FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)™

HORMIS NAGAR, MOOKKANNOOR

ANGAMALY-683577

'FOCUS ON EXCELLENCE'

DATA SCIENCE

……

LABORATORY RECORD

Name: NEEMA POULOSE

Branch: MASTER OF COMPUTER APPLICATION

Semester: 3 **Batch: B** **Roll No: 23**

FEDERAL INSTITUTE OF

SCIENCE AND TECHNOLOGY

(FISAT)™

HORMIS NAGAR, MOOKKANNOOR

ANGAMALY-683577

'FOCUS ON EXCELLENCE'

Name : NEEMA POULOSE

Branch : MASTER OF COMPUTER APPLICATION

Semester : 3 **Roll No: 23**

University Exam.Reg. No:

CERTIFICATE

This is to certify that this is a Bonafide record of the Practical work done and submitted to Kerala Technological University in partial fulfillment for the award of the Master Of Computer Applications is a record of the original research work done by ***NEEMA POULOSE*** *in the* ***DATA SCIENCE*** *Laboratory of the Federal Institute of Science and Technology during the academic year 2021-2022.*

Signature of Staff in Charge

Name:

Date:

Signature of H.O.D

Name:

Date of University practical examination ……

Signature of

Internal Examiner

Signature of

External Examiner

CONTENT

8	21/12/2022	Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm	48	
9	13/01/2022	Program to implement k-means clustering technique using any standard dataset available in the public domain	50	
10	27/01/2022	Programs on feedforward network to classify any standard dataset available in the public domain	55	
11	27/01/2022	Programs on convolutional neural network to classify images from any standard dataset in the public domain.	57	

AIM

1. Matrix operations (using vectorization) and transformation using python and SVD using Python.

CODE:

```
import numpy as np
y=np.arange(1,26)
print(y)

y=y.reshape(5,5)
print(y)

print(y[:5,:5])
print(y[:,:-1])
print(y[::,::2])
print(y[1::2,::])
print(y[1::2,1::2])
```

OUTPUT:

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
[[ 1  2  3  4]
 [ 6  7  8  9]
 [11 12 13 14]
 [16 17 18 19]
 [21 22 23 24]]
[[ 1  3  5]
 [ 6  8 10]
 [11 13 15]
 [16 18 20]
 [21 23 25]]
[[ 6  7  8  9 10]
 [16 17 18 19 20]]
[[ 7  9]
 [17 19]]
```

CODE:

```
import numpy
x=numpy.array([[1,2],[4,5]])
y=numpy.array([[7,8],[9,10]])
print(np.add(x,y))
print(np.subtract(x,y))
print(np.divide(x,y))
print(np.dot(x,y))
```

OUTPUT:

```
[[ 8 10]
 [13 15]]
[[-6 -6]
 [-5 -5]]
[[0.14285714 0.25      ]
 [0.44444444 0.5       ]]
[[25 28]
 [73 82]]
```

CODE:

```
print(x.sum())
print(x.sum(axis=0))
print(x.sum(axis=1))
print(x.max())
print(x.transpose())
```

OUTPUT:

```
12
[5 7]
[3 9]
5
[[1 4]
 [2 5]]
```

CODE:

```
print(y[4:-1])
```

OUTPUT:

```
[ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
```

AIM

2. Programs using matplotlib / plotly / bokeh / seaborn for data visualisation.

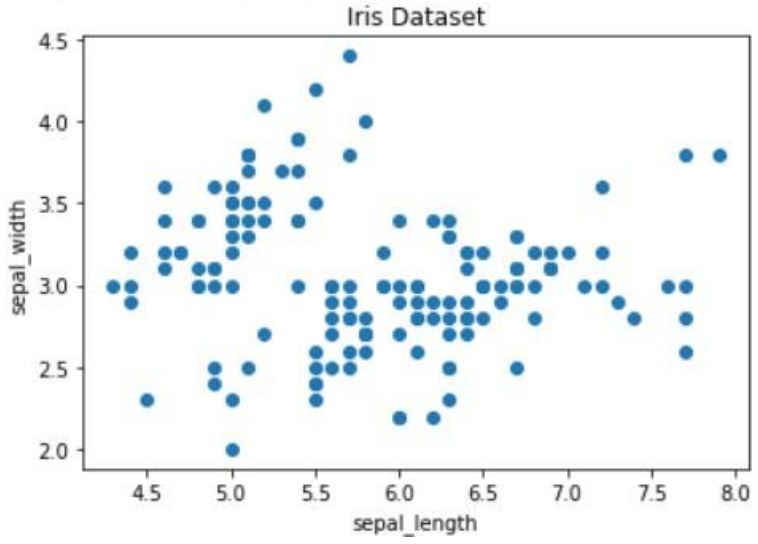
Dataset used: iris.csv

CODE:

```
import pandas as pd
iris = pd.read_csv('/content/iris.csv')

import matplotlib.pyplot as plt
fig, ax = plt.subplots()

# scatter the sepal_length against the sepal_width
ax.scatter(iris['sepal.length'], iris['sepal.width'])
# set a title and labels
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```

OUTPUT:

```
Text(0, 0.5, 'sepal_width')
```

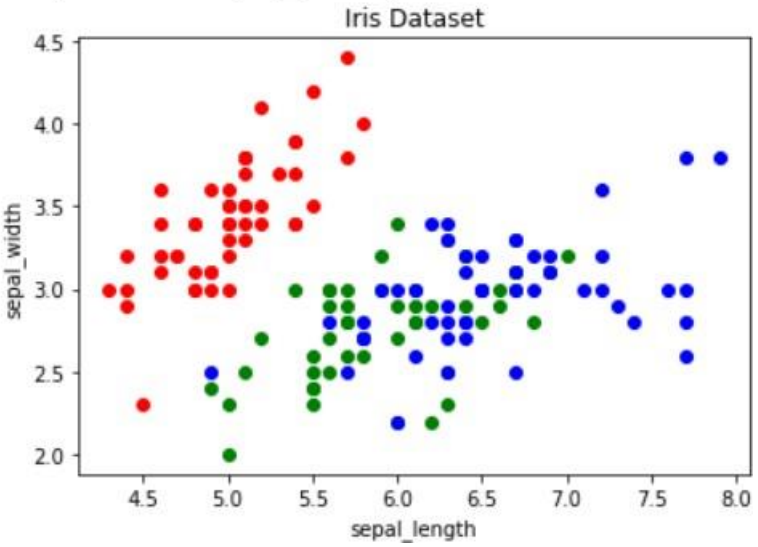


CODE:

```
#matplotlib plot with diferent colors for Iris flower varities
fig, ax = plt.subplots()
colors = {'Setosa':'r', 'Versicolor':'g', 'Virginica':'b'}

for i in range(len(iris['sepal.length'])):
    ax.scatter(iris['sepal.length'][i], iris['sepal.width'][i],co
lor=colors[iris['variety'][i]])
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```

OUTPUT:

```
Text(0, 0.5, 'sepal_width')
```

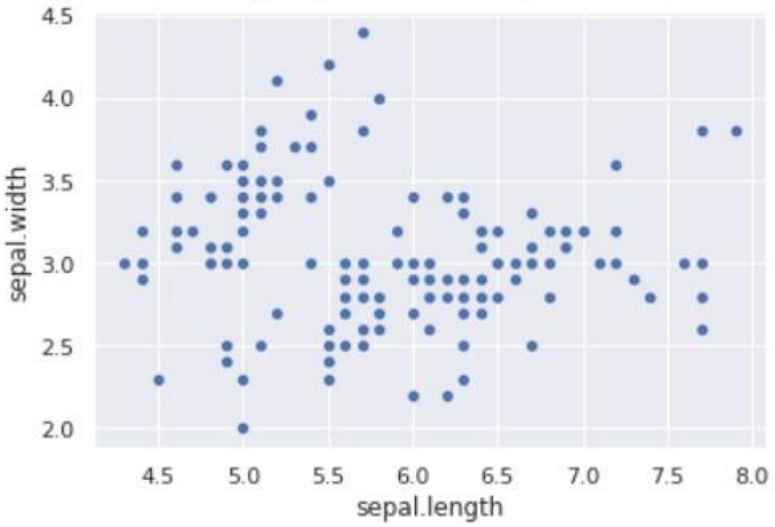


CODE:

