UNIVERSITY SCHOOL OF AUTOMATION & ROBOTICS (USAR)

Guru Gobind Singh Indraprastha University, East Delhi Campus, SurajmalVihar, Delhi 110092

Data Analytics Lab

(ARI351)

Submitted To:

Dr. Sanjay Kumar Singh

Asst. Professor, USAR

Submitted By:

Name: Vipul Goyal

Batch: IIOT-B1

Roll No: 02419011721

INDEX

S.No	Topic	Remark's
1.	Program to read data (csv/inbuilt sklearn) from local drive/google drive. Understand shape, size, features, classes, number of samples / classes, countplot, pairplot, heatmap, etc. from dataframe.	
2.	Program to implement handle missing values and categorical data.	
3.	Program to implement feature scaling, feature extraction and selection.	
4.	Program to implement simple and multiple linear regression	
5.	Program to implement regularized and nonlinear regression.	
6.	Program to implement classification algorithms Perceptron, logistic regression and SVM.	
7.	Program to implement classification algorithm Decision Tree, Naive Bayes and KNN.	
8.	Program to implement ensemble learning classification algorithms.	
9.	Program to implement K-Means clustering techniques.	
10.	Program to implement Hierarchical and density-based clustering techniques.	

LAB-1

AIM: Program to read data (csv/inbuilt sklearn) from local drive/google drive. Understand shape, size, features, classes, number of samples/ classes, countplot, pairplot, heatmap, etc. from dataframe.

CODE:

```
import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/iris.csv',header=None)
df.columns=['SL','SW','PL','PW','Target']
df
```

OUTPUT:

	SL	SW	PL	PW	Target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

CODE:

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

OUTPUT:

```
0    50
1    50
2    50
Name: Target, dtype: int64
```

CODE:

```
[5] df.shape
```

```
(150, 5)
```

CODE:

```
df.describe()
```

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

CODE:

```
[7] df.isna().sum()
```

OUTPUT:

```
SL        0
SW        0
PL        0
PW        0
Target    0
dtype: int64
```

CODE:

```
[8] df.info()
```

OUTPUT:

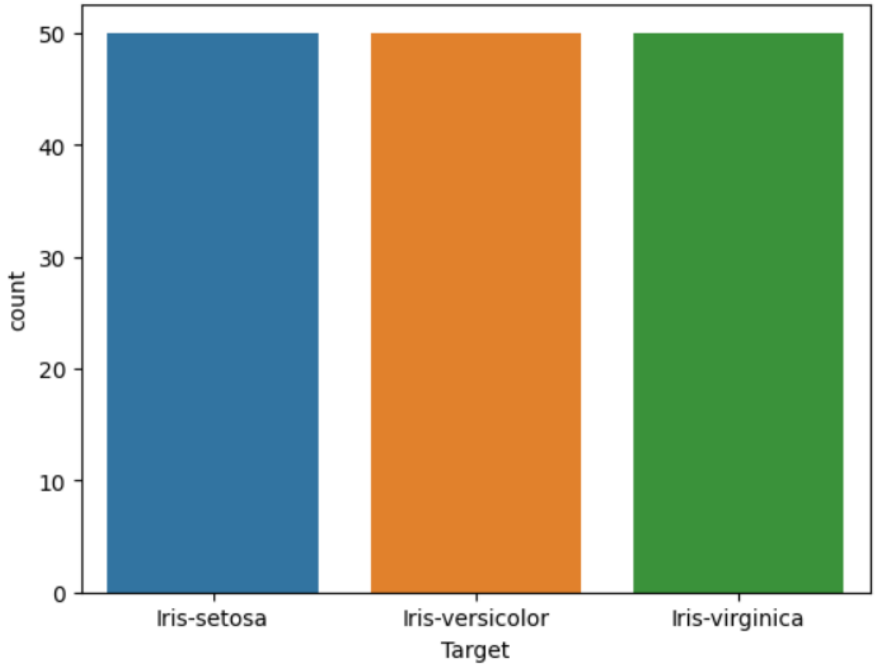
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   SL      150 non-null    float64
 1   SW      150 non-null    float64
 2   PL      150 non-null    float64
 3   PW    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

CODE:

```
[9] import seaborn as sns
    sns.countplot(x=df['Target'])
```

OUTPUT:

<Axes: xlabel='Target', ylabel='count'>

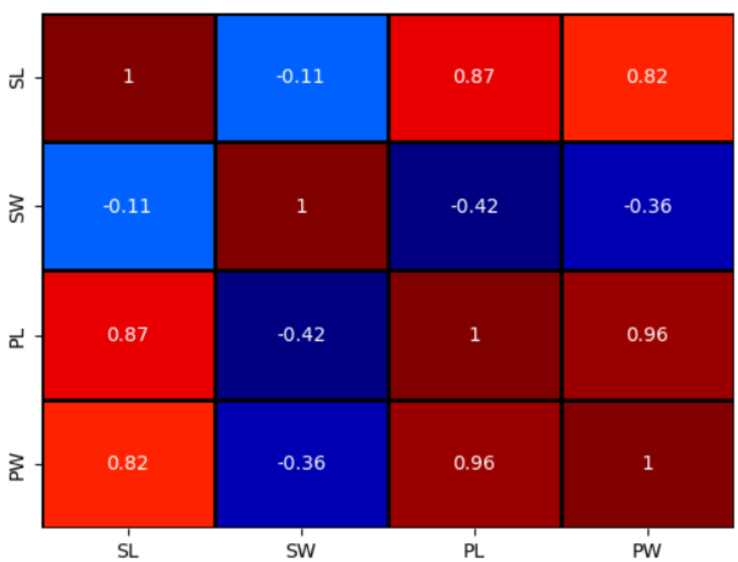


CODE:

```
cor=df.corr()
sns.heatmap(cor,annot=True,cbar=False,cmap='jet',linewidth=2,linecolor='black')
```

OUTPUT:

