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


main_data=pd.read_csv('/content/dataR2.csv')


print(main_data.head())
print(main_data.columns)
```

```
   Age        BMI  Glucose  ...  Resistin    MCP.1  Classification
0   48  23.500000       70  ...   7.99585  417.114               1
1   83  20.690495       92  ...   4.06405  468.786               1
2   82  23.124670       91  ...   9.27715  554.697               1
3   68  21.367521       77  ...  12.76600  928.220               1
4   86  21.111111       92  ...  10.57635  773.920               1

[5 rows x 10 columns]
Index(['Age', 'BMI', 'Glucose', 'Insulin', 'HOMA', 'Leptin', 'Adiponectin',
       'Resistin', 'MCP.1', 'Classification'],
      dtype='object')
```

```
main_data.style
```

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Cla
0	48	23.500000	70	2.707000	0.467409	8.807100	9.702400	7.995850	417.114000	1
1	83	20.690495	92	3.115000	0.706897	8.843800	5.429285	4.064050	468.786000	1
2	82	23.124670	91	4.498000	1.009651	17.939300	22.432040	9.277150	554.697000	1
3	68	21.367521	77	3.226000	0.612725	9.882700	7.169560	12.766000	928.220000	1
4	86	21.111111	92	3.549000	0.805386	6.699400	4.819240	10.576350	773.920000	1
5	49	22.854458	92	3.226000	0.732087	6.831700	13.679750	10.317600	530.410000	1
6	89	22.700000	77	4.690000	0.890787	6.964000	5.589865	12.936100	1256.083000	1
7	76	23.800000	118	6.470000	1.883201	4.311000	13.251320	5.104200	280.694000	1
8	73	22.000000	97	3.350000	0.801543	4.470000	10.358725	6.284450	136.855000	1
9	75	23.000000	83	4.952000	1.013839	17.127000	11.578990	7.091300	318.302000	1
10	34	21.470000	78	3.469000	0.667436	14.570000	13.110000	6.920000	354.600000	1
11	29	23.010000	82	5.663000	1.145436	35.590000	26.720000	4.580000	174.800000	1
12	25	22.860000	82	4.090000	0.827271	20.450000	23.670000	5.140000	313.730000	1
13	24	18.670000	88	6.107000	1.330000	8.880000	36.060000	6.850000	632.220000	1
14	38	23.340000	75	5.782000	1.069670	15.260000	17.950000	9.350000	165.020000	1
15	44	20.760000	86	7.553000	1.600000	14.090000	20.320000	7.640000	63.610000	1
16	47	22.030000	84	2.869000	0.590000	26.650000	38.040000	3.320000	191.720000	1
17	61	32.038959	85	18.077000	3.790144	30.772900	7.780255	13.683920	444.395000	1
18	64	34.529723	95	4.427000	1.037394	21.211700	5.462620	6.701880	252.449000	1
19	32	36.512637	87	14.026000	3.009980	49.372700	5.100000	17.102230	588.460000	1
20	36	28.576676	86	4.345000	0.921719	15.124800	8.600000	9.153900	534.224000	1
21	34	31.975015	87	4.530000	0.972138	28.750200	7.642760	5.625920	572.783000	1
22	29	32.270788	84	5.810000	1.203832	45.619600	6.209635	24.603300	904.981000	1
23	35	30.276817	84	4.376000	0.906707	39.213400	9.048185	16.437060	733.797000	1
24	54	30.483158	90	5.537000	1.229214	12.331000	9.731380	10.192990	1227.910000	1
25	45	37.035608	83	6.760000	1.383997	39.980200	4.617125	8.704480	586.173000	1
26	50	38.578759	106	6.703000	1.752611	46.640100	4.667645	11.783880	887.160000	1
27	66	31.446541	90	9.245000	2.052390	45.962400	10.355260	23.381900	1102.110000	1
28	35	35.250761	90	6.817000	1.513374	50.609400	6.966895	22.037030	667.928000	1
29	36	34.174890	80	6.590000	1.300427	10.280900	5.065915	15.721870	581.313000	1
30	66	36.212279	101	15.533000	3.869788	74.706900	7.539550	22.320240	864.968000	1
31	53	36.790166	101	10.175000	2.534932	27.184100	20.030000	10.263090	695.754000	1
32	28	35.855815	87	8.576000	1.840410	68.510200	4.794200	21.443660	358.624000	1
33	43	34.422174	89	23.194000	5.091856	31.212800	8.300955	6.710260	960.246000	1
34	51	27.688778	77	3.855000	0.732193	20.092000	3.192090	10.375180	473.859000	1
35	67	29.606767	79	5.819000	1.133929	21.903300	2.194280	4.207500	585.307000	1
36	66	31.238590	82	4.181000	0.845677	16.224700	4.267105	3.291750	634.602000	1
37	69	35.092702	101	5.646000	1.406607	83.482100	6.796985	82.100000	263.499000	1