```
import seaborn as sns
sns.scatterplot(x='sepal.length', y='sepal.width', data=iris)
```

OUTPUT:

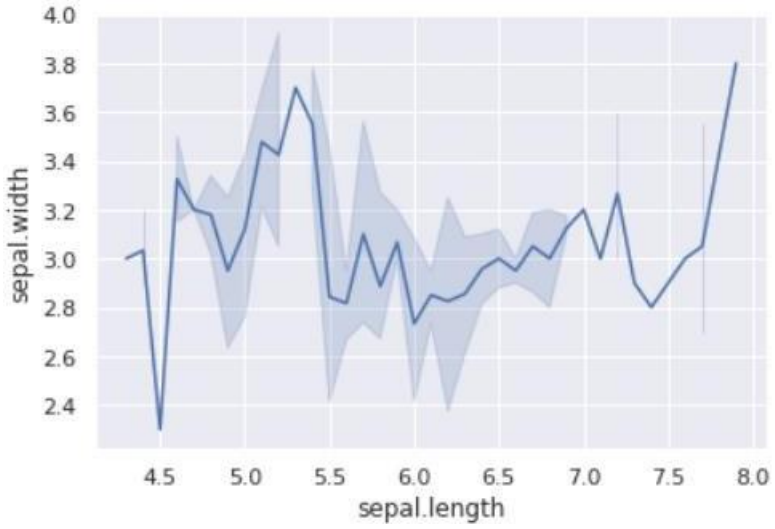
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01f4191210>
```



CODE:

```
sns.lineplot(x="sepal.length", y="sepal.width", data=iris)
plt.show()
```

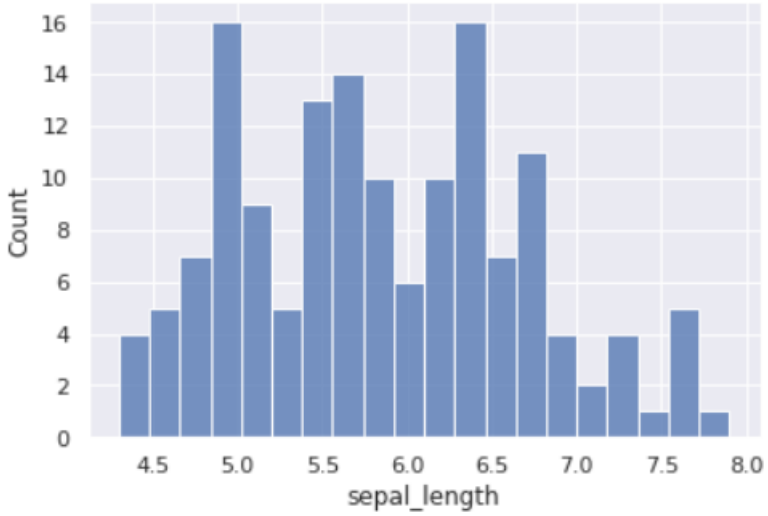
OUTPUT:



CODE:

```
#seaborn histogram plot
sns.set(style="darkgrid")
df = sns.load_dataset("iris")


sns.histplot(data=df, x="sepal_length",bins=20)
plt.show()
```

OUTPUT:

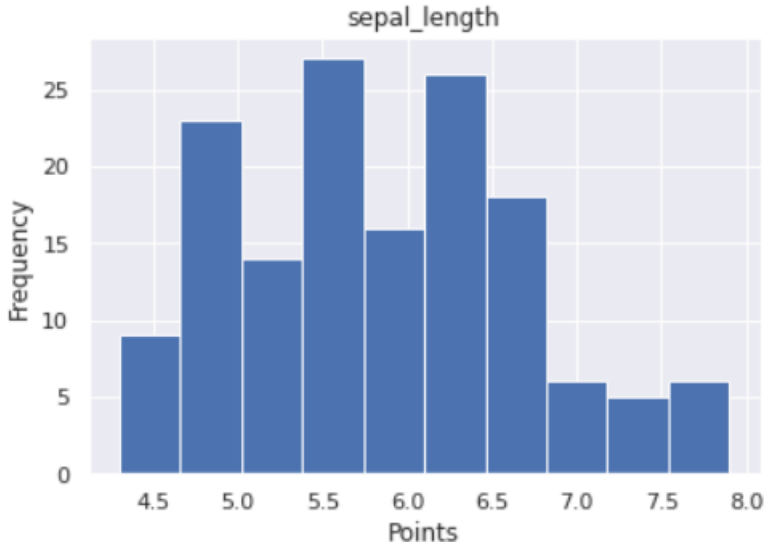


CODE:

```
#matplotlib histogram plot
iris_feat = iris.iloc[:,:-1]
iris_species = iris.iloc[:,-1]
fig, ax = plt.subplots()
# plot histogram
ax.hist(iris_feat['sepal.length'])
# set title and labels
ax.set_title('sepal_length')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```

OUTPUT:

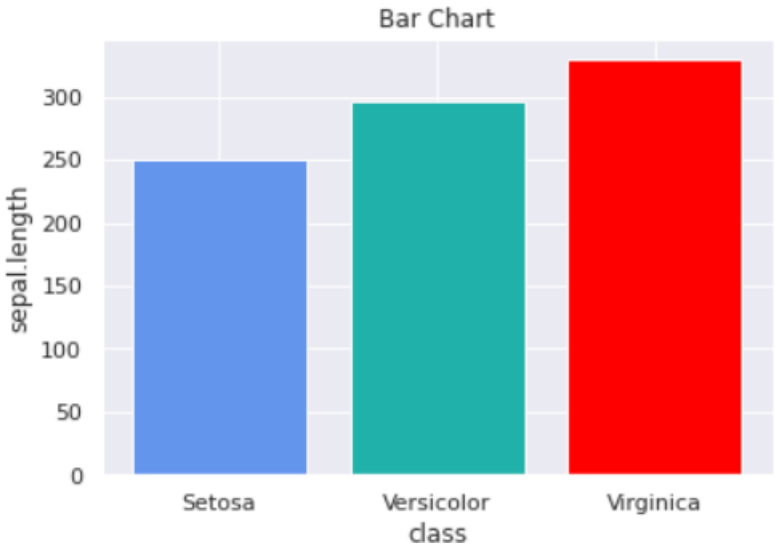
```
Text(0, 0.5, 'Frequency')
```



CODE:

```
#Bar chart using Matplotlib
df = iris.groupby('variety')['sepal.length'].sum().to_frame().rese
t_index()
#Creating the bar chart
plt.bar(df['variety'],df['sepal.length'],color = ['cornflowerblue'
,'lightseagreen','red'])
#Adding the aesthetics
plt.title('Bar Chart')
plt.xlabel('class')
plt.ylabel('sepal.length')
#Show the plot
plt.show()
```

OUTPUT:

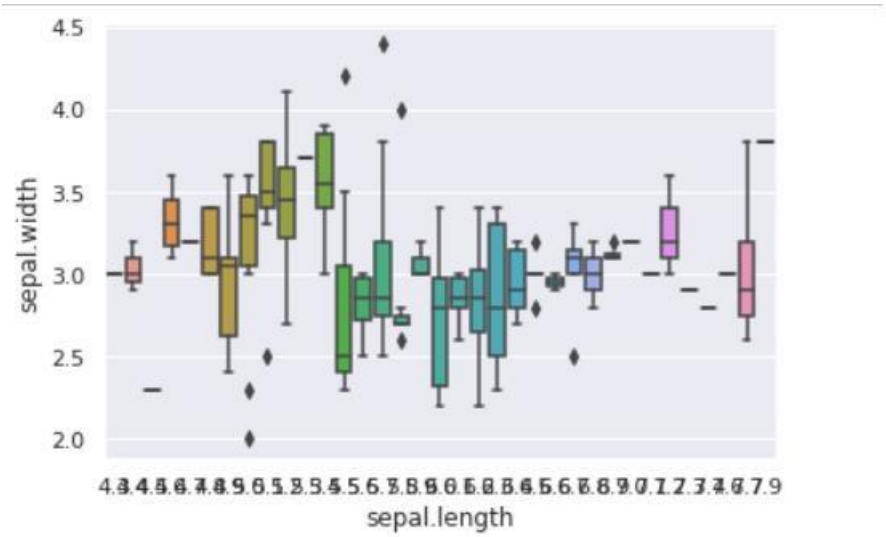


CODE

```
import seaborn as sns
sns.boxplot('sepal.length', 'sepal.width', data=iris)
```

OUTPUT:

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y.
From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f01ef6944d0>
```



CODE:

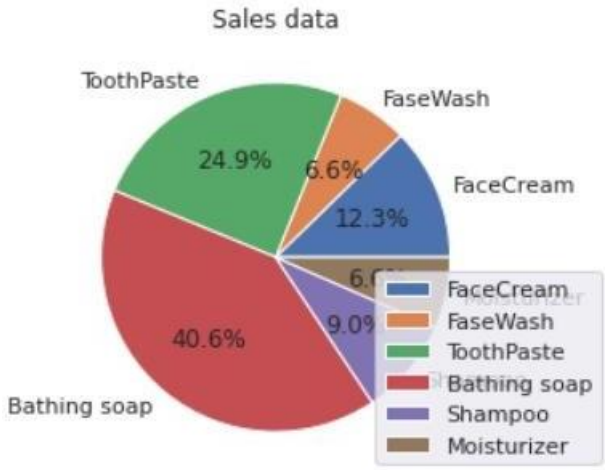
```
import pandas as pd
iris = pd.read_csv('/content/company_sales_data.csv')


#Line plot with matplotlib
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data.csv")
profitList = df ['total_profit'].tolist()
monthList  = df ['month_number'].tolist()
plt.plot(monthList, profitList, label = 'Month-
wise Profit data of last year')
plt.xlabel('Month number')
plt.ylabel('Profit in dollar')
plt.xticks(monthList)
plt.title('Company profit per month')
plt.yticks([100000, 200000, 300000, 400000, 500000])
plt.show()
```

OUTPUT:

CODE:

```
monthList  = df ['month_number'].tolist()

labels = ['FaceCream', 'FaseWash', 'ToothPaste', 'Bathing soap',
'Shampoo', 'Moisturizer']
salesData   = [df ['facecream'].sum(), df ['facewash'].sum(), df
['toothpaste'].sum(),
         df ['bathingsoap'].sum(), df ['shampoo'].sum(), df ['moi
sturizer'].sum()]
plt.axis("equal")
plt.pie(salesData, labels=labels, autopct='%1.1f%%')
plt.legend(loc='lower right')
plt.title('Sales data')
plt.show()
```

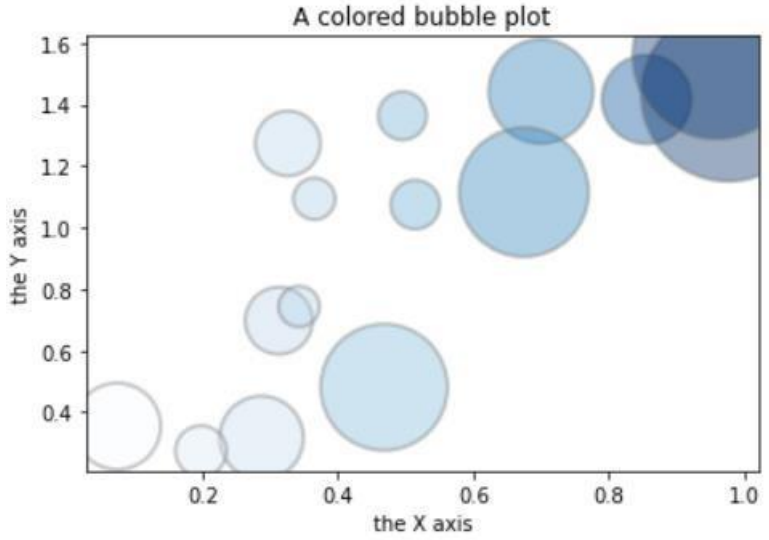
OUTPUT:



CODE:

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

# create data
x =  np.random.rand(15)
y = x+np.random.rand(15)
z = x+np.random.rand(15)
z=z*z

# Change color with c and transparency with alpha.
# I map the color to the X axis value.
plt.scatter(x, y, s=z*2000, c=x, cmap="Blues", alpha=0.4, edgecol
ors="grey", linewidth=2)

# Add titles (main and on axis)
plt.xlabel("the X axis")
plt.ylabel("the Y axis")
plt.title("A colored bubble plot")

# Show the graph
plt.show()
```

OUTPUT:


A colored bubble plot
the Y axis
the X axis
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.4
0.6
0.8
1.0

AIM

3. Programs to handle data using pandas

CODE:

```
import numpy as np

import pandas as pd

s = pd.Series([1, 3, 5, 6, 8])
print(s)
```

OUTPUT:

```
0    1
1    3
2    5
3    6
4    8
dtype: int64
```

CODE:

```
dict = {"country": ["Brazil", "Russia", "India", "China", "South
Africa"],
       "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing",
"Pretoria"],
       "area": [8.516, 17.10, 3.286, 9.597, 1.221],
       "population": [200.4, 143.5, 1252, 1357, 52.98] }
b = pd.DataFrame(dict)
print(b)
```

OUTPUT:

	country	capital	area	population
0	Brazil	Brasilia	8.516	200.40
1	Russia	Moscow	17.100	143.50
2	India	New Dehli	3.286	1252.00
3	China	Beijing	9.597	1357.00
4	South Africa	Pretoria	1.221	52.98

CODE:

```
b.index = ["BR", "RU", "IN", "CH", "SA"]
print(b)
```

OUTPUT:

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Dehli	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Dataset used: cars1.csv

CODE:

```
import pandas as pd
cars = pd.read_csv('cars1.csv')
print(cars)
```

OUTPUT:

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105
5	VW	Up!	1000	929	105
6	Skoda	Fabia	1400	1109	90
7	Mercedes	A-Class	1500	1365	92
8	Ford	Fiesta	1500	1112	98
9	Audi	A1	1600	1150	99
10	Hyundai	I20	1100	980	99
11	Suzuki	Swift	1300	990	101
12	Ford	Fiesta	1000	1112	99
13	Honda	Civic	1600	1252	94
14	Hundai	I30	1600	1326	97
15	Opel	Astra	1600	1330	97
16	BMW	1	1600	1365	99
17	Mazda	3	2200	1280	104
18	Skoda	Rapid	1600	1119	104
19	Ford	Focus	2000	1328	105
20	Ford	Mondeo	1600	1584	94
21	Opel	Insignia	2000	1428	99
22	Mercedes	C-Class	2100	1365	99
23	Skoda	Octavia	1600	1415	99
24	Volvo	S60	2000	1415	99
25	Mercedes	CLA	1500	1465	102
26	Audi	A4	2000	1490	104
27	Audi	A6	2000	1725	114
28	Volvo	V70	1600	1523	109
29	BMW	5	2000	1705	114
30	Mercedes	E-Class	2100	1605	115
31	Volvo	XC70	2000	1746	117
32	Ford	B-Max	1600	1235	104
33	BMW	216	1600	1390	108
34	Opel	Zafira	1600	1405	109
35	Mercedes	SLK	2500	1395	120

CODE:

```
# Print out first 4 observations
print(cars[0:4])

# Print out fifth and sixth observation
print(cars[4:6])
```

OUTPUT:

```
          Car       Model  Volume  Weight  CO2
0      Toyoty        Aygo    1000     790   99
1  Mitsubishi  Space Star    1200    1160   95
2       Skoda      Citigo    1000     929   95
3        Fiat         500     900     865   90
    Car   Model  Volume  Weight  CO2
4  Mini  Cooper    1500    1140  105
5    VW     Up!    1000     929  105
```

CODE:

```
import pandas as pd
cars = pd.read_csv('cars1.csv', index_col = 0) #first column is t
aen as index column


print(cars.iloc[2])
```

OUTPUT:

```
Model     Citigo
Volume      1000
Weight       929
CO2           95
Name: Skoda, dtype: object
```

CODE:

```
#Slicing dataframe
import pandas as pd

df = pd.DataFrame([['Jay','M',18],['Jennifer','F',17],
                   ['Preity','F',19],['Neil','M',17]],
                  columns = ['Name','Gender','Age'])
print(df)
df1 = df.iloc[2:,:]
df2 = df.iloc[:2,:]
print(df1)
print(df2)
```

OUTPUT:

```
      Name Gender  Age
0      Jay      M   18
1  Jennifer      F   17
2    Preity      F   19
3      Neil      M   17
     Name Gender  Age
2  Preity      F   19
3    Neil      M   17
      Name Gender  Age
0      Jay      M   18
1  Jennifer      F   17
```

CODE:

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print(s)

print ("The actual data series is:")
print( s.values)
```

OUTPUT:

```
0   -1.138968
1   -1.097746
2    0.109717
3    1.159537
dtype: float64
The actual data series is:
[-1.13896826 -1.09774589  0.10971687  1.15953676]
```

CODE:

```
print (s.head(2))
```

OUTPUT:

```
0   -1.138968
1   -1.097746
dtype: float64
```

CODE:

```
print(s.tail(3))
```

OUTPUT:

```
1   -1.097746
2    0.109717
3    1.159537
dtype: float64
```

CODE:

```
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith
','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print(df)
print ("The transpose of the data series is:")
print(df.T)
```