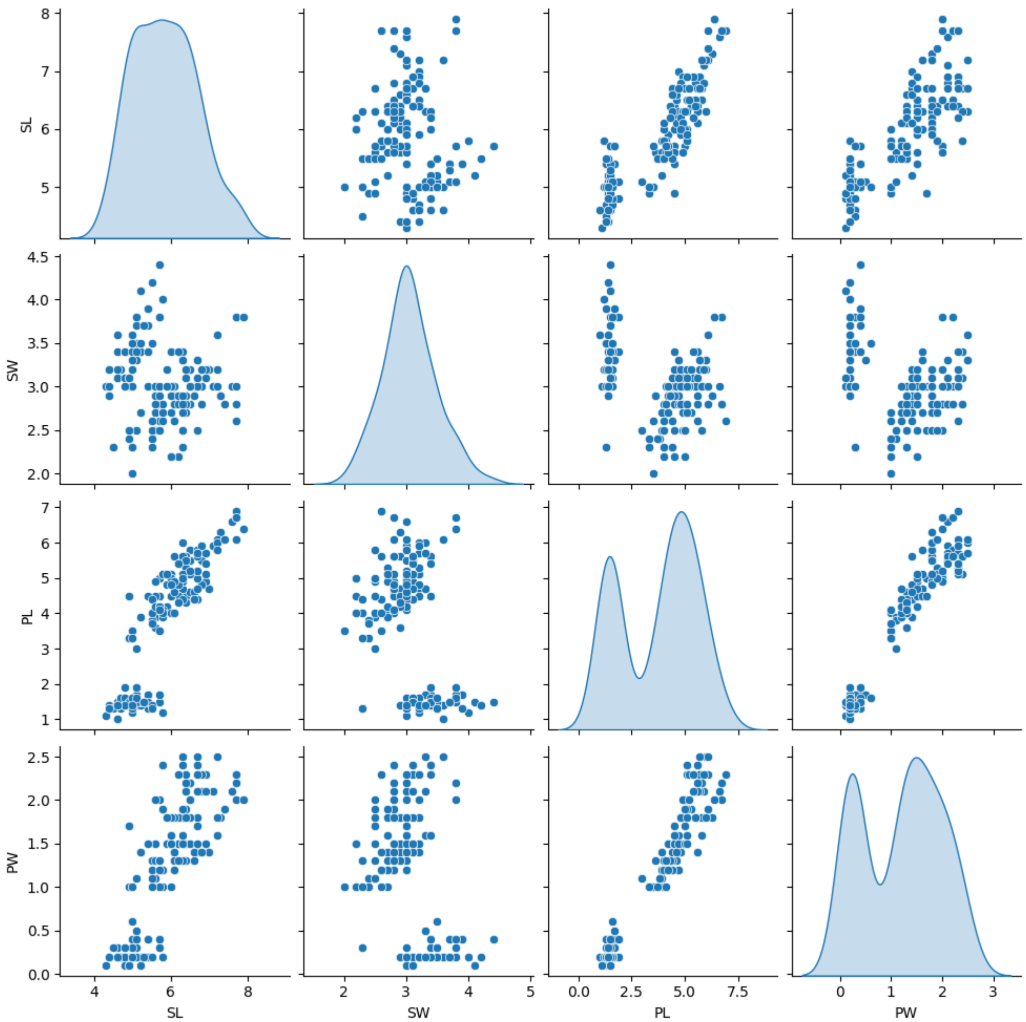
```
<ipython-input-10-cef91cb3c4cd>:1: FutureWarning: The default value of r
  cor=df.corr()
<Axes: >
```



CODE:

```
# Exploratory data analysis (EDA)
sns.pairplot(df,diag_kind='kde')
```

OUTPUT:



CODE:

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

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Target, dtype: int64
```

CODE:

```
dict={'Iris-setosa':0,'Iris-versicolor':1,'Iris-virginica':2}
df['Target']=df['Target'].map(dict)
df['Target'].value_counts()
```

```
0    50
1    50
2    50
Name: Target, dtype: int64
```

CODE:

```
[14] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   SL      150 non-null    float64
 1   SW      150 non-null    float64
 2   PL      150 non-null    float64
 3   PW      150 non-null    float64
 4   Target  150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

CODE:

```
y=df['Target']
x=df.drop(['Target'],axis=1)
print(x.shape)
print(y.shape)
```

OUTPUT:

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

LAB-2

AIM: Program to implement handle missing values and categorical data.

CODE:

```
import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/basics_data.csv')
print(df)
```

OUTPUT:

```
    F1  F2  F3    F4    F5  F6 Country  Color  Target
0    0   1   2   3.0   4.0   5     IND    red    Male
1    1   2   3   4.0   5.0   6      UK    red    Male
2    2   3   4   5.0   6.0   7     IND    red    Male
3    3   4   5   6.0   7.0   8      UK    red    Male
4    4   5   6   7.0   8.0   9     IND  green    Male
5    5   6   7   8.0   9.0  10      UK  green    Male
6    6   7   8   9.0  10.0  11     IND  green    Male
7    7   8   9   NaN  11.0  12      UK  green  Female
8    8   9  10   NaN  12.0  13     IND  green  Female
9    9  10  11  12.0  13.0  14      UK  green  Female
10  10  11  12  13.0   NaN  15     IND   blue  Female
11  11  12  13  14.0   NaN  16      UK   blue  Female
12  12  13  14  15.0  16.0  17     IND   blue  Female
13  13  14  15  16.0  17.0  18      UK   blue  Female
```

CODE:

```
df.isna().sum()
```

OUTPUT:

```
F1         0
F2         0
F3         0
F4         2
F5         2
F6         0
Country    0
Color      0
Target     0
dtype: int64
```

CODE:

```
df1=df.dropna() # Delete all NaN
df1.isna().sum()
```

OUTPUT:

```
F1         0
F2         0
F3         0
F4         0
F5         0
F6         0
Country    0
Color      0
Target     0
dtype: int64
```

CODE:

```
df.values
```

OUTPUT:

```
array([[0, 1, 2, 3.0, 4.0, 5, 'IND', 'red', 'Male'],
       [1, 2, 3, 4.0, 5.0, 6, 'UK', 'red', 'Male'],
       [2, 3, 4, 5.0, 6.0, 7, 'IND', 'red', 'Male'],
       [3, 4, 5, 6.0, 7.0, 8, 'UK', 'red', 'Male'],
       [4, 5, 6, 7.0, 8.0, 9, 'IND', 'green', 'Male'],
       [5, 6, 7, 8.0, 9.0, 10, 'UK', 'green', 'Male'],
       [6, 7, 8, 9.0, 10.0, 11, 'IND', 'green', 'Male'],
       [7, 8, 9, nan, 11.0, 12, 'UK', 'green', 'Female'],
       [8, 9, 10, nan, 12.0, 13, 'IND', 'green', 'Female'],
       [9, 10, 11, 12.0, 13.0, 14, 'UK', 'green', 'Female'],
       [10, 11, 12, 13.0, nan, 15, 'IND', 'blue', 'Female'],
       [11, 12, 13, 14.0, nan, 16, 'UK', 'blue', 'Female'],
       [12, 13, 14, 15.0, 16.0, 17, 'IND', 'blue', 'Female'],
       [13, 14, 15, 16.0, 17.0, 18, 'UK', 'blue', 'Female']], dtype=object)
```

CODE:

```
[16] df.describe()
```

OUTPUT:

	F1	F2	F3	F4	F5	F6
count	14.0000	14.0000	14.0000	12.000000	12.000000	14.0000
mean	6.5000	7.5000	8.5000	9.333333	9.833333	11.5000
std	4.1833	4.1833	4.1833	4.519319	4.152400	4.1833
min	0.0000	1.0000	2.0000	3.000000	4.000000	5.0000
25%	3.2500	4.2500	5.2500	5.750000	6.750000	8.2500
50%	6.5000	7.5000	8.5000	8.500000	9.500000	11.5000
75%	9.7500	10.7500	11.7500	13.250000	12.250000	14.7500
max	13.0000	14.0000	15.0000	16.000000	17.000000	18.0000

CODE:

```
[9] # impute
import numpy as np
from sklearn.impute import SimpleImputer
imp=SimpleImputer(missing_values=np.nan,strategy='constant',fill_value=-99)
x_imp=imp.fit_transform(df[['F4','F5']].values)
print(x_imp)
df1=df.drop(['F4','F5'],axis=1)
df2=pd.DataFrame(x_imp,columns=['F4','F5'])
df3=pd.concat([df1,df2],axis=1)
df
```

OUTPUT:

```
[[  3.   4.]
 [  4.   5.]
 [  5.   6.]
 [  6.   7.]
 [  7.   8.]
 [  8.   9.]
 [  9.  10.]
 [-99.  11.]
 [-99.  12.]
 [ 12.  13.]
 [ 13. -99.]
 [ 14. -99.]
 [ 15.  16.]
 [ 16.  17.]]
```

