

**FEDERAL INSTITUTE OF
SCIENCE AND TECHNOLOGY
(FISAT)TM**

HORMIS NAGAR, MOOKKANNOOR

ANGAMALY-683577

‘FOCUS ON EXCELLENCE’

DATA SCIENCE

.....
LABORATORY RECORD

Name: JINCY JOSE

Branch: MASTER OF COMPUTER APPLICATION

Semester: 3

Batch: B

Roll No: 03

**FEDERAL INSTITUTE OF
SCIENCE AND TECHNOLOGY
(FISAT)TM**

HORMIS NAGAR, MOOKKANNOOR

ANGAMALY-683577



Name : JINCY JOSE

Branch : MASTER OF COMPUTER APPLICATION

Semester : 3

Roll No: 03

University Exam.Reg. No: FIT20MCA2061

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 **JINCY JOSE** 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

| SI No | Date : | Name of Experiment: | Page No: | Signature of Staff –In – Charge: |
|--------------|---------------|--|-----------------|---|
| 1 | 19/11/2021 | Matrix operations (using vectorization) and transformation using python and SVD using Python | 1 | |
| 2 | 30/11/2021 | Programs using matplotlib / plotly / bokeh / seaborn for data visualisation. | 4 | |
| 3 | 26/11/2021 | Programs to handle data using pandas | 15 | |
| 4 | 07/12/2021 | Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm. | 22 | |
| 5 | 14/12/2021 | Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm | 28 | |
| 6 | 04/01/2022 | Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance. | 31 | |
| 7 | 14/01/2022 | Program to implement text classification using Support vector machine. | 34 | |

| | | | | |
|----|------------|--|----|--|
| 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 | 38 | |
| 9 | 25/01/2022 | Program to implement k-means clustering technique using any standard dataset available in the public domain | 40 | |
| 10 | 28/01/2022 | Programs on feedforward network to classify any standard dataset available in the public domain | 45 | |
| 11 | 28/01/2022 | Programs on convolutional neural network to classify images from any standard dataset in the public domain. | 47 | |
| | | | | |
| | | | | |

AIM

1: Matrix operations(using vectorixation) and transformation using python and SVD

CODE:

```
a = np.arange(0,4).reshape((2,2))
b = np.eye(2)
print(np.dot(a,b)) ##Matrix multiplication
```

OUTPUT:

```
[[0. 1.]
 [2. 3.]]
```

CODE:

```
x = np.arange(1,10).reshape(3,3)
print(x)
```

OUTPUT:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

CODE:

#SVD image compresion

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

img_eg = mpimg.imread("rose.jpg")
plt.imshow(img_eg)
print(img_eg.shape) #Operation results: (800, 1280,3)

#Converting image data into two-dimensional matrix and singular value decomposition
img_temp = img_eg.reshape(800, 1280 * 3)
U,Sigma,VT = np.linalg.svd(img_temp)

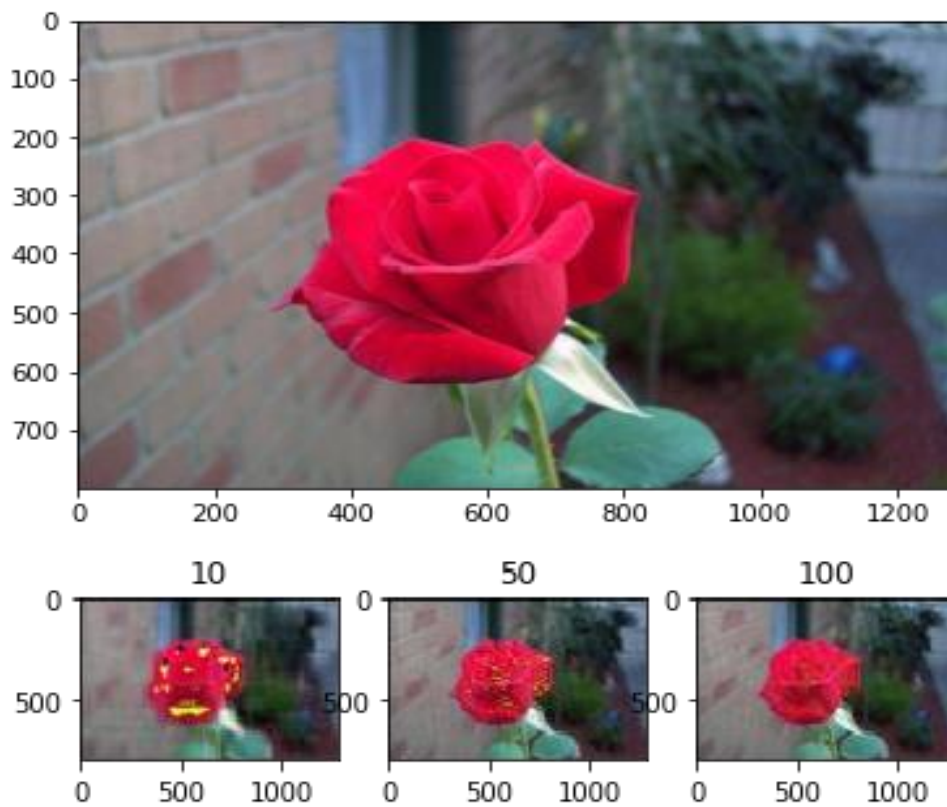
# Take the first 10 singular values
sval_nums = 10
```

```
img_re-
struct1 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:
sval_nums,:])
img_restruct1 = img_restruct1.reshape(800, 1280,3)
img_restruct1.tolist()

# Take the first 50 singular values
sval_nums = 50
img_re-
struct2 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:
sval_nums,:])
img_restruct2 = img_restruct2.reshape(800, 1280,3)

# Take the first 100 singular values
sval_nums = 100
img_re-
struct3 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:
sval_nums,:])
img_restruct3 = img_restruct3.reshape(800, 1280,3)

#Exhibition
fig, ax = plt.subplots(nrows=1, ncols=3)
ax[0].imshow(img_restruct1.astype(np.uint8))
ax[0].set(title = "10")
ax[1].imshow(img_restruct2.astype(np.uint8))
ax[1].set(title = "50")
ax[2].imshow(img_restruct3.astype(np.uint8))
ax[2].set(title = "100")
plt.show()
```

OUTPUT:

AIM

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

Dataset used: iris.csv

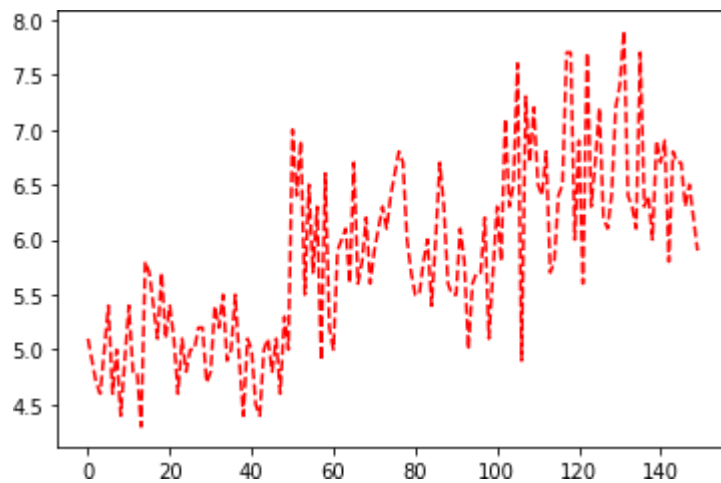
CODE:

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