OUTPUT:

```
     Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The transpose of the data series is:
          0      1      2     3      4      5     6
Name     Tom  James  Ricky   Vin  Steve  Smith  Jack
Age       25     26     25    23     30     29    23
Rating  4.23   3.24   3.98  2.56    3.2    4.6   3.8
```

CODE:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith
','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
```

```
print(df)
print ("Row axis labels and column axis labels are:")
print (df.axes)
```

OUTPUT:

```
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
Row axis labels and column axis labels are:
[RangeIndex(start=0, stop=7, step=1), Index(['Name', 'Age', 'Rating'], dtype='object')]
```

CODE:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith
','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("The data types of each column are:")
print (df.dtypes)
```

OUTPUT:

```
The data types of each column are:
Name       object
Age         int64
Rating    float64
dtype: object
```

CODE:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith
','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Is the object empty?")
print (df.empty)
```

OUTPUT:

```
Is the object empty?
False
```

CODE:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith
','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),  'Rating':pd.Series([
4.23,3.24,3.98,2.56,3.20,4.6,3.8])
   }

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print (df)
print ("The dimension of the object is:")
print (df.ndim)
```

OUTPUT:

```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The dimension of the object is:
2
```

CODE:

```
 d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smit
h','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,30]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print(df)
print ("Our object is:")
print ("The shape of the object is:")
print (df.shape)
```

OUTPUT:

```
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   30    3.80
Our object is:
The shape of the object is:
(7, 3)
```

CODE:

```
print (df.size)

print (df.values)
```

OUTPUT:

```
21


  [['Tom' 25 4.23]
   ['James' 26 3.24]
   ['Ricky' 25 3.98]
   ['Vin' 23 2.56]
   ['Steve' 30 3.2]
   ['Smith' 29 4.6]
   ['Jack' 30 3.8]]
```

CODE:

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

OUTPUT:

```
Name      0
Age       0
Rating    0
dtype: int64
```

CODE:

```
df = pd.DataFrame(np.arange(12).reshape(3, 4),

                  columns=['A', 'B', 'C', 'D'])
print(df)
```

OUTPUT:

```
   A  B   C   D
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

AIM

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

CODE:

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Over
cast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']

# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool'
,'Mild','Mild','Mild','Hot','Mild']

# Label or target varible

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Ye
s','Yes','Yes','No']



 from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```

OUTPUT:

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
```

CODE:

```
temp_encoded=le.fit_transform(temp)
print(temp_encoded)
print(" ")
label=le.fit_transform(play)
print(label)
```

OUTPUT:

```
[1 1 1 2 0 0 0 2 0 2 2 2 1 2]

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

CODE:

```
features=list(zip(weather_encoded,temp_encoded))
print(features)
```

OUTPUT:

```
[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2),
(2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
```

CODE:

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)


from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)
predicted= model.predict([[0,1]]) # 0:Overcast, 1:Hot
print(predicted)
```

OUTPUT:

```
[1]
```

Dataset used: iris.csv

CODE:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
dataset = pd.read_csv("iris.csv")
print(dataset.describe)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
```

OUTPUT:

```
<bound method NDFrame.describe of      sepal.length  sepal.width  petal.length  petal.width    variety
0             5.1          3.5           1.4          0.2     Setosa
1             4.9          3.0           1.4          0.2     Setosa
2             4.7          3.2           1.3          0.2     Setosa
3             4.6          3.1           1.5          0.2     Setosa
4             5.0          3.6           1.4          0.2     Setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]>
```

CODE:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze=0.20)


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
print(X_test)
```

OUTPUT:

```
[[ 2.17968894 -0.14585275  1.67223212  1.24723193]
 [ 1.09336496  0.52731379  1.16075926  1.24723193]
 [ 0.61055431 -1.26779698  0.76294703  0.98465678]
 [ 1.09336496 -0.14585275  0.76294703  0.72208164]
 [-0.23436434 -0.59463044  0.70611671  1.11594435]
 [-1.32068831  0.30292494 -1.33977476 -1.24723193]
 [-0.47576967  0.75170264 -1.11245349 -1.24723193]
 [-0.11366168 -0.59463044  0.47879543  0.19693136]
 [-0.83787766 -1.26779698 -0.37365934 -0.06564379]
 [-1.07928299  1.20048033 -1.28294444 -1.3785195 ]
 [ 0.61055431 -0.81901929  0.70611671  0.85336921]
 [-0.83787766  1.64925802 -0.99879285 -0.98465678]
 [ 0.12774365  0.30292494  0.64928639  0.85336921]
 [-0.11366168 -0.3702416   0.30830448  0.19693136]
 [ 0.61055431 -1.26779698  0.70611671  0.4595065 ]
 [ 0.61055431 -0.59463044  0.81977735  0.4595065 ]
 [ 0.00704099  2.09803572 -1.39660508 -1.24723193]
 [-0.83787766  1.64925802 -1.16928381 -1.24723193]
 [-1.07928299 -1.26779698  0.47879543  0.72208164]
 [-0.71717499  0.75170264 -1.28294444 -1.24723193]
 [ 1.45547296  0.30292494  0.59245607  0.32821893]
 [ 0.12774365 -0.14585275  0.81977735  0.85336921]
 [-1.19998565 -0.14585275 -1.28294444 -1.3785195 ]
 [ 0.00704099 -1.04340814  0.19464384  0.06564379]
 [ 1.21406763  0.30292494  1.27441989  1.50980707]
 [-0.71717499 -0.81901929  0.13781352  0.32821893]
 [ 0.36914898 -0.59463044  0.59245607  0.06564379]
 [ 0.36914898 -1.04340814  1.10392894  0.32821893]
 [ 0.00704099 -0.59463044  0.81977735  1.64109464]

[-0.83787766  0.75170264 -1.22611412 -1.24723193]
[-1.44139098  0.0785361  -1.22611412 -1.24723193]
[-1.07928299 -1.49218583 -0.20316839 -0.19693136]
[-1.07928299 -0.14585275 -1.28294444 -1.24723193]
[ 0.73125697  0.30292494  0.93343798  1.50980707]
[ 0.48985164  0.75170264  0.9902683   1.50980707]
[ 1.33477029  0.30292494  1.16075926  1.50980707]
[-0.83787766  1.42486918 -1.22611412 -0.98465678]
[-1.68279631 -0.14585275 -1.33977476 -1.24723193]
 .                                              .
[ 0.85195964 -0.59463044  0.53562575  0.4595065 ]
[-1.44139098  1.20048033 -1.51026572 -1.24723193]
[-0.83787766  1.64925802 -1.22611412 -1.11594435]
[-0.71717499  2.32242456 -1.22611412 -1.3785195 ]
[ 0.73125697  0.0785361   1.04709862  0.85336921]
[-0.95858032 -0.14585275 -1.16928381 -1.24723193]
[-0.95858032  0.97609148 -1.33977476 -1.11594435]
```

```
[ 1.33477029  0.0785361   0.9902683   1.24723193]
[-1.44139098  0.75170264 -1.28294444 -1.11594435]
[ 0.61055431  0.52731379  1.33125021  1.77238221]
[-0.23436434 -1.26779698  0.13781352 -0.06564379]
[ 0.48985164 -0.59463044  0.64928639  0.85336921]
[ 0.85195964 -0.14585275  1.04709862  0.85336921]
[-0.355067   -1.49218583  0.02415289 -0.19693136]
[-0.23436434 -0.81901929  0.30830448  0.19693136]
[ 1.21406763 -0.14585275  1.04709862  1.24723193]
[ 0.61055431  0.75170264  1.10392894  1.64109464]
[ 1.69687828  1.20048033  1.38808053  1.77238221]
[ 2.3003916  -0.14585275  1.38808053  1.50980707]
[-1.07928299  0.0785361  -1.22611412 -1.24723193]
[ 0.61055431 -0.3702416   1.10392894  0.85336921]
[ 0.36914898 -0.59463044  0.19464384  0.19693136]
[-0.47576967  1.87364687 -1.11245349 -0.98465678]
[-0.47576967 -0.14585275  0.47879543  0.4595065 ]
[ 1.93828361 -0.59463044  1.38808053  0.98465678]
[ 1.09336496 -0.14585275  0.87660766  1.50980707]
[-1.80349897 -0.14585275 -1.4534354  -1.3785195 ]
[-0.11366168  2.9955911  -1.22611412 -0.98465678]
[-0.95858032 -1.71657468 -0.20316839 -0.19693136]
[-0.95858032 -2.38974122 -0.08950775 -0.19693136]
[-0.23436434 -0.14585275  0.47879543  0.4595065 ]
[ 0.36914898 -0.14585275  0.53562575  0.32821893]
[ 0.24844632 -0.14585275  0.64928639  0.85336921]
[-1.19998565  0.0785361  -1.16928381 -1.24723193]

[ 0.12774365 -0.14585275  0.30830448  0.4595065 ]
[ 0.73125697  0.30292494  0.47879543  0.4595065 ]
[-0.95858032  0.97609148 -1.16928381 -0.72208164]
[ 2.3003916   1.64925802  1.72906244  1.3785195 ]
[ 0.73125697 -0.59463044  1.10392894  1.24723193]
[ 1.33477029  0.0785361   0.81977735  1.50980707]
[-0.83787766  0.52731379 -1.11245349 -0.85336921]
[ 1.57617562 -0.14585275  1.27441989  1.24723193]
[-0.355067    0.97609148 -1.33977476 -1.24723193]
[-0.47576967  0.75170264 -1.22611412 -0.98465678]
[-1.19998565  0.75170264 -0.99879285 -1.24723193]
[-0.355067   -1.26779698  0.19464384  0.19693136]
[ 0.73125697 -0.59463044  1.10392894  1.3785195 ]
[ 0.00704099 -0.81901929  0.81977735  0.98465678]
[ 1.09336496  0.0785361   1.10392894  1.64109464]
[-0.11366168 -0.59463044  0.25147416  0.19693136]
[ 0.61055431  0.52731379  0.59245607  0.59079407]
[ 0.48985164 -1.94096352  0.47879543  0.4595065 ]
[ 0.85195964  0.30292494  0.81977735  1.11594435]
[-0.83787766  0.97609148 -1.28294444 -1.11594435]
[ 1.69687828 -0.14585275  1.21758958  0.59079407]
[ 1.09336496 -1.26779698  1.21758958  0.85336921]]
```

