20MCA241 - DATA SCIENCE
LAB RECORD

Submitted by,

Roshan Thomas

RMCA-B

Roll No.15

DATE: 18/11/2021

EXPERIMENT -1

AIM: Write a python program that takes an input n and calculate n+nn+nnn

Program

```
n=int(input("Enter the number : "))

print(n+n*n+n*n*n)
```

Output

```
Enter the number : 4
84
```

EXPERIMENT -2

AIM: Write a python program to get a largest number from a list

Program

```
list1 = [10, 20, 40, 45, 100]

# printing the maximum element
print("Largest element is:", max(list1))
```

Output

```
Largest element is: 100

Process finished with exit code 0
```

EXPERIMENT -3

AIM: Write a python program to clone or copy a list.

Program

```
# Using the in-built function list()
def Cloning(li1):
    li_copy = list(li1)
    return li_copy

li1 = [4, 8, 2, 10, 15, 18]
```

```
li2 = Cloning(li1)
print("Original List:", li1)
print("After Cloning:", li2)
```

Output

```
Original List: [4, 8, 2, 10, 15, 18]
After Cloning: [4, 8, 2, 10, 15, 18]

Process finished with exit code 0
```

EXPERIMENT -4

AIM: Write a python program to shuffle and print a specified list.

Program

```
import random
number_list = [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
print ("Original list : ",  number_list)
#shuffle method
random.shuffle(number_list)
print ("List after shuffle  : ",  number_list)
```

Output

```
Original list :  [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
List after shuffle  :  [42, 28, 14, 56, 21, 49, 63, 70, 35, 7]

Process finished with exit code 0
```

EXPERIMENT -5

AIM: Write a python program script to sort a python dictionary by value.

Program

```
import operator

d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
s = sorted(d.items(), key=operator.itemgetter(1))
print('ascending order : ', s)
s1 = dict(sorted(d.items(), key=operator.itemgetter(1), reverse=True))
print('descending order : ', s1)
```

Output

```
ascending order :  [(0, 0), (2, 1), (1, 2), (4, 3), (3, 4)]
descending order :  {3: 4, 4: 3, 1: 2, 2: 1, 0: 0}

Process finished with exit code 0
```

EXPERIMENT -6

AIM: Write a python program script to add key to a dictionary.

Program

```
dict = {'Name':'Tom', 'Roll no':20}
print(dict)
dict.update({'Name':'Ammu', 'Roll no':15})
print(dict)
```

Output

```
{'Name': 'Tom', 'Roll no': 20}
{'Name': 'Ammu', 'Roll no': 15}

Process finished with exit code 0
```

EXPERIMENT -7

AIM: Write a python program script to merge two dictionaries.

Program

```
def Merge(dict1, dict2):
    return (dict2.update(dict1))

dict1 = {'a': 10, 'b': 20}
dict2 = {'c': 30, 'd': 40}

print(Merge(dict1, dict2))
print(dict2)
```

Output

```
{'c': 30, 'd': 40, 'a': 10, 'b': 20}

Process finished with exit code 0
```

Date: 25/11/2021

EXPERIMENT -8

AIM: Working with pandas

Program

```
import pandas as pd
#Create Dataframe from  dict
student_dict
={'Name':['Joe','Nat','Harry'],'Age':[20,21,19],'Marks':[85.10,77.80,91.54]}
student_df = pd.DataFrame(student_dict)
#Display dataframe
print("DataFrame:",student_df)
#select top 2 rows
print(student_df.head(2))
#select bottom 2 rows
print(student_df.tail(2))
#select value at row index 0 and column 'Name'
print(student_df.at[0,'Name'])
#select value at first row and column
print(student_df.iat[0,0])
#select values of 'Name' column
print(student_df.get('Name'))
#select values from row index 0 to 2 and 'Name' column
print(student_df.loc[0:2,['Name']])
student_df=student_df.sort_values(by=['Marks'])
print(student_df)
print(student_df.iloc[0:2,0:2])

print(dict)
filter=student_df['Marks']>80
student_df['Marks'].where(filter,other=0,inplace=True)
print(student_df)
student_df=student_df.filter(like='N',axis='columns')
print(student_df)
```

Output

```
DataFrame:     Name  Age  Marks
0    Joe   20  85.10
1    Nat   21  77.80
2  Harry   19  91.54
  Name  Age  Marks
0  Joe   20   85.1
1  Nat   21   77.8
    Name  Age  Marks
1    Nat   21  77.80
2  Harry   19  91.54
Joe
Joe
0      Joe
1      Nat
2    Harry
Name: Name, dtype: object
    Name
0    Joe
1    Nat
2  Harry
    Name  Age  Marks
1    Nat   21  77.80
0    Joe   20  85.10
2  Harry   19  91.54
```

```
  Name  Age
1  Nat   21
0  Joe   20
<class 'dict'>
    Name  Age  Marks
1    Nat   21   0.00
0    Joe   20  85.10
2  Harry   19  91.54
    Name
1    Nat
0    Joe
2  Harry

Process finished with exit code 0
```

EXPERIMENT - 9

AIM: Write a python program to demonstrate basic array characterstics

Program

```
import numpy as np #creating
array object
arr=np.array([[1,2,3],[4,2,5]])
#printing type of arr object
print("Array is of type :",type(arr))
#printing type of arr dimensions(axes)
print("No. of type :",arr.ndim)
print("Shape of the array :",arr.shape)
#printing size of the array
print("Size of array :",arr.size)
#printing type of elemmts in array
print("Array  stores elements of type: :",arr.dtype)
```

Output

```
Array is of type : <class 'numpy.ndarray'>
No. of type : 2
Shape of the array : (2, 3)
Size of array : 6
Array  stores elements of type: : int32

Process finished with exit code 0
```

EXPERIMENT-10

AIM: Program to demonstrate the basic array techniques

Program

```
#program to demonstrate
#array creation technique

import numpy as np

#creating array from list  with type float

a=np.array([[1,2,4],[5,8,7]],dtype='float')
print("Array created using passed list:\n",a)

#creating array from tuple

b=np.array((1,3,4))
print("\nArray created using passed tuple:\n",b)
#Creating a 3X4 array with all zeros
c=np.zeros((3,4))
print("\n An array initialized with all zeros:\n",c)
#create a constant value array of complex type
d=np.full((3,3),6,dtype='complex')
print("\n An array initialized with all 6s and with type complex \n",d)
#create an array with random values
e=np.random.random((2,2))
print("\nA random array : \n",e)
#create a sequence of integers from 0 to 30 with steps of 5
f=np.arange(0,30,5)
print("\n A Seqential array with steps of 5 :\n",f)
#Creating a sequence of 10 values in range 0 to 5
g=np.linspace(0,5,10)
print("\n A sequential aray with 10 values between 0 and 5 : ",g)
#Reshaping 3X4 array to 2X2X3 array
arr=np.array([[1,2,3,4],
              [5,2,4,2],
              [1,2,0,1]])
newarr=arr.reshape(2,2,3)
print("\nOriginal array : \n",arr)
print("Reshaped array :\n",newarr)

#Flatten array
arr=np.array([[1,2,3],[4,5,6]])
flarr=arr.flatten()
```

```
print("\nOriginal array:\n",arr)
print("\n Flatted array :\n ",flarr)
```