```
## Plotting Using Matplotlib
```

```
import matplotlib.pyplot as plt
plt.plot(iris["sepal.length"], "r--")
plt.show
```

OUTPUT:

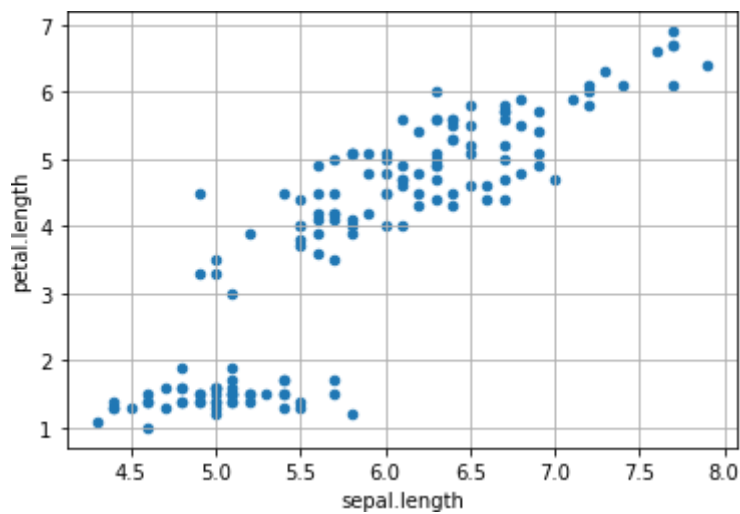


CODE:

```
## Scatter Plot
```

```
iris.plot(kind="scatter",
          x='sepal.length',
          y='petal.length')
plt.grid()
```

OUTPUT:

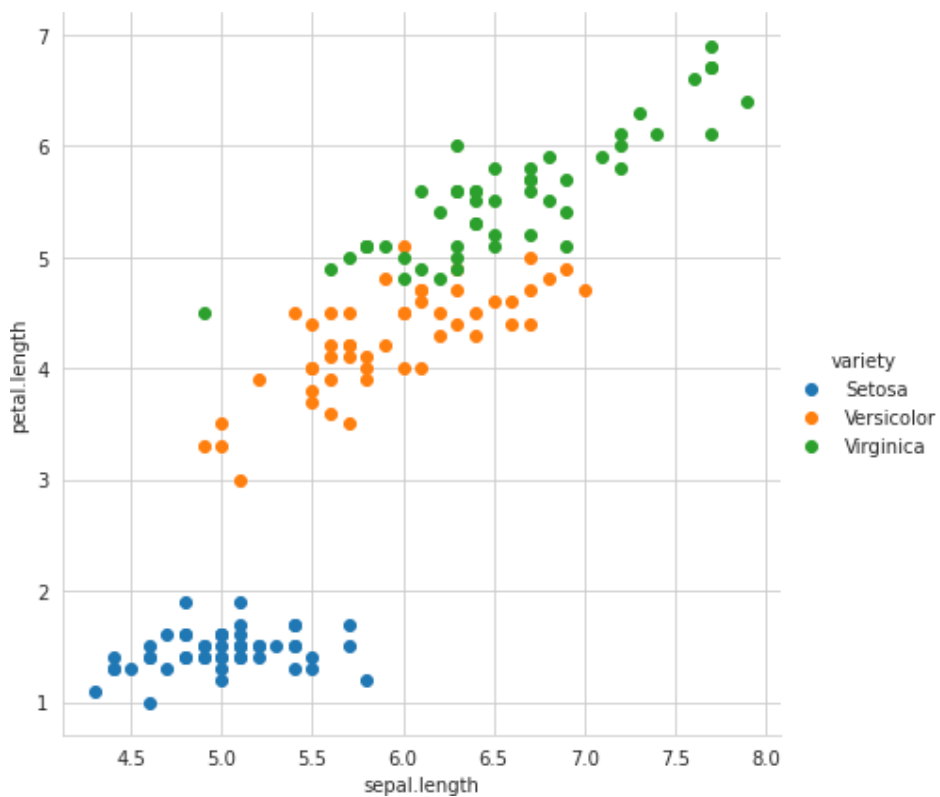


CODE:

```
## Plotting using Seaborn
```

```
import seaborn as sns
sns.set_style("whitegrid")
sns.FacetGrid(iris, hue = "variety", height = 6).map(plt.scatter, 'sepal.length',
'petal.length').add_legend()
```

OUTPUT:

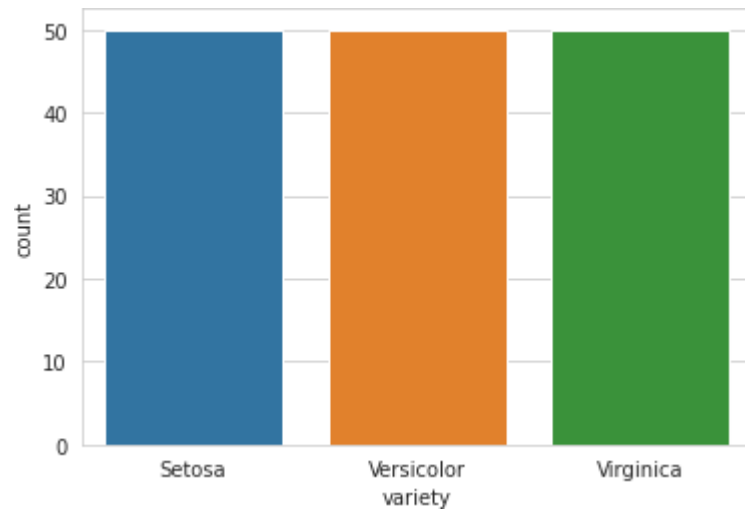


CODE:

```
# Distribution Chart  
#Visualizing the target(class label) column
```

```
sns.countplot(x='variety', data=iris, )  
plt.show()
```

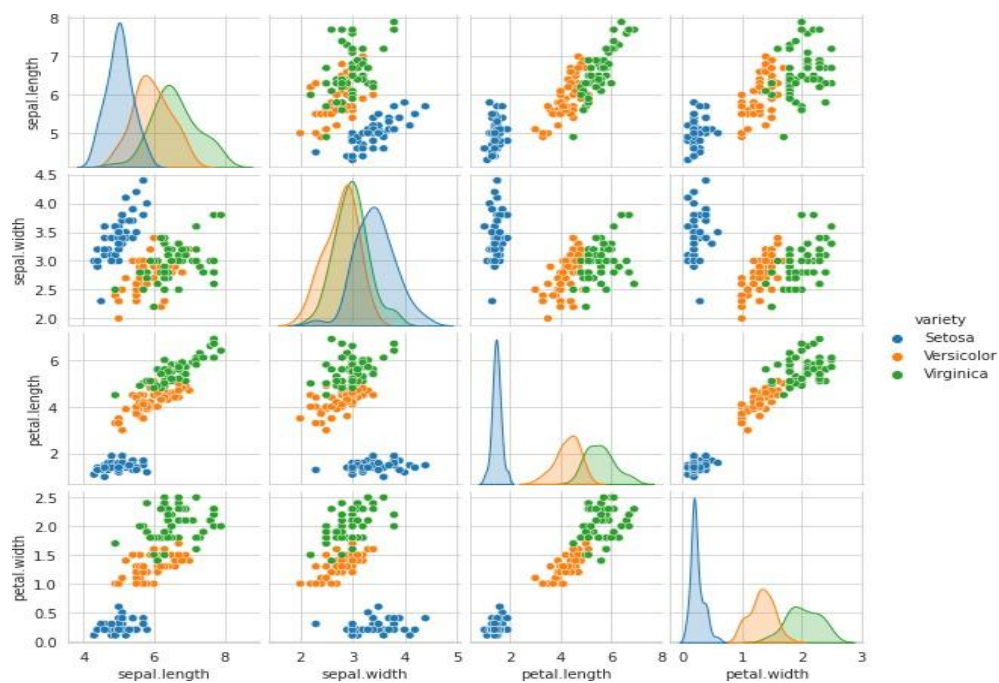
OUTPUT:



CODE:

```
#plotting all the column's relationships using a pairplot. It can be used for multivariate analysis.  
sns.pairplot(iris,hue='variety', height=2)
```

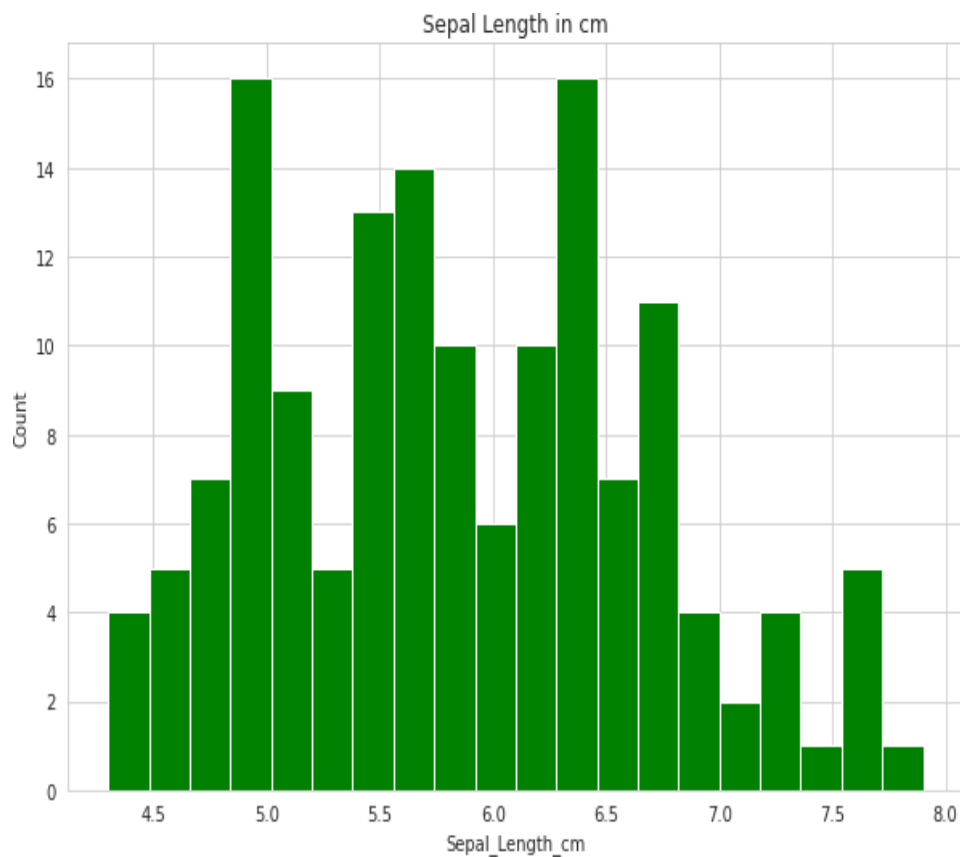
OUTPUT:



#Histogram for Sepal Length

```
plt.figure(figsize = (10, 7))  
x = iris["sepal.length"]  
plt.hist(x, bins = 20, color = "green")  
plt.title("Sepal Length in cm")  
plt.xlabel("Sepal_Length_cm")  
plt.ylabel("Count")
```

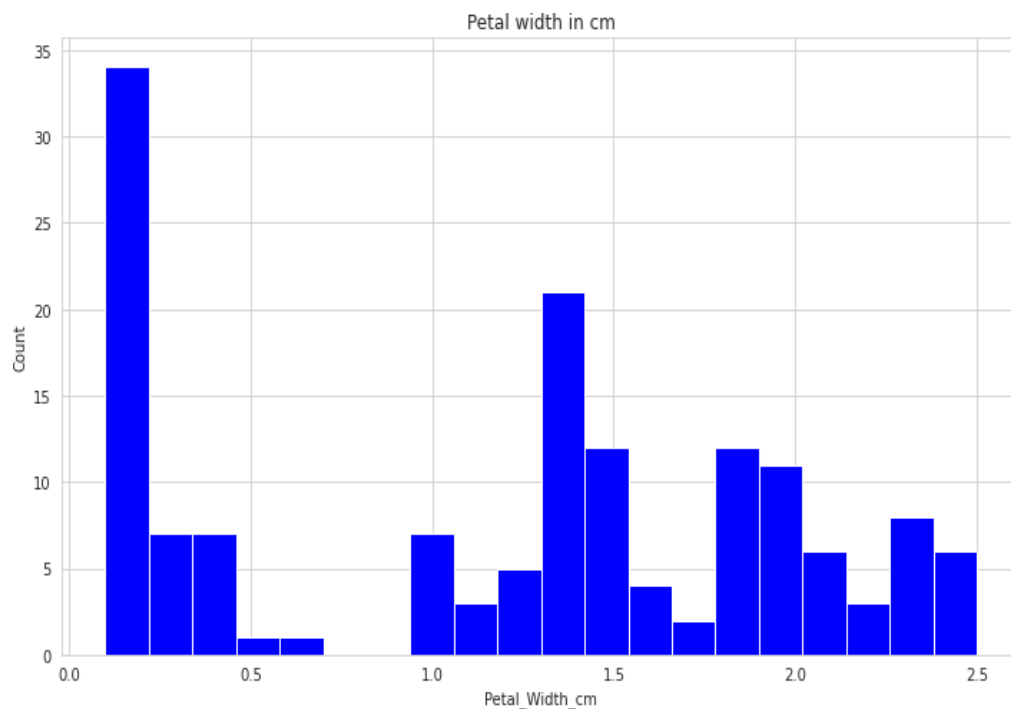
OUTPUT:



CODE:

```
#Histogram for Petal Width  
plt.figure(figsize = (12, 7))  
x = iris["petal.width"]  
  
plt.hist(x, bins = 20, color = "blue")  
plt.title("Petal width in cm")  
plt.xlabel("Petal_Width_cm")  
plt.ylabel("Count")
```

OUTPUT:



CODE:

#Histograms allow seeing the distribution of data for various columns.
It can be used for uni as well as bi-variate analysis.