	F1	F2	F3	F4	F5	F6	Country	Color	Target
0	0	1	2	3.0	4.0	5	IND	red	Male
1	1	2	3	4.0	5.0	6	UK	red	Male
2	2	3	4	5.0	6.0	7	IND	red	Male
3	3	4	5	6.0	7.0	8	UK	red	Male
4	4	5	6	7.0	8.0	9	IND	green	Male
5	5	6	7	8.0	9.0	10	UK	green	Male
6	6	7	8	9.0	10.0	11	IND	green	Male
7	7	8	9	NaN	11.0	12	UK	green	Female
8	8	9	10	NaN	12.0	13	IND	green	Female
9	9	10	11	12.0	13.0	14	UK	green	Female
10	10	11	12	13.0	NaN	15	IND	blue	Female
11	11	12	13	14.0	NaN	16	UK	blue	Female
12	12	13	14	15.0	16.0	17	IND	blue	Female
13	13	14	15	16.0	17.0	18	UK	blue	Female

CODE:

```
[ ] df3.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   F1       14 non-null     int64  
 1   F2       14 non-null     int64  
 2   F3       14 non-null     int64  
 3   F6       14 non-null     int64  
 4   Country  14 non-null     object 
 5   Color    14 non-null     object 
 6   Target   14 non-null     object 
 7   F4       14 non-null     float64
 8   F5       14 non-null     float64
dtypes: float64(2), int64(4), object(3)
memory usage: 1.1+ KB
```

CODE:

```python
# dictionary mapping
dict={'Male':0,'Female':1}
df3['Target']=df3['Target'].map(dict)
print(df3)
```

OUTPUT:

```
    F1  F2  F3  F6 Country  Color  Target    F4    F5
0    0   1   2   5     IND    red       0   3.0   4.0
1    1   2   3   6      UK    red       0   4.0   5.0
2    2   3   4   7     IND    red       0   5.0   6.0
3    3   4   5   8      UK    red       0   6.0   7.0
4    4   5   6   9     IND  green       0   7.0   8.0
5    5   6   7  10      UK  green       0   8.0   9.0
6    6   7   8  11     IND  green       0   9.0  10.0
7    7   8   9  12      UK  green       1 -99.0  11.0
8    8   9  10  13     IND  green       1 -99.0  12.0
9    9  10  11  14      UK  green       1  12.0  13.0
10  10  11  12  15     IND   blue       1  13.0 -99.0
11  11  12  13  16      UK   blue       1  14.0 -99.0
12  12  13  14  17     IND   blue       1  15.0  16.0
13  13  14  15  18      UK   blue       1  16.0  17.0
```

CODE:

```
[ ]  dict={'UK':0,'IND':1}
     df3['Country']=df3['Country'].map(dict)
     df3
```

OUTPUT:

[ ]

	F1	F2	F3	F6	Country	Color	Target	F4	F5
0	0	1	2	5	1	red	0	3.0	4.0
1	1	2	3	6	0	red	0	4.0	5.0
2	2	3	4	7	1	red	0	5.0	6.0
3	3	4	5	8	0	red	0	6.0	7.0
4	4	5	6	9	1	green	0	7.0	8.0
5	5	6	7	10	0	green	0	8.0	9.0
6	6	7	8	11	1	green	0	9.0	10.0
7	7	8	9	12	0	green	1	-99.0	11.0
8	8	9	10	13	1	green	1	-99.0	12.0
9	9	10	11	14	0	green	1	12.0	13.0
10	10	11	12	15	1	blue	1	13.0	-99.0
11	11	12	13	16	0	blue	1	14.0	-99.0
12	12	13	14	17	1	blue	1	15.0	16.0
13	13	14	15	18	0	blue	1	16.0	17.0

CODE:

```
[ ]  df4=pd.get_dummies(df3['Color'],prefix='Color')
     df5=pd.concat([df3,df4],axis=1)
     df6=df5.drop(['Color'],axis=1)
     df6
```

OUTPUT:

	F1	F2	F3	F6	Country	Target	F4	F5	Color_blue	Color_green	Color_red
0	0	1	2	5	1	0	3.0	4.0	0	0	1
1	1	2	3	6	0	0	4.0	5.0	0	0	1
2	2	3	4	7	1	0	5.0	6.0	0	0	1
3	3	4	5	8	0	0	6.0	7.0	0	0	1
4	4	5	6	9	1	0	7.0	8.0	0	1	0
5	5	6	7	10	0	0	8.0	9.0	0	1	0
6	6	7	8	11	1	0	9.0	10.0	0	1	0
7	7	8	9	12	0	1	-99.0	11.0	0	1	0
8	8	9	10	13	1	1	-99.0	12.0	0	1	0
9	9	10	11	14	0	1	12.0	13.0	0	1	0
10	10	11	12	15	1	1	13.0	-99.0	1	0	0
11	11	12	13	16	0	1	14.0	-99.0	1	0	0
12	12	13	14	17	1	1	15.0	16.0	1	0	0
13	13	14	15	18	0	1	16.0	17.0	1	0	0

CODE:

```
# x, y
y=df6['Target']
x=df6.drop(['Target'],axis=1)
print(x.shape)
print(y.shape)
```

OUTPUT:

```
(14, 10)
(14,)
```

LAB-3

AIM: Program to implement feature scaling, feature extraction and selection.

CODE:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
# Feature Scaling
# Method-1: Standardization
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_std=sc.fit_transform(x_train)
x_test_std=sc.transform(x_test)
print(x_train_std)
```

OUTPUT:

```
[[ 1.42411021  1.42411021  1.42411021  1.42411021 -0.70710678  0.53577386
   0.54591027  1.41421356 -0.70710678 -0.70710678]
 [ 0.52467218  0.52467218  0.52467218  0.52467218 -0.70710678  0.41958194
   0.43030574 -0.70710678  1.41421356 -0.70710678]
 [-1.27420387 -1.27420387 -1.27420387 -1.27420387 -0.70710678  0.1871981
   0.19909669 -0.70710678 -0.70710678  1.41421356]
 [ 0.07495317  0.07495317  0.07495317  0.07495317 -0.70710678 -2.80474387
   0.37250348 -0.70710678  1.41421356 -0.70710678]
 [ 0.74953169  0.74953169  0.74953169  0.74953169  1.41421356  0.44862992
  -2.80662105  1.41421356 -0.70710678 -0.70710678]
 [-0.82448486 -0.82448486 -0.82448486 -0.82448486 -0.70710678  0.24529406
   0.25689895 -0.70710678 -0.70710678  1.41421356]
 [-1.49906338 -1.49906338 -1.49906338 -1.49906338  1.41421356  0.15815011
   0.17019556 -0.70710678 -0.70710678  1.41421356]
 [-0.37476584 -0.37476584 -0.37476584 -0.37476584 -0.70710678  0.30339002
   0.31470122 -0.70710678  1.41421356 -0.70710678]
 [ 1.1992507   1.1992507   1.1992507   1.1992507   1.41421356  0.50672588
   0.51700914  1.41421356 -0.70710678 -0.70710678]]
```

CODE:

```
# Method 2- Normalization
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x_train_mms=mms.fit_transform(x_train)
x_test_mms=mms.transform(x_test)
x_train_mms
```

OUTPUT:

```
array([[1.        , 1.        , 1.        , 1.        , 0.        ,
        1.        , 1.        , 1.        , 0.        , 0.        ],
       [0.69230769, 0.69230769, 0.69230769, 0.69230769, 0.        ,
        0.96521739, 0.96551724, 0.        , 1.        , 0.        ],
       [0.07692308, 0.07692308, 0.07692308, 0.07692308, 0.        ,
        0.89565217, 0.89655172, 0.        , 0.        , 1.        ],
       [0.53846154, 0.53846154, 0.53846154, 0.53846154, 0.        ,
        0.        , 0.94827586, 0.        , 1.        , 0.        ],
       [0.76923077, 0.76923077, 0.76923077, 0.76923077, 1.        ,
        0.97391304, 0.        , 1.        , 0.        , 0.        ],
       [0.23076923, 0.23076923, 0.23076923, 0.23076923, 0.        ,
        0.91304348, 0.9137931 , 0.        , 0.        , 1.        ],
       [0.        , 0.        , 0.        , 0.        , 1.        ,
        0.88695652, 0.88793103, 0.        , 0.        , 1.        ],
       [0.38461538, 0.38461538, 0.38461538, 0.38461538, 0.        ,
        0.93043478, 0.93103448, 0.        , 1.        , 0.        ],
       [0.92307692, 0.92307692, 0.92307692, 0.92307692, 1.        ,
        0.99130435, 0.99137931, 1.        , 0.        , 0.        ]])
```

CODE:

```
# Feature Extraction
# Method-1: PCA
print(x.shape)
from sklearn.decomposition import PCA
pca=PCA(n_components=2) # [1 to no_features]
x_pca=pca.fit_transform(x)# unsupervised
print(x_pca.shape)
x_pca
```

OUTPUT:

```
(14, 10)
(14, 2)
array([[ -1.1079453 , -14.10878741],
       [ -1.04390054, -15.39666555],
       [ -0.97993049, -16.6839798 ],
       [ -0.91588573, -17.97185794],
       [ -0.85805127, -19.2537908 ],
       [ -0.79400651, -20.54166894],
       [ -0.73003645, -21.82898319],
       [-76.67746766,  54.98436328],
       [-77.3108506 ,  54.41357402],
       [ -0.53797688, -25.69205373],
       [ 80.38946785,  51.53371816],
       [ 81.16900567,  50.94063505],
       [ -0.33323342, -29.55331251],
       [ -0.26918866, -30.84119065]])
```

CODE:

```
##### Method-2: LDA
print(x.shape)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda=LDA(n_components=1) # [1 to min(fetures, class-1)]
x_lda=lda.fit_transform(x,y) #supervised
print(x_lda.shape)
x_lda
```

OUTPUT:

```
(14, 10)
(14, 1)
array([[-4.66767768],
       [-3.73733329],
       [-2.76043416],
       [-1.83008978],
       [-3.51889565],
       [-2.58855127],
       [-1.61165214],
       [ 2.74242688],
       [ 3.75073642],
       [ 1.22593576],
       [ 2.77324181],
       [ 3.72035334],
       [ 2.78579768],
       [ 3.71614207]])
```

LAB-4

AIM: Program to implement simple and multiple linear regression.

CODE:

```
[2] import pandas as pd
    df=pd.read_csv('/content/drive/MyDrive/housing.csv',header=None,sep='\s+')
    df
```

OUTPUT:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

CODE:

```
# single varible regression
x=df.iloc[0] # 0th row - Sample
x=df.iloc[:,0:1] #0th col - feature
y=df.iloc[:,13] # 13th col as target
y
```

OUTPUT:

```
0        24.0
1        21.6
2        34.7
3        33.4
4        36.2
         ...
501      22.4
502      20.6
503      23.9
504      22.0
505      11.9
Name: 13, Length: 506, dtype: float64
```

CODE:

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x,y)
y_pred=model.predict(x)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y,y_pred)
r2=r2_score(y,y_pred)
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

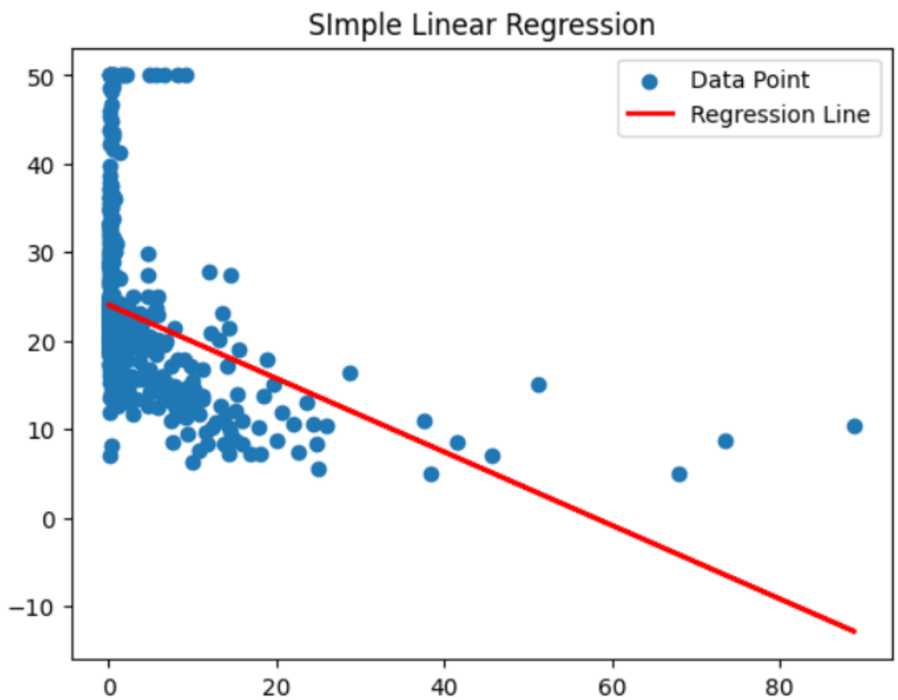
```
MSE= 71.69073588196659
R2-Score= 0.15078046904975717
```

CODE:

```
import matplotlib.pyplot as plt
plt.scatter(x,y,label='Data Point')
plt.plot(x,y_pred,c='r',lw='2',label='Regression Line')
plt.legend()
plt.title('SImple Linear Regression')
```

OUTPUT:

```
Text(0.5, 1.0, 'SImple Linear Regression')
```



CODE:

```
[6] # Multiple linear regression
    # Dataset
    x=df.iloc[:,0:13] # features
    y=df.iloc[:,13] # Target
    # train_test split
    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
    print(x.shape)
    print(x_train.shape)
    print(x_test.shape)
```

OUTPUT:

```
(506, 13)
(354, 13)
(152, 13)
```

CODE:

```
[7] from sklearn.linear_model import LinearRegression
    model=LinearRegression()
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    from sklearn.metrics import mean_squared_error,r2_score
    mse=mean_squared_error(y_test,y_pred)
    r2=r2_score(y_test,y_pred)
    print('Testing Performance')
    print('MSE=',mse)
    print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 27.195965766883493
R2-Score= 0.6733825506400162
```

LAB-5

AIM: Program to implement regularized and nonlinear regression.

CODE:

```
# Regularized Regression
# Lasso (Penalty-L1), Ridge(Penalty-L2), ElasticNet(L1+L2)
from sklearn.linear_model import Lasso
model=Lasso()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print('Testing Performance')
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 32.34503899856862
R2-Score= 0.6115433359595555
```

CODE:

```
from sklearn.linear_model import Ridge
model=Ridge()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print('Testing Performance')
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 27.7622245921665
R2-Score= 0.6665819091486692
```

CODE:

```
[10] from sklearn.linear_model import ElasticNet
model=ElasticNet()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print('Testing Performance')
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 31.87361081774105
R2-Score= 0.6172050826795714
```

CODE:

```
####### Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
model=DecisionTreeRegressor()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print('Testing Performance')
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 27.62085526315789
R2-Score= 0.6682797230838062
```

CODE:

```
####### Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print('Testing Performance')
print('MSE=',mse)
print('R2-Score=',r2)
```

OUTPUT:

```
Testing Performance
MSE= 14.059126927631587
R2-Score= 0.8311530387744892
```

CODE:

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
r1=LinearRegression()
r2=Lasso()
r3=Ridge()
r4=ElasticNet()
r5=SVR()
r6=DecisionTreeRegressor()
r7=RandomForestRegressor()
reg=[r1,r2,r3,r4,r5,r6,r7]
names=['LR','LASSO','RIDGE','EL','SVR','DTR','RFR']
mse={}
r2s={}
t={}
```

```python
import time
from sklearn.metrics import mean_squared_error,r2_score
##################################################################################
for model,name in zip(reg,names):
  t1=time.time()
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  t2=time.time()
  mse[name]=mean_squared_error(y_test,y_pred)
  r2s[name]=r2_score(y_test,y_pred)
  t[name]=t2-t1
########################################
for i,j in r2s.items():
  print(i,'=:=',j)
```

OUTPUT:

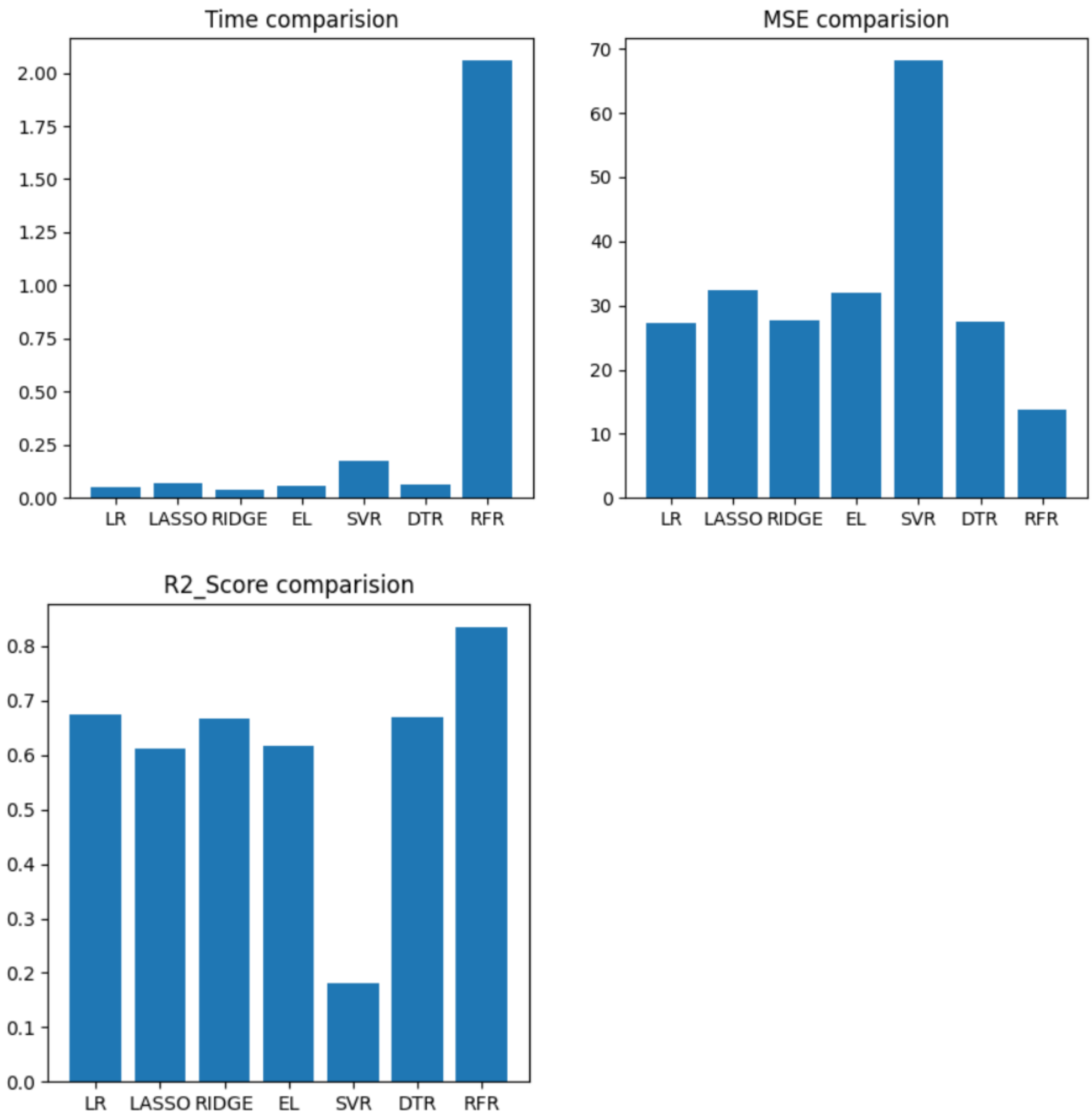
```
LR =:= 0.6733825506400162
LASSO =:= 0.6115433359595555
RIDGE =:= 0.6665819091486692
EL =:= 0.6172050826795714
SVR =:= 0.1811277097860169
DTR =:= 0.6699650422687811
RFR =:= 0.8349930153080197
```

CODE:

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.bar(t.keys(),t.values())
plt.title('Time comparision')
plt.subplot(2,2,2)
plt.bar(mse.keys(),mse.values())
plt.title('MSE comparision')
plt.subplot(2,2,3)
plt.bar(r2s.keys(),r2s.values())
plt.title('R2_Score comparision')
```

OUTPUT:

```
Text(0.5, 1.0, 'R2_Score comparision')
```



LAB-6

AIM: Program to implement classification algorithms Perceptron, logistic regression and SVM.

CODE:

```
[16] from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
     print(x_train.shape)
     print(x_test.shape)
     print(y_train.shape)
     print(y_test.shape)
```

OUTPUT:

```
(105, 4)
(45, 4)
(105,)
(45,)
```

CODE:

```
[17] from sklearn.linear_model import Perceptron,LogisticRegression
     from sklearn.svm import SVC
     clf1=Perceptron()
     clf2=LogisticRegression()
     clf3=SVC()
     clf1.fit(x_train,y_train)
     clf2.fit(x_train,y_train)
     clf3.fit(x_train,y_train)
     clf1_pred=clf1.predict(x_test)
     clf2_pred=clf2.predict(x_test)
     clf3_pred=clf3.predict(x_test)
```

```python
from sklearn.metrics import accuracy_score
clf1_acc=accuracy_score(y_test,clf1_pred)
clf2_acc=accuracy_score(y_test,clf2_pred)
clf3_acc=accuracy_score(y_test,clf3_pred)
print('<----- Testing Accuracy---->')
print('Perceptron Accuracy:',clf1_acc)
print('Logistic Regression Accuracy:',clf2_acc)
print('SVC Accuracy:',clf3_acc)
```

OUTPUT:

```
<----- Testing Accuracy---->
Perceptron Accuracy: 0.8
Logistic Regression Accuracy: 0.9777777777777777
SVC Accuracy: 0.9777777777777777
```

CODE:

```python
[18] from sklearn.metrics import classification_report
     cr=classification_report(y_test,clf1_pred)
     print(cr)
```

OUTPUT:

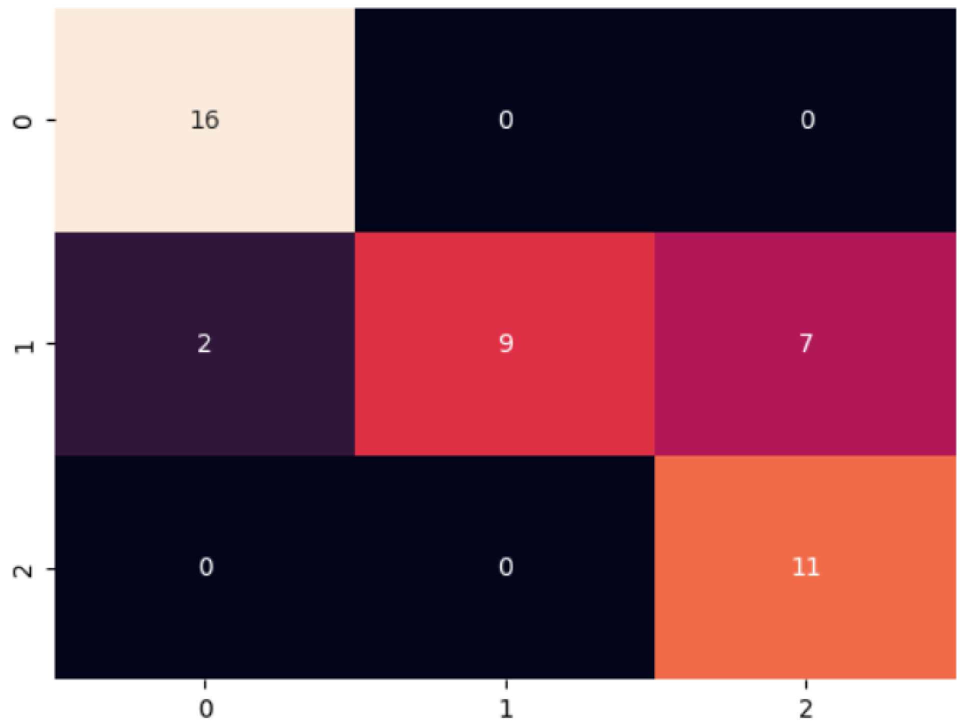
```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        16
           1       1.00      0.50      0.67        18
           2       0.61      1.00      0.76        11

    accuracy                           0.80        45
   macro avg       0.83      0.83      0.79        45
weighted avg       0.87      0.80      0.79        45
```

CODE:

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,clf1_pred)
sns.heatmap(cm,annot=True,cbar=False)
```

OUTPUT:

<Axes: >



LAB-7

AIM: Program to implement classification algorithm Decision Tree, Naive Bayes and KNN.

CODE:

```
[7]  # loading Dataset
     from sklearn.datasets import load_breast_cancer
     x,y=load_breast_cancer(return_X_y=True)
     print(x.shape)
     print(y.shape)
     print(y)
```

OUTPUT:

```
(569, 30)
(569,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

CODE:

```
[8] # train test split
    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
    print(x_train.shape)
    print(x_test.shape)
    print(y_train.shape)
    print(y_test.shape)
```

OUTPUT:

```
(398, 30)
(171, 30)
(398,)
(171,)
```

CODE:

```
from sklearn.linear_model import Perceptron,LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
clf1=Perceptron()
clf2=LogisticRegression()
clf3=SVC()
clf4=GaussianNB()
clf5=KNeighborsClassifier()
clf6=DecisionTreeClassifier()
clf=[clf1,clf2,clf3,clf4,clf5,clf6]
clf_name=['PERC','LR','SVM','GNB','KNN','DT']
acc={}
T={}
import time
from sklearn.metrics import accuracy_score
for model,model_name in zip(clf,clf_name):
  st=time.time()
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  et=time.time()
  acc[model_name]=accuracy_score(y_pred,y_test)
  T[model_name]=et-st
```

OUTPUT:

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

CODE:

```
[13] for i,j in acc.items():
       print(i,":-",j*100)
```

OUTPUT:

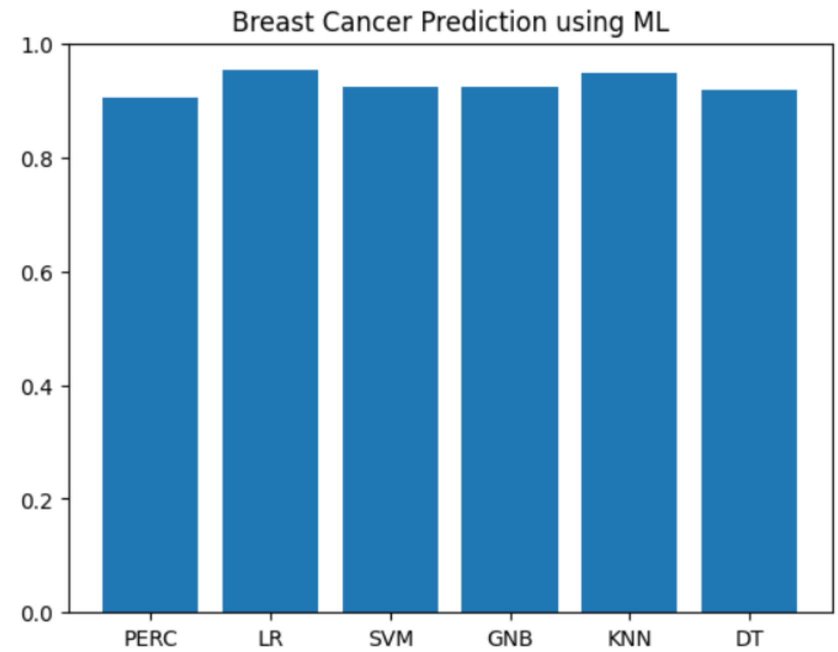
```
DT :- 92.98245614035088
PERC :- 90.64327485380117
LR :- 95.32163742690058
SVM :- 92.39766081871345
GNB :- 92.39766081871345
KNN :- 94.73684210526315
```

CODE:

```
import matplotlib.pyplot as plt
plt.bar(acc.keys(),acc.values())
plt.title('Breast Cancer Prediction using ML')
```

OUTPUT:

```
Text(0.5, 1.0, 'Breast Cancer Prediction using ML')
```



CODE:

```
for i,j in acc.items():
  print(i,":-",j*100)
```

OUTPUT:

```
DT :- 0.008004426956176758
PERC :- 0.004202365875244141
LR :- 0.03667259216308594
SVM :- 0.009318351745605469
GNB :- 0.0016100406646728516
KNN :- 0.11841917037963867
```

LAB-8

AIM: Program to implement ensemble learning classification algorithms.

CODE:

```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier

clf1 = RandomForestClassifier(n_estimators=100)
clf2 = ExtraTreesClassifier(n_estimators=100)
clf3 = BaggingClassifier(n_estimators=100, estimator= DecisionTreeClassifier(),bootstrap=True)
clf4 = AdaBoostClassifier(n_estimators=100)
clf5 = GradientBoostingClassifier(n_estimators=100)
clf6 = VotingClassifier(estimators=[('rf',clf1),('et',clf2),('bag',clf3),
('ada',clf4),('gbc',clf5)], voting='hard',weights=[1,1,1,1,1])
clf=[clf1,clf2,clf3,clf4,clf5,clf6]
clf_name=['RF','ET','BAG','ADA','GBC','VT']
acc={}
T={}
import time
from sklearn.metrics import accuracy_score
for model,model_name in zip(clf,clf_name):
  st=time.time()
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  et=time.time()
  acc[model_name]=accuracy_score(y_pred,y_test)
  T[model_name]=et-st
```

OUTPUT