CODE:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_test)
print(' ')
print(y_pred)
```

OUTPUT:

```
['Virginica' 'Virginica' 'Setosa' 'Virginica' 'Versicolor' 'Virginica'
 'Virginica' 'Setosa' 'Versicolor' 'Versicolor' 'Setosa' 'Virginica'
 'Virginica' 'Virginica' 'Setosa' 'Virginica' 'Setosa' 'Setosa' 'Setosa'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Virginica' 'Setosa' 'Virginica' 'Virginica']

['Virginica' 'Virginica' 'Setosa' 'Virginica' 'Versicolor' 'Versicolor'
 'Virginica' 'Setosa' 'Versicolor' 'Versicolor' 'Setosa' 'Virginica'
 'Virginica' 'Virginica' 'Setosa' 'Virginica' 'Setosa' 'Setosa' 'Setosa'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Virginica' 'Setosa' 'Virginica' 'Virginica']
```

CODE:

```
from sklearn.metrics import classification_report, confusion_matr
ix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

OUTPUT:

```
[[ 8  0  0]
 [ 0  7  0]
 [ 0  1 14]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         8
  Versicolor       0.88      1.00      0.93         7
   Virginica       1.00      0.93      0.97        15

    accuracy                           0.97        30
   macro avg       0.96      0.98      0.97        30
weighted avg       0.97      0.97      0.97        30
```

CODE:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fruits=pd.read_table('/content/fruit_data_with_colors.txt')


fruits.head()
```

OUTPUT:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

CODE:

```
fruits.shape
predct = dict(zip(fruits.fruit_label.unique(), fruits.fruit_name.u
nique()))
predct
```

OUTPUT:

```
(59, 7)
```

```
{1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}
```

CODE:

```
apple_data=fruits[fruits['fruit_name']=='apple']
orange_data=fruits[fruits['fruit_name']=='orange']
lemon_data=fruits[fruits['fruit_name']=='lemon']
mandarin_data=fruits[fruits['fruit_name']=='mandarin']


apple_data.head()
```

OUTPUT:

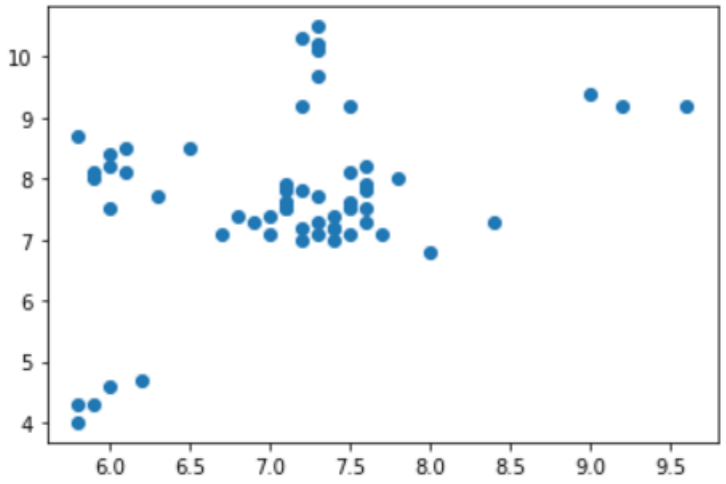
	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89

CODE:

```
plt.scatter(fruits['width'],fruits['height'])
```

OUTPUT:

```
<matplotlib.collections.PathCollection at 0x7f1a659c7690>
```

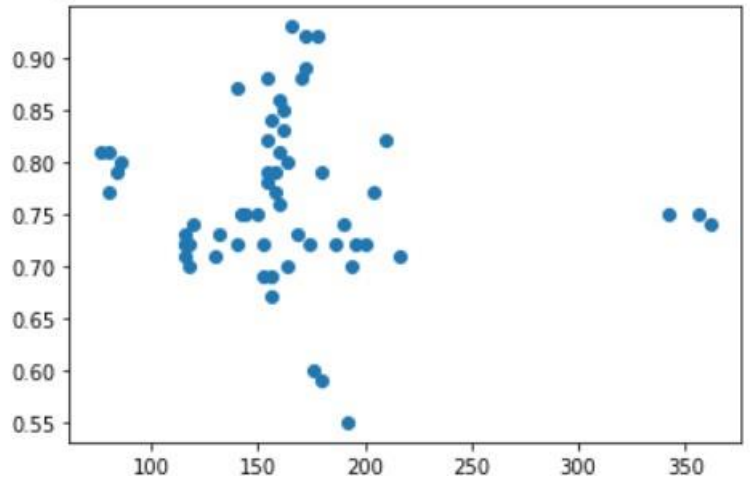


CODE:

```
plt.scatter(fruits['mass'],fruits['color_score'])
```

OUTPUT:

```
<matplotlib.collections.PathCollection at 0x7f1a65485a50>
```



CODE:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


X=fruits[['mass','width','height']]
Y=fruits['fruit_label']
X_train,X_test,y_train,y_test=train_test_split(X,Y,random_state=0
)

X_train.describe()
```

OUTPUT:

	mass	width	height
count	44.000000	44.000000	44.000000
mean	159.090909	7.038636	7.643182
std	53.316876	0.835886	1.370350
min	76.000000	5.800000	4.000000
25%	127.500000	6.175000	7.200000
50%	157.000000	7.200000	7.600000
75%	172.500000	7.500000	8.250000
max	356.000000	9.200000	10.500000

CODE:

```
X_test.describe()
```

OUTPUT:

	mass	width	height
count	15.000000	15.00000	15.000000
mean	174.933333	7.30000	7.840000
std	60.075508	0.75119	1.369463
min	84.000000	6.00000	4.600000
25%	146.000000	7.10000	7.250000
50%	166.000000	7.20000	7.600000
75%	185.000000	7.45000	8.150000
max	362.000000	9.60000	10.300000

CODE:

```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

OUTPUT:

```
KNeighborsClassifier()
```

CODE:

```
knn.score(X_test,y_test)
```

OUTPUT:

```
0.5333333333333333
```

CODE:

```
prediction1=knn.predict([['100','6.3','8']])
predct[prediction1[0]]
```

OUTPUT:

lemon

CODE:

```
prediction2=knn.predict([['300','7','10']])
predct[prediction2[0]]
```

OUTPUT:

orange

AIM

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

CODE:

Dataset used: `Social_Network_Ads.csv`

```
import pandas as pd
dataset = pd.read_csv("/content/Social_Network_Ads.csv")
print(dataset.describe())
print(dataset.head())
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze = 0.20, random_state = 0)
```

OUTPUT:

```
            User ID         Age  EstimatedSalary   Purchased
count  4.000000e+02  400.000000       400.000000  400.000000
mean   1.569154e+07   37.655000     69742.500000    0.357500
std    7.165832e+04   10.482877     34096.960282    0.479864
min    1.556669e+07   18.000000     15000.000000    0.000000
25%    1.562676e+07   29.750000     43000.000000    0.000000
50%    1.569434e+07   37.000000     70000.000000    0.000000
75%    1.575036e+07   46.000000     88000.000000    1.000000
max    1.581524e+07   60.000000    150000.000000    1.000000
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