```
fig, axes = plt.subplots(2, 2, figsize=(10,10))
```

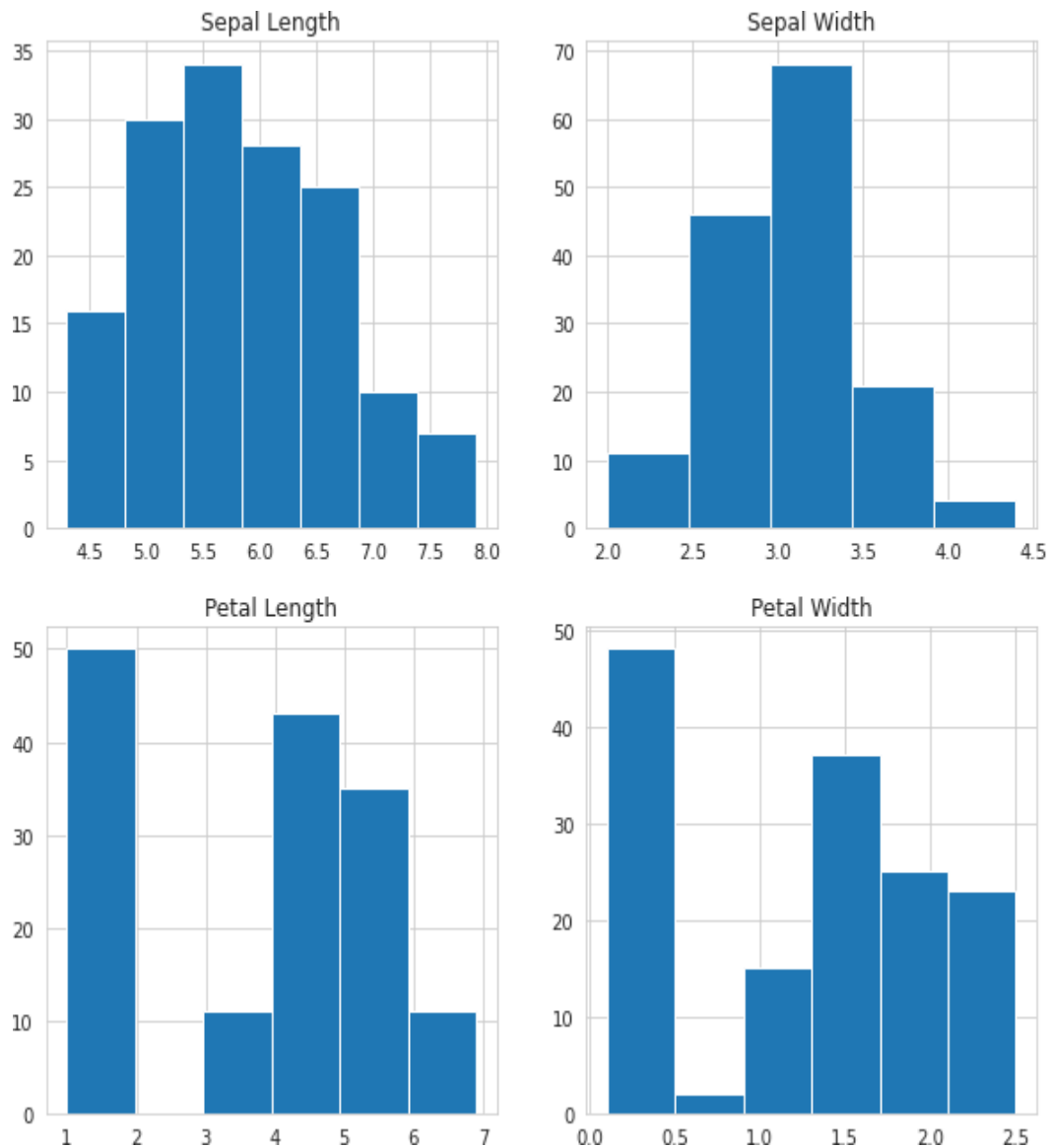
```
axes[0,0].set_title("Sepal Length")  
axes[0,0].hist(iris['sepal.length'], bins=7)
```

```
axes[0,1].set_title("Sepal Width")  
axes[0,1].hist(iris['sepal.width'], bins=5);
```

```
axes[1,0].set_title("Petal Length")  
axes[1,0].hist(iris['petal.length'], bins=6);
```

```
axes[1,1].set_title("Petal Width")  
axes[1,1].hist(iris['petal.width'], bins=6);
```

OUTPUT:



CODE:

```
#Histograms with Distplot Plot
```

```
plot = sns.FacetGrid(iris, hue="variety")  
plot.map(sns.distplot, "sepal.length").add_legend()
```

```
plot = sns.FacetGrid(iris, hue="variety")  
plot.map(sns.distplot, "sepal.width").add_legend()
```

```
plot = sns.FacetGrid(iris, hue="variety")  
plot.map(sns.distplot, "petal.length").add_legend()
```

