and Y using plt.scatter() function.

Step 6: Set the x axis, y axis. And call function sns.kdeplot().

Step 6: Print the graph using plt.show() function.

Step 7: For contour plot ,create a data frame for the given data.

Step 8: Set the x axis, y axis. And call function sns.scatterplot().

Step 9: Set the x axis, y axis. And call function sns.kdeplot().

Step 10: Stop

Density plots

```
import seaborn as sns
```

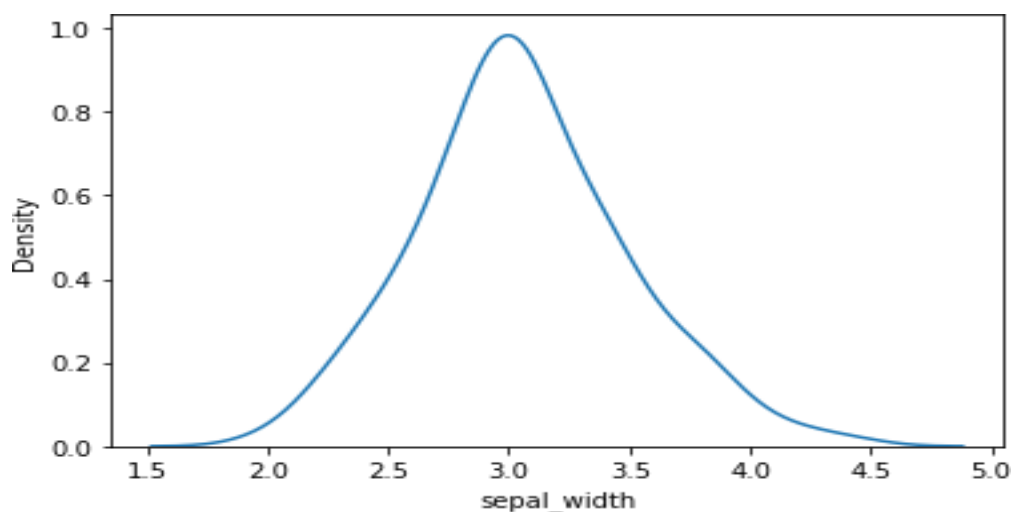
```
df = sns.load_dataset('iris.csv')
```

```
# Make default density plot
```

```
sns.kdeplot(df['sepal_width'])
```

```
out: <AxesSubplot:xlabel='sepal_width', ylabel='Density'>
```

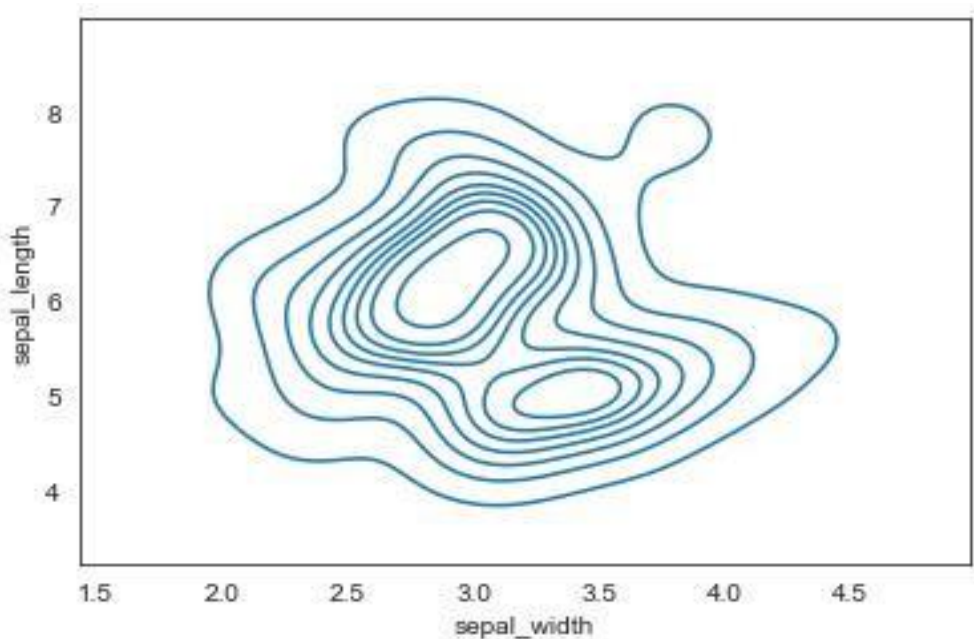
Output:

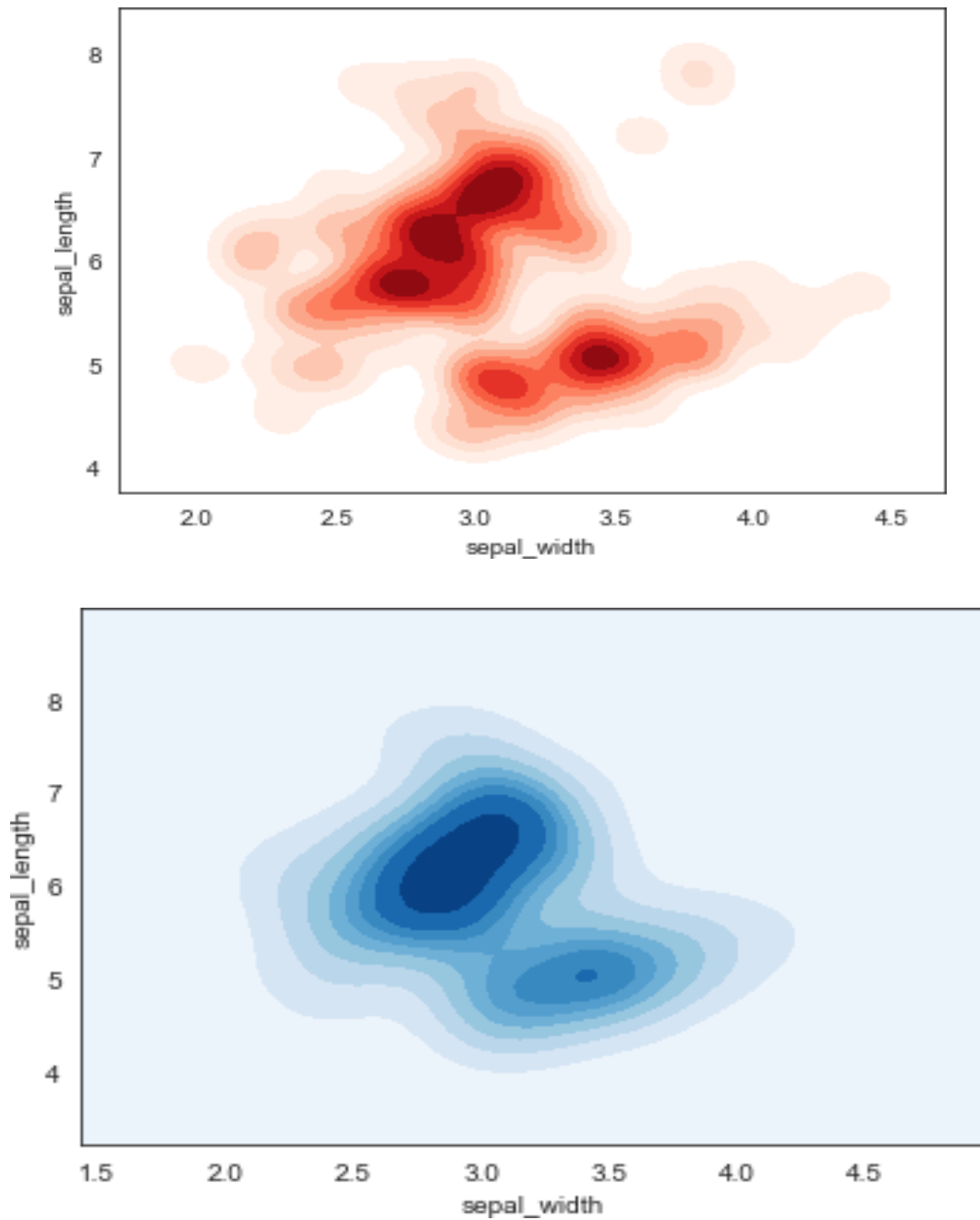


Contour plots

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('iris')
# set seaborn style
sns.set_style("white")
# Basic 2D density plot
sns.kdeplot(x=df.sepal_width, y=df.sepal_length)
plt.show()
# Custom the color, add shade and bandwidth
sns.kdeplot(x=df.sepal_width, y=df.sepal_length, cmap="Reds", shade=True, bw_adjust=.5)
plt.show()
# Add thresh parameter
sns.kdeplot(x=df.sepal_width, y=df.sepal_length, cmap="Blues", shade=True, thresh=0)
plt.show()
```

Output:





Result:

Thus, the python program to plot density and contour plots was executed successfully.

Exp No. 6C	Correlation and scatter plots
Date:	

Aim:

To write a Python Program to plot correlation and scatter plots using UCI dataset.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib.pyplot to plot the graph.

Step 3: Import the library numpy, pandas, scipy and seaborn

Step 4: Create the two random arrays X and Y.

Step 5: Scatter plot the array X and Y using plt.scatter() function.

Step 6: Print the graph using plt.show() function.

Step 7: Create a data frame for the given data.

Step 8: Set the x axis, y axis. And call function sns.scatterplot().

Step 9: Add the linear model fit line to a scatterplot using sea born's lmlot() method.

Step 10: Stop

Scatter plots

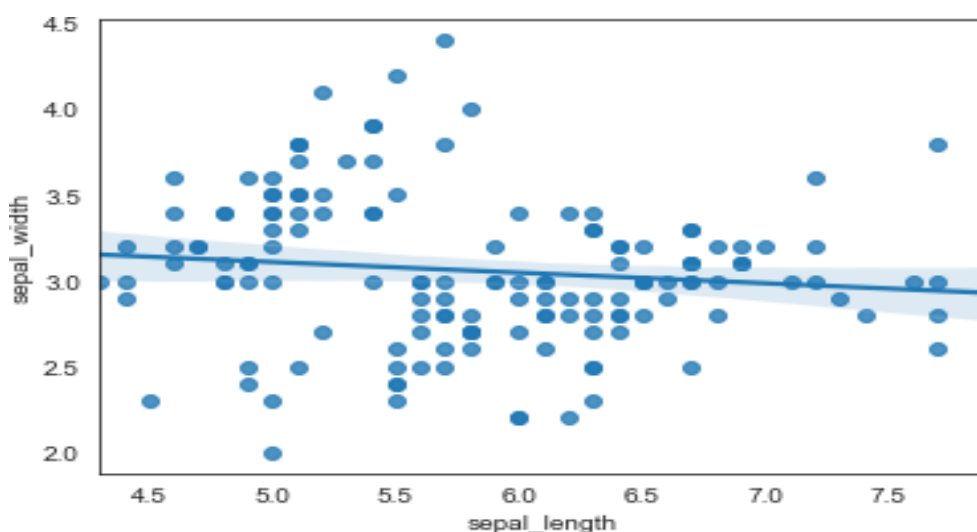
```
import seaborn as sns
```