```
for i,j in acc.items():
  print(i,":-",j*100)
```

```
DT :- 92.39766081871345
PERC :- 90.64327485380117
LR :- 95.32163742690058
SVM :- 92.39766081871345
GNB :- 92.39766081871345
KNN :- 94.73684210526315
```

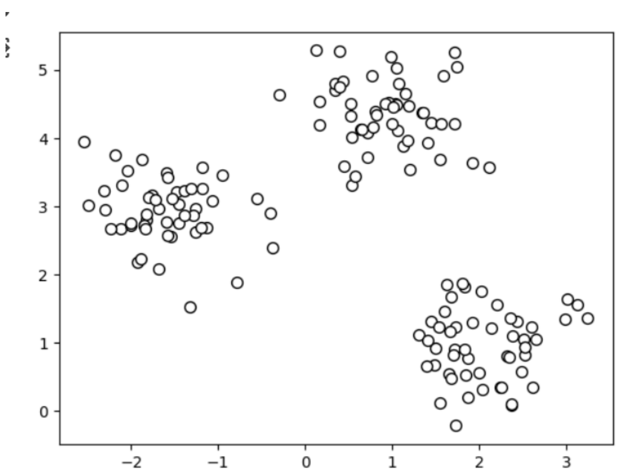
LAB-9

AIM: Program to implement K-Means clustering techniques.

CODE:

```
from sklearn.datasets import make_blobs
x,y=make_blobs(n_samples=150,n_features=2,centers=3,cluster_std=0.5,

shuffle=True,random_state=0)

import matplotlib.pyplot as plt
plt.scatter(x[:,0],x[:,1],c='white',edgecolor='black',marker='o',s=50)
plt.show()
```

OUTPUT:



CODE:

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3,init='random',max_iter=100,n_init=10)
y_km = km.fit_predict(x)
print('Distortion=',km.inertia_)
```

OUTPUT:

```
Distortion= 72.47601670996696
[[ 0.9329651   4.35420712]
 [-1.5947298   2.92236966]
 [ 2.06521743  0.96137409]]
```

CODE:

```
print(km.cluster_centers_)
```

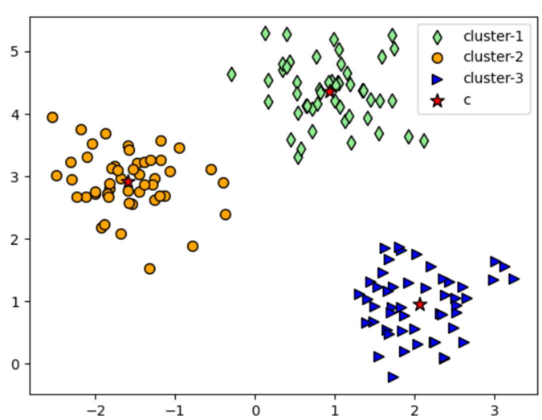
OUTPUT:

```
[[ 0.9329651   4.35420712]
 [-1.5947298   2.92236966]
 [ 2.06521743  0.96137409]]
```

CODE:

```
[14] plt.figure()
     plt.scatter(x[y_km==0,0],x[y_km==0,1],s=50,c='lightgreen',marker='d',edgecolor='black',label='cluster-1')
     plt.scatter(x[y_km==1,0],x[y_km==1,1],s=50,c='orange',marker='o',edgecolor='black',label='cluster-2')
     plt.scatter(x[y_km==2,0],x[y_km==2,1],s=50,c='blue',marker='>',edgecolor='black',label='cluster-3')
     plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],s=100,c='red',marker='*',edgecolor='black',label='c')
     plt.legend()
     plt.show()
```

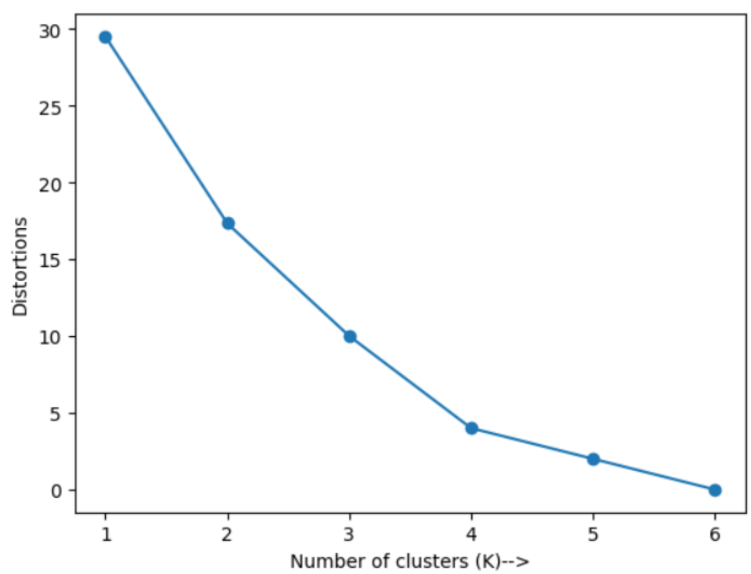
OUTPUT:



CODE:

```
dist=[]
for i in range(1,7):
    km=KMeans(n_clusters=i,init='k-means++',n_init=10,max_iter=100)
    km.fit(x)
    dist.append(km.inertia_)
plt.plot(range(1,7),dist,marker='o')
plt.xlabel('Number of clusters (K)-->')
plt.ylabel('Distortions')
plt.show()
```

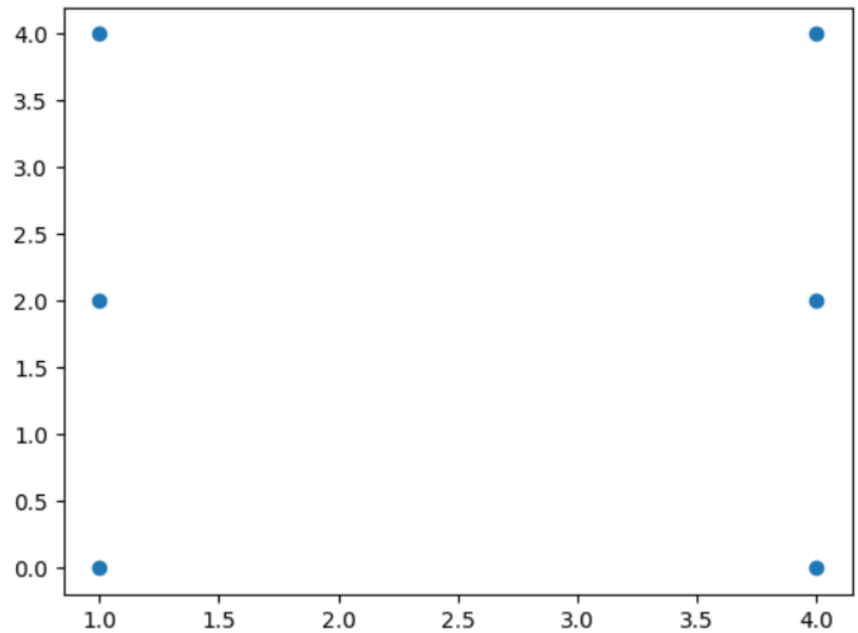
OUTPUT:



CODE:

```
#### Agglomerative

from sklearn.cluster import AgglomerativeClustering
import numpy as np
x=np.array([[1,2],[1,4],[1,0],[4,2],[4,4],[4,0]])
import matplotlib.pyplot as plt
plt.scatter(x[:,0],x[:,1])
plt.show()
ac=AgglomerativeClustering(n_clusters=2,metric='euclidean',linkage='single') # 'complete'
labels=ac.fit_predict(x)
print(labels)
```

OUTPUT:




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

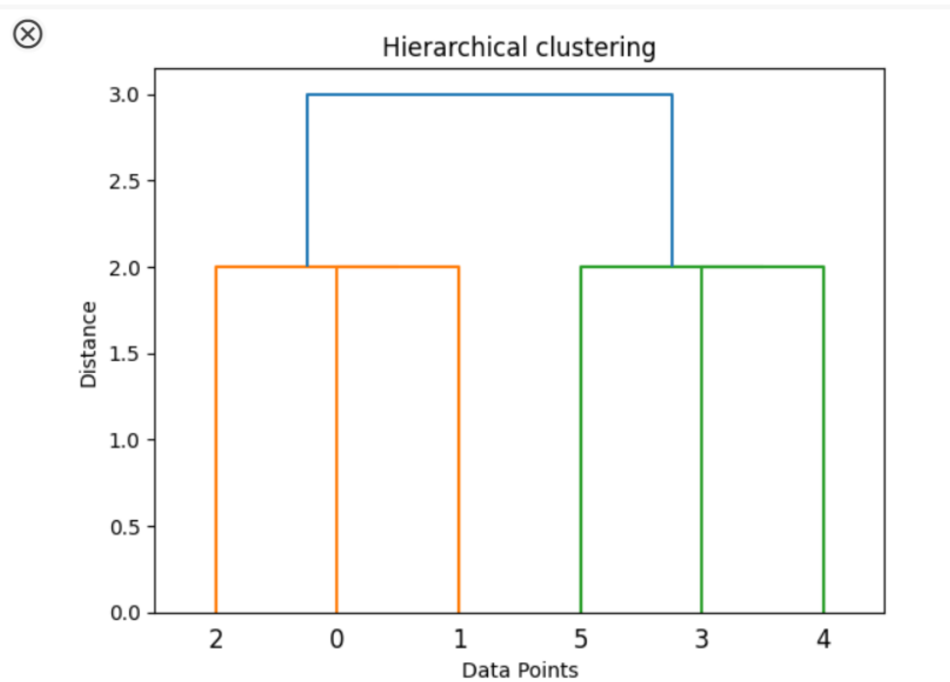
LAB-10

AIM: Program to implement Hierarchical and density-based clustering techniques.

CODE:

```
import numpy as np
x=np.array([[1,2],[1,4],[1,0],[4,2],[4,4],[4,0]])
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram,linkage
z=linkage(x,'single') # 'complete'
dendrogram(z)
plt.title('Hierarchical clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```

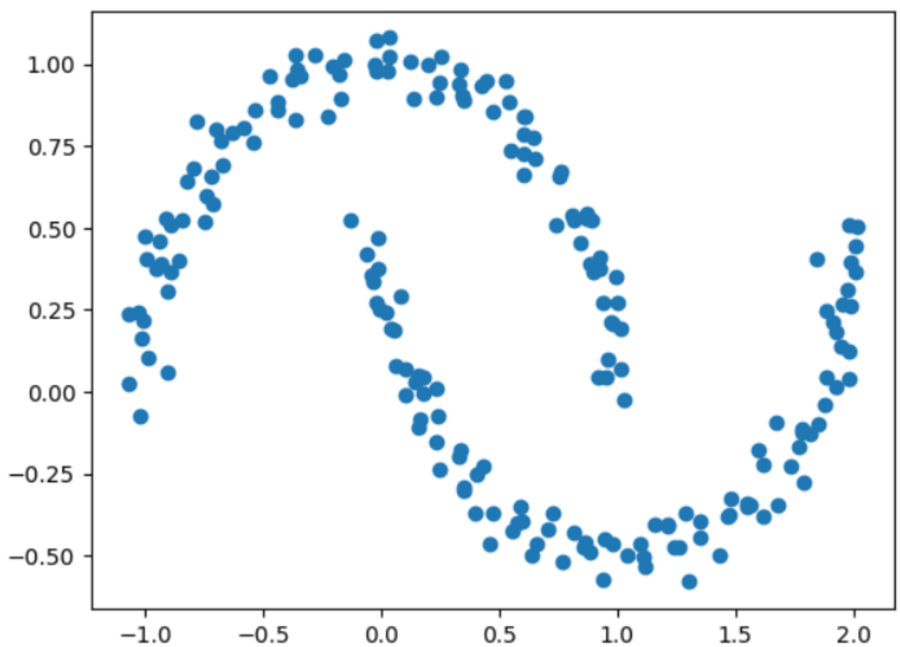
OUTPUT:



CODE:

```
from sklearn.datasets import make_moons
x,y=make_moons(n_samples=200,noise=0.05,random_state=0)
plt.scatter(x[:,0],x[:,1])
plt.show()
```

OUTPUT:

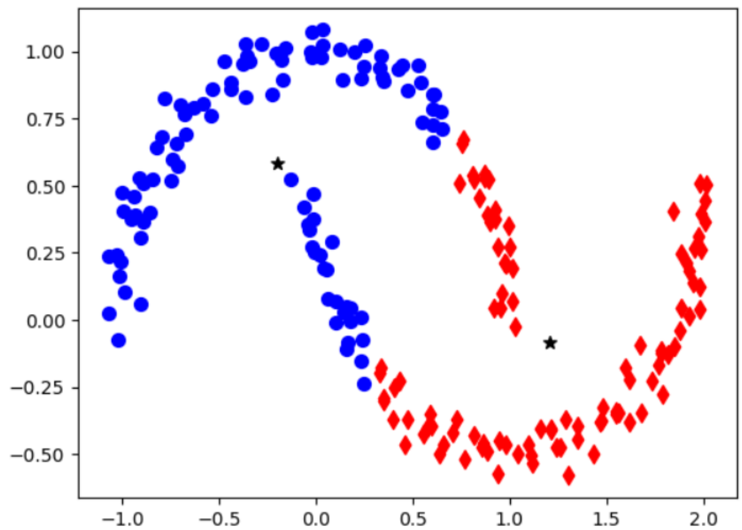


CODE:

```
from sklearn.cluster import KMeans
km=KMeans(n_clusters=2,random_state=0)
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],c='black',s=50,marker='*')
plt.show()
```

OUTPUT:

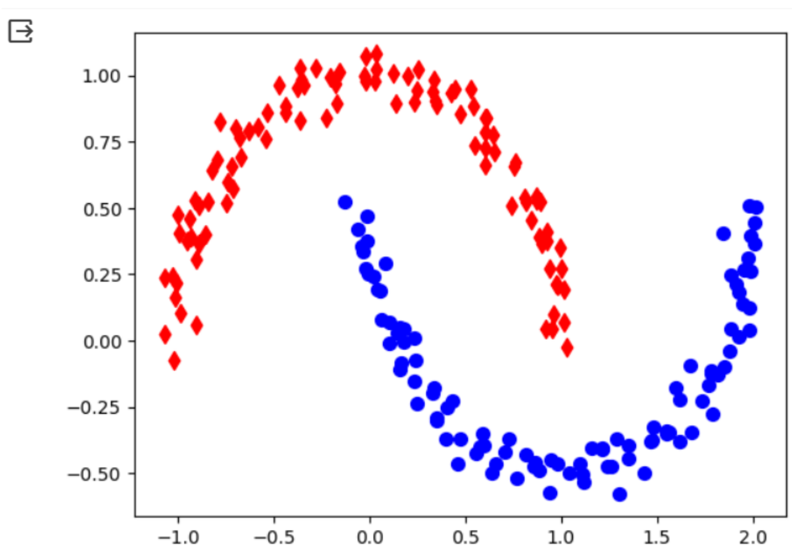
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```



CODE:

```
from sklearn.cluster import AgglomerativeClustering
km=AgglomerativeClustering(n_clusters=2,linkage='single') #'compete
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
plt.show()
```

OUTPUT:



CODE:

```
# CODE
from sklearn.cluster import DBSCAN
km=DBSCAN(eps=0.2,min_samples=5,metric='euclidean')
y_km=km.fit_predict(x)
plt.scatter(x[y_km==0,0],x[y_km==0,1],c='blue',s=50,marker='o')
plt.scatter(x[y_km==1,0],x[y_km==1,1],c='red',s=50,marker='d')
print("OUTPUT")
plt.show()
```

OUTPUT: