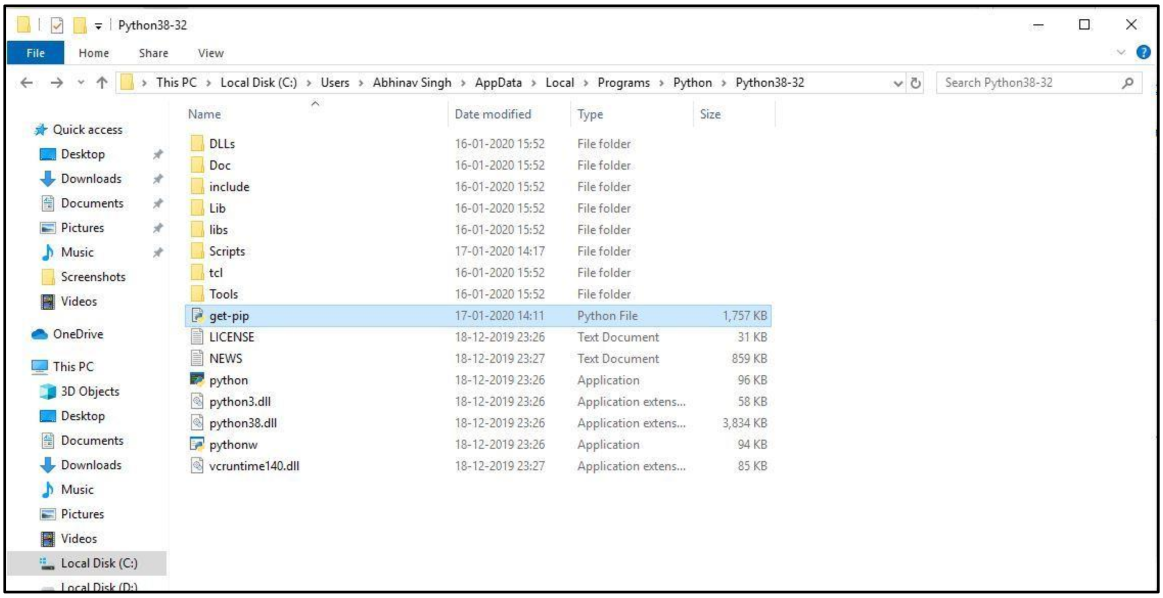
Experiment 1:
How to Install PIP on Windows?

Before we start with how to install pip for Python on Windows, let's first go through the basic introduction to Python. Python is a widely-used general-purpose, high-level programming language. Python is a programming language that lets you work quickly and integrate systems more efficiently.
PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI).
pip uses PyPI as the default source for packages and their dependencies. So whenever you type:

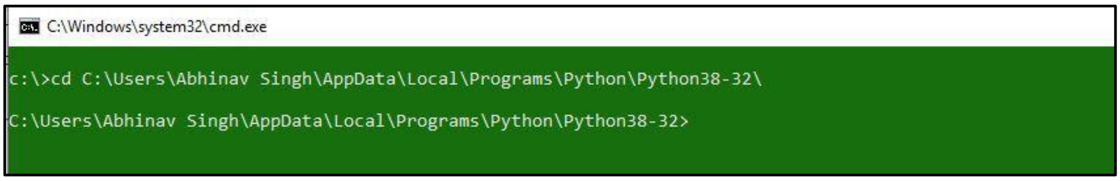
Download and Install pip:

pip can be downloaded and installed using command-line by going through the following steps:

- Download the **get-pip.py** file and store it in the same directory as python is installed.



- Change the current path of the directory in the command line to the path of the directory where the above file exists.



- Run the command given below:
  python get-pip.py

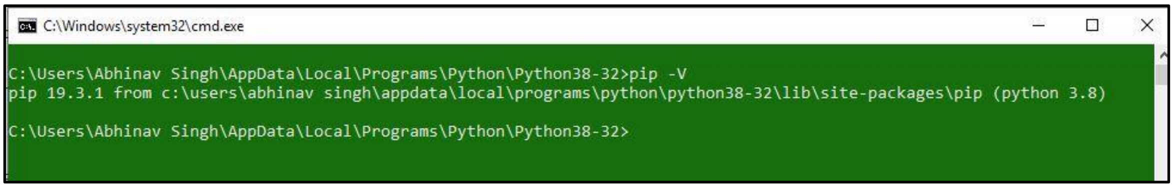
  and wait through the installation process.

- pip is now installed on your system.

Verification of the Installation process:

One can easily verify if the pip has been installed correctly by performing a version check on the same. Just go to the command line and execute the following command:

pip -V



To Install Various Packages using PIP :

Syntax : pip install <package_name>

pip will look for that package on PyPI and if found, it will download and install the package on your local system.

Packages :

a) Numpy:

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

NumPy – A Replacement for MatLab

NumPy is often used along with packages like **SciPy** (Scientific Python) and **Mat−plotlib** (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.

It is open source, which is an added advantage of NumPy.

b) Scipy:

SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations.

The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

SciPy Sub-packages

SciPy is organized into sub-packages covering different scientific computing domains. These are summarized in the following table −

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

c) matplotlib

- **plot(x, y):** plot x and y using default line style and color.
- **plot.axis([xmin, xmax, ymin, ymax])**: scales the x-axis and y-axis from minimum to maximum values
- **plot.(x, y, color='green', marker='o', linestyle='dashed', linewidth=2, markersize=12):** x and y co-ordinates are marked using circular markers of size 12 and green color line with — style of width 2
- **plot.xlabel('X-axis')**: names x-axis
- **plot.ylabel('Y-axis')**: names y-axis
- **plot(x, y, label = 'Sample line ')** plotted Sample Line will be displayed as a legend

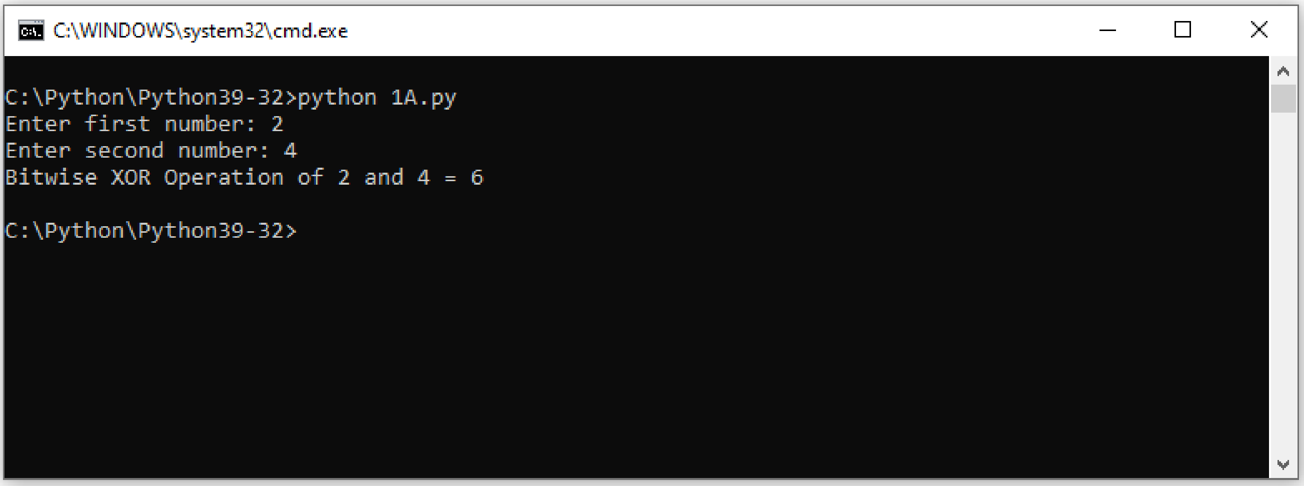
d) scikit-learn

Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling. Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, and statistical tools for analyzing these models. It also provides functionality for dimensionality reduction, feature selection, feature extraction, ensemble techniques, and inbuilt datasets. We will be looking into these features one by one.

This library is built upon NumPy, SciPy, and Matplotlib.

Write a program to read two numbers from user and display the result using bitwise & , | and ^ operators on the numbers

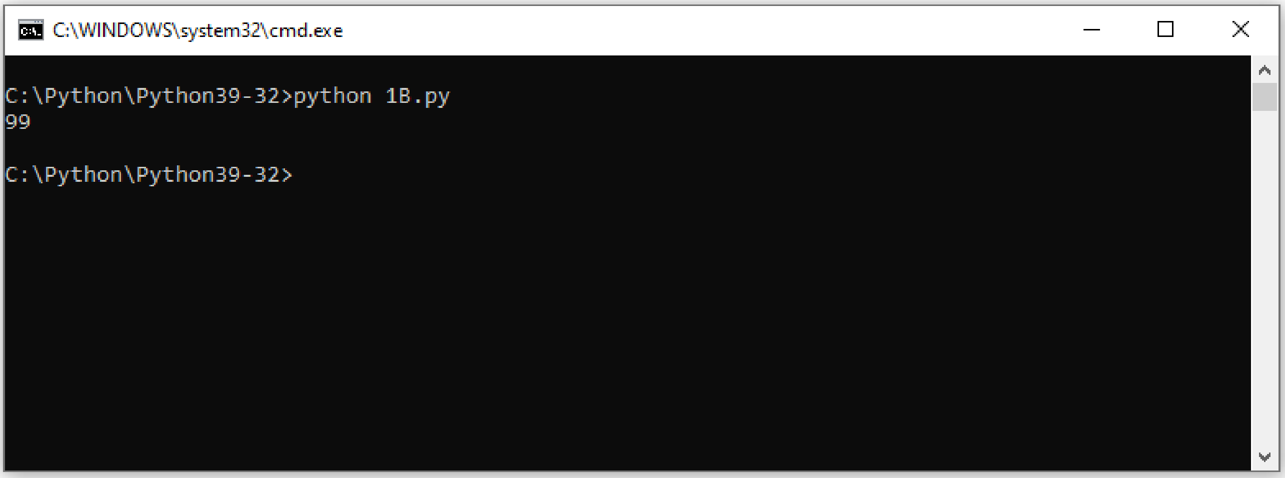
```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = a^b
print ("Bitwise XOR Operation of", a, "and", b, "=", c)
```



Write a program to calculate the sum of numbers from 1 to 20 which are not divisible by 2, 3 or 5.

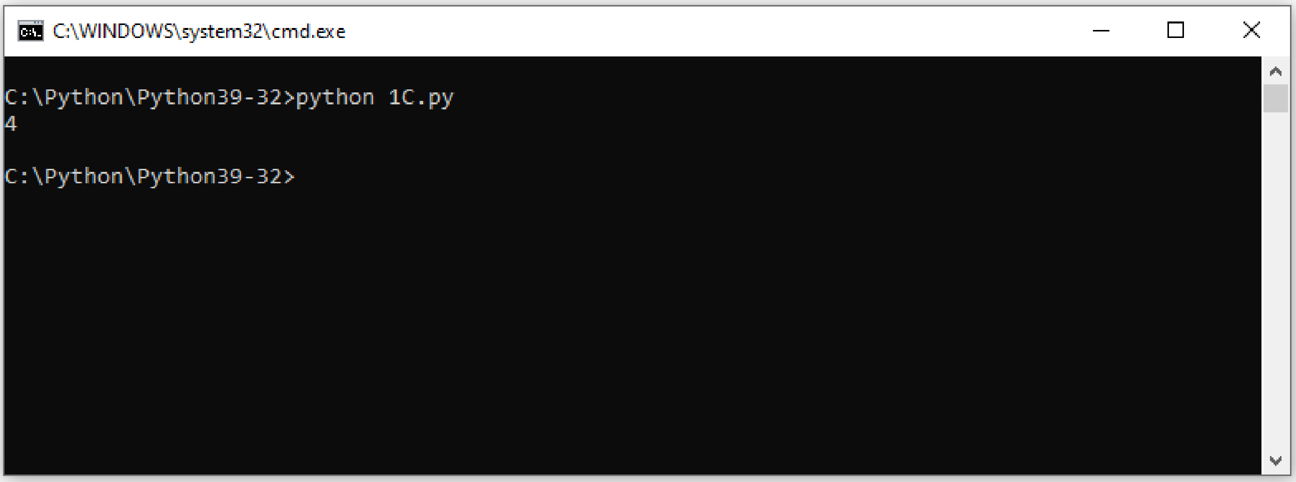
```
   def findSum(n, k):
# Find the last multiple of N
       val = (k // (n - 1)) * n;
       rem = k % (n - 1);
       # Find the K-th non-multiple of N
       if (rem == 0):
           val = val - 1;
       else:
           val = val + rem;
      # Calculate the sum of
       # all elements from 1 to val
       sum = (val * (val + 1)) // 2;
       # Calculate the sum of
       # all multiples of N
       # between 1 to val
       x = k // (n - 1);
```

```
        sum_of_multiples = (x * (x + 1) * n) // 2;
        sum -= sum_of_multiples;
        return sum;
    # Driver code
    n = 7; k = 13;
    print(findSum(n, k))
```



Write a program to find the maximum of two numbers using functions.

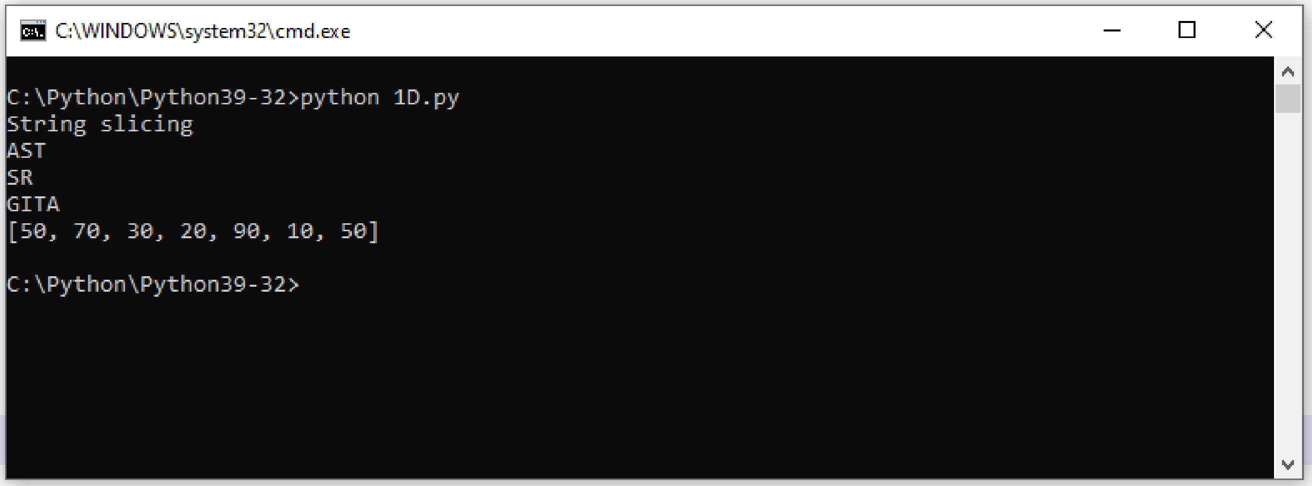
```
def maximum(a, b):

    if a >= b:
        return a
    else:
        return b

# Driver code
a = 2
b = 4
print(maximum(a, b))
```



Implement slicing operation on strings and lists.

```
# String slicing
String ='ASTRING'
# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
```

```
s3 = slice(-1, -12, -2)
print("String slicing")
print(String[s1])
print(String[s2])
print(String[s3])
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]
# Display list
print(Lst[-7::1])
```

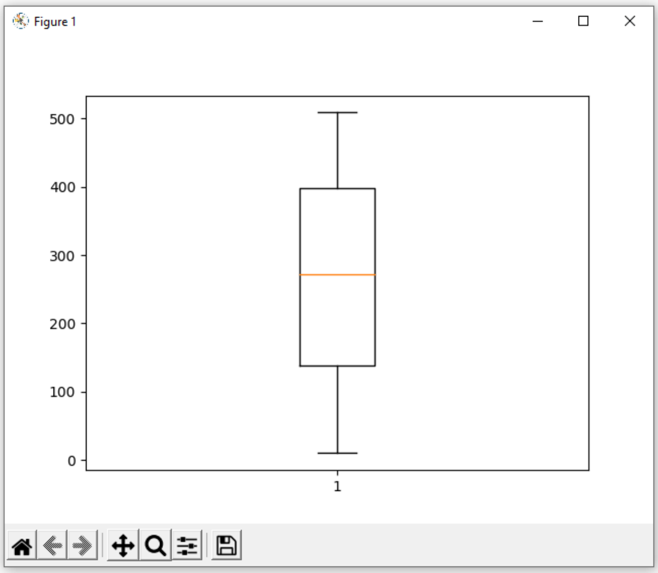


Experiment 2:

Implement python program to load structured data onto Data Frame and perform exploratory data analysis

```
import pandas as pd
import matplotlib.pyplot as plt
Df = pd.read_csv('Carseats.csv')
print(Df.describe())
print(Df["Education"].value_counts())
print(Df.groupby(['Education', 'Age']).mean())
y = list(Df.Population)
plt.boxplot(y)
plt.show()
```

```
C:\WINDOWS\system32\cmd.exe - python  2A.py

C:\Python\Python39-32>python 2A.py
       Unnamed: 0       Sales   CompPrice  ...       Price         Age   Education
count  400.000000  400.000000  400.000000  ...  400.000000  400.000000  400.000000
mean   200.500000    7.496325  124.975000  ...  115.795000   53.322500   13.900000
std    115.614301    2.824115   15.334512  ...   23.676664   16.200297    2.620528
min      1.000000    0.000000   77.000000  ...   24.000000   25.000000   10.000000
25%    100.750000    5.390000  115.000000  ...  100.000000   39.750000   12.000000
50%    200.500000    7.490000  125.000000  ...  117.000000   54.500000   14.000000
75%    300.250000    9.320000  135.000000  ...  131.000000   66.000000   16.000000
max    400.000000   16.270000  175.000000  ...  191.000000   80.000000   18.000000

[8 rows x 9 columns]
17    49
12    49
10    48
11    48
16    47
13    43
14    40
18    40
15    36
Name: Education, dtype: int64
               Unnamed: 0   Sales  CompPrice  Income  Advertising  Population  Price
Education Age
10        26        141.0  10.500      120.0    52.0          4.5       217.0   94.5
          28        122.0  11.670      125.0    89.0         10.0       380.0   87.0
          29        305.0  11.930      123.0    98.0         12.0       408.0  134.0
          30        221.0  10.590      131.0   120.0         15.0       262.0  124.0
          32        332.0  10.100      135.0    63.0         15.0       213.0  134.0
...                   ...     ...        ...     ...          ...         ...    ...
18        73        192.5   8.155      109.5    78.0          9.0       270.0   92.0
          76         16.0   8.710      149.0    95.0          5.0       400.0  144.0
          77        180.0   7.780      144.0    25.0          3.0        70.0  116.0
          78        247.0   6.900      120.0    56.0         20.0       266.0   90.0
          80        184.0   5.320      118.0    74.0          6.0       426.0  102.0

[283 rows x 7 columns]
```



Implement python program for data preparation activities such as filtering, grouping, ordering and joining of datasets.

```
import pandas as pd
import matplotlib.pyplot as plt
Df = pd.read_csv('Carseats.csv')
# Filter top scoring students
df = df[df['Age'] >= 60]
print(df)
```

```
C:\WINDOWS\system32\cmd.exe

C:\Python\Python39-32>python 2B.py
     Unnamed: 0  Sales  CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  Education Urban   US
1             2  11.22        111      48           16         260     83      Good   65         10   Yes  Yes
5             6  10.81        124     113           13         501     72       Bad   78         16    No  Yes
6             7   6.63        115     105            0          45    108    Medium   71         15   Yes   No
7             8  11.85        136      81           15         425    120      Good   67         10   Yes  Yes
8             9   6.54        132     110            0         108    124    Medium   76         10    No   No
..          ...    ...        ...     ...          ...         ...    ...       ...  ...        ...   ...  ...
383         384   9.35         98     117            0          76     68    Medium   63         10   Yes   No
385         386   5.87        131      73           13         455    132    Medium   62         17   Yes  Yes
387         388   8.67        142      73           14         238    115    Medium   73         14    No  Yes
388         389   8.14        135      89           11         245     78       Bad   79         16   Yes  Yes
390         391   5.47        108      75            9          61    111    Medium   67         12   Yes  Yes

[163 rows x 12 columns]

C:\Python\Python39-32>
```

Merging

```
# import module
import pandas as pd
# creating DataFrame for Student Details
details = pd.DataFrame({
        'ID': [101, 102, 103, 104, 105, 106,
                107, 108, 109, 110],
        'NAME': ['Jagroop', 'Praveen', 'Harjot',
                        'Pooja', 'Rahul', 'Nikita',
                        'Saurabh', 'Ayush', 'Dolly', "Mohit"],
        'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',
                        'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})
# printing details
print(details)
```

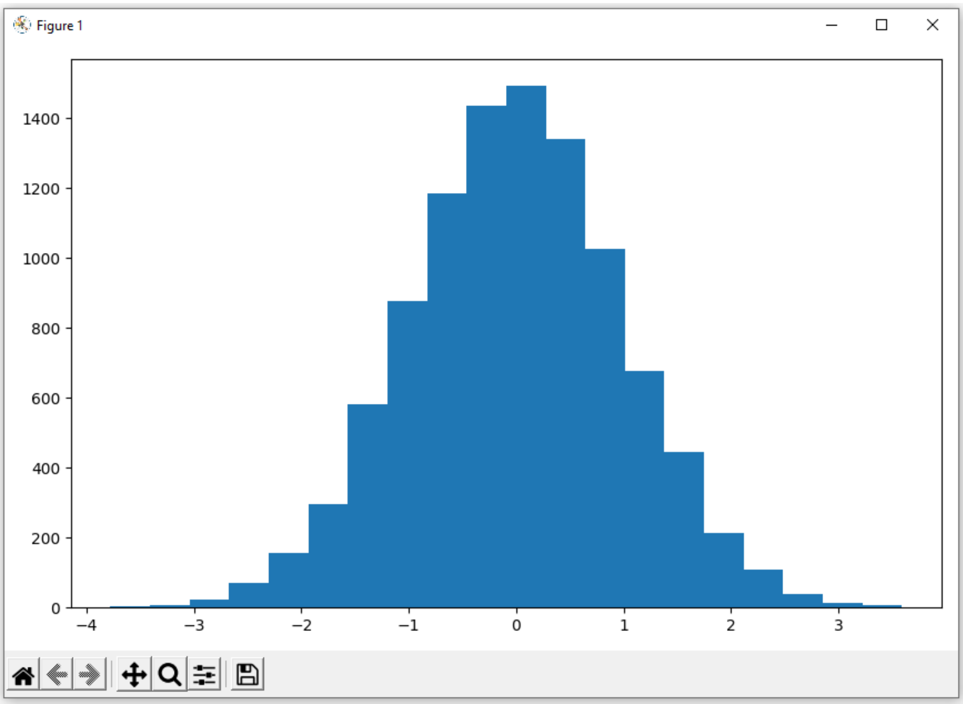
```
C:\WINDOWS\system32\cmd.exe

C:\Python\Python39-32>python 2B(1).py
    ID     NAME BRANCH
0  101  Jagroop    CSE
1  102  Praveen    CSE
2  103   Harjot    CSE
3  104    Pooja    CSE
4  105    Rahul    CSE
5  106   Nikita    CSE
6  107  Saurabh    CSE
7  108    Ayush    CSE
8  109    Dolly    CSE
9  110    Mohit    CSE

C:\Python\Python39-32>
```

Experiment 3:

Implement Python program to prepare plots such as bar plot, histogram, distribution plot, box plot, scatter plot.

Histogram:

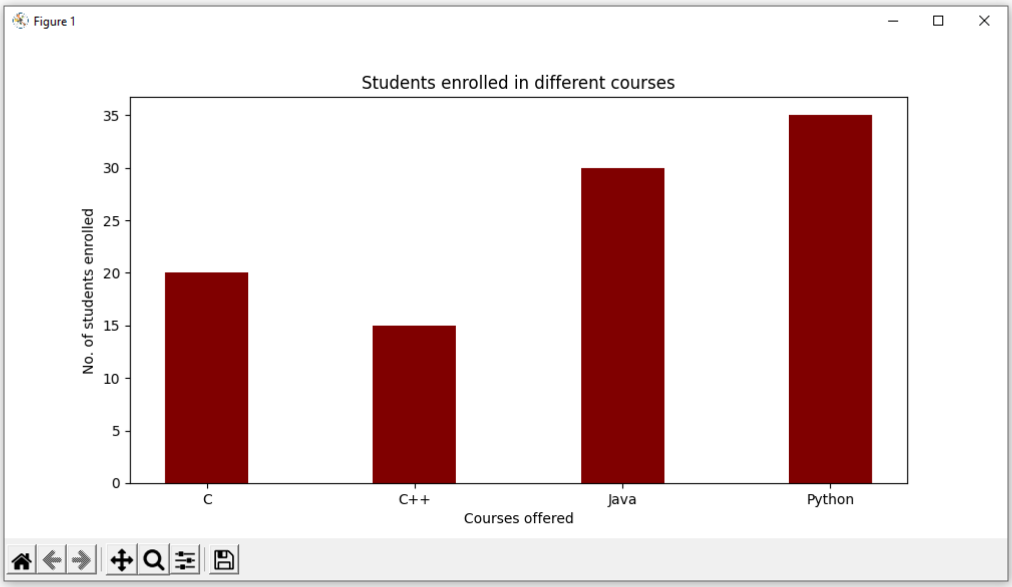
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter

# Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20

# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25

# Creating histogram
fig, axs = plt.subplots(1, 1,figsize =(10, 7),tight_layout = True)

axs.hist(x, bins = n_bins)

# Show plot
plt.show()
```



barplot:

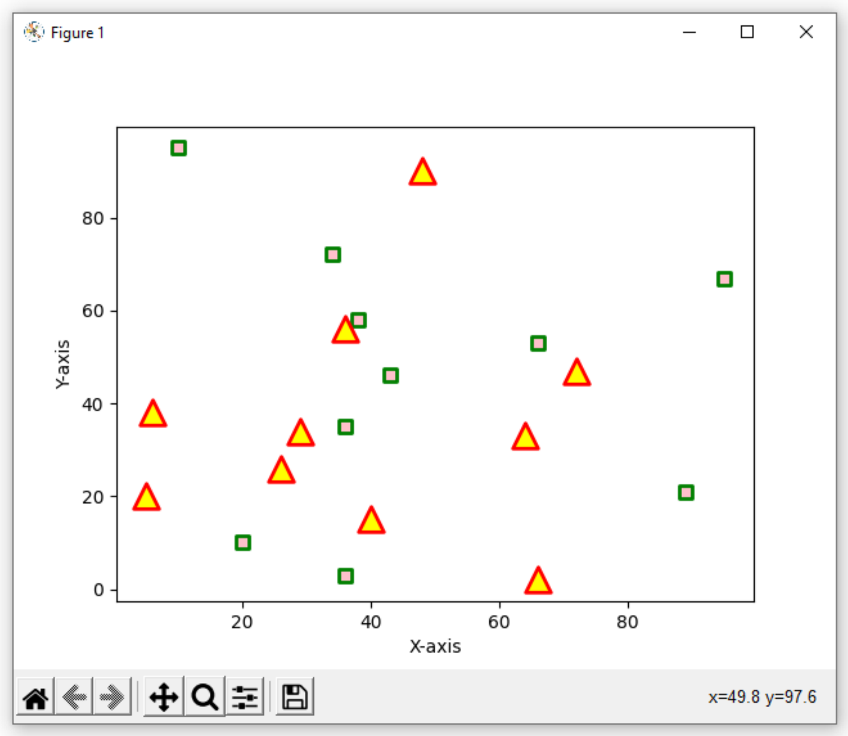
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
                 'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
                 width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```



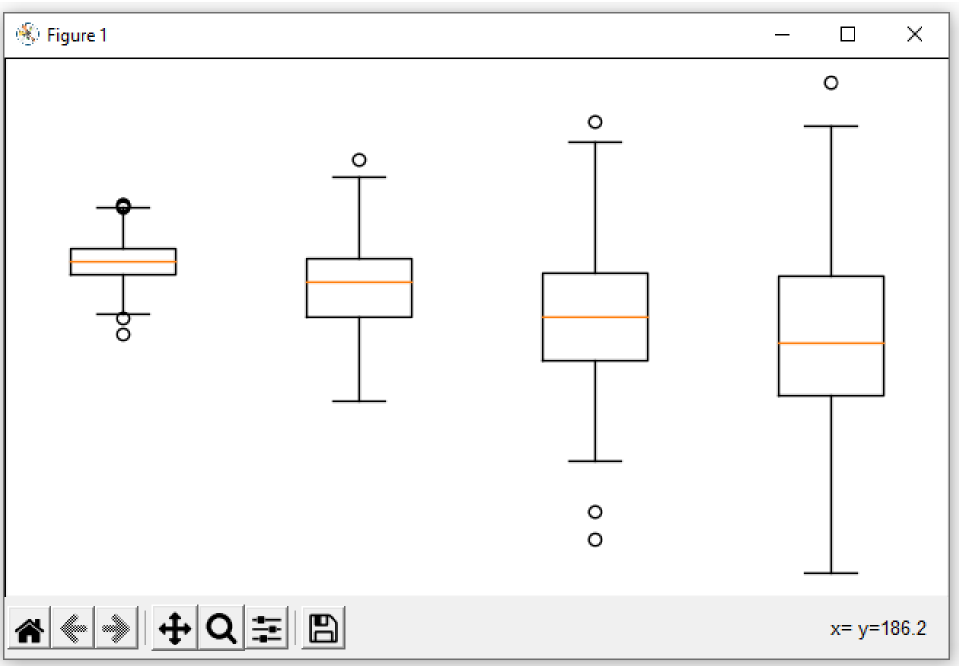
scatter plot:

```
import matplotlib.pyplot as plt

# dataset-1
x1 = [89, 43, 36, 36, 95, 10,66, 34, 38, 20]
y1 = [21, 46, 3, 35, 67, 95,53, 72, 58, 10]

# dataset2
x2 = [26, 29, 48, 64, 6, 5,36, 66, 72, 40]
y2 = [26, 34, 90, 33, 38,20, 56, 2, 47, 15]

plt.scatter(x1, y1, c ="pink", linewidths = 2, marker ="s", edgecolor ="green", s = 50)
plt.scatter(x2, y2, c ="yellow", linewidths = 2, marker ="^", edgecolor ="red", s = 200)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```



boxplot:

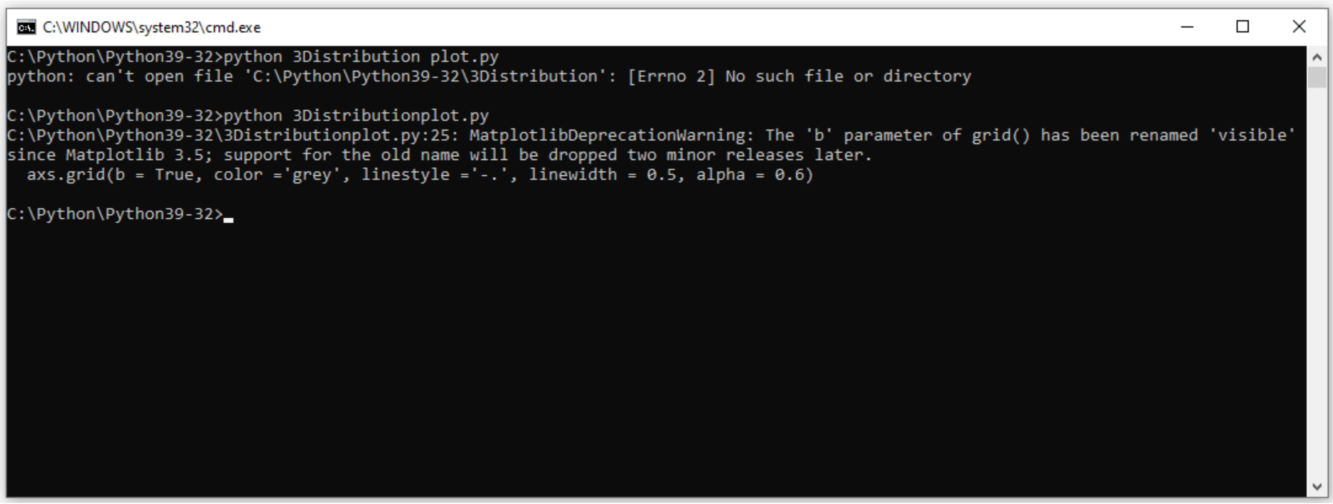
```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np
# Creating dataset
np.random.seed(10)
data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(90, 20, 200)
data_3 = np.random.normal(80, 30, 200)
data_4 = np.random.normal(70, 40, 200)
data = [data_1, data_2, data_3, data_4]
fig = plt.figure(figsize =(10, 7))
# Creating axes instance
ax = fig.add_axes([0, 0, 1, 1])

# Creating plot
bp = ax.boxplot(data)
# show plot
plt.show()
```

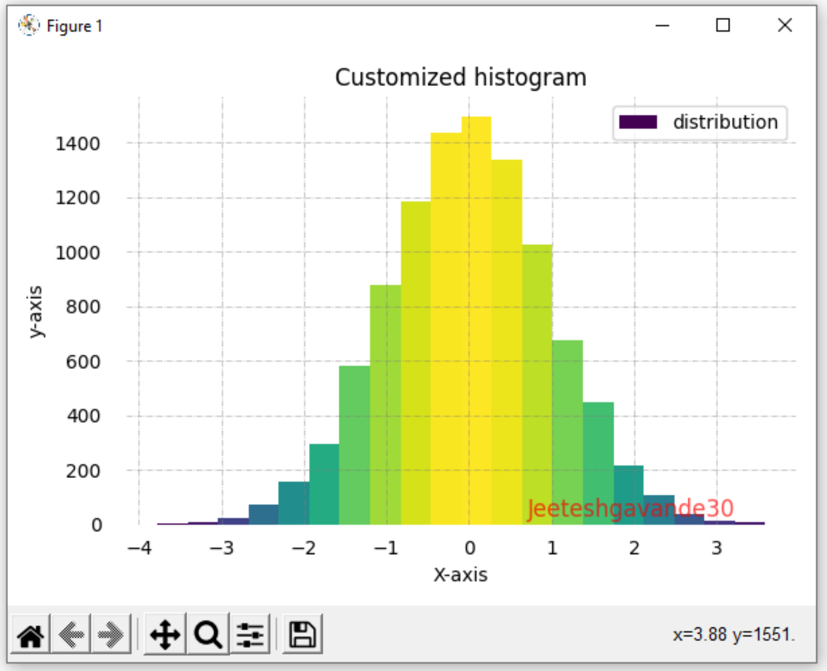


Distribution plot:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
# Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20
# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25
legend = ['distribution']
# Creating histogram
fig, axs = plt.subplots(1, 1, figsize =(10, 7), tight_layout = True)
# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
        axs.spines[s].set_visible(False)
# Remove x, y ticks
axs.xaxis.set_ticks_position('none')
axs.yaxis.set_ticks_position('none')
# Add padding between axes and labels
axs.xaxis.set_tick_params(pad = 5)
axs.yaxis.set_tick_params(pad = 10)
# Add x, y gridlines
axs.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.6)
# Add Text watermark
fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='red', ha ='right', va ='bottom', alpha =
0.7)
# Creating histogram
N, bins, patches = axs.hist(x, bins = n_bins)
# Setting color
fracs = ((N**(1 / 5)) / N.max())
```

```
norm = colors.Normalize(fracs.min(), fracs.max())
for thisfrac, thispatch in zip(fracs, patches):
        color = plt.cm.viridis(norm(thisfrac))
        thispatch.set_facecolor(color)
# Adding extra features
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.legend(legend)
plt.title('Customized histogram')
# Show plot
plt.show()
```

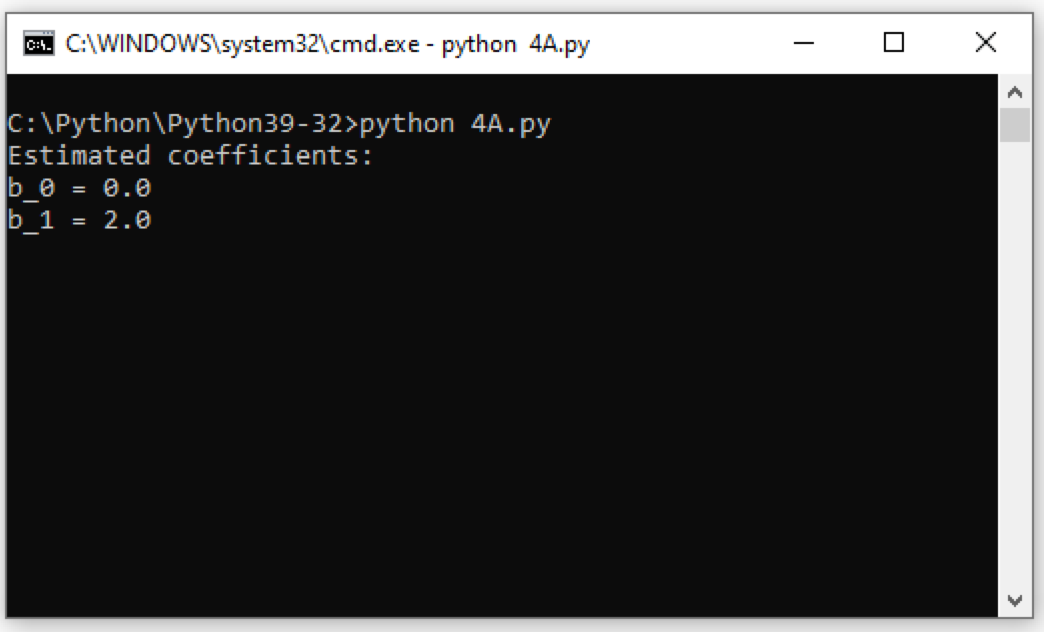


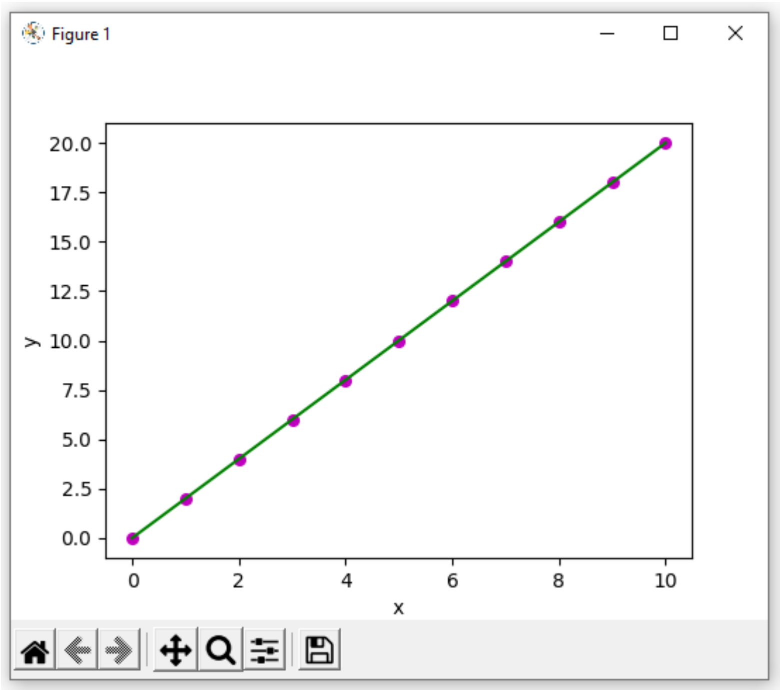


Experiment 4

Implement Simple Linear regression algorithm in Python.

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
	# number of observations/points
	n = np.size(x)
	# mean of x and y vector
	m_x = np.mean(x)
	m_y = np.mean(y)
	# calculating cross-deviation and deviation about x
	SS_xy = np.sum(y*x) - n*m_y*m_x
	SS_xx = np.sum(x*x) - n*m_x*m_x
	# calculating regression coefficients
	b_1 = SS_xy / SS_xx
	b_0 = m_y - b_1*m_x
	return (b_0, b_1)

def plot_regression_line(x, y, b):
	# plotting the actual points as scatter plot
	plt.scatter(x, y, color = "m",
				marker = "o", s = 30)
	# predicted response vector
	y_pred = b[0] + b[1]*x
	# plotting the regression line
	plt.plot(x, y_pred, color = "g")
	# putting labels
	plt.xlabel('x')
	plt.ylabel('y')
	# function to show plot
	plt.show()

def main():
	# observations / data
  x = np.array([i for i in range(11)])
  y = np.array([2*i for i in range(11)])
	# estimating coefficients
  b = estimate_coef(x, y)
  print("Estimated coefficients:\nb_0 = {} \
  \nb_1 = {}".format(b[0], b[1]))
	# plotting regression line
  plot_regression_line(x, y, b)

if __name__ == "__main__":
	main()
```



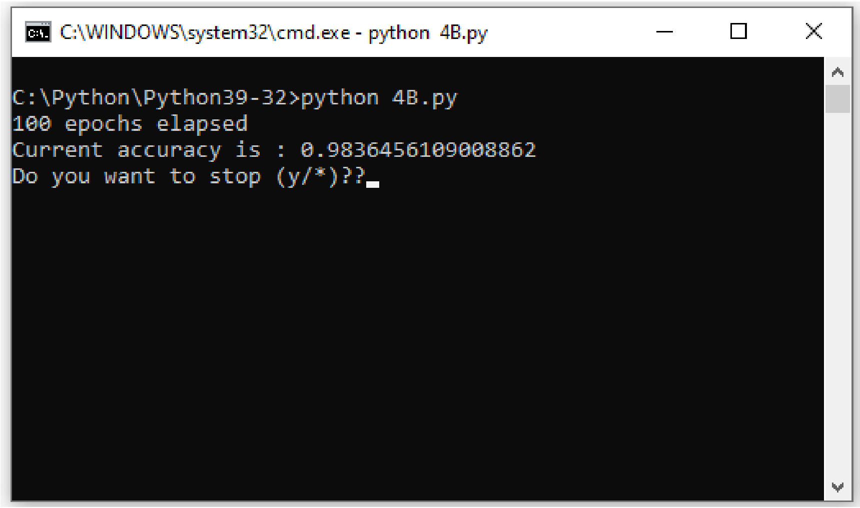


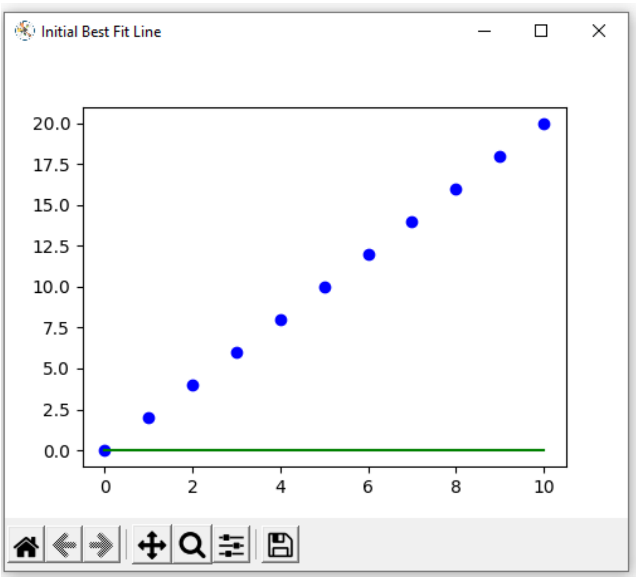
Implement Gradient Descent algorithm for the above linear regression model.

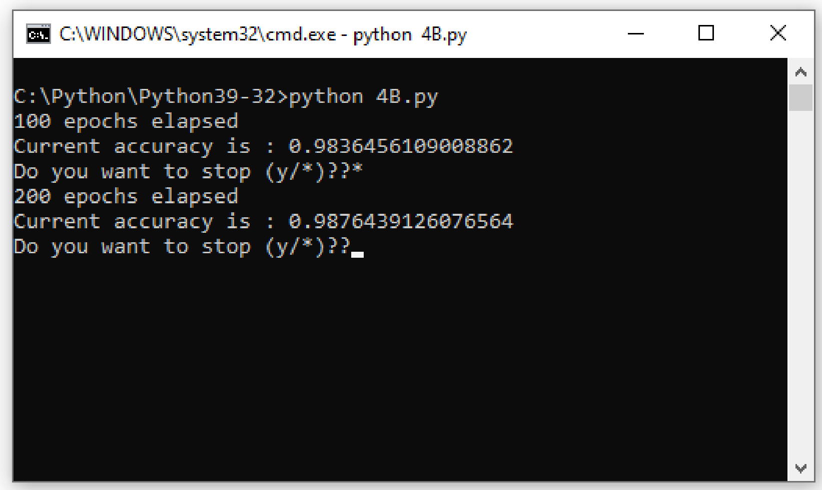
```
# Implementation of gradient descent in linear regression
import numpy as np
import matplotlib.pyplot as plt

class Linear_Regression:
	def __init__(self, X, Y):
		self.X = X
		self.Y = Y
		self.b = [0, 0]

	def update_coeffs(self, learning_rate):
```

```
            Y_pred = self.predict()
            Y = self.Y
            m = len(Y)
            self.b[0] = self.b[0] - (learning_rate * ((1/m) * np.sum(Y_pred - Y)))

            self.b[1] = self.b[1] - (learning_rate * ((1/m) * np.sum((Y_pred - Y) * self.X)))

        def predict(self, X=[]):
            Y_pred = np.array([])
            if not X: X = self.X
            b = self.b
            for x in X:
                Y_pred = np.append(Y_pred, b[0] + (b[1] * x))
            return Y_pred

        def get_current_accuracy(self, Y_pred):
            p, e = Y_pred, self.Y
            n = len(Y_pred)
            return 1-sum([abs(p[i]-e[i])/e[i] for i in range(n)  if e[i] != 0])/n
        def compute_cost(self, Y_pred):
            m = len(self.Y)
            J = (1 / 2*m) * (np.sum(Y_pred - self.Y)**2)
            return J
        def plot_best_fit(self, Y_pred, fig):
                    f = plt.figure(fig)
                    plt.scatter(self.X, self.Y, color='b')
                    plt.plot(self.X, Y_pred, color='g')
                    f.show()
def main():
   X = np.array([i for i in range(11)])
   Y = np.array([2*i for i in range(11)])
   regressor = Linear_Regression(X, Y)
   iterations = 0
   steps = 100
   learning_rate = 0.01
   costs = []
        #original best-fit line
   Y_pred = regressor.predict()
   regressor.plot_best_fit(Y_pred, 'Initial Best Fit Line')
   while 1:
      Y_pred = regressor.predict()
      cost = regressor.compute_cost(Y_pred)
      costs.append(cost)
      regressor.update_coeffs(learning_rate)
      iterations += 1
      if iterations % steps == 0:
         print(iterations, "epochs elapsed")
         print("Current accuracy is :",regressor.get_current_accuracy(Y_pred))
         stop = input("Do you want to stop (y/*)??")
         if stop == "y":
            break
```

```
        #final best-fit line
    regressor.plot_best_fit(Y_pred, 'Final Best Fit Line')
        #plot to verify cost function decreases
    h = plt.figure('Verification')
    plt.plot(range(iterations), costs, color='b')
    h.show()
        # if user wants to predict using the regressor:
    regressor.predict([i for i in range(10)])
if __name__ == '__main__':
        main()
```

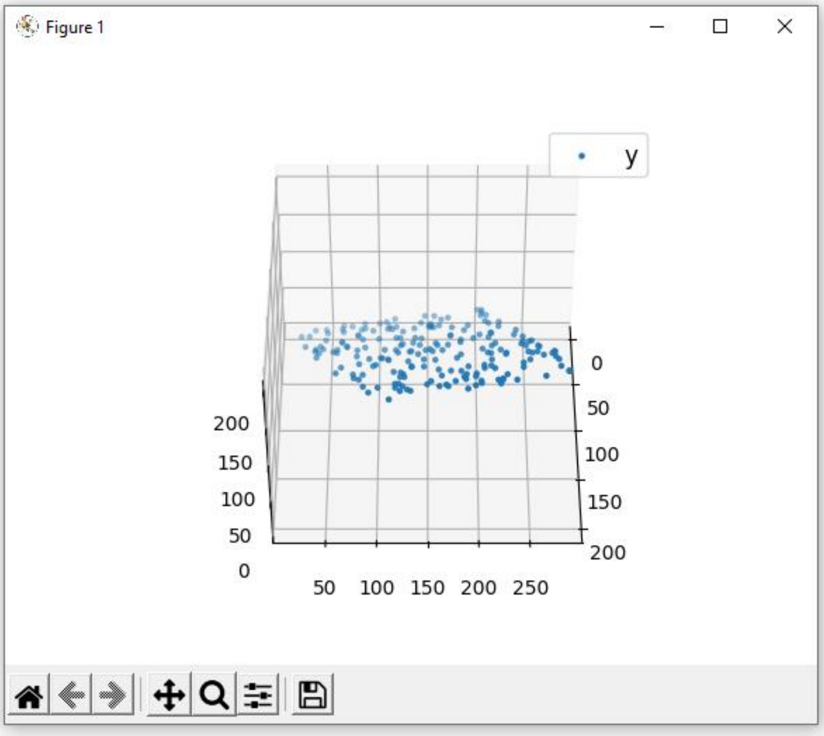






Experiment 5:
Implement Multiple linear regression algorithm using Python.

```
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def generate_dataset(n):
        x = []
        y = []
        random_x1 = np.random.rand()
        random_x2 = np.random.rand()
        for i in range(n):
                x1 = i
                x2 = i/2 + np.random.rand()*n
                x.append([1, x1, x2])
                y.append(random_x1 * x1 + random_x2 * x2 + 1)
        return np.array(x), np.array(y)
x, y = generate_dataset(200)
mpl.rcParams['legend.fontsize'] = 12
ax = plt.axes(projection ='3d')
ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)
plt.show()
```



Experiment 6:
Implement Python Program to build logistic regression and decision tree models using the Python package stats model and sklearn APIs.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn import metrics


col_names =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']

# load dataset

pima = pd.read_csv("diabetes.csv", header=None, names=col_names)

feature_cols =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']

X = pima[feature_cols] # Features

y = pima.Outcome # Target variable


X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=1)

logreg = LogisticRegression()
```

logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)

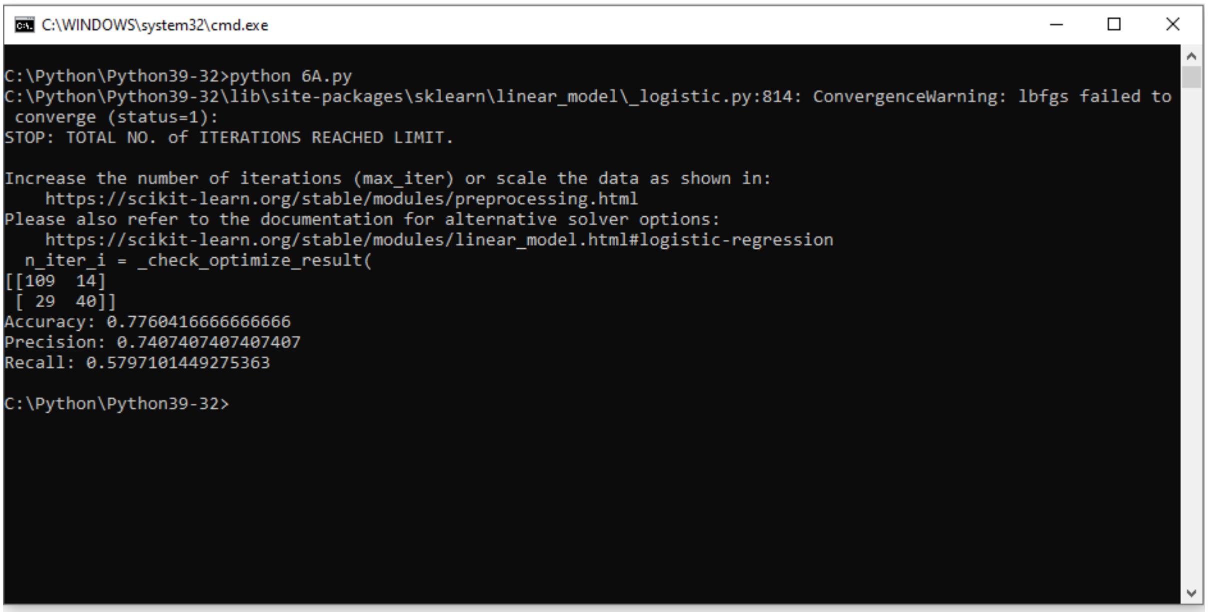
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)

print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

print("Precision:",metrics.precision_score(y_test, y_pred))

print("Recall:",metrics.recall_score(y_test, y_pred))



6b) decision tree

import pandas as pd

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

from sklearn.model_selection import train_test_split # Import train_test_split function

from sklearn import metrics

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

load dataset

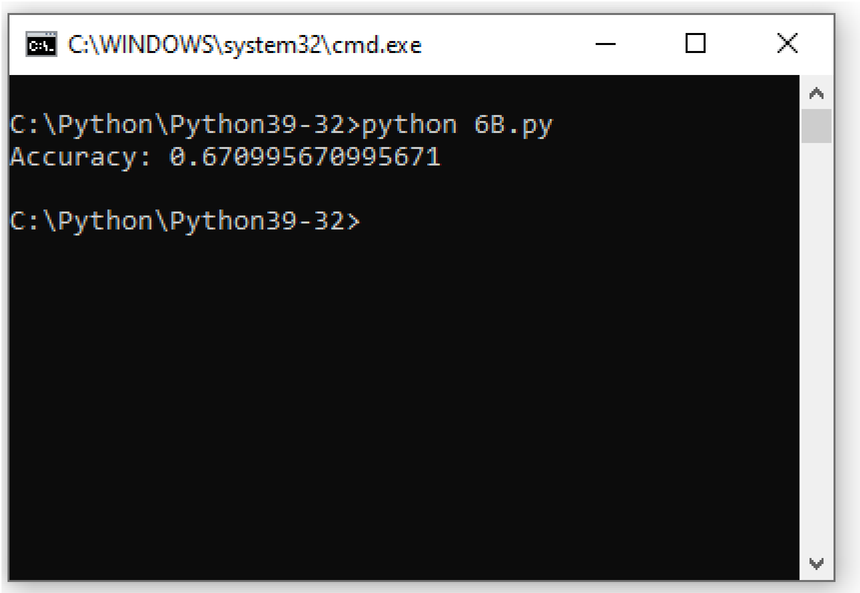
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)

feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

X = pima[feature_cols] # Features

y = pima.label

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

clf = DecisionTreeClassifier()

clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```



Experiment 7:
Write a Python program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions

```
#k-Nearest Neighbour algorithm(lab)

from sklearn.datasets import load_iris

iris = load_iris()

print("Feature Names:",iris.feature_names,"Iris Data:",iris.data,"Target
Names:",iris.target_names,"Target:",iris.target)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size = .25)

from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()

clf.fit(X_train, y_train)

print(" Accuracy=",clf.score(X_test, y_test))

print("Predicted Data")

print(clf.predict(X_test))
```

prediction=clf.predict(X_test)

print("Test data :")

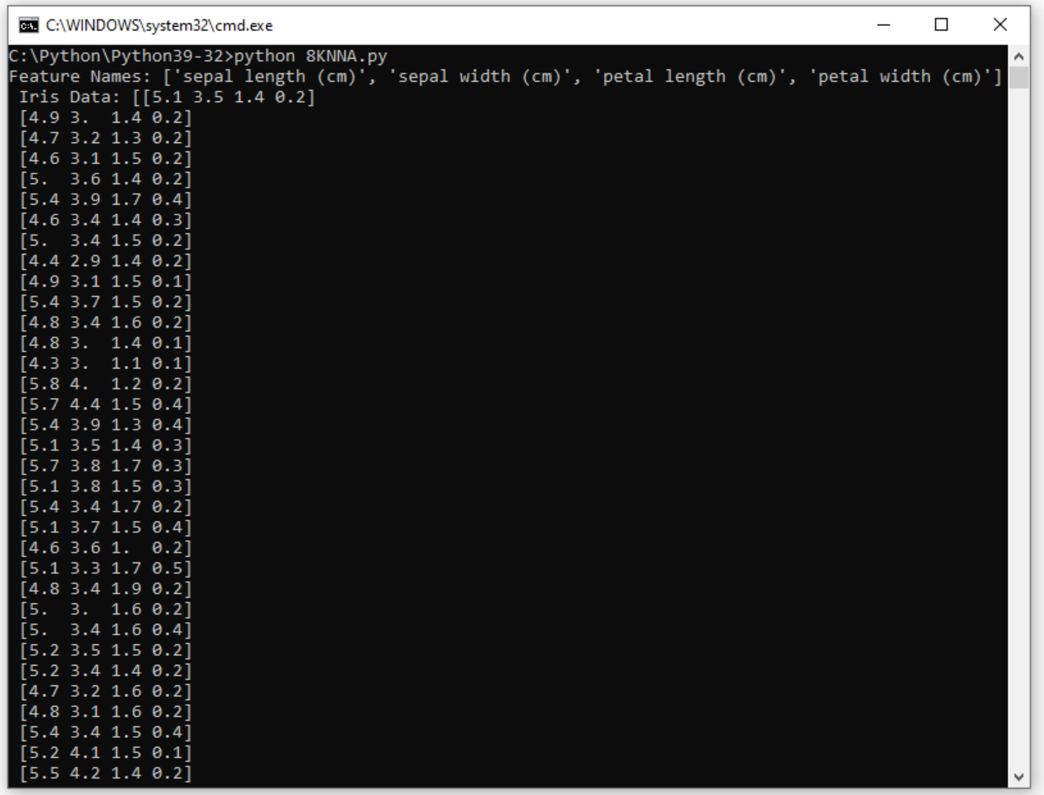
print(y_test)

diff=prediction-y_test

print("Result is ")

print(diff)

print('Total no of samples misclassied =', sum(abs(diff)))



C:\WINDOWS\system32\cmd.exe

```
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
```

C:\WINDOWS\system32\cmd.exe

```
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
```

```
C:\WINDOWS\system32\cmd.exe
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]] Target Names: ['setosa' 'versicolor' 'virginica'] Target: [0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
 Accuracy= 0.9736842105263158
Predicted Data
[2 2 2 0 2 1 0 1 2 0 0 2 1 1 0 2 1 1 0 1 0 0 0 1 1 1 0 2 2 0 0 0 2 0 2 0 2
 0]
Test data :
[2 2 2 0 2 1 0 1 2 0 0 2 1 1 0 2 1 1 0 2 0 0 0 1 1 1 0 2 2 0 0 0 2 0 2 0 2
 0]
Result is
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
Total no of samples misclassied = 1

C:\Python\Python39-32>
```

Experiment 8:
Implement Support vector Machine algorithm on any data set

```
#SupportVectorMachine(lab)

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn import svm

from sklearn import metrics

cancer_data = datasets.load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer_data.data, cancer_data.target,
test_size=0.4,random_state=109)

#create a classifier

cls = svm.SVC(kernel="linear")

#train the model
```

```
cls.fit(X_train,y_train)

#predict the response

pred = cls.predict(X_test)

#accuracy

print("acuracy:", metrics.accuracy_score(y_test,y_pred=pred))

#precision score

print("precision:", metrics.precision_score(y_test,y_pred=pred))

#recall score

print("recall" , metrics.recall_score(y_test,y_pred=pred))

print(metrics.classification_report(y_test, y_pred=pred))
```

C:\WINDOWS\system32\cmd.exe

```
C:\Python\Python39-32>python 9.py
acuracy: 0.9649122807017544
precision: 0.9642857142857143
recall 0.9782608695652174
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        90
           1       0.96      0.98      0.97       138

    accuracy                           0.96       228
   macro avg       0.97      0.96      0.96       228
weighted avg       0.96      0.96      0.96       228


C:\Python\Python39-32>
```

Experiment 9:
Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB



data = pd.read_csv('tennisdata.csv')

print("The first 5 values of data is :\n",data.head())
```

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

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("\nNow the Train data is :\n",X.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)

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))
```

```
C:\WINDOWS\system32\cmd.exe

C:\Python\Python39-32>python 10.py
The first 5 values of data is :
     Outlook Temperature Humidity  Windy PlayTennis
0      Sunny         Hot     High  False         No
1      Sunny         Hot     High   True         No
2   Overcast         Hot     High  False        Yes
3      Rainy        Mild     High  False        Yes
4      Rainy        Cool   Normal  False        Yes

The First 5 values of train data is
     Outlook Temperature Humidity  Windy
0      Sunny         Hot     High  False
1      Sunny         Hot     High   True
2   Overcast         Hot     High  False
3      Rainy        Mild     High  False
4      Rainy        Cool   Normal  False

The first 5 values of Train output is
 0      No
1      No
2     Yes
3     Yes
4     Yes
Name: PlayTennis, dtype: object

Now the Train data is :
    Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666

C:\Python\Python39-32>
```

Experiment 10:

Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

```
#Bayesian network(lab)

import bayespy as bp

import numpy as np

import csv

from colorama import init

from colorama import Fore, Back, Style

init()

ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3,'Teen':4}

genderEnum = {'Male':0, 'Female':1}

familyHistoryEnum = {'Yes':0, 'No':1}

dietEnum = {'High':0, 'Medium':1, 'Low':2}
```

```
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}

cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}

heartDiseaseEnum = {'Yes':0, 'No':1}

with open('heartdisease.csv') as csvfile:

   lines = csv.reader(csvfile)

   dataset = list(lines)

   data = []

   for x in dataset:

data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEnu
m[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])

data = np.array(data)

N = len(data)

p_age = bp.nodes.Dirichlet(1.0*np.ones(5))

age = bp.nodes.Categorical(p_age, plates=(N,))

age.observe(data[:,0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))

gender = bp.nodes.Categorical(p_gender, plates=(N,))

gender.observe(data[:,1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))

familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))

familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))

diet = bp.nodes.Categorical(p_diet, plates=(N,))

diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))

lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))

lifestyle.observe(data[:,4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))

cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))

cholesterol.observe(data[:,5])

p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
```

```
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle,

cholesterol], bp.nodes.Categorical, p_heartdisease)

heartdisease.observe(data[:,6])

p_heartdisease.update()

m = 0

while m == 0:

   print("\n")

   res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' +
str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter
dietEnum: ' + str(dietEnum))),int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter
Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical,
p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]

   print("Probability(HeartDisease) = " + str(res))

   m = int(input("Enter for Continue:0, Exit :1 "))
```

```
C:\Python\Python310>python 11BayesianNetwork.py


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}2
Enter Gender: {'Male': 0, 'Female': 1}1
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}1
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}1
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1

C:\Python\Python310>
```

Experiment 11:

Assuming a set of documents that need to be classified, use the naive Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision and recall for your data set

```
from sklearn.datasets import fetch_20newsgroups

twenty_train = fetch_20newsgroups(subset='train', shuffle=True)

print("lenth of the twenty_train--------->", len(twenty_train))

print("**First Line of the First Data File**")

from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()

X_train_counts = count_vect.fit_transform(twenty_train.data)

print('dim=',X_train_counts.shape)

from sklearn.feature_extraction.text import TfidfTransformer
```

```
tfidf_transformer = TfidfTransformer()

X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

print(X_train_tfidf.shape)

from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)

from sklearn.pipeline import Pipeline

text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf',MultinomialNB())])

text_clf = text_clf.fit(twenty_train.data, twenty_train.target)

# Performance of NB Classifier

import numpy as np

twenty_test = fetch_20newsgroups(subset='test', shuffle=True)

predicted = text_clf.predict(twenty_test.data)

accuracy=np.mean(predicted == twenty_test.target)

print("Predicted Accuracy = ",accuracy)

#To Calculate Accuracy,Precision,Recall

from sklearn import metrics

print("Accuracy= ",metrics.accuracy_score(twenty_test.target,predicted))

print("Precision=",metrics.precision_score(twenty_test.target,predicted,average=None))

print("Recall=",metrics.recall_score(twenty_test.target,predicted,average=None))

print(metrics.classification_report(twenty_test.target,predicted,target_names=twenty_test.target_n
ames))
```

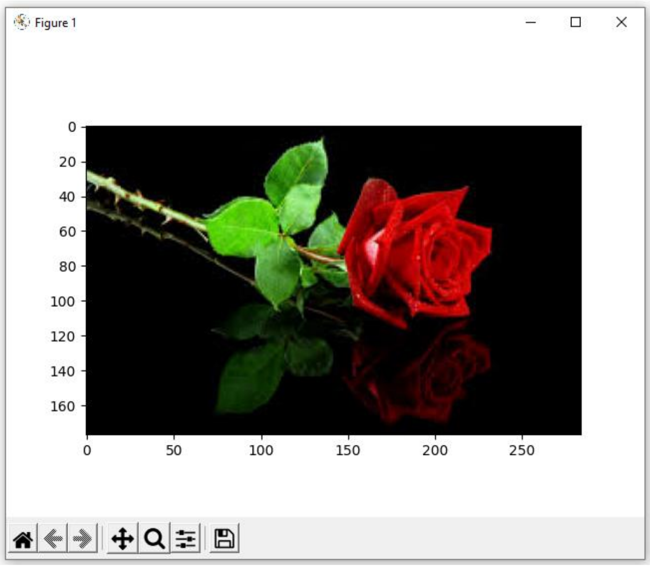
```
C:\WINDOWS\system32\cmd.exe
C:\Python\Python39-32>python 12NaiveBayesianClassifierAPI.py
lenth of the twenty_train---------> 5
**First Line of the First Data File**
dim= (11314, 130107)
(11314, 130107)
Predicted Accuracy =  0.7738980350504514
Accuracy=  0.7738980350504514
Precision= [0.80193237 0.81028939 0.81904762 0.67180617 0.85632184 0.88955224
 0.93127148 0.84651163 0.93686869 0.92248062 0.89170507 0.59379845
 0.83629893 0.92113565 0.84172662 0.43896976 0.64339623 0.92972973
 0.95555556 0.97222222]
Recall= [0.52037618 0.64781491 0.65482234 0.77806122 0.77402597 0.75443038
 0.69487179 0.91919192 0.9321608  0.89924433 0.96992481 0.96717172
 0.59796438 0.73737374 0.89086294 0.98492462 0.93681319 0.91489362
 0.41612903 0.13944223]
                          precision    recall  f1-score   support

             alt.atheism       0.80      0.52      0.63       319
           comp.graphics       0.81      0.65      0.72       389
 comp.os.ms-windows.misc       0.82      0.65      0.73       394
comp.sys.ibm.pc.hardware       0.67      0.78      0.72       392
   comp.sys.mac.hardware       0.86      0.77      0.81       385
          comp.windows.x       0.89      0.75      0.82       395
            misc.forsale       0.93      0.69      0.80       390
               rec.autos       0.85      0.92      0.88       396
         rec.motorcycles       0.94      0.93      0.93       398
      rec.sport.baseball       0.92      0.90      0.91       397
        rec.sport.hockey       0.89      0.97      0.93       399
               sci.crypt       0.59      0.97      0.74       396
         sci.electronics       0.84      0.60      0.70       393
                 sci.med       0.92      0.74      0.82       396
               sci.space       0.84      0.89      0.87       394
  soc.religion.christian       0.44      0.98      0.61       398
      talk.politics.guns       0.64      0.94      0.76       364
   talk.politics.mideast       0.93      0.91      0.92       376
      talk.politics.misc       0.96      0.42      0.58       310
      talk.religion.misc       0.97      0.14      0.24       251

                accuracy                           0.77      7532
               macro avg       0.83      0.76      0.76      7532
            weighted avg       0.82      0.77      0.77      7532


C:\Python\Python39-32>
```

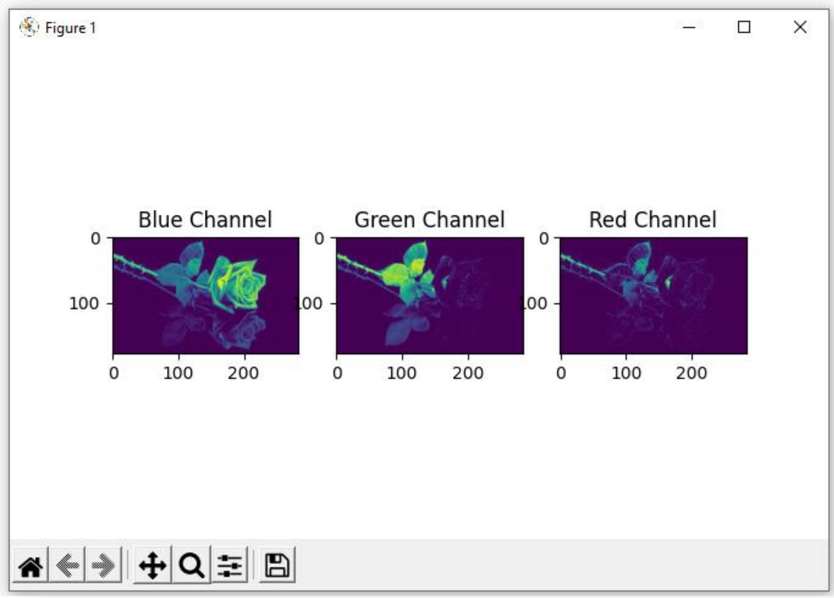
Experiment 12:

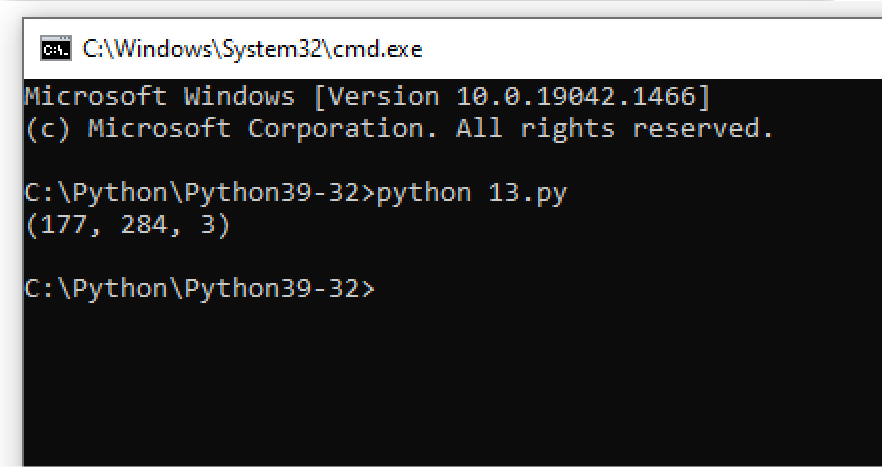
Implement PCA on any Image dataset for dimensionality reduction and classification of images into different classes

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

import cv2

from scipy.stats import stats

import matplotlib.image as mpimg

img = cv2.cvtColor(cv2.imread('rose.jpg'), cv2.COLOR_BGR2RGB)

plt.imshow(img)

plt.show()

print(img.shape)

#Splitting into channels

blue,green,red = cv2.split(img)
```

```
# Plotting the images
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.imshow(blue)
fig.add_subplot(132)
plt.title("Green Channel")
plt.imshow(green)
fig.add_subplot(133)
plt.title("Red Channel")
plt.imshow(red)
plt.show()
```







Experiment 13:
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
#Locally Weighted Regressionalgorithm(lab)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# kernel smoothing function

def kernel(point, xmat, k):

    m,n = np.shape(xmat)

    weights = np.mat(np.eye((m)))

    for j in range(m):

        diff = point - X[j]
```

```
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k**2))
    return weights
# function to return local weight of eah traiining example
def localWeight(point, xmat, ymat, k):
    wt = kernel(point, xmat, k)
    W = (X.T * (wt*X)).I * (X.T * wt * ymat.T)
    return W
# root function that drives the algorithm
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)
    return ypred
#import data
data = pd.read_csv('tips.csv')
colA = np.array(data.total_bill)
colB = np.array(data.tip)
mcolA = np.mat(colA)
mcolB = np.mat(colB)
m = np.shape(mcolB)[1]
one = np.ones((1, m), dtype = int)
X = np.hstack((one.T, mcolA.T))
print(X.shape)

# predicting values using LWLR
ypred = localWeightRegression(X, mcolB, 0.8)

# plotting the predicted graph
xsort = X.copy()
xsort.sort(axis=0)
```

```
plt.scatter(colA, colB, color='red')
plt.plot(xsort[:, 1], ypred[X[:, 1].argsort(0)], color='green', linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```