```
df = sns.load_dataset('iris')
```

```
# use the function regplot to make a scatterplot
```

```
sns.regplot(x=df["sepal_length"], y=df["sepal_width"])
```

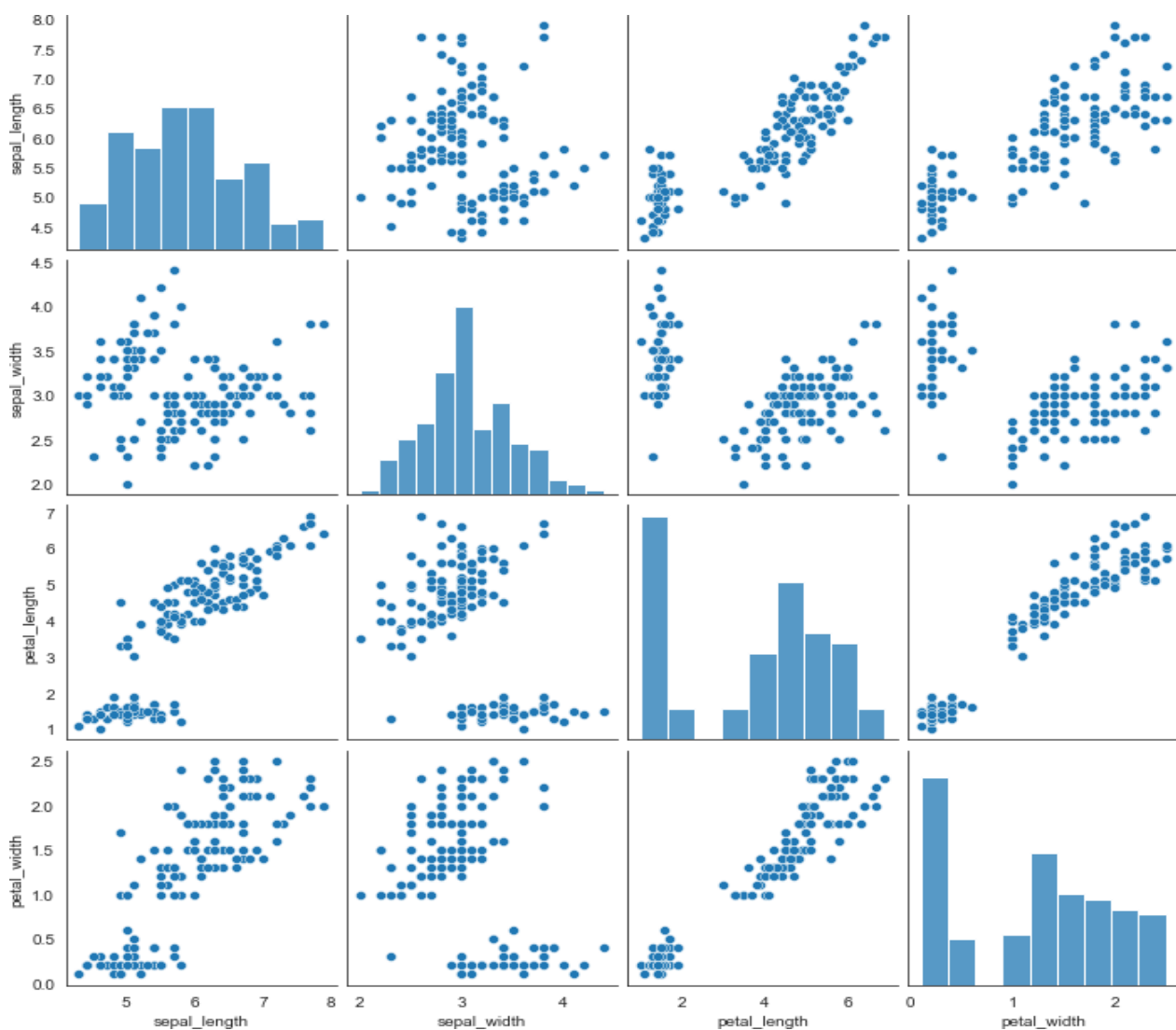
```
out : <AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>
```



Correlation plots

```
import seaborn as sns  
df = sns.load_dataset('iris')  
import matplotlib.pyplot as plt  
# Basic correlogram  
sns.pairplot(df)  
sns.plt.show()
```

Output:



Result:

Thus, the python program to plot density and contour plots was executed successfully.

Exp No. 6D	Histograms
Date:	

Aim:

To write a Python Program to plot histograms using UCI dataset.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib. pyplot to plot the graph.

Step 3: Create two lists, one represents the programming language and another represents its popularity.

Step 4: Plot the bar graph use the function plt.bar().

Step 5 :Denote the label for x and y axis using xlabel() and ylabel() function. And add title using plt.title().

Step 6: Plot the horizontal bar graph by plt.barh() function.

Step 7: Create a list with the given data. And also the bin values.

Step 8: Plot the histogram using the function plt.hist() and Print the graph using plt.show() function

Step 9: Stop

Histograms

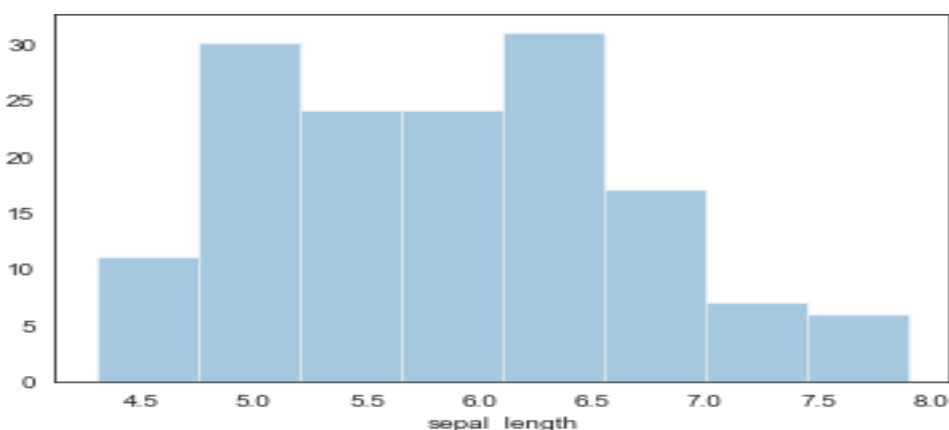
```
import seaborn as sns
```

```
df = sns.load_dataset('iris')
```

```
# Plot the histogram thanks to the distplot function
```

```
sns.distplot( a=df["sepal_length"], hist=True, kde=False, rug=False )
```

Output:



Result:

Thus, the python program for histogram was executed successfully.

Exp No. 6E	Three dimensional plotting
Date:	

Aim:

To write a Python Program to plot three dimensional plotting using UCI dataset.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib. pyplot to plot the graph.

Step 3: Create two lists, one represents the programming language and another represents its popularity.

Step 4: Plot the bar graph use the function plt.bar().

Step 5 :Denote the label for x and y axis using xlabel() and ylabel() function. And add title using plt.title().

Step 6: Plot the horizontal bar graph by plt.barh() function.

Step 7: Create a list with the given data. And also the bin values.

Step 8: Plot the histogram using the function plt.hist() and Print the graph using plt.show() function

Step 9: Stop

Program:

```
import matplotlib.pyplot as plt
# unused but required import for doing 3d projections with matplotlib < 3.2
import mpl_toolkits.mplot3d # noqa: F401
from sklearn import datasets
from sklearn.decomposition import PCA
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
y = iris.target
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
plt.figure(2, figsize=(8, 6))
plt.clf()
# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```

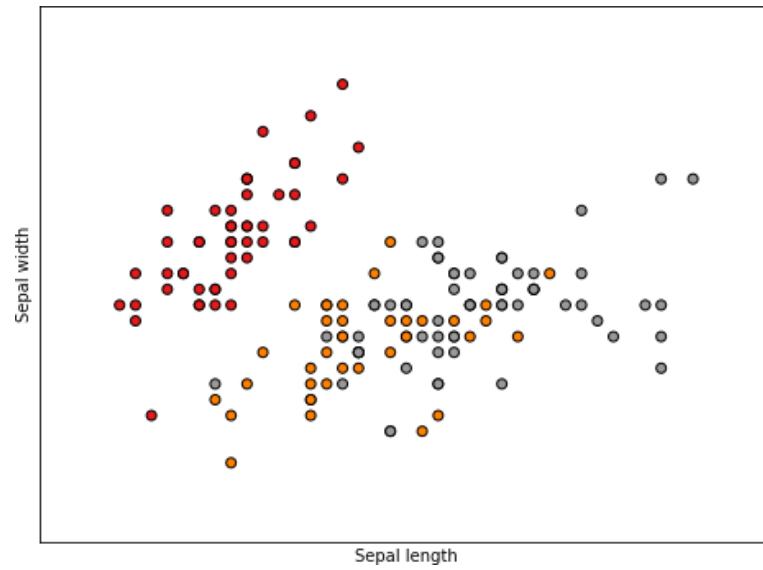
```

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks()
plt.yticks()

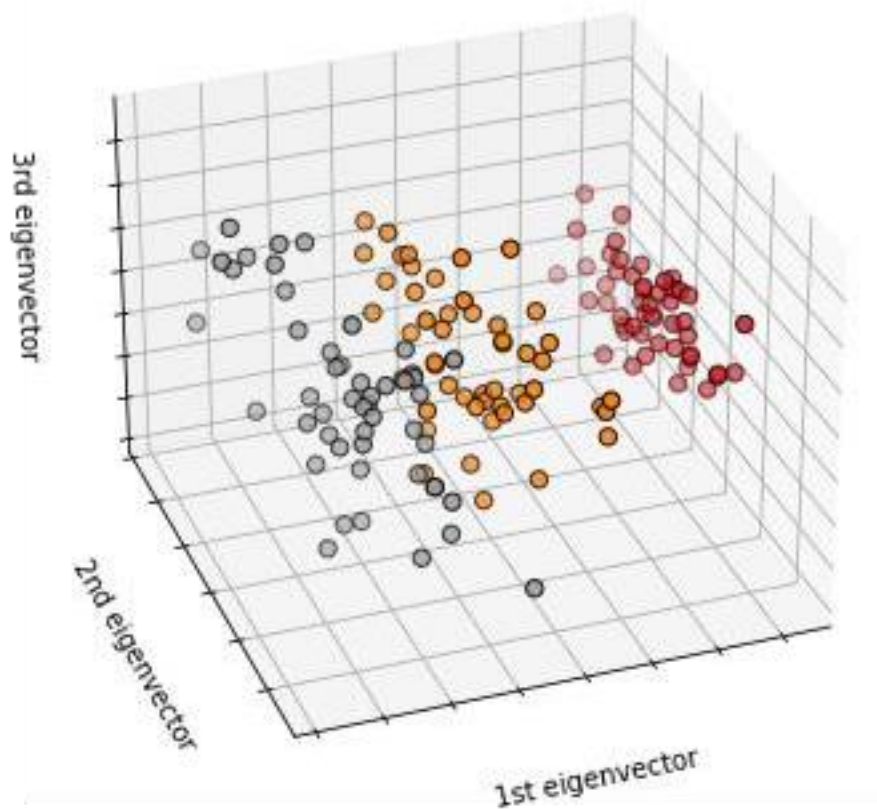
# To get a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azimuth=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=y,
    cmap=plt.cm.Set1,
    edgecolor="k",
    s=40,
)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])
plt.show()

```

Output:



First three PCA directions



Result:

Thus, the python program for three dimensional plotting was executed successfully.

Exp No. 7	Visualizing Geographic Data with Basemap
Date:	

Aim:

To write a Python Program to Visualize Geographic Data with Basemap.

Algorithm:

Step 1: Start

Step 2: Import the library matplotlib. pyplot to plot the graph.

Step 3: Import Basemap. Installation of Basemap is straightforward; if you're using conda you can type this and the package will be downloaded: \$ conda install basemap

Step 4: The Basemap toolkit installed and imported, geographic plots are just a few lines away also require the pillow package in Python 3.

Step 5: Map (long, lat) to (x, y) for plotting.

Step 6 : Draw a shaded-relief image,lats and longs are returned as a dictionary.

Step 7: Map keys contain the plt.Line2D instances and cycle through these lines and set the desired style.

Step 8: Stop

Program:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
-----
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```



```

fig = plt.figure(figsize=(8, 8))
m= Basemap(projection='lcc',resolution=None,width=8E6,height=8E6,lat_0=45,lon_0=-100,)
m.etopo(scale=0.5, alpha=0.5)
# Map (long, lat) to (x, y) for plotting
x, y = m(-122.3, 47.6)
plt.plot(x, y, 'ok', markersize=5)
plt.text(x, y, ' Seattle', fontsize=12);

#Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

```



```

-----
from itertools import chain
def draw_map(m, scale=0.2):
    # draw a shaded-relief image
    m.shadedrelief(scale=scale)
    # lats and longs are returned as a dictionary
    lats = m.drawparallels(np.linspace(-90, 90, 13))
    lons = m.drawmeridians(np.linspace(-180, 180, 13))
    # keys contain the plt.Line2D instances
    lat_lines = chain(*(tup[1][0] for tup in lats.items()))
    lon_lines = chain(*(tup[1][0] for tup in lons.items()))
    all_lines = chain(lat_lines, lon_lines)
    # cycle through these lines and set the desired style
    for line in all_lines:

```

```
line.set(linestyle='-', alpha=0.3, color='w')
```

```
-----
```

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')  
m = Basemap(projection='cyl', resolution=None,  
llcrnrlat=-90, urcrnrlat=90,  
llcrnrlon=-180, urcrnrlon=180, )  
draw_map(m)
```



```
fig = plt.figure(figsize=(8, 6), edgecolor='w')  
m = Basemap(projection='moll', resolution=None,  
lat_0=0, lon_0=0)  
draw_map(m)
```



```
fig = plt.figure(figsize=(8, 8))  
m = Basemap(projection='ortho', resolution=None,  
lat_0=50, lon_0=0)  
draw_map(m);
```



```
fig = plt.figure(figsize=(8, 8))  
m = Basemap(projection='lcc', resolution=None,  
lon_0=0, lat_0=50, lat_1=45, lat_2=55,width=1.6E7, height=1.2E7)  
draw_map(m)
```



Result:

Thus, the python program for Visualizing Geographic Data with Basemap was executed successfully.

ADDITIONAL EXPERIMENTS

Exp No. 1	Normal distribution for two dimension array and to use sea born library for visualization of curve.
Date:	

Aim:

To write a python program to calculate the normal distribution for two-dimension array and to use sea born library for visualization of curve.

Algorithm:

Step 1: Import the library numpy, pandas, matplotlib, statsmodels and seaborn.

Step 2: Create an numpy array with size (2,3).

Step 3: Calculate the distribution values by “.normal()”.

Step 4: To calculate the distribution with mean and Standard deviation use “.normal(loc=mean, scale, size_array)” function.

Step 5: Plot the distribution curve by “sns.distplot()” function.

Program code:

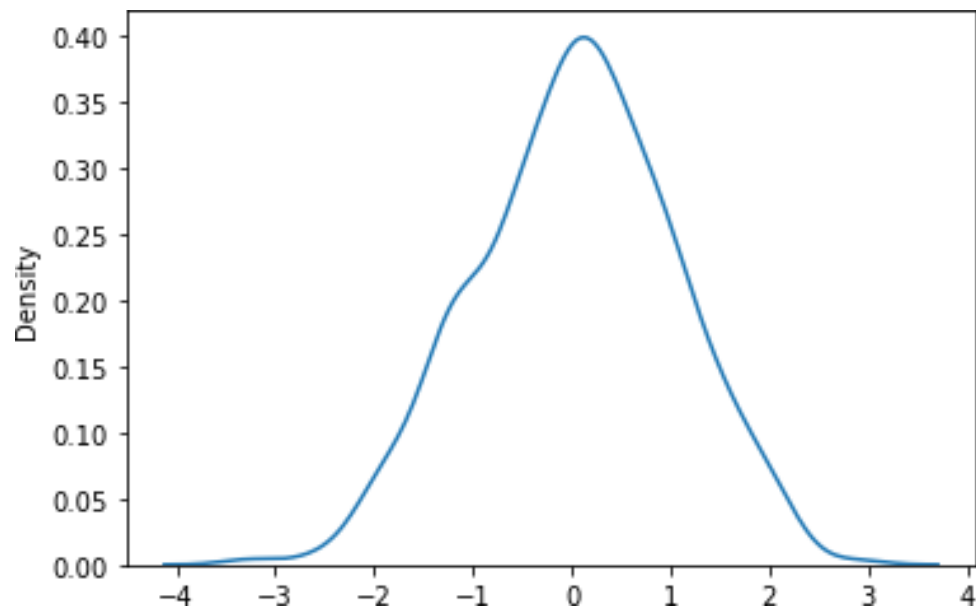
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
x1 = random.normal( size = (2,3))
print(x1)
# with mean 1 and standard deviation of 2
x2=random.normal(loc=1,scale=2,size=(2,3))
print(x2)
#visualization:
sns.distplot(random.normal(size=1000),hist=False)
plt.show()
```

Output:

```
[[ 1.36897055  1.11968637 -1.04295062]
 [-0.56357028 -0.5674646  1.62733457]]
```

```
[[ 2.37046635  1.85176364 -1.22936495]
```

```
[-0.34335274  3.195814  -0.87871991]]
```



Result:

Thus, the python program to calculate normal distribution for two-dimension array was executed successfully.

Exp No. 2	CORRELATION AND SCATTER PLOTS
Date:	

Correlation:

Correlation is simple with seaborn and pandas. It is a way to determine if two variables in a dataset are related in any way.

If features are strongly correlated we can remove them to prevent duplicating information. In data science, we can see the r values, also called Pearson's correlation. This measures how closely two sequences of number are correlated. The r value is a number between -1 to +1.

1. Write a python program to find the correlation between variables of given data. Also create a heatmap using seaborn to present their relations.

Sample data:

Temperature: 3, 39, 46, 45, 34, 38, 30, 22, 34, 38, 31, 14, 27, 39

wind speed: 4.3, 3.86, 9.73, 6.86, 7.8, 8.7, 16.46, 3.1, 9.1, 9.76, 17.9, 2.2, 8.23, 11.26

Aim:

To write a python program to find correlation between temperature and wind speed and to plot scatter plot and heat map of the correlation.

Algorithm:

Step 1: Load the library pandas, scipy and seaborn

Step 2: Create a data frame for the given data.

Step 3: Set the x axis, y axis. And call function sns.scatterplot().

Step 4: Add the linear model fit line to a scatterplot using sea born's lmplot() method.

Step 5: Calculate the pearson correlation coefficient values for x and y using the method pearsonr(), which is provided by scipy library.

Program :

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
weather={'temp':[3,39,46,45,34,38,30,22,34,38,31,14,27,39],
        'wind_speed':[4.3,3.86,9.73,6.86,7.8,8.7,16.46,3.1,9.1,9.76,17.9,2.2,8.23,11.26]}
weather=pd.DataFrame(weather)
#Scatter plot
ax = sns.scatterplot(x="temp", y="wind_speed", data=weather)
ax.set_title("Temperaure Vs Wind_speed")
```

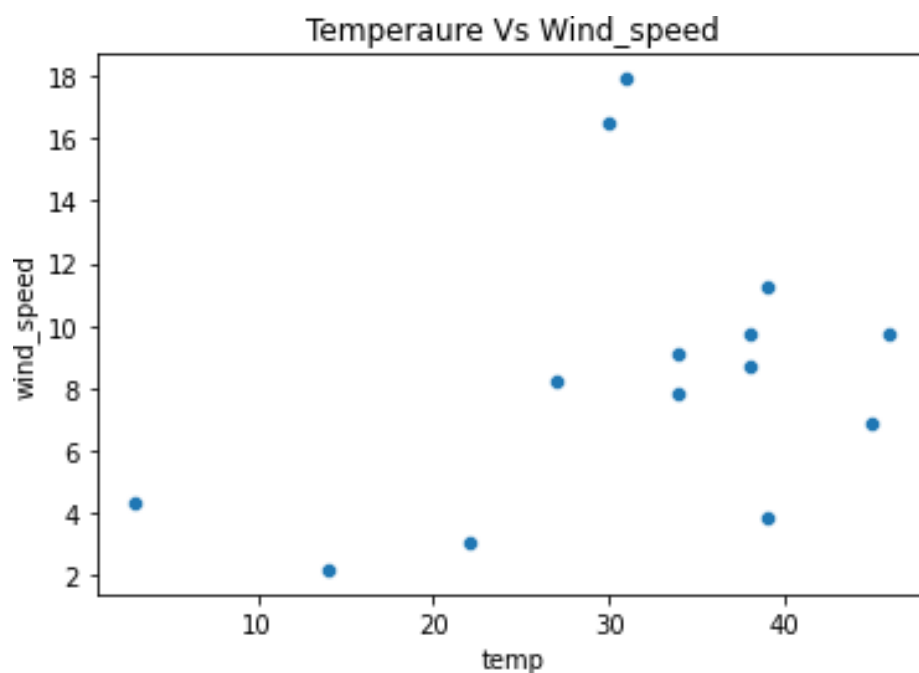
```

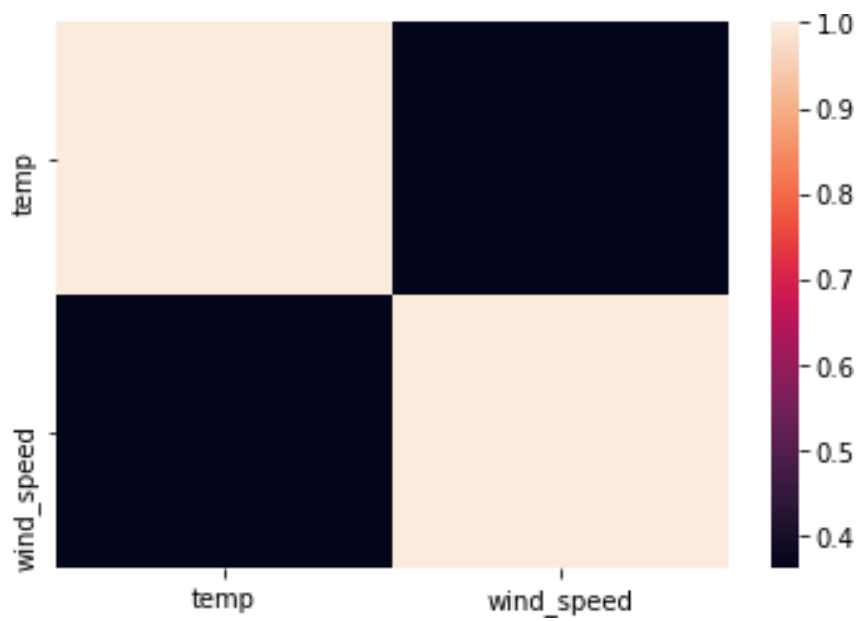
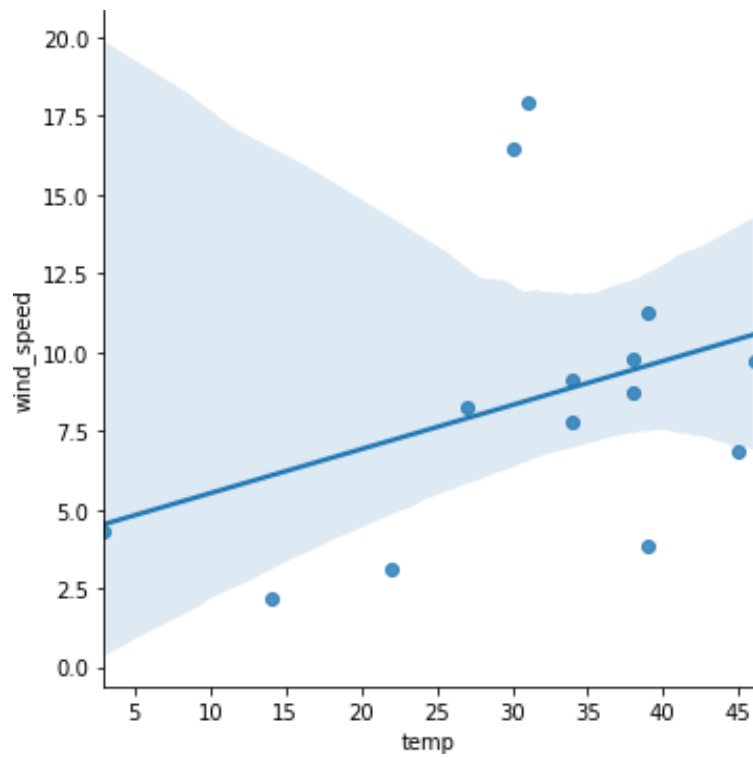
#adding best fit line
sns.lmplot(x="temp",y="wind_speed", data=weather)
plt.show()
#finding correlation between temp and wind speed
stats.pearsonr(weather['temp'],weather['wind_speed'])
cormat = weather.corr()
round(cormat,2)
print(cormat)
#correlation matrix to heatmap
sns.heatmap(cormat)

```

Output:

	temp	wind_speed
temp	1.000000	0.362074
wind_speed	0.362074	1.000000





Result:

Thus, python program to find correlation between temperature and wind speed and to plot scatter plot and heat map of the correlation was executed successfully.

Ex.No:3	Implement Navie Bayes Models
Date:	

Aim:

To implement navie bayes using navie classifier methods

Procedure:

- Step 1 - Import basic libraries.
- Step 2 - Importing the dataset.
- Step 3 - Data preprocessing.
- Step 4 - Training the model.
- Step 5 - Testing and evaluation of the model.
- Step 6 - Visualizing the model.

Program:

```
# import necessary libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Load Data from CSV
data = pd.read_csv('tennisdata.csv')
print("The first 5 Values of data is :\n", data.head())
```

The first 5 Values of data is :

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

obtain train data and train output

```
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())
```

The First 5 values of the train data is

	Outlook	Temperature	Humidity	Windy
0	Sunny	Hot	High	Weak
1	Sunny	Hot	High	Strong
2	Overcast	Hot	High	Weak
3	Rain	Mild	High	Weak

4 Rain Cool Normal Weak

```
y = data.iloc[:, -1]
print("\n\nThe First 5 values of train output is\n", y.head())
```

The First 5 values of train output is

0 No

1 No

2 Yes

3 Yes

4 Yes

Name: PlayTennis, dtype: object

convert them in numbers

```
le_outlook = LabelEncoder()
```

```
X.Outlook = le_outlook.fit_transform(X.Outlook)
```

```
le_Temperature = LabelEncoder()
```

```
X.Temperature = le_Temperature.fit_transform(X.Temperature)
```

```
le_Humidity = LabelEncoder()
```

```
X.Humidity = le_Humidity.fit_transform(X.Humidity)
```

```
le_Windy = LabelEncoder()
```

```
X.Windy = le_Windy.fit_transform(X.Windy)
```

```
print("\n\nNow the Train output is\n", X.head())
```

Now the Train output is

	Outlook	Temperature	Humidity	Windy
--	---------	-------------	----------	-------

0	2	1	0	1
---	---	---	---	---

1	2	1	0	0
---	---	---	---	---

2	0	1	0	1
---	---	---	---	---

3	1	2	0	1
---	---	---	---	---

4	1	0	1	1
---	---	---	---	---

```
le_PlayTennis = LabelEncoder()
```

```
y = le_PlayTennis.fit_transform(y)
```

```
print("\n\nNow the Train output is\n", y)
```

Now the Train output is

```
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

```
classifier = GaussianNB()
```

```
classifier.fit(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score
```

```
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

Output:

Accuracy is: 0.3333333333333333

Result:

Thus, the naive baye program was executed and output is verified.

VERSION - II

Ex.No:4	Correlation coefficient
Date:	

AIM:

To write a python program to compute correlation coefficient.

ALGORITHM:

- Step 1: Start the Program
- Step 2: Import math package
- Step 3: Define correlation coefficient function
- Step 4: Calculate correlation using formula
- Step 5: Print the result
- Step 6 : Stop the process

PROGRAM:

Python Program to find correlation coefficient.

import math

function that returns correlation coefficient.

def correlationCoefficient(X, Y, n) :

sum_X = 0

sum_Y = 0

sum_XY = 0

squareSum_X = 0

squareSum_Y = 0

i = 0

while i < n :

sum of elements of array X.

sum_X = sum_X + X[i]

sum of elements of array Y.

sum_Y = sum_Y + Y[i]

```

# sum of X[i] * Y[i].
sum_XY = sum_XY + X[i] * Y[i]

# sum of square of array elements.
squareSum_X = squareSum_X + X[i] * X[i]
squareSum_Y = squareSum_Y + Y[i] * Y[i]

i = i + 1

# use formula for calculating correlation
# coefficient.
corr = (float)(n * sum_XY - sum_X * sum_Y)/
(float)(math.sqrt((n * squareSum_X -
sum_X * sum_X)* (n * squareSum_Y -
sum_Y * sum_Y)))
return corr

# Driver function
X = [15, 18, 21, 24, 27]
Y = [25, 25, 27, 31, 32]

# Find the size of array.
n = len(X)

# Function call to correlationCoefficient.
print ('{0:.6f}'.format(correlationCoefficient(X, Y, n)))

```

Output :

0.953463

RESULT:

Thus, the computation for correlation coefficient was successfully completed