Output

```
Array created using passed list:
 [[1. 2. 4.]
 [5. 8. 7.]]

Array created using passed tuple:
 [1 3 2]

An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

 An array initialized with all 6s.Array type is complex:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]
```

```
A random array:
[[0.08925937 0.78551309]
[0.53828337 0.31812399]]

A sequential array with steps of 5 :
[ 0  5 10 15 20 25]

A sequential array with 10 values between0 and 5:
[0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
3.33333333 3.88888889 4.44444444 5.        ]
```

```
Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]

Reshaped array :
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]

Original array:
 [[1 2 3]
 [4 5 6]]

Flattened array :
 [1 2 3 4 5 6]

Process finished with exit code 0
```

EXPERIMENT-11

AIM: Program to demonstrate indexing in numpy

Program

```
#3.python program to demonstrate
#indexing in numpy
import numpy as np
#An exempler array
arr=np.array([[-1,2,0,4],
              [4,-0.5,6,0],
              [2.6,0,7,8],
              [3,-7,4,2.0]])
#Slicing array
temp=arr[:2, ::2]
print("Array with first 2 rows and alternate colums(0 and 2) : \n",temp)

#Integer array indexing example

temp=arr[[0,1,2,3],[3,2,1,0]]
print("\nElements at indices (0,3), (1,2),(2,1),(3,0) :\n",temp)

#boolean array indexing example#print the shape of an array
print("shape of an array : ",arr.size)
cond=arr>0  #cond is a boolean array
temp=arr[cond]
print("\nElements greater than 0 : \n",temp)
```

Output

```
Array with first 2 rows  alternate columns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

 Elements at indices (0,3),(1,2),(2,1),(3,0):
 [4. 6. 0. 3.]

 Elements greater than 0:
 [2.  4.  4.  6.  2.6 7.  8.  3.  4.  2. ]

Process finished with exit code 0
```

EXPERIMENT-12

AIM: Program to demonstrate basic operations on single array.

Program

```
#basic operations on single array.
import numpy as np
a=np.array([1,2,5,3])
#add 1 to every element
print("Adding 1 to every element:",a+1)
#Substracting 3 from each element
print("Substracting 3 from each element:",a-3)
#multipy each element by 10 print("Multiplying
each element by 10:",a*10) #Square each
element
print("Squaring each element :",a**2)
#modify existing array
a*=2
print("Doubled each element of original array:",a)
#transpose of array
a=np.array([[1,2,3],[3,4,5],[9,6,0]])
print("\nOriginal array :\n",a)
print("\nTranspose of array :\n",a.T)
```

Output

```
Adding 1 to every element: [2 3 6 4]
Substracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element : [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array :
 [[1 2 3]
 [3 4 5]
 [9 6 0]]

Transpose of array :
 [[1 3 9]
 [2 4 6]
 [3 5 0]]

Process finished with exit code 0
```

EXPERIMENT-13

AIM: Program to demonstrate basic operations on single array.

Program

```
#basic operations on single array.
import numpy as np
a=np.array([1,2,5,3])
#add 1 to every element
print("Adding 1 to every element:",a+1)
#Substracting 3 from each element
print("Substracting 3 from each element:",a-3)
#multipy each element by 10 print("Multiplying
each element by 10:",a*10) #Square each
element
print("Squaring each element :",a**2)
#modify existing array
a*=2
print("Doubled each element of original array:",a)
#transpose of array
a=np.array([[1,2,3],[3,4,5],[9,6,0]])
print("\nOriginal array :\n",a)
print("\nTranspose of array :\n",a.T)
```

Output

```
Adding 1 to every element: [2 3 6 4]
Substracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element : [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array :
 [[1 2 3]
 [3 4 5]
 [9 6 0]]

Transpose of array :
 [[1 3 9]
 [2 4 6]
 [3 5 0]]

Process finished with exit code 0
```

EXPERIMENT-14

AIM: Using csv file plot a graph by matplotlib library

Program

```
import pandas as pd df=pd.read_csv("C:/Users/ajcemca/Desktop/Datascience
Lab/35_Teena RoseMathew/data.csv")
print(df.head(5))
print(df.tail(5))
print(df.shape)
print(df.head(5))
```

Output

```
   Roll_no  Name  Cloud  IOT  DAA
0        1   ANU     87   59   80
1        2  BINU     40   56   87
2        3  CINU     35   40   60
3        4  DILU     50   60   70
   Roll_no  Name  Cloud  IOT  DAA
0        1   ANU     87   59   80
1        2  BINU     40   56   87
2        3  CINU     35   40   60
3        4  DILU     50   60   70
(4, 5)
   Roll_no  Name  Cloud  IOT  DAA
0        1   ANU     87   59   80
1        2  BINU     40   56   87
2        3  CINU     35   40   60
3        4  DILU     50   60   70

Process finished with exit code 0
```