38	60	26.349292	103	5.138000	1.305395	24.299800	2.194280	20.253500	378.996000	1
39	77	35.587929	76	3.881000	0.727558	21.786300	8.125550	17.261500	618.272000	1
40	76	29.218408	83	5.376000	1.100646	28.562000	7.369960	8.043750	698.789000	1
41	76	27.200000	94	14.070000	3.262364	35.891000	9.346630	8.415600	377.227000	1
42	75	27.300000	85	5.197000	1.089638	10.390000	9.000805	7.576700	335.393000	1
43	69	32.500000	93	5.430000	1.245642	15.145000	11.787960	11.787960	270.142000	1
44	71	30.300000	102	8.340000	2.098344	56.502000	8.130000	4.298900	200.976000	1

45	66	27.700000	90	6.042000	1.341324	24.846000	7.652055	6.705200	225.880000	1
46	75	25.700000	94	8.079000	1.873251	65.926000	3.741220	4.496850	206.802000	1
47	78	25.300000	60	3.508000	0.519184	6.633000	10.567295	4.663800	209.749000	1
48	69	29.400000	89	10.704000	2.349885	45.272000	8.286300	4.530000	215.769000	1
49	85	26.600000	96	4.462000	1.056602	7.850000	7.931700	9.613500	232.006000	1
50	76	27.100000	110	26.211000	7.111918	21.778000	4.935635	8.493950	45.843000	1
51	77	25.900000	85	4.580000	0.960273	13.740000	9.753260	11.774000	488.829000	1
52	45	21.303949	102	13.852000	3.485163	7.647600	21.056625	23.034080	552.444000	2
53	45	20.829995	74	4.560000	0.832352	7.752900	8.237405	28.032300	382.955000	2
54	49	20.956608	94	12.305000	2.853119	11.240600	8.412175	23.117700	573.630000	2
55	34	24.242424	92	21.699000	4.924226	16.735300	21.823745	12.065340	481.949000	2
56	42	21.359915	93	2.999000	0.687971	19.082600	8.462915	17.376150	321.919000	2
57	68	21.082813	102	6.200000	1.559920	9.699400	8.574655	13.742440	448.799000	2
58	51	19.132653	93	4.364000	1.001102	11.081600	5.807620	5.570550	90.600000	2
59	62	22.656250	92	3.482000	0.790182	9.864800	11.236235	10.695480	703.973000	2
60	38	22.499637	95	5.261000	1.232828	8.438000	4.771920	15.736060	199.055000	2
61	69	21.513859	112	6.683000	1.846290	32.580000	4.138025	15.698760	713.239000	2
62	49	21.367521	78	2.640000	0.507936	6.333900	3.886145	22.942540	737.672000	2
63	51	22.892820	103	2.740000	0.696143	8.016300	9.349775	11.554920	359.232000	2
64	59	22.832879	98	6.862000	1.658774	14.903700	4.230105	8.204900	355.310000	2
65	45	23.140496	116	4.902000	1.402626	17.997300	4.294705	5.263300	518.586000	2
66	54	24.218750	86	3.730000	0.791257	8.687400	3.705230	10.344550	635.049000	2
67	64	22.222222	98	5.700000	1.377880	12.190500	4.783985	13.912450	395.976000	2
68	46	20.830000	88	3.420000	0.742368	12.870000	18.550000	13.560000	301.210000	2
69	44	19.560000	114	15.890000	4.468268	13.080000	20.370000	4.620000	220.660000	2
70	45	20.260000	92	3.440000	0.780651	7.650000	16.670000	7.840000	193.870000	2
71	44	24.740000	106	58.460000	15.285341	18.160000	16.100000	5.310000	244.750000	2
72	51	18.370000	105	6.030000	1.561770	9.620000	12.760000	3.210000	513.660000	2
73	72	23.620000	105	4.420000	1.144780	21.780000	17.860000	4.820000	195.940000	2
74	46	22.210000	86	36.940000	7.836205	10.160000	9.760000	5.680000	312.000000	2
75	43	26.562500	101	10.555000	2.629602	9.800000	6.420295	16.100000	806.724000	2
76	55	31.975015	92	16.635000	3.775036	37.223400	11.018455	7.165140	483.377000	2
77	43	31.250000	103	4.328000	1.099601	25.781600	12.718960	38.653100	775.322000	2
78	86	26.666667	201	41.611000	20.630734	47.647000	5.357135	24.370100	1698.440000	2
79	41	26.672763	97	22.033000	5.271762	44.705900	13.494865	27.832500	783.796000	2
80	59	28.672626	77	3.188000	0.605507	17.022000	16.440480	31.690400	910.489000	2
81	81	31.640368	100	9.669000	2.385020	38.806600	10.636525	29.558300	426.175000	2
82	48	32.461911	99	28.677000	7.002923	46.076000	21.570000	10.157260	738.034000	2
83	71	25.510204	112	10.395000	2.871792	19.065300	5.486100	42.744700	799.898000	2
84	42	29.296875	98	4.172000	1.008511	12.261700	6.695585	53.671700	1041.843000	2

```
print(main_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
```

```
 0   Age             116 non-null    int64
 1   BMI             116 non-null    float64
 2   Glucose         116 non-null    int64
 3   Insulin         116 non-null    float64
 4   HOMA            116 non-null    float64
 5   Leptin          116 non-null    float64
 6   Adiponectin     116 non-null    float64
 7   Resistin        116 non-null    float64
 8   MCP.1           116 non-null    float64
 9   Classification  116 non-null    int64
dtypes: float64(7), int64(3)
memory usage: 9.2 KB
None
```

```
target=main_data['Classification']
```

```
data=main_data.drop(['Classification'],axis=1)
```

```
print(data.isna().sum())
```

```
Age            0
BMI            0
Glucose        0
Insulin        0
HOMA           0
Leptin         0
Adiponectin    0
Resistin       0
MCP.1          0
dtype: int64
```

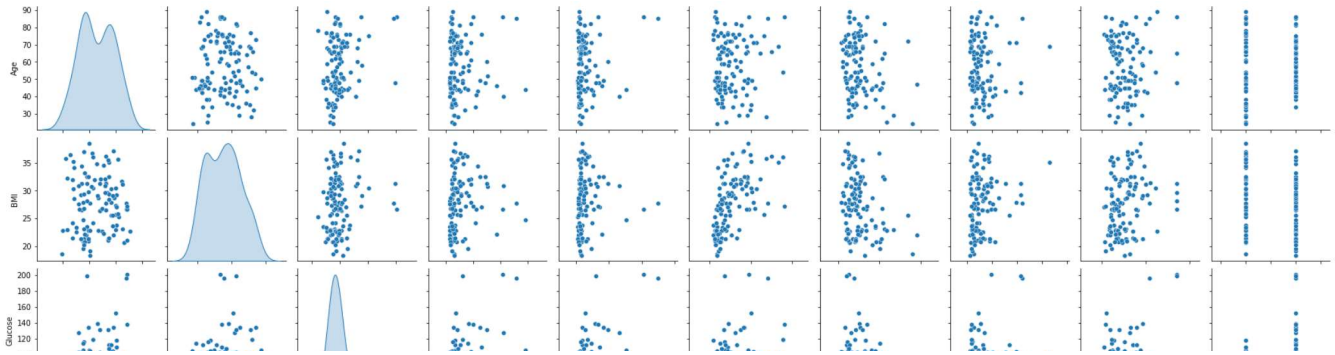
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(data,target,test_size=0.3,random_state=10)
print(x_train.shape)
print(x_test.shape)
```

```
(81, 9)
(35, 9)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
sc.fit(x_train)
x_train_std=sc.transform(x_train)
x_test_std=sc.transform(x_test)


import seaborn as sns
data=pd.DataFrame(x_train_std)
sns.pairplot(main_data,diag_kind='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x7fe87aa40e90>
```




```
from sklearn.metrics import accuracy_score
```

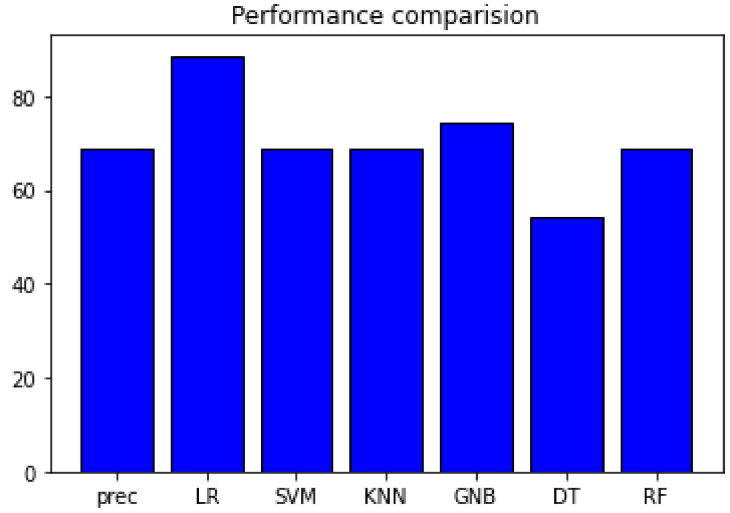
```
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
clf1=Perceptron(eta0=1)
clf2=LogisticRegression(penalty='l2',C=10)
clf3=SVC(C=100,kernel='rbf') # kernel='linear'
clf4=KNeighborsClassifier(n_neighbors=3) # 3-NN
clf5=GaussianNB()
clf6=DecisionTreeClassifier(max_depth=5)
clf7=RandomForestClassifier(max_depth=5)


clf=[clf1,clf2,clf3,clf4,clf5,clf6,clf7]
clf_names=['prec','LR','SVM','KNN','GNB','DT','RF']
test={}
T={}
import time
for model,name in zip(clf,clf_names):
  st=time.time()
  model.fit(x_train_std,y_train)
  y_pred=model.predict(x_test_std)
  et=time.time()
  acc=accuracy_score(y_test,y_pred)
  test[name]=np.round(acc*100,decimals=1)
  T[name]=np.round((et-st)*1000,decimals=1) # ms
print(test)
print(T)
```

```
{'prec': 68.6, 'LR': 88.6, 'SVM': 68.6, 'KNN': 68.6, 'GNB': 74.3, 'DT': 54.3, 'RF': 68.
{'prec': 1.6, 'LR': 5.9, 'SVM': 1.5, 'KNN': 3.5, 'GNB': 0.8, 'DT': 3.1, 'RF': 166.2}
```

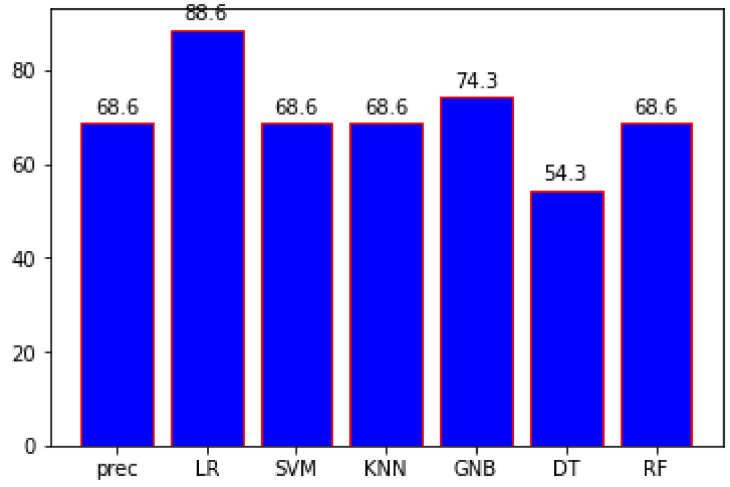
```
print(test.keys())
```

```
print(test.values())
plt.bar(test.keys(),test.values(),color='b',edgecolor='k')
plt.title('Performance comparision')
```

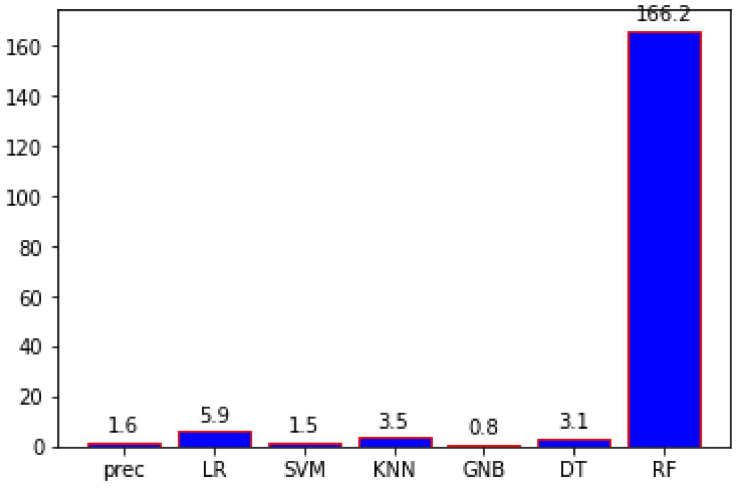
```
dict_keys(['prec', 'LR', 'SVM', 'KNN', 'GNB', 'DT', 'RF'])
dict_values([68.6, 88.6, 68.6, 68.6, 74.3, 54.3, 68.6])
Text(0.5, 1.0, 'Performance comparision')
```




```
fig,ax=plt.subplots()
#print(ax)
#print(fig)
#plt.subplots()
rect=plt.bar(test.keys(),test.values(),color='b',edgecolor='r')

def autolabel(rects):
  for rect in rects:
    height=rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x()+rect.get_width()/2,height),
                xytext=(0,3),
                textcoords='offset points',
                ha='center',va='bottom')
autolabel(rect)
```

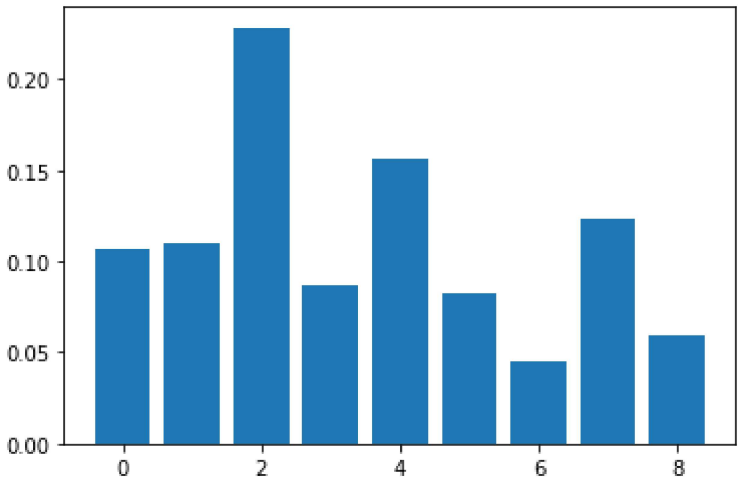


```
fig,ax=plt.subplots()
rect=plt.bar(T.keys(),T.values(),color='b',edgecolor='r')
def autolabel(rects):
  for rect in rects:
    height=rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x()+rect.get_width()/2,height),
                xytext=(0,3),
                textcoords='offset points',
                ha='center',va='bottom')
autolabel(rect)
```




```
print(clf7.feature_importances_)
plt.bar(range(0,9),clf7.feature_importances_)
```

```
[0.10680234 0.11020628 0.2284356  0.08662353 0.15683116 0.08272607
 0.04550312 0.12371492 0.05915699]
<BarContainer object of 9 artists>
```



```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=3,max_depth=3)
rf.fit(x_train_std,y_train)
pred=rf.predict(x_test_std)
from sklearn.metrics import accuracy_score
print('Test acc=',accuracy_score(y_test,pred))
```

```
Test acc= 0.6571428571428571
```

```
rf=pd.DataFrame(rf)
```

```
from sklearn.feature_selection import SelectFromModel
model=SelectFromModel(RandomForestClassifier(n_estimators=100,max_depth=None),
                      prefit=False,
                      max_features=7)
model.fit(x_train,y_train)
x_train_new=model.transform(x_train)
x_test_new=model.transform(x_test)
print(x_train.shape)
print(x_train_new.shape)
```

```
(81, 9)
(81, 4)
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train_new,y_train)
pred_lr=lr.predict(x_test_new)
from sklearn.metrics import accuracy_score
print('Accuracy with model selection',accuracy_score(pred_lr,y_test))
```

```
Accuracy with model selection 0.6857142857142857
```

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=3,max_depth=3)
rf.fit(x_train_new,y_train)
pred=rf.predict(x_test_new)
from sklearn.metrics import accuracy_score
print('Test acc=',accuracy_score(y_test,pred))
```

```
Test acc= 0.6857142857142857
```

```
from sklearn.decomposition import PCA
pca=PCA(n_components=7)
pca.fit(x_train_std)
x_train_pca=pca.transform(x_train_std)
x_test_pca=pca.transform(x_test_std)
print(x_train_pca.shape)
```

```
(81, 7)
```

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(max_depth=3)
dt.fit(x_train_pca,y_train)
pred=dt.predict(x_test_pca)
from sklearn.metrics import accuracy_score
print('Accuracy with 2-PCA',accuracy_score(y_test,pred))
```

```
Accuracy with 2-PCA 0.8
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train_pca,y_train)
pred_lr=lr.predict(x_test_pca)
from sklearn.metrics import accuracy_score
print('Accuracy with 2-PCA',accuracy_score(pred_lr,y_test))
```

```
Accuracy with 2-PCA 0.8571428571428571
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda=LDA(n_components=1)
lda.fit(x_train_std,y_train)
x_train_lda=lda.transform(x_train_std)
x_test_lda=lda.transform(x_test_std)
print(x_train_lda.shape)
```

```
(81, 1)
```

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=3,max_depth=3)
rf.fit(x_train_lda,y_train)
pred=rf.predict(x_test_lda)
from sklearn.metrics import accuracy_score
print('Test acc=',accuracy_score(y_test,pred))
```

```
Test acc= 0.5714285714285714
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train_lda,y_train)
pred_lr=lr.predict(x_test_lda)
from sklearn.metrics import accuracy_score
print('Accuracy with lda',accuracy_score(pred_lr,y_test))
```

```
Accuracy with lda 0.7714285714285715
```

```
from sklearn.metrics import classification_report
cr=classification_report(y_test,pred_lr)
print(cr)
```

```
                  precision    recall  f1-score   support

               1       0.75      0.75      0.75        16
               2       0.79      0.79      0.79        19

        accuracy                           0.77        35
       macro avg       0.77      0.77      0.77        35
    weighted avg       0.77      0.77      0.77        35
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(pred_lr,y_test)
import seaborn as sns
print(cm)
sns.heatmap(cm,annot=True,fmt='.0f',cbar=False)
```

```
[[12  4]
 [ 4 15]]
```

```
tn, fp, fn, tp = confusion_matrix(pred_lr,y_test).ravel()
print(tn, fp, fn, tp)
print('Specificity=',(tn/(tn+fp))*100)
```

```
12 4 4 15
Specificity= 75.0
```


```
from sklearn.preprocessing import label_binarize
target1=label_binarize(target,classes=[0,1])
n_classes=target1.shape[1]
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train_pca,y_train)
pred_lr=lr.predict(x_test_pca)
from sklearn.metrics import accuracy_score
acc=accuracy_score(pred,y_test)
print('Testing Accuracy=',acc)
```

```
Testing Accuracy= 0.5714285714285714
```

```
print(target1)
```

```
from sklearn.metrics import roc_curve,auc
fpr,tpr,_=roc_curve(pred,y_test,pos_label=1)
area=auc(fpr,tpr)
import matplotlib.pyplot as plt
plt.plot(fpr,tpr,label='ROC class 1 (area=%.2f)'%area)
plt.legend()
plt.plot([0,1],[0,1],ls='--')
```

```
[<matplotlib.lines.Line2D at 0x7fe8683b2c90>]
```

```
x=pd.read_csv('/content/dataR2.csv')
y=x['Classification']
x=x.drop(['Classification'],axis=1)
```


```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
clf=SVC()
scores=cross_val_score(estimator=clf,X=x,y=y,cv=2)
print(scores)
print('Overall performance: %.2f +/- %.2f'%(np.mean(scores),np.std(scores)))
```

```
[0.55172414 0.55172414]
Overall performance: 0.55 +/- 0.00
```

```
x=pd.read_csv('/content/dataR2.csv')
y=x['Classification']
x=x.drop(['Classification'],axis=1)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
pipe_svc=Pipeline([('std',StandardScaler()),
                   ('pca',PCA()),
                   ('clf',SVC())])
par={'clf__C':[0.001,0.01,0.1,1,10,100],
     'clf__kernel':['linear','rbf'],
     'pca__n_components':[1,2,3,4]}
from sklearn.model_selection import GridSearchCV
gs=GridSearchCV(estimator=pipe_svc,param_grid=par,cv=10,scoring='accuracy')
gs.fit(x_train,y_train)
print(gs.best_score_)
#print(gs.best_params_)
#print(gs.best_estimator_)
```

```
0.7544444444444445
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
```