CODE:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

OUTPUT:

```
GaussianNB()
```

CODE:

```
y_pred  =  classifier.predict(X_test)


y_pred
```

OUTPUT:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

CODE:

```
y_pred  =  classifier.predict(X_test)


y_test
```

OUTPUT:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1])
```

CODE:

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
print(cm)
print(ac)
```

OUTPUT:

```
[[56  2]
 [ 4 18]]
0.925
```

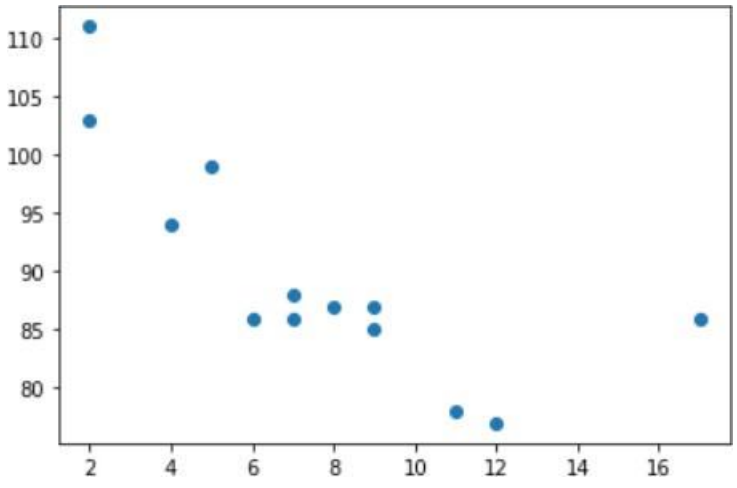
AIM

6. Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

CODE:

```
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:



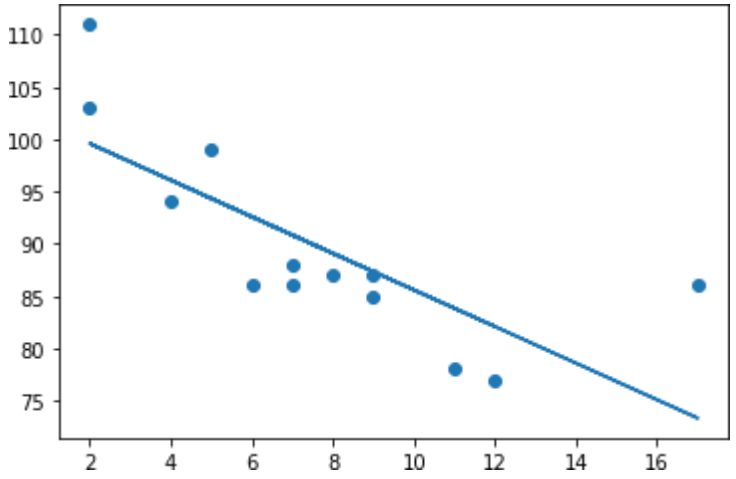
CODE:

```
import matplotlib.pyplot as plt
from scipy import stats

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]

slope, intercept, r, p, std_err = stats.linregress(x, y) # r corre
lation coefficiant   # p probability of hypothesis

def myfunc(x):
```

```
return slope * x + intercept

mymodel = list(map(myfunc, x))


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

OUTPUT:

```
-0.758591524376155
```



CODE:

```
import pandas
import warnings
warnings.filterwarnings("ignore")

df = pandas.read_csv("cars1.csv")


X = df[['Weight', 'Volume']]
y = df['CO2']
```

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

OUTPUT:

```
LinearRegression()
```

CODE:

```
predictedCO2 = regr.predict([[2300, 1000]])
print(predictedCO2)
```

OUTPUT:

```
[104.86715554]
```

AIM

7. Program to implement text classification using Support vector machine.

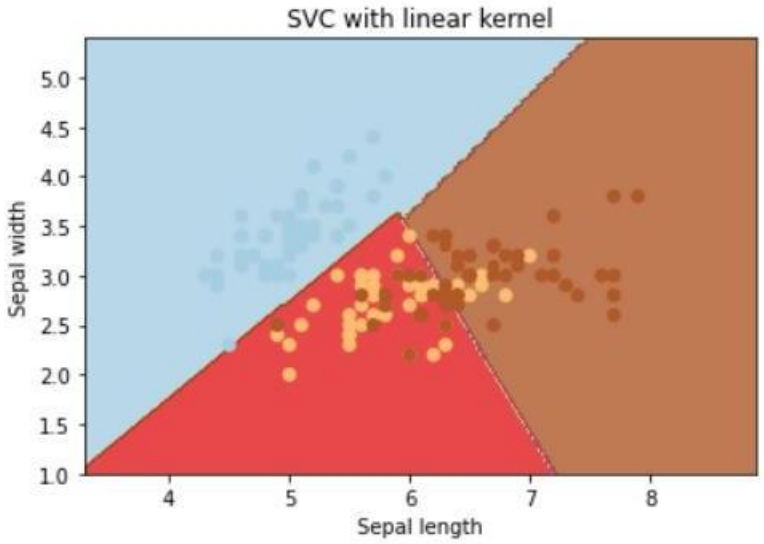
CODE:

Dataset used: iris.csv

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We co
uld
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target

# we create an instance of SVM and fit out data. We do not scale
our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=1,gamma='auto').fit(X, y)

# create a mesh to plot in
#x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
#y_min, y_))max = X[:, 1].min() - 1, X[:, 1].max() + 1
#h = (x_max / x_min)/100
#xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
#np.arange(y_min, y_max, h


plt.subplot(1, 1, 1)
Z = svc.predict(np.c_ravel[xx.(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
```

```
plt.title('SVC with linear kernel')
plt.show()
```

OUTPUT:



CODE:

Dataset used: True.csv, Fake.csv

```
 #Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics import accuracy_score, confusion_matrix,class
ification_report

from sklearn.svm import LinearSVC

import csv
true = pd.read_csv("True.csv")
fake = pd.read_csv("Fake.csv")
```

```
 fake['target'] = 'fake'
true['target'] = 'true'
#News dataset
news = pd.concat([fake, true]).reset_index(drop = True)
news.head()
news.dropna()
```

OUTPUT:

	title	text	subject	date	target
0	you were wrong! 70-year-old men don t change ...	News	"December 31	2017"	fake
165	look at me! I m violating the U.S. flag code ...	News	"October 29	2017"	fake
277	particularly those where people are dying. Ob...	News	"September 29	2017"	fake
294	utterly and completely misunderstanding it. T...	News	"September 25	2017"	fake
379	I salute you.Featured image via David Becker/...	News	"September 10	2017"	fake
...	...	...	...	...	...
39998	rescuers pulled Maria s body from the rubble....	worldnews	"September 21	2017 "	true
40742	adding she had a Spanish passport but chose t...	worldnews	"September 14	2017 "	true
40788	adding the Rohingya belong in camps for displ...	worldnews	"September 14	2017 "	true
40824	said Reick. "	worldnews	"September 14	2017 "	true
41394	in general. "	worldnews	"September 7	2017 "	true

236 rows × 5 columns

CODE:

```
#Train-test split
x_train,x_test,y_train,y_test = train_test_split(news['text'], new
s.target, test_size=0.2, random_state=1)

#Term frequency(TF)=count(word)/total(words)6+    0ZXCVBNM,./
#TF-
IDF: we can even reduce the weightage of more common words like (t
he, is, an etc.) which occurs in all document.
#This is called as TF-
IDF i.e Term Frequency times inverse document frequency.
#count vectorizer : involves counting the number of occurrences ea
ch word appears in a document
```

```
pipe2 = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTran
sformer()), ('model', LinearSVC())])


model_svc = pipe2.fit(x_train.astype('U'), y_train.astype('U'))
svc_pred = model_svc.predict(x_test.astype('U'))

print("Accuracy of SVM Classifier: {}%".format(round(accuracy_scor
e(y_test, svc_pred)*100,2)))
print("\nConfusion Matrix of SVM Classifier:\n")
print(confusion_matrix(y_test, svc_pred))
print("\nClassification Report of SVM Classifier:\n")
print(classification_report(y_test, svc_pred))
```

OUTPUT:

```
Accuracy of SVM Classifier: 51.43%

Confusion Matrix of SVM Classifier:

[[4302    3]
 [4085   26]]