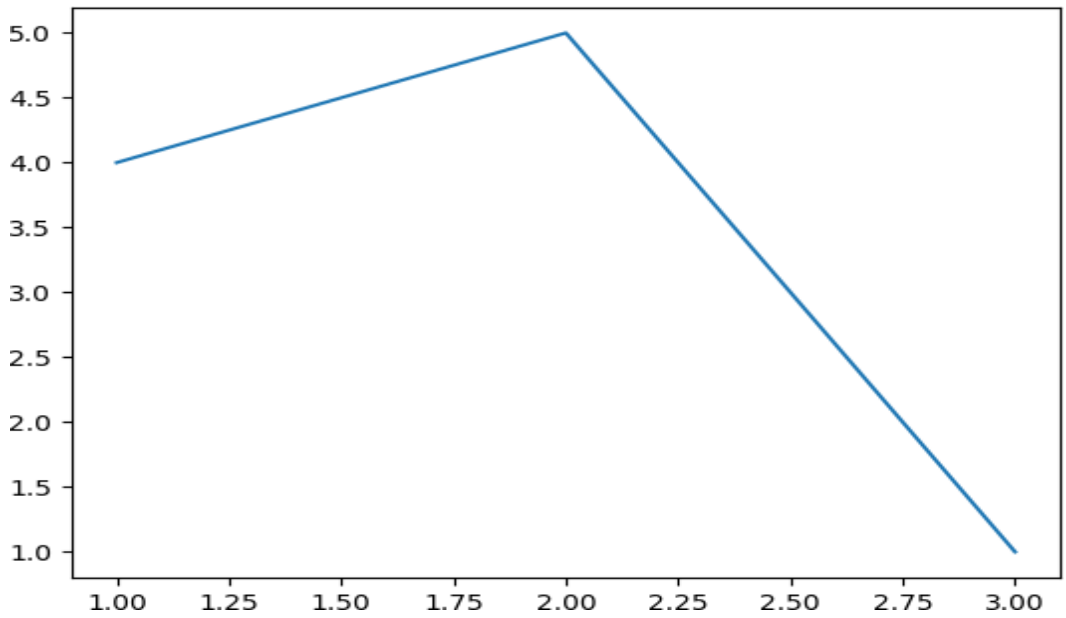
EXPERIMENT- 15

AIM: Using csv file plot a graph by matplotlib library

Program

```
import matplotlib.pyplot as plt
import csv

Subjects =
[]Scores =
[]

with open('C:/Users/ajcemca/Desktop/Datascience Lab/35_Teena Rose
Mathew/marks_9.csv', 'r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        Subjects.append(row[0])
        Scores.append(int(row[1]))

plt.pie(Scores, labels=Subjects, autopct='%.2f%%')
plt.title('Marks of a Student', fontsize=20) plt.show()
```

Output



Date: 02/12/2021

EXPERIMENT- 16

AIM: Program to implement K-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

Program

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
irisData = load_iris()
m = irisData.data
n = irisData.target
m_train, m_test, n_train, n_test = train_test_split(m,n, test_size=0.2, random_state=46)
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(m_train, n_train)
print(knn.predict(m_test))
p = knn.predict(m_test)
q = accuracy_score(n_test, p)
print("accuracy of the algorithm is:", q)
```

Output

```
C:\Users\hp\PycharmProjects\pythonProject3\venv\Scripts\python.exe C:/Users/hp/P
[0 1 1 0 0 2 0 1 1 1 1 2 0 2 0 0 0 0 1 2 0 2 0 1 2 0 1 1 2 2]
accuracy of the algorithm is: 0.9

Process finished with exit code 0
```

EXPERIMENT- 17

AIM: Program to implement K-NN classification using any random data, without using inbuilt package.

Program

```
from math import sqrt
# calculate the Euclidean distance between two vectors    distance = 0.0

def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1) - 1):
        distance += (row1[i] - row2[i]) ** 2
```

```
    return sqrt(distance)

# Locate the closest neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# Test distance function
dataset = [[2.7810836, 2.550537003, 0],
           [1.465489372, 2.362125076, 0],
           [3.396561688, 4.400293529, 0],
           [1.38807019, 1.850220317, 0],
           [3.06407232, 3.005305973, 0],
           [7.627531214, 2.759262235, 1],
           [5.332441248, 2.088626775, 1],
           [6.922596716, 1.77106367, 1],
           [8.675418651, -0.242068655, 1],
           [7.673756466, 3.508563011, 1]]

prediction = predict_classification(dataset, dataset[0], 5)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
neighbors = get_neighbors(dataset, dataset[0], 3)

for neighbor in neighbors:
    print(neighbor)
```

Output

```
C:\Users\Student\PycharmProjects\pythonProject\venv\Scripts\python.exe
Expected 0, Got 0.
[2.7810836, 2.550537003, 0]
[3.06407232, 3.005305973, 0]
[1.465489372, 2.362125076, 0]

Process finished with exit code 0
```

EXPERIMENT- 18

AIM: Confusion Matrix

Program

```
# confusion matrix in sklearn
from sklearn.metrics import confusion_matrix from
sklearn.metrics import classification_report

# actual values
actual = [1,0,0,1,0,0,1,0,0,1]
# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]

# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0])
print('Confusion matrix : \n',matrix)

# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n',matrix)
```

Output

```
Confusion matrix :
 [[2 2]
 [1 5]]
Outcome values :
 2 2 1 5
Classification report :
              precision    recall  f1-score   support

           1       0.67      0.50      0.57         4
           0       0.71      0.83      0.77         6

    accuracy                           0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10


Process finished with exit code 0
```

Date: 09/12/2021

EXPERIMENT-19

AIM: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm

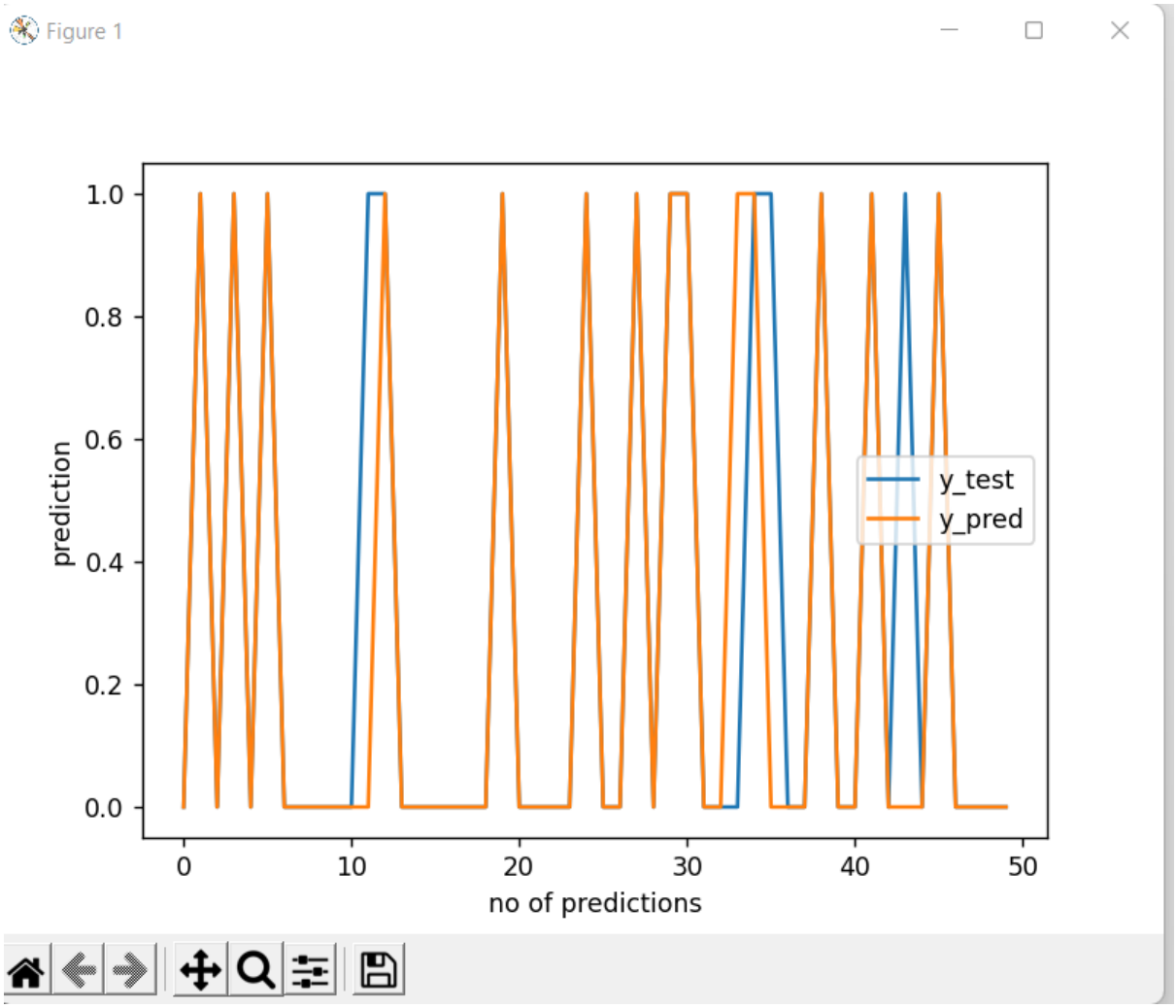
Program

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
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, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred)

print("Accuracy is:", ac)
print("Confusion Matrix\n",cm)
plt.plot([i for i in range(0, 50)], y_test[20:70])
plt.plot([i for i in range(0, 50)], y_pred[20:70])
plt.xlabel("no of predictions")
```

```
plt.ylabel("prediction")
plt.legend(["y_test", "y_pred"])
plt.show()
```