```
plot = sns.FacetGrid(iris, hue="variety")
```

```
plot.map(sns.distplot, "petal.width").add_legend()
```

```
plt.show()
```

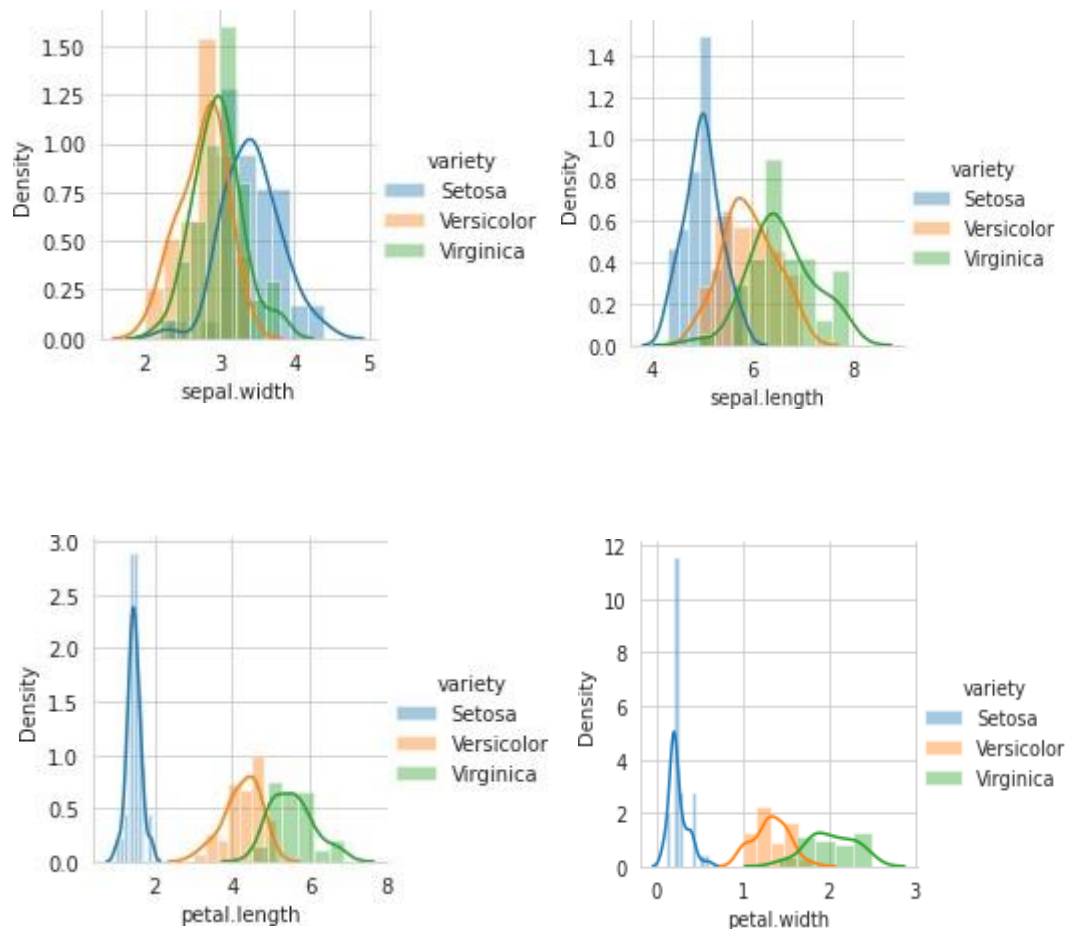
#In the case of Sepal Length, there is a huge amount of overlapping.

#In the case of Sepal Width also, there is a huge amount of overlapping.

#In the case of Petal Length, there is a very little amount of overlapping.

#In the case of Petal Width also, there is a very little amount of overlapping.

OUTPUT:

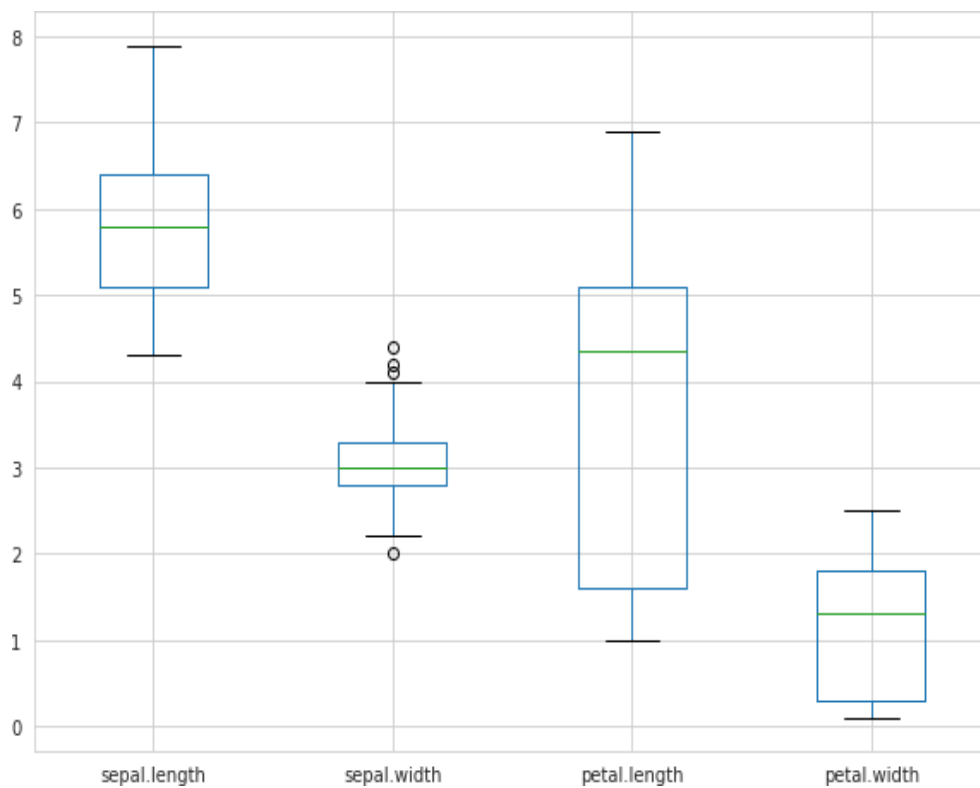


CODE:

```
# Box Plot for Iris Data
```

```
plt.figure(figsize = (10, 7))  
iris.boxplot()
```

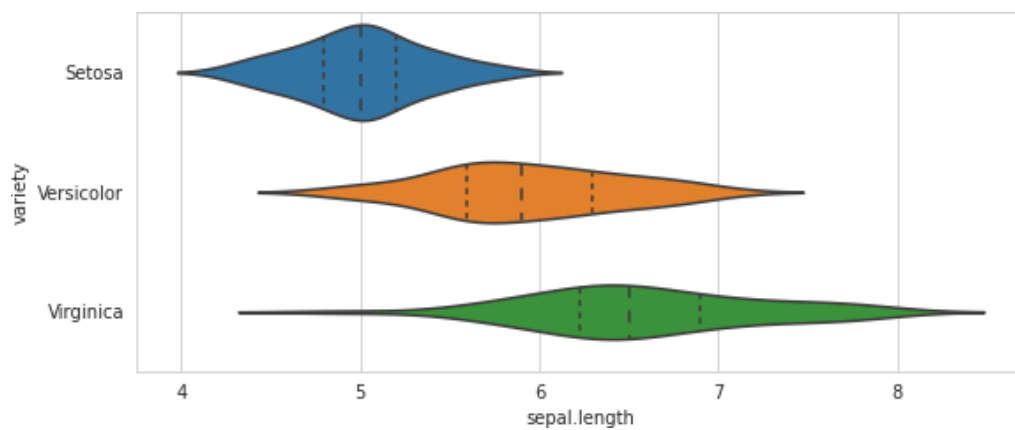
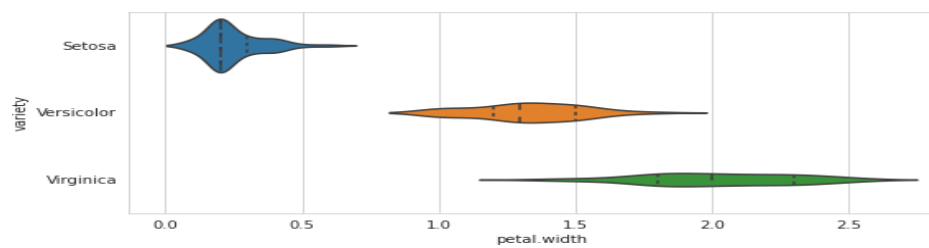
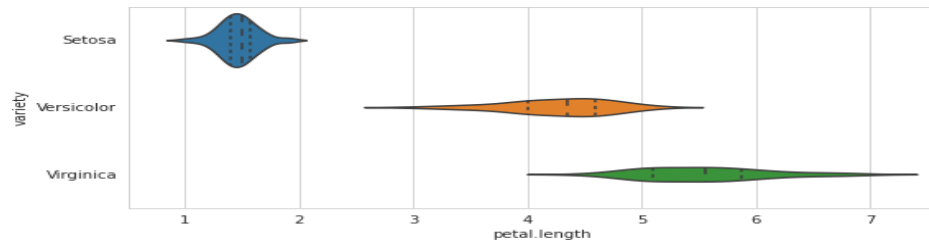
OUTPUT:

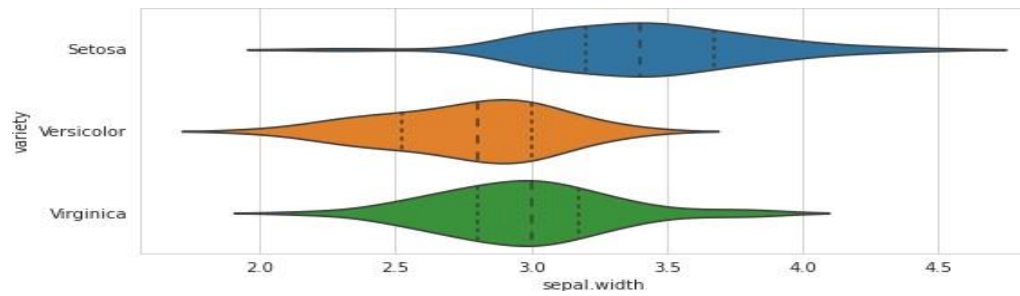


CODE:

```
import matplotlib.gridspec as gridspec  
fig = plt.figure(figsize=(9, 40))  
outer = gridspec.GridSpec(4, 1, wspace=0.2, hspace=0.2)  
for i, col in enumerate(iris.columns[:-1]):  
    inner = gridspec.GridSpecFromSubplotSpec(2, 1, subplot_spec=outer[i], wspace=0.2,  
hspace=0.4)  
    ax = plt.Subplot(fig, inner[1])  
    _ = sns.violinplot(y="variety", x=f"{col}", data=iris, inner='quartile', ax=ax)  
    fig.add_subplot(ax)  
fig.show()
```


OUTPUT:

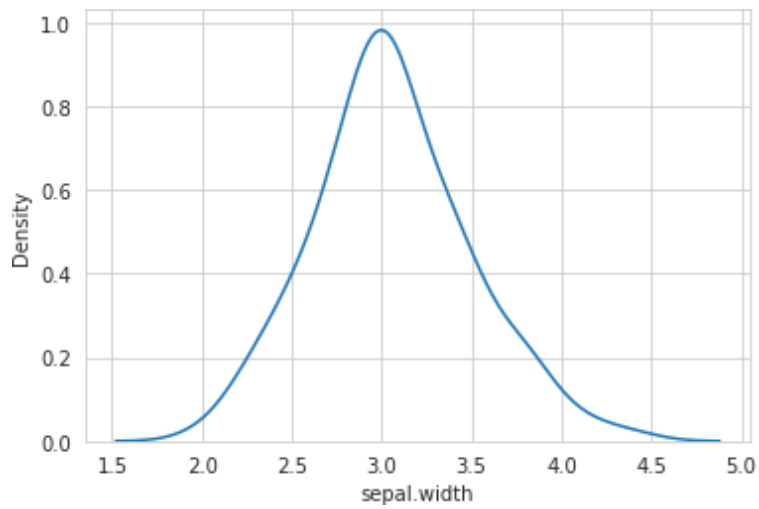




CODE:

```
# Make default density plot  
sns.kdeplot(iris['sepal.width'])
```

OUTPUT:



AIM**3. Programs to handle data using pandas****CODE:**

```
#Pandas is a Python library.
```

```
#Pandas is used to analyze data.
```

```
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 |

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 |

CODE:

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

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

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

import pandas as pd
cars = pd.read_csv('cars1.csv', index_col = 0) #first column is taen 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]
CodeText
```

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 ("Our object is:")
print (df)
print ("The dimension of the object is:")
print (df.ndim)
```

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)
```

OUTPUT:

```
21
```

CODE:

```
print(df.values)
```

OUTPUT:

```
[['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() #sum returns the number of missing values
```

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:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd
```

```
df = pd.read_csv("iris.csv")
print(df)
```

OUTPUT:

```
      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:

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

OUTPUT:

```
Setosa      50
Versicolor  50
Virginica   50
Name: variety, dtype: int64
```

CODE:

```
X = df.drop('variety', axis=1)
```

```
y = df['variety']
```

```
# splitting to trainset and Test set in the ratio 70:30
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

```
print(X_train)
```

```
print(" ")
```

```
print(X_test)
```

OUTPUT:

```

    sepal.length  sepal.width  petal.length  petal.width
46             5.1           3.8           1.6           0.2
95             5.7           3.0           4.2           1.2
67             5.8           2.7           4.1           1.0
45             4.8           3.0           1.4           0.3
143            6.8           3.2           5.9           2.3
..            ...           ...           ...           ...
116            6.5           3.0           5.5           1.8
41             4.5           2.3           1.3           0.3
62             6.0           2.2           4.0           1.0
91             6.1           3.0           4.6           1.4
123            6.3           2.7           4.9           1.8