Classification Report of SVM Classifier:

              precision    recall  f1-score   support

        fake       0.51      1.00      0.68      4305
        true       0.90      0.01      0.01      4111

    accuracy                           0.51      8416
   macro avg       0.70      0.50      0.35      8416
weighted avg       0.70      0.51      0.35      8416
```

AIM

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

CODE:

Dataset used: iris

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

data=load_iris()
X=data.data
y=data.target
print(X.shape,y.shape)
```

OUTPUT:

```
(150, 4) (150,)
```

CODE:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier#for checking testi
ng results
from sklearn.metrics import classification_report, confusion_matri
x#for visualizing tree
from sklearn.tree import plot_tree
X_train, X_test, y_train, y_test = train_test_split(X , y, test_si
ze = 25, random_state = 10)

clf=DecisionTreeClassifier()
clf.fit(X_train,y_train)
```

OUTPUT:

```
DecisionTreeClassifier()
```

CODE:

```
y_pred =clf.predict(X_test)
print("Classification report - \n", classification_report(y_test,y
_pred))
```

OUTPUT:

```
Classification report -
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      0.90      0.95        10
           2       0.86      1.00      0.92         6

    accuracy                           0.96        25
   macro avg       0.95      0.97      0.96        25
weighted avg       0.97      0.96      0.96        25
```

CODE:

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
from sklearn import tree
fig,axes = plt.subplots(nrows=1,ncols=1,figsize =(3,3),dpi=200)
tree.plot_tree(clf,feature_names=data.feature_names,class_names=da
ta.target_names,filled=True)
plt.show()
fig.savefig("/content/iris_tree.png")
```

OUTPUT:

```
[[9 0 0]
 [0 9 1]
 [0 0 6]]
```

AIM

9. Program to implement k-means clustering technique using any standard dataset available in the public domain.

CODE:

Dataset used: GENERAL.csv

```
# importing the libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as  plt

dataset= pd.read_csv('./CC GENERAL.csv')

# checking the presence of null values
print(dataset.isnull().sum())
 #CREDIT_LIMIT                           1
 #MINIMUM_PAYMENTS                     313
```

OUTPUT:

```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        1
PAYMENTS                            0
MINIMUM_PAYMENTS                  313
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

CODE:

```
dataset['CREDIT_LIMIT'].fillna(dataset.CREDIT_LIMIT.mean(), inplac
e = True)
dataset['MINIMUM_PAYMENTS'].fillna(dataset.MINIMUM_PAYMENTS.mean()
, inplace = True) # unfilled vaues replaced using mean

print(dataset.isnull().sum())

print(dataset.describe())
```

OUTPUT:

```
CUST_ID                           0
BALANCE                           0
BALANCE_FREQUENCY                 0
PURCHASES                         0
ONEOFF_PURCHASES                  0
INSTALLMENTS_PURCHASES            0
CASH_ADVANCE                      0
PURCHASES_FREQUENCY               0
ONEOFF_PURCHASES_FREQUENCY        0
PURCHASES_INSTALLMENTS_FREQUENCY  0
CASH_ADVANCE_FREQUENCY            0
CASH_ADVANCE_TRX                  0
PURCHASES_TRX                     0
CREDIT_LIMIT                      0
PAYMENTS                          0
MINIMUM_PAYMENTS                  0
PRC_FULL_PAYMENT                  0
TENURE                            0
dtype: int64
             BALANCE  BALANCE_FREQUENCY  ...  PRC_FULL_PAYMENT       TENURE
count    8950.000000       8950.000000  ...       8950.000000  8950.000000
mean     1564.474828          0.877271  ...          0.153715    11.517318
std      2081.531879          0.236904  ...          0.292499     1.338331
min         0.000000          0.000000  ...          0.000000     6.000000
25%       128.281915          0.888889  ...          0.000000    12.000000
50%       873.385231          1.000000  ...          0.000000    12.000000
75%      2054.140036          1.000000  ...          0.142857    12.000000
max     19043.138560          1.000000  ...          1.000000    12.000000
```

CODE:

```
dataset.drop(['CUST_ID'], axis= 1, inplace = True) #no relevance f
or custid

# No Categorical Values found
X = dataset.iloc[:,:].values
```

```
# Using standard scaler
from sklearn.preprocessing import StandardScaler
standardscaler= StandardScaler()
X = standardscaler.fit_transform(X)  #scaling the values
print(X)
```

OUTPUT:

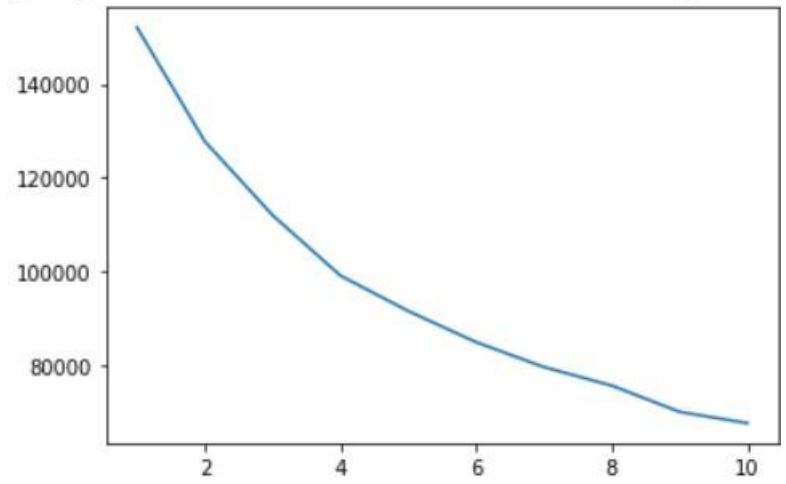
```
[[-0.73198937 -0.24943448 -0.42489974 ... -0.31096755 -0.52555097
   0.36067954]
 [ 0.78696085  0.13432467 -0.46955188 ...  0.08931021  0.2342269
   0.36067954]
 [ 0.44713513  0.51808382 -0.10766823 ... -0.10166318 -0.52555097
   0.36067954]
 ...
 [-0.7403981  -0.18547673 -0.40196519 ... -0.33546549  0.32919999
  -4.12276757]
 [-0.74517423 -0.18547673 -0.46955188 ... -0.34690648  0.32919999
  -4.12276757]
 [-0.57257511 -0.88903307  0.04214581 ... -0.33294642 -0.52555097
  -4.12276757]]
```

CODE:

```
"""K MEANS CLUSTERING """
#Inertia, or the within-
cluster sum of squares criterion, can be recognized as a measure o
f how internally coherent clusters are
from sklearn.cluster import KMeans
wss= []
for i in range(1, 11):
    kmeans= KMeans(n_clusters = i, init = 'k-
means++', random_state = 0)
    kmeans.fit(X)
    wss.append(kmeans.inertia_)
plt.plot(range(1,11), wss) # selecting 4
```

OUTPUT:

```
[<matplotlib.lines.Line2D at 0x7f74661e8a90>]
```



CODE:

```
wss_mean=np.array(wss).mean()
print(wss)
print(wss_mean)
print([abs(wss_mean-x) for x in wss])
k=np.argmin([abs(wss_mean-x) for x in wss])+1
```

OUTPUT:

```
[152149.99999999983, 127784.92103208725, 111986.41162208859,
99073.93826774803, 91502.98328256077, 84851.13240432573,
79532.40237691796, 75568.97609993909, 69954.91393943134,
67546.56302862825]
95995.22420537268
[56154.775794627145, 31789.69682671457, 15991.187416715911,
3078.714062375351, 4492.240922811907, 11144.091801046947,
16462.82182845472, 20426.248105433595, 26040.31026594134,
28448.661176744426]
```

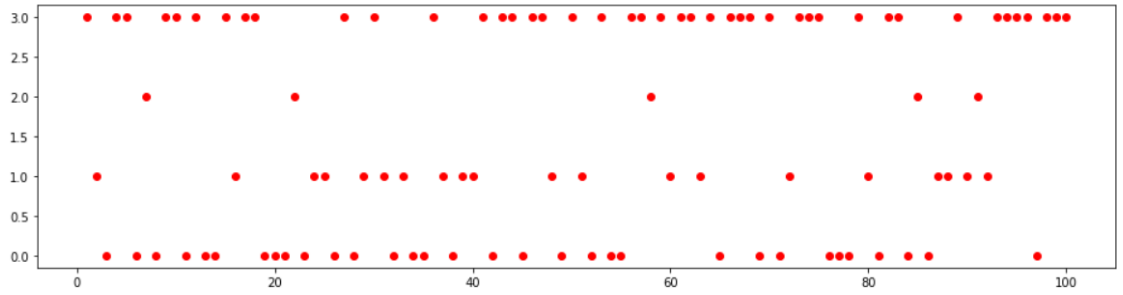
CODE:

```
kmeans = KMeans(n_clusters = k, init= 'k-
means++', random_state =  0)
kmeans.fit(X)

Y_pred_K= kmeans.predict(X)
print(Y_pred_K)
```

```
#showing the clusters of first 100 persons
plt.figure(figsize=(16,4))
plt.plot(range(1,100+1),Y_pred_K[:100],'ro')
```

OUTPUT:

```
[3 1 0 ... 0 3 3]
[<matplotlib.lines.Line2D at 0x7f745836e290>]
```



AIM

10. Programs on feedforward network to classify any standard dataset available in the public domain.

Dataset used: HR_comma_sep.csv

CODE:

```
import numpy as np
import pandas as pd

# Load data
data=pd.read_csv('HR_comma_sep.csv')
data.head()
```

OUTPUT:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

CODE:

```
from sklearn import preprocessing
# Creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
data['salary']=le.fit_transform(data['salary'])
data['sales']=le.fit_transform(data['sales'])
```

```
X=data[['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hour
s', 'time_spend_company', 'Work_accident', 'promotion_last_5years', 'sales', 'salary']]


y=data['left']
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
) # 70% training and 30% test

from sklearn.neural_network import MLPClassifier
# Create model object
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=False,
                    learning_rate_init=0.01)
# Fit data onto the model
clf.fit(X_train,y_train)
```

OUTPUT:

```
MLPClassifier(hidden_layer_sizes=(6, 5), learning_rate_init=0.01,
              random_state=5)
```

CODE:

```
ypred=clf.predict(X_test)
# Import accuracy score
from sklearn.metrics import accuracy_score
# Calcuate accuracy
accuracy_score(y_test,ypred)
```

OUTPUT:

```
0.9386666666666666
```

AIM

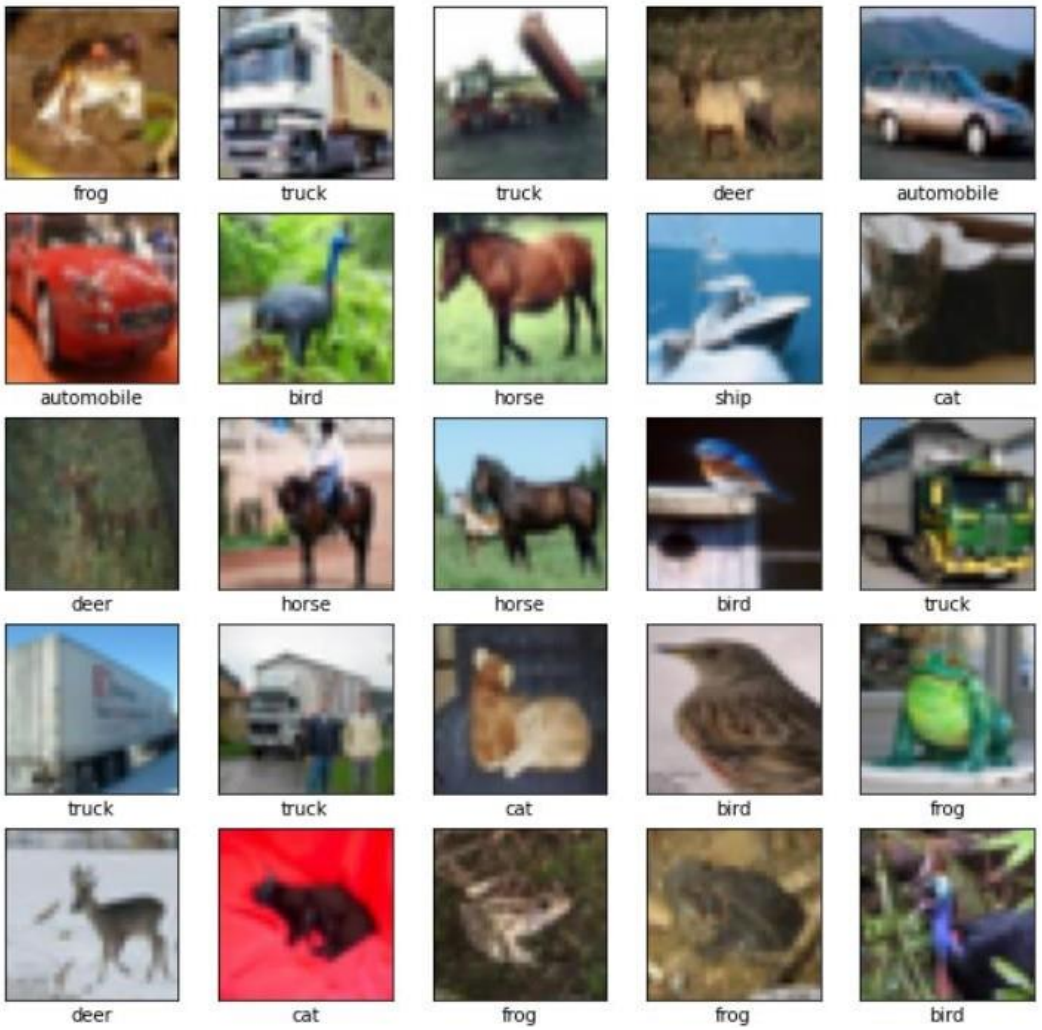
11. Programs on convolutional neural network to classify images from any standard dataset in the public domain.

Dataset used: cifar10

CODE:

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

#The CIFAR10 dataset contains 60,000 color images in 10 classes, w
ith 6,000 images in each class
(train_images, train_labels), (test_images, test_labels) = dataset
s.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 25
5.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

OUTPUT:



CODE:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape
=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

CODE:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

CODE:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(f
rom_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

OUTPUT:

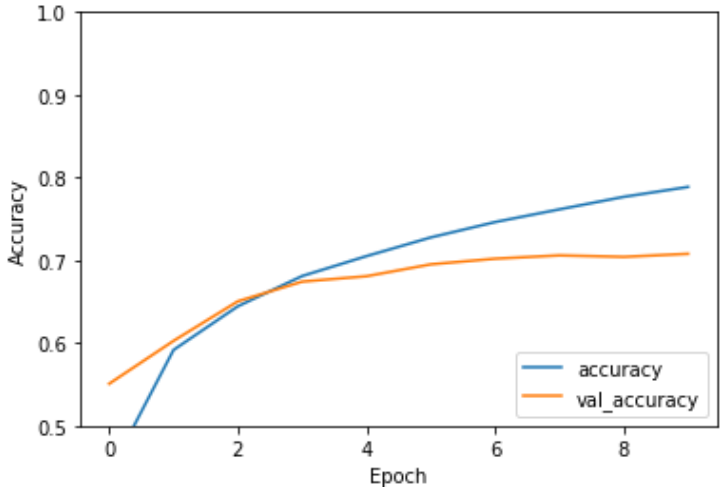
```
Epoch 1/10
1563/1563 [==============================] - 93s 59ms/step - loss: 1.5275 - accuracy: 0.4426 - val_loss: 1.2727 - val_accuracy: 0.5508
Epoch 2/10
1563/1563 [==============================] - 74s 47ms/step - loss: 1.1513 - accuracy: 0.5916 - val_loss: 1.1043 - val_accuracy: 0.6026
Epoch 3/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.0104 - accuracy: 0.6444 - val_loss: 1.0100 - val_accuracy: 0.6504
Epoch 4/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.9112 - accuracy: 0.6809 - val_loss: 0.9435 - val_accuracy: 0.6742
Epoch 5/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.8387 - accuracy: 0.7050 - val_loss: 0.9262 - val_accuracy: 0.6807
Epoch 6/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.7794 - accuracy: 0.7276 - val_loss: 0.8774 - val_accuracy: 0.6951
Epoch 7/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.7285 - accuracy: 0.7462 - val_loss: 0.8627 - val_accuracy: 0.7019
Epoch 8/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.6793 - accuracy: 0.7616 - val_loss: 0.8734 - val_accuracy: 0.7059
Epoch 9/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.6346 - accuracy: 0.7767 - val_loss: 0.8788 - val_accuracy: 0.7041
Epoch 10/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.6008 - accuracy: 0.7887 - val_loss: 0.8842 - val_accuracy: 0.7078
```

CODE:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, ve
rbose=2)
```

OUTPUT:

```
313/313 - 4s - loss: 0.8842 - accuracy: 0.7078 - 4s/epoch - 12ms/step
```



CODE:

```
print(test_acc)
```

OUTPUT:

```
0.7077999711036682
```