Output

```
C:\Users\hp\PycharmProjects\pythonProject3\venv\Script
Accuracy is: 0.9125
Confusion Matrix
 [[55  3]
 [ 4 18]]
```



DATE: 13/12/2021

EXPERIMENT- 20

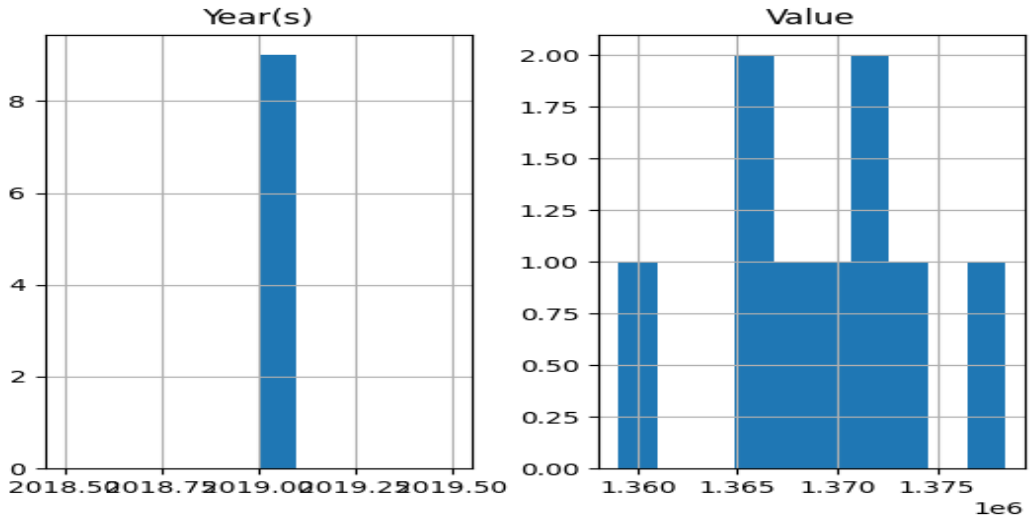
AIM: Data visualization Graphs

1. HISTOGRAM

PROGRAM

```
import pandas as pd
import matplotlib.pyplot as plt
data = [
['India', 2019, 'Medium', 1368737.513],
['India', 2019, 'High', 1378419.072],
['India', 2019, 'Low', 1359043.965],
['India', 2019, 'Constant fertility', 1373707.838],
['India', 2019,'Instant replacement', 1366687.871],
['India', 2019, 'Zero migration', 1370868.782],
['India', 2019,'Constant mortality', 1366282.778],
['India', 2019, 'No change', 1371221.64],
['India', 2019, 'Momentum', 1367400.614],]
df = pd.DataFrame(data, columns = ([ 'Country or Area', 'Year(s)', 'Variant', 'Value']))
df.hist()
plt.show()
```

OUTPUT



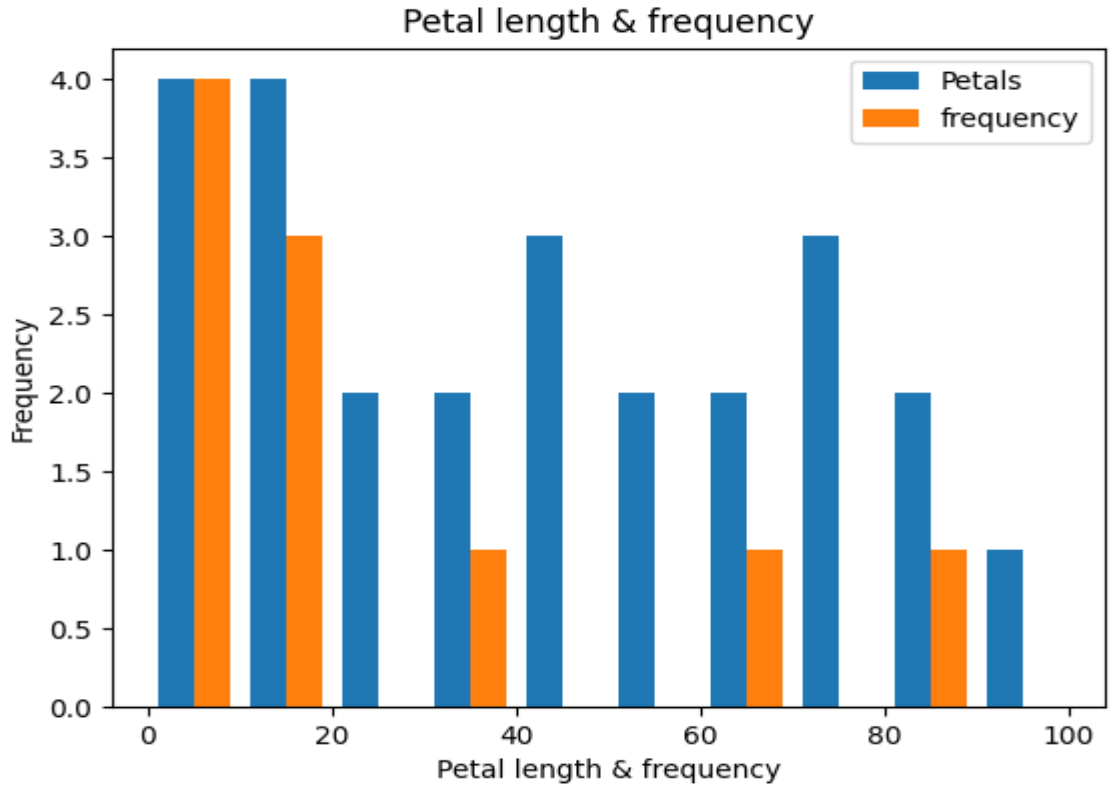
2. CLASS STRATIFIED HISTOGRAM

PROGRAM

```
from matplotlib import pyplot as plt
import numpy as np
a = np.array([2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,
        71,73,79,83,89,97]) # primes
```

```
b=np.array([2,3,5,7,13,17,19,31,61,89]) # exponents
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist([a,b],bins,label=['Petals','frequency'])
plt.legend(loc='upper right')
plt.title("Petal length & frequency")
plt.xlabel("Petal length & frequency")
plt.ylabel("Frequency")
plt.show()
```

OUTPUT



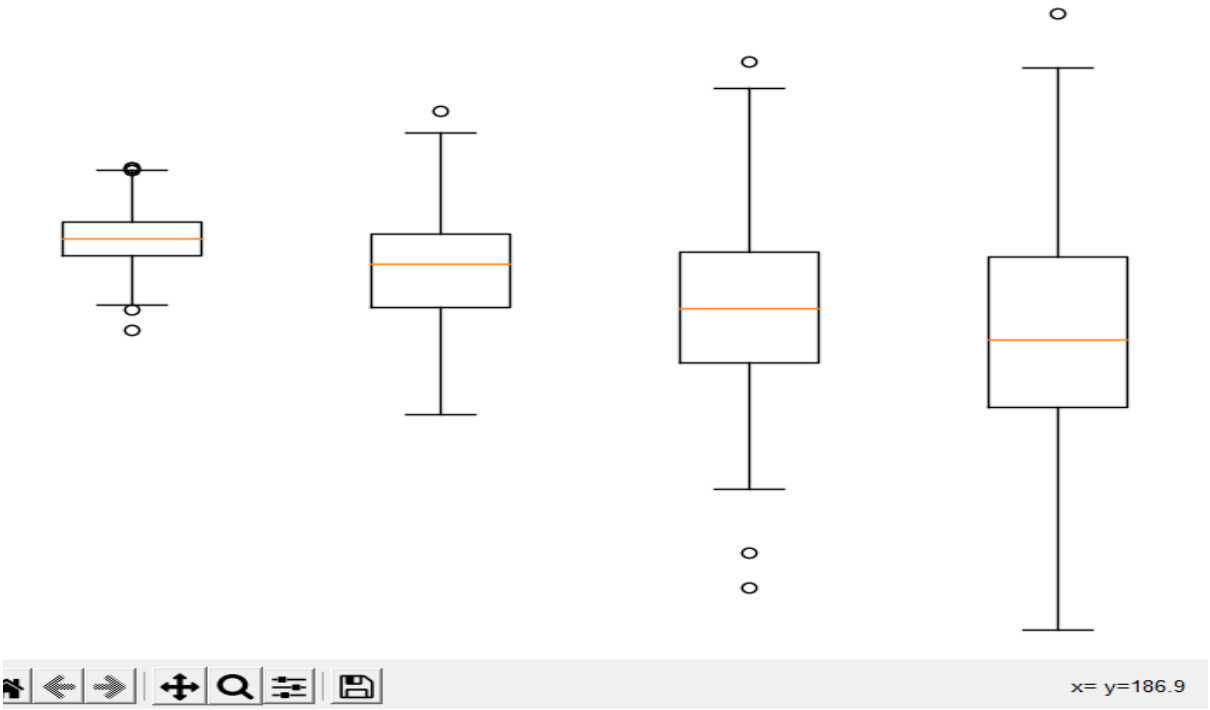
3. BOX WHISKER PLOT(QUANTILE PLOT)

PROGRAM

```
# 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()
```

OUTPUT



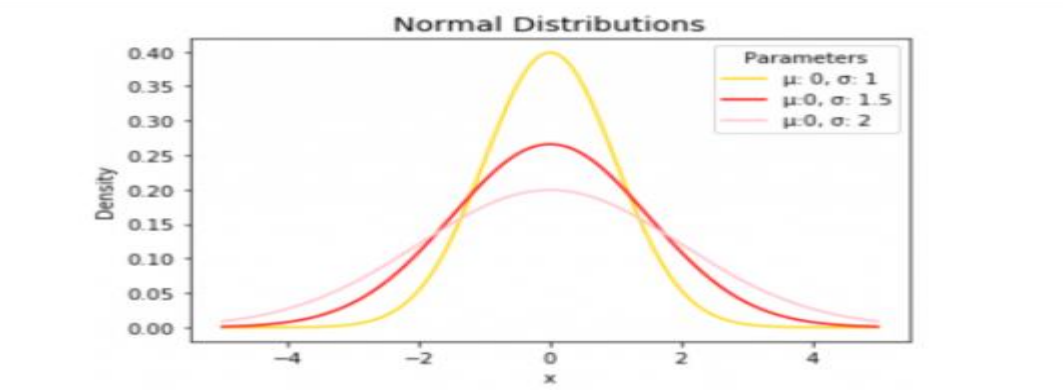
4. DISTRIBUTION PLOT

PROGAM

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm

#x-axis ranges from -5 and 5 with .001 steps

x = np.arange(-5, 5, 0.001)

#define multiple normal distributions

plt.plot(x, norm.pdf(x, 0, 1), label='μ: 0, σ: 1', color='gold')

plt.plot(x, norm.pdf(x, 0, 1.5), label='μ:0, σ: 1.5', color='red')
```

```
plt.plot(x, norm.pdf(x, 0, 2), label='μ:0, σ: 2', color='pink')

#add legend to plot
plt.legend(title='Parameters')

#add axes labels and a title
plt.ylabel('Density')
plt.xlabel('x')
plt.title('Normal Distributions', fontsize=14)
```

OUTPUT

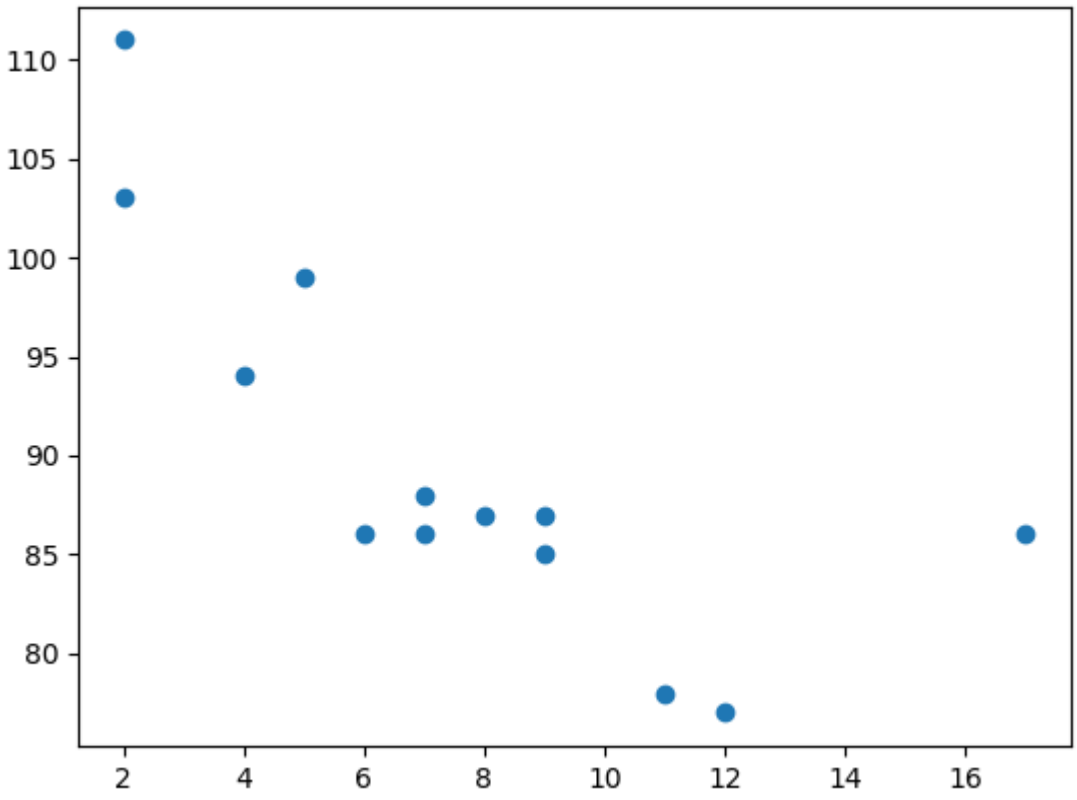


5. SCATTER PLOT

PROGRAM

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

OUTPUT

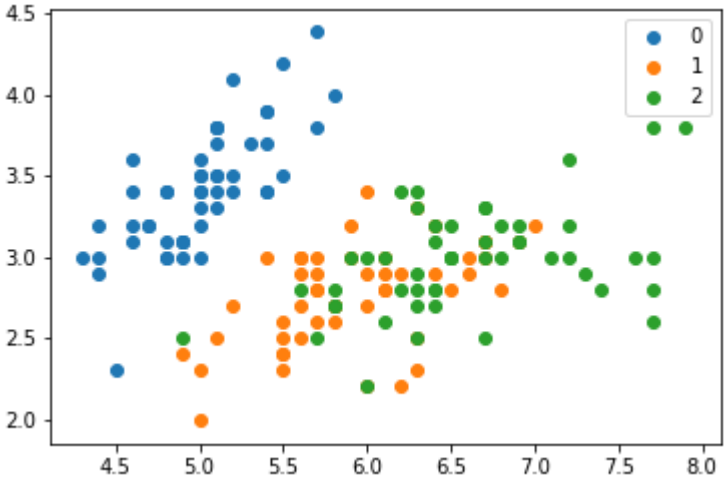


6. MULTIPLE SCATTER

PROGRAM

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
feats = load_iris()['data']
target = load_iris()['target']

f, ax = plt.subplots(1)
for i in np.unique(target):
    mask = target == i
    plt.scatter(feats[mask, 0], feats[mask, 1], label=i)
ax.legend()
```

OUTPUT

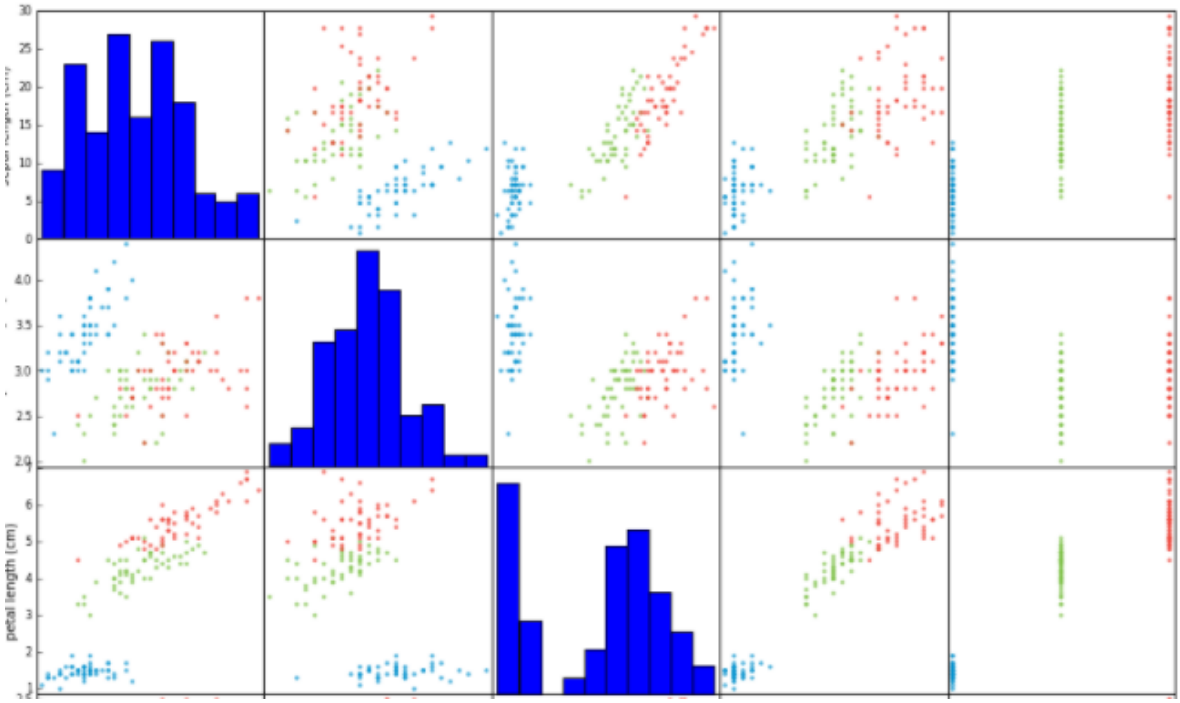


7. MULTIPLE SCATTER MATRIX

PROGRAM

```
from pandas.tools.plotting import scatter_matrix
import pandas as pd
from sklearn import datasets

iris = datasets.load_iris()
iris_data = pd.DataFrame(data=iris['data'],columns=iris['feature_names'])
iris_data["target"] = iris['target']

color_wheel = {1: "#0392cf",
               2: "#7bc043",
               3: "#ee4035"}
colors = iris_data["target"].map(lambda x: color_wheel.get(x + 1))
ax = scatter_matrix(iris_data, color=colors, alpha=0.6, figsize=(15, 15), diagonal='hist')
```

OUTPUT

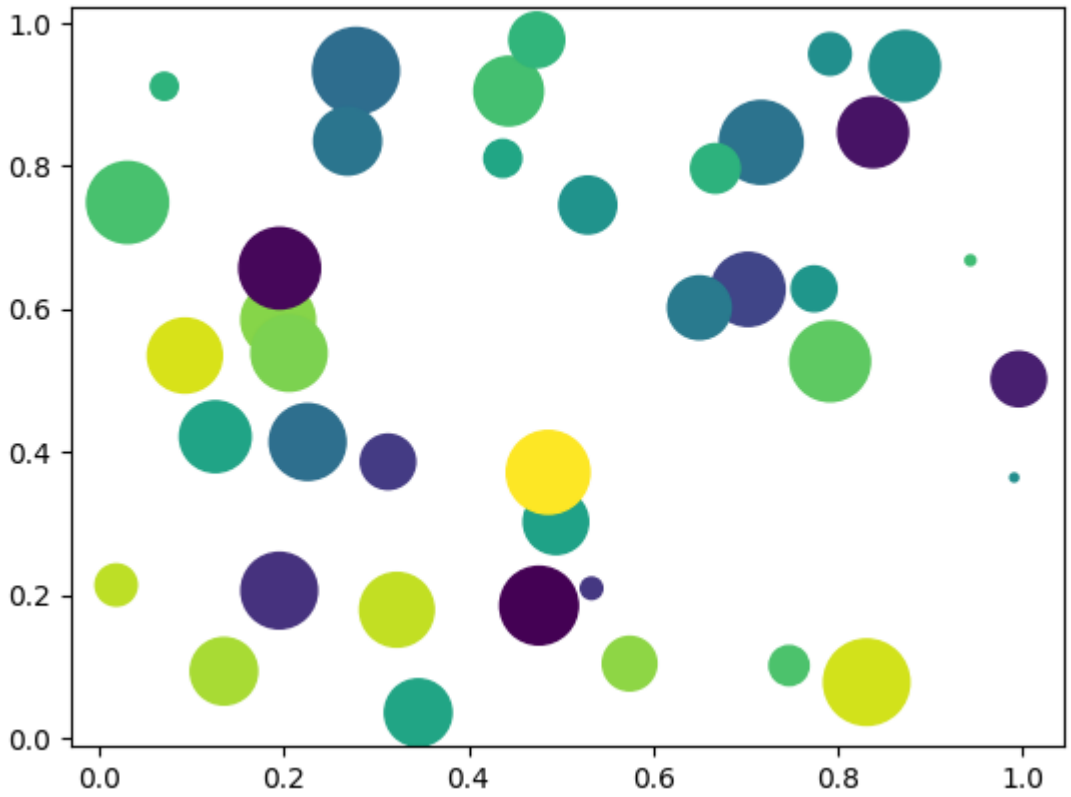


8. BUBBLE PLOT

PROGRAM

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

# create data
x = np.random.rand(40)
y = np.random.rand(40)
z = np.random.rand(40)
colors = np.random.rand(40)
# use the scatter function
plt.scatter(x, y, s=z * 1000, c=colors)
plt.show()
```

OUTPUT

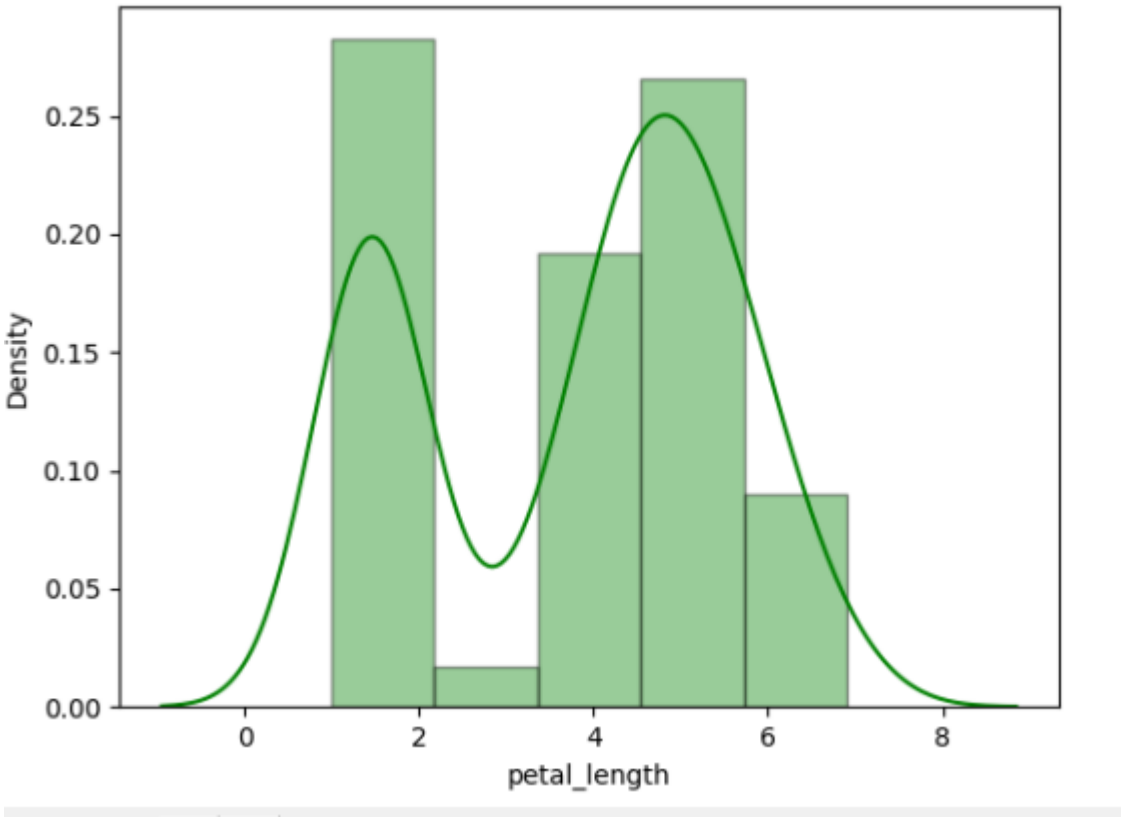


9. DENSITY CHART

PROGRAM

```
#importing various libraries
import seaborn as sns
import matplotlib..pyplot as plt
#importing iris dataset from the library
df2=sns.load dataset('iris')
#plotting histogram and density plot for
#petal length using displot() by setting color
sns.distplot(a=df2.petal_length,color='green',
                hist_kws={"edgecolor":'black'})
#visualizing plot using matplotlib.pyplot library
plt.show()
```

OUTPUT

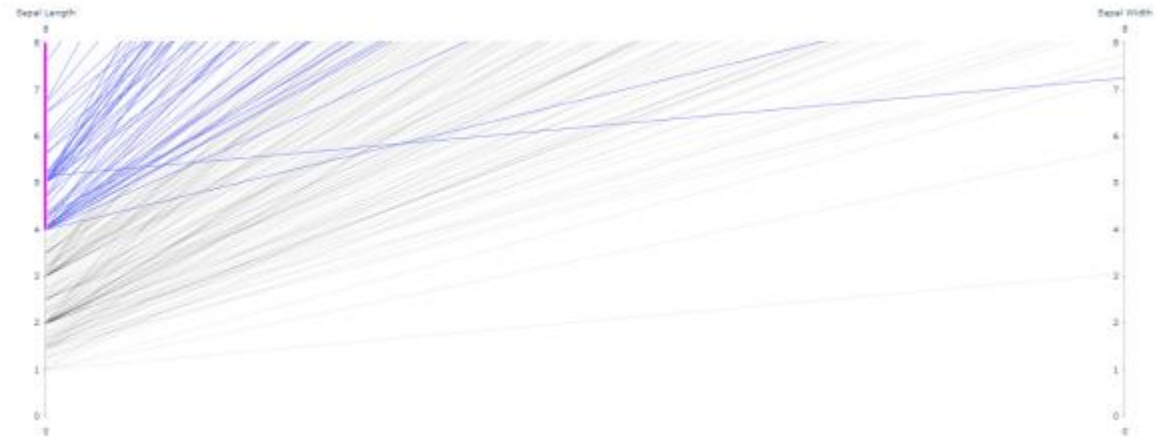


10. PARALLEL PLOT

PROGRAM

```
# importing various package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# making data frame from csv file
df = pd.read_csv(
    'https://raw.github.com/pandas-dev/'
    'pandas/master/pandas/tests/io/data/csv/iris.csv'
)

# Creating Andrews curves
x = pd.plotting.andrews_curves(df, 'Name')

# ploting the Curve
x.plot()

# Display
plt.show()
```

OUTPUT

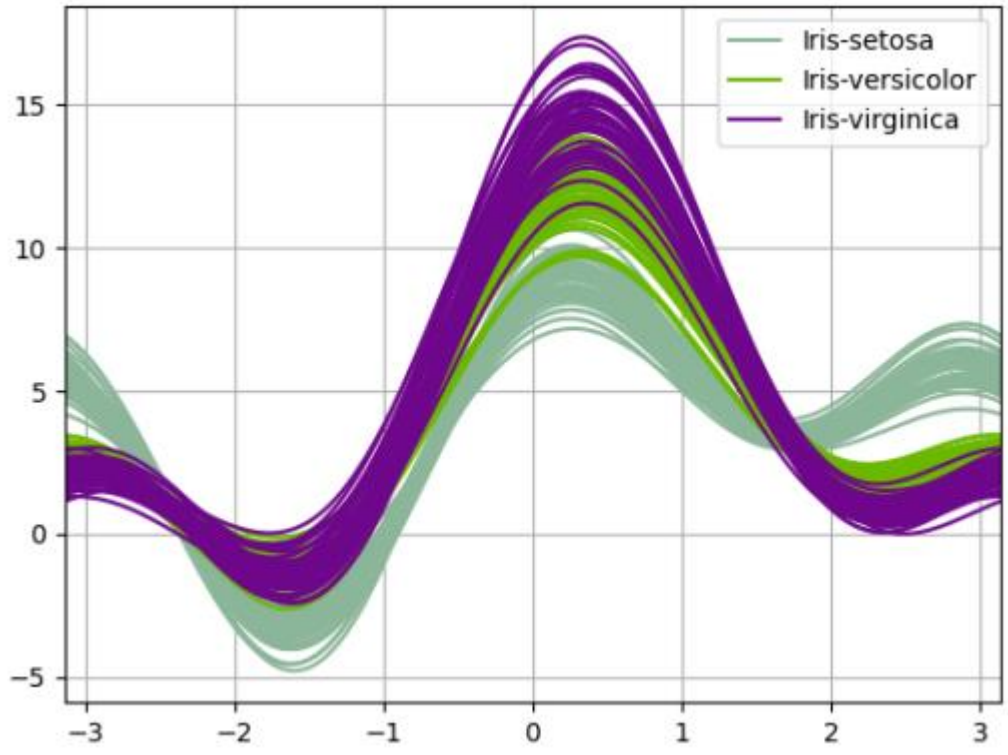


11. ANDREWS CURVE

PROGRAM

```
# importing various package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# making data frame from csv file
df = pd.read_csv(
    'https://raw.github.com/pandas-dev/'
    'pandas/master/pandas/tests/io/data/csv/iris.csv'
)

# Creating Andrews curves
x = pd.plotting.andrews_curves(df, 'Name')

# ploting the Curve
x.plot()

# Display
plt.show()
```

OUTPUT


Iris-setosa
Iris-versicolor
Iris-virginica
15
10
5
0
−5
−3
−2
−1
0
1
2
3

DATE: 06/01/2022

EXPERIMENT- 21

AIM: Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Program

```
# Run this program on your local python
# interpreter, provided you have installed
# the required libraries.

# Importing the required packages

import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# Function importing Dataset
def importdata():
    balance_data = pd.read_csv(
        'https://archive.ics.uci.edu/ml/machine-learning-' +
        'databases/balance-scale/balance-scale.data',
        sep=',', header=None)

    # Printing the dataswet shape
    print("Dataset Length: ", len(balance_data))
    print("Dataset Shape: ", balance_data.shape)

    # Printing the dataset obseravtions
    print("Dataset: ", balance_data.head())
    return balance_data


# Function to split the dataset
def splitdataset(balance_data):
    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=0.3, random_state=100)

    return X, Y, X_train, X_test, y_train, y_test
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
```

```
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion="gini",
                                      random_state=100, max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini


# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):
    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
        criterion="entropy", random_state=100,
        max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):
    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ",
          confusion_matrix(y_test, y_pred))

    print("Accuracy : ",
          accuracy_score(y_test, y_pred) * 100)



# Driver code
def main():
    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
```

```
    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__ == "__main__":
    main()
```

Output

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
```

```
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
Dataset Length:  625
Dataset Shape:  (625, 5)
```

```
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
```

```
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383

Process finished with exit code 0
```