```

```
[105 rows x 4 columns]
```

```

    sepal.length  sepal.width  petal.length  petal.width
25             5.0           3.0           1.6           0.2
141            6.9           3.1           5.1           2.3
125            7.2           3.2           6.0           1.8
102            7.1           3.0           5.9           2.1
128            6.4           2.8           5.6           2.1
122            7.7           2.8           6.7           2.0
76             6.8           2.8           4.8           1.4
103            6.3           2.9           5.6           1.8
14             5.8           4.0           1.2           0.2
37             4.9           3.6           1.4           0.1
100            6.3           3.3           6.0           2.5
63             6.1           2.9           4.7           1.4
64             5.6           2.9           3.6           1.3
61             5.9           3.0           4.2           1.5
17             5.1           3.5           1.4           0.3
74             6.4           2.9           4.3           1.3
111            6.4           2.7           5.3           1.9
120            6.9           3.2           5.7           2.3
79             5.7           2.6           3.5           1.0
85             6.0           3.4           4.5           1.6
49             5.0           3.3           1.4           0.2
21             5.1           3.7           1.5           0.4
110            6.5           3.2           5.1           2.0
149            5.9           3.0           5.1           1.8
72             6.3           2.5           4.9           1.5
11             4.8           3.4           1.6           0.2
36             5.5           3.5           1.3           0.2
6              4.6           3.4           1.4           0.3
68             6.2           2.2           4.5           1.5
144            6.7           3.3           5.7           2.5
43             5.0           3.5           1.6           0.6
80             5.5           2.4           3.8           1.1
32             5.2           4.1           1.5           0.1
7              5.0           3.4           1.5           0.2
55             5.7           2.8           4.5           1.3

129            7.2           3.0           5.8           1.6
117            7.7           3.8           6.7           2.2

```

12

4.8

3.0

1.4

0.1

CODE:

```
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

OUTPUT:

```
Number transactions X_train dataset: (105, 4)
Number transactions y_train dataset: (105,)
Number transactions X_test dataset: (45, 4)
Number transactions y_test dataset: (45,)
```

CODE:

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

OUTPUT:

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

```
63      Versicolor
64      Versicolor
```

```
61      Versicolor
17      Setosa
74      Versicolor
111     Virginica
120     Virginica
79      Versicolor
85      Versicolor
49      Setosa
21      Setosa
110     Virginica
149     Virginica
72      Versicolor
11      Setosa
36      Setosa
6       Setosa
```

```
68      Versicolor
```

```

144     Virginica
43       Setosa
47       Setosa
77     Versicolor
80     Versicolor
32       Setosa
7       Setosa
148     Virginica
88     Versicolor
137     Virginica
55     Versicolor
112     Virginica
29       Setosa
129     Virginica
117     Virginica
12       Setosa
Name: variety, dtype: object

```

CODE:

```

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

```

OUTPUT:

```

[[15  0  0]
 [ 0 11  2]
 [ 0  0 17]]

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Setosa | 1.00 | 1.00 | 1.00 | 15 |
| Versicolor | 1.00 | 0.85 | 0.92 | 13 |
| Virginica | 0.89 | 1.00 | 0.94 | 17 |
| accuracy | | | 0.96 | 45 |
| macro avg | 0.96 | 0.95 | 0.95 | 45 |
| weighted avg | 0.96 | 0.96 | 0.95 | 45 |

CODE:

```

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

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

# Label or target variable

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_en-
coded)
print(" ") la-
bel=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]
```


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_size = 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
```

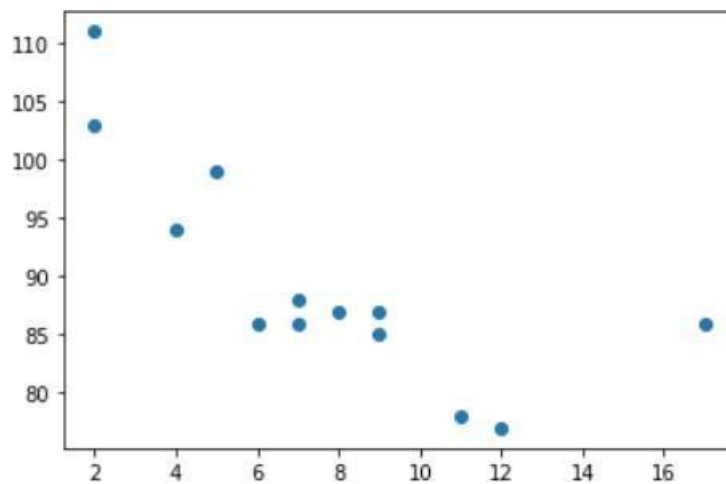
AIM

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
correlation coefficient # p probability of hypothesis

def myfunc(x):
```

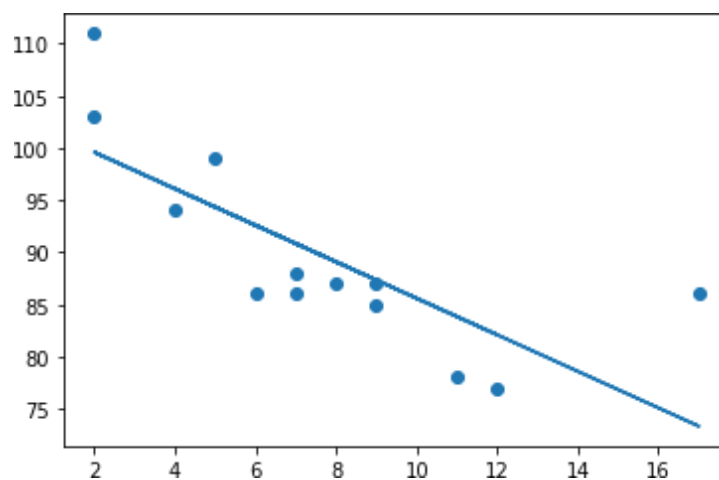
```
return slope * x + intercept

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 could
# 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)

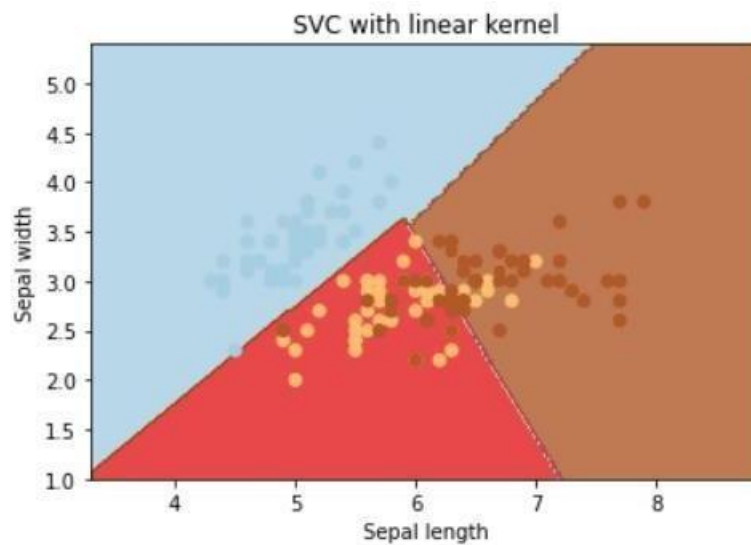
# create a mesh to plot in
#x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
#y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
#h = (x_max - x_min)/100
#xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
#np.arange(y_min, y_max, h))

plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), 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, classification_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', TfidfTransformer()), ('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_score(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 testing results
from sklearn.metrics import classification_report, confusion_matrix#for visualizing tree
from sklearn.tree import plot_tree
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 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("/con-
tent/iris_tree.png")

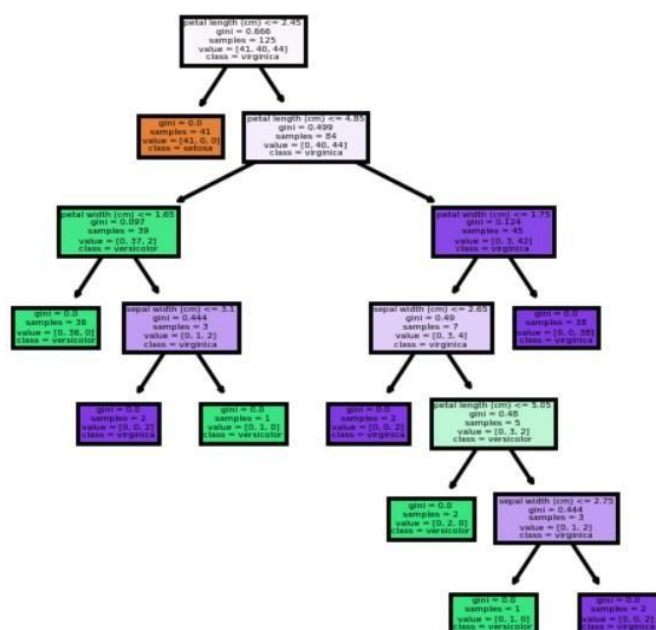
```

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 da-

taset= 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(), inplace = 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:

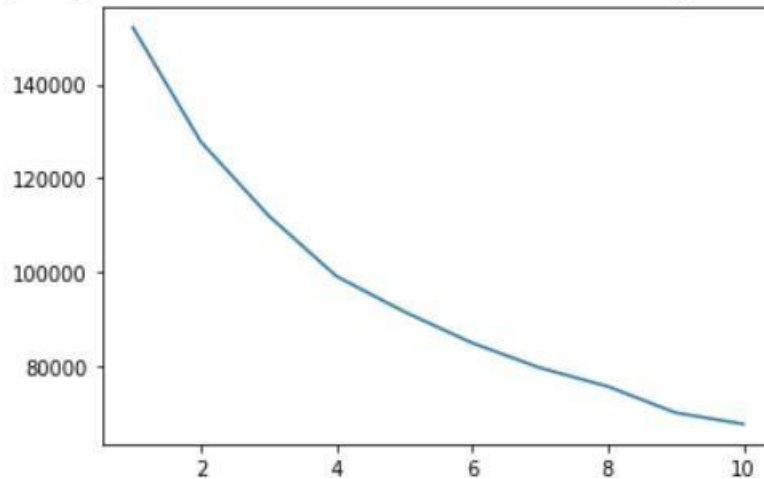
```
[[ -0.73198937 -0.24943448 -0.42489974 ... -0.31096755 -0.52555097
   0.36067954]
 [  0.78696085  0.13432467 -0.46955188 ...  0.08931021  0.2342269
   0.36067954]
 [  0.44713513  0.51808382 -0.10766823 ... -0.10166318 -0.52555097
   0.36067954]
 ...
 [-0.7403981  -0.18547673 -0.40196519 ... -0.33546549  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.ap-
    pend(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:

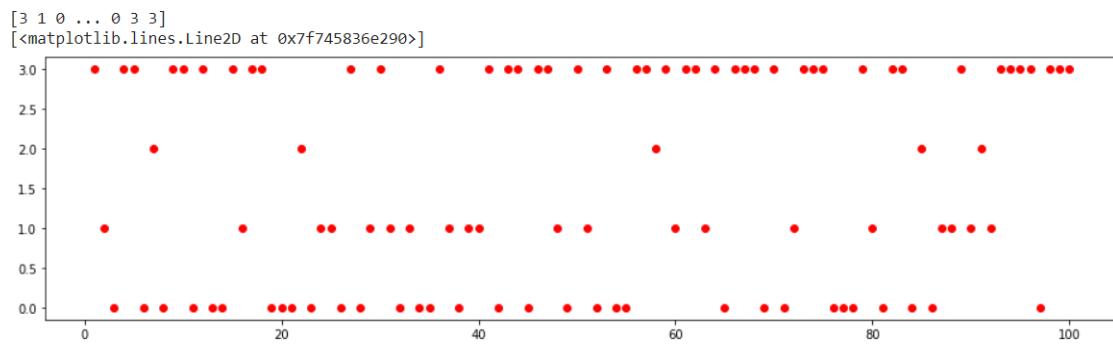
```
kmeans = KMeans(n_clusters = k, init= 'k-
means++', random_state = 0)
kmeans.fit(X)

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:



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_monthly_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_monthly_hours', '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
```

```
# Calculate 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.

CODE:

```
import numpy as np
import pandas as pd

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

data.head()
```

Output:

| | satis- fac- tion_l evel | last_e valu- ation | num- ber_p ro- ject | aver- age_monthly _hours | time_spen d_com- pany | Work _acci- dent | le ft | promo- tion_last_ 5years | sal es | sal ary |
|---|----------------------------------|--------------------------|------------------------------|--------------------------------|-----------------------------|------------------------|----------|--------------------------------|-----------|----------------|
| 0 | 0.38 | 0.53 | 2 | 157 | 3 | 0 | 1 | 0 | sal es | lo w |
| 1 | 0.80 | 0.86 | 5 | 262 | 6 | 0 | 1 | 0 | sal es | me diu m |
| 2 | 0.11 | 0.88 | 7 | 272 | 4 | 0 | 1 | 0 | sal es | me diu m |
| 3 | 0.72 | 0.87 | 5 | 223 | 5 | 0 | 1 | 0 | sal es | lo w |
| 4 | 0.37 | 0.52 | 2 | 159 | 3 | 0 | 1 | 0 | sal es | lo w |

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_monthly_hours',
```

```
'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)

ypred=clf.predict(X_test)
```

OUTPUT:

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

CODE:

```
# Import accuracy score
from sklearn.metrics import accuracy_score
# Calculate accuracy
print ("Accuracy:",accuracy_score(y_test,ypred))
```

OUTPUT:

```
Accuracy: 0.9386666666666666
```

CODE:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, ypred))
print(classification_report(y_test, ypred))
```

OUTPUT:

```
[[3248 180]
 [ 96 976]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.97 | 0.95 | 0.96 | 3428 |
| 1 | 0.84 | 0.91 | 0.88 | 1072 |
| accuracy | | | 0.94 | 4500 |
| macro avg | 0.91 | 0.93 | 0.92 | 4500 |
| weighted avg | 0.94 | 0.94 | 0.94 | 4500 |

