```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import datasets
```

```
pip install datawig
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: datawig in /usr/local/lib/python3.7/dist-packages (0.
Requirement already satisfied: typing==3.6.6 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scikit-learn[alldeps]==0.22.1 in /usr/local/lib/pytho
Requirement already satisfied: mxnet==1.4.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pandas==0.25.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: numpy<1.15.0,>=1.8.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
```

```
import datawig
```

```
path = "/content/app_data.csv"
```

```
df = pd.read_csv(path)
df
```

	Age	BMI	Sex	Height	Weight	AlvaradoScore	PediatricAppendiciti
0	12.531143	16.494601	male	159.0	41.7	7	
1	12.410678	12.595222	female	152.0	29.1	8	
2	10.537988	15.991247	male	133.5	28.5	3	
3	10.425736	16.185025	male	146.0	34.5	4	
4	13.270363	20.449137	female	164.0	55.0	2	
...	...	...	...	...	...	...	

✓ 0s completed at 11:12 PM ● ✕

```
426  12.528405  29.316297    male  152.3  68.0        7
427  12.013689  28.906250    male  160.0  74.0        5
428   7.739904  22.038188  female  120.5  32.0        5
429  10.157426  21.017920  female  142.2  42.5        9

430 rows × 41 columns
```

```
#df.info()
```

```
#column dropping considering y3= AppendicitisComplications
df.drop(['AppendicitisComplications','TreatmentGroupBinar'],axis=1,inplace=True)

# Ultrasound
df.drop(['AppendixOnSono','AppendixDiameter','AppendixWallLayers','Kokarde','TissuePerfusi
        'BowelWallThick','Ileus','Enteritis','Peritonitis'],axis=1,inplace=True)
```

```
#df.info()
```

```
df_numerical = df.filter(['Age','BMI','Height','Weight','AlvaradoScore','PediatricAppendic
                    'AppendixDiameter','BodyTemp','WBCCount','NeutrophilPerc','CRPEntry'],
```

```
#df_numerical.info()
```

```
df_categorical = df.filter(['Sex','KetonesInUrine','ErythrocytesInUrine','WBCInUrine',
                           'Peritonitis','AppendixWallLayers','TissuePerfusion'],axis=1).c
```

```
#df_categorical.info()
```

```
#df_categorical.head()
```

```
df_boolean = df.filter(['AppendixOnSono','MigratoryPain','LowerAbdominalPainRight','Reboun
                    'Nausea','AppetiteLoss','Dysuria','FreeFluids','Kokarde',
                    'SurroundingTissueReaction','PathLymphNodes','MesentricLymphadenitis',
                    'FecalImpaction','Meteorism','Enteritis','DiagnosisByCriteria',
                     'PsoasSign','Stool'],axis=1).copy()
```

```
#df_boolean.info()
```

```
#df_boolean.sample(10)
```

```
#pandas profiling
#from pandas_profiling import ProfileReport
```

```
#profile = ProfileReport(df)
#profile.to_file(output_file = "AppendicitisComplications_profiling.html")
```

```
#perform label Encoding for categorical data

from sklearn.preprocessing import LabelEncoder
from pandas import Series
df_categorical = df_categorical.apply(lambda series:pd.Series(
      LabelEncoder().fit_transform(series[series.notnull()]),
      index = series[series.notnull()].index
   ))
```

```
#df_categorical.info()
```

```
#df_categorical.head()
```

```
#concatanation two dataframe
df_new = pd.concat([df_numerical,df_categorical],axis=1)
```

```
#df_new.info()
```

```
# Datawig imputation

from datawig import SimpleImputer
```

```
# impute missing values using Datawig
df_dw_imputed = datawig.SimpleImputer.complete(df_new)
```

```
#df_dw_imputed.head()
```

```
df_dw_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 14 columns):
Age                              430 non-null float64
BMI                              430 non-null float64
```

```
    BMI                            430 non-null float64
    Height                         430 non-null float64
    Weight                         430 non-null float64
    AlvaradoScore                  430 non-null float64
    PediatricAppendicitisScore     430 non-null float64
    BodyTemp                       430 non-null float64
    WBCCount                       430 non-null float64
    NeutrophilPerc                 430 non-null float64
    CRPEntry                       430 non-null float64
    Sex                            430 non-null float64
    KetonesInUrine                 430 non-null float64
    ErythrocytesInUrine            430 non-null float64
    WBCInUrine                     430 non-null float64
    dtypes: float64(14)
    memory usage: 47.2 KB
```

```
#df_dw_imputed.isnull()
```

```
#perform labelEncoding for Boolean data
df_boolean = df_boolean.apply(lambda series:pd.Series(
      LabelEncoder().fit_transform(series[series.notnull()]),
      index = series[series.notnull()].index
   ))
```

```
#df_boolean.head()
```

```
df_boolean = df_boolean.fillna(df_boolean.mode().iloc[0])
```

```
#df_boolean.sample(20)
```

```
#df_boolean.info()
```

```
#concatanation two dataframe
df_final = pd.concat([df_dw_imputed,df_boolean],axis=1)
```

```
df_final.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 430 entries, 0 to 429
    Data columns (total 29 columns):
    Age                            430 non-null float64
    BMI                            430 non-null float64
    Height                         430 non-null float64
    Weight                         430 non-null float64
    AlvaradoScore                  430 non-null float64
    PediatricAppendicitisScore     430 non-null float64
    BodyTemp                       430 non-null float64
    WBCCount                       430 non-null float64
```

```
 NeutrophilPerc               430 non-null float64
 CRPEntry                     430 non-null float64
 Sex                          430 non-null float64
 KetonesInUrine               430 non-null float64
 ErythrocytesInUrine          430 non-null float64
 WBCInUrine                   430 non-null float64
 MigratoryPain                430 non-null int64
 LowerAbdominalPainRight      430 non-null float64
 ReboundTenderness            430 non-null float64
 CoughingPain                 430 non-null float64
 Nausea                       430 non-null int64
 AppetiteLoss                 430 non-null float64
 Dysuria                      430 non-null float64
 FreeFluids                   430 non-null float64
 PathLymphNodes               430 non-null float64
 MesentricLymphadenitis       430 non-null float64
 FecalImpaction               430 non-null float64
 Meteorism                    430 non-null float64
 DiagnosisByCriteria          430 non-null int64
 PsoasSign                    430 non-null float64
 Stool                        430 non-null float64
dtypes: float64(26), int64(3)
memory usage: 97.5 KB
```

```
#correlation and pvalue

from scipy import stats
corr_df=pd.DataFrame(columns=['r','p'])

for col in df_final:
    print(col)
    if pd.api.types.is_numeric_dtype(df_final[col]):
        r,p = stats.pearsonr(df_final.DiagnosisByCriteria,df_final[col])
        corr_df.loc[col]=[round(r,3),round(p,3)]

corr_df
```

```
Age
BMI
Height
Weight
AlvaradoScore
PediatricAppendicitisScore
BodyTemp
WBCCount
NeutrophilPerc
CRPEntry
Sex
KetonesInUrine
ErythrocytesInUrine
WBCInUrine
MigratoryPain
LowerAbdominalPainRight
```

```
ReboundTenderness
CoughingPain
Nausea
AppetiteLoss
Dysuria
FreeFluids
PathLymphNodes
MesentricLymphadenitis
FecalImpaction
Meteorism
DiagnosisByCriteria
PsoasSign
Stool
```

	r	p
Age	0.072	0.136
BMI	0.109	0.024
Height	0.050	0.301
Weight	0.094	0.051
AlvaradoScore	-0.439	0.000
PediatricAppendicitisScore	-0.373	0.000
BodyTemp	-0.198	0.000
WBCCount	-0.411	0.000
NeutrophilPerc	-0.441	0.000
CRPEntry	-0.262	0.000
Sex	-0.102	0.034
KetonesInUrine	0.098	0.042
ErythrocytesInUrine	0.063	0.191
WBCInUrine	-0.038	0.434
MigratoryPain	-0.141	0.003
LowerAbdominalPainRight	-0.067	0.166
ReboundTenderness	-0.158	0.001
CoughingPain	-0.144	0.003
Nausea	-0.138	0.004
AppetiteLoss	-0.067	0.164
Dysuria	0.098	0.043
FreeFluids	-0.191	0.000

PathLymphNodes	0.018	0.709
MesentricLymphadenitis	-0.047	0.327
FecalImpaction	0.038	0.426
Meteorism	0.064	0.186
DiagnosisByCriteria	1.000	0.000
PsoasSign	0.080	0.097
Stool	0.071	0.144

```
df_final.shape
```

```
     (430, 29)
```

```
df_final['DiagnosisByCriteria'].value_counts()
```

```
     0    246
     1    184
     Name: DiagnosisByCriteria, dtype: int64
```

1 = yes, 0 = NO

```
no = df_final[df_final.DiagnosisByCriteria==0]
yes = df_final[df_final.DiagnosisByCriteria==1]
```

```
print(no.shape)
print(yes.shape)
```

```
     (246, 29)
     (184, 29)
```

```
#spliting the data for training and testing

X=df_final.drop(columns='DiagnosisByCriteria',axis=1)
Y=df_final['DiagnosisByCriteria']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2, stratify=Y, random
```

```
print(X.shape)
print(X_train.shape)
print(X_test.shape)
```

```
     (430, 28)
     (344, 28)
     (86, 28)
```

```
print(Y.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
     (430,)
     (344,)
     (86,)
```

Logistic Regression

```
# model training using logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
```

```
     /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
     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
       extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
```

```
# accuracy score for training data and testing data
X_train_prediction=model.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=model.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)


print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)
```

```
     Accuracy score for training data:  0.7267441860465116
     Accuracy score for testing data:  0.7906976744186046
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(model , X_train, Y_train, cv = kf)
result
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
    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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.6947058823529412
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(model , X_test, Y_test, cv = kf)
result
```

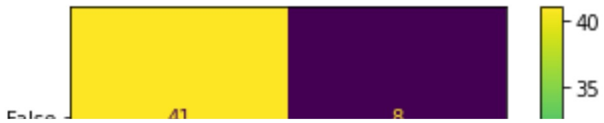
```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

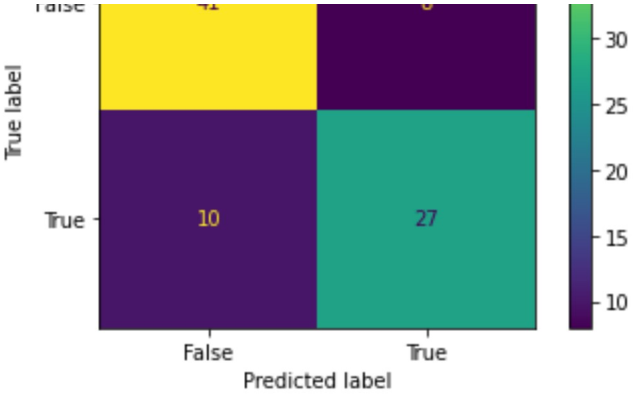
```
          STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

          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
            extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
          /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
          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
            extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
          /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
          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
            extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
          /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
          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
            extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
          /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
          STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.7305555555555555
```

```
from sklearn import metrics
import matplotlib.pyplot as plt

# make predictions
predicted = model.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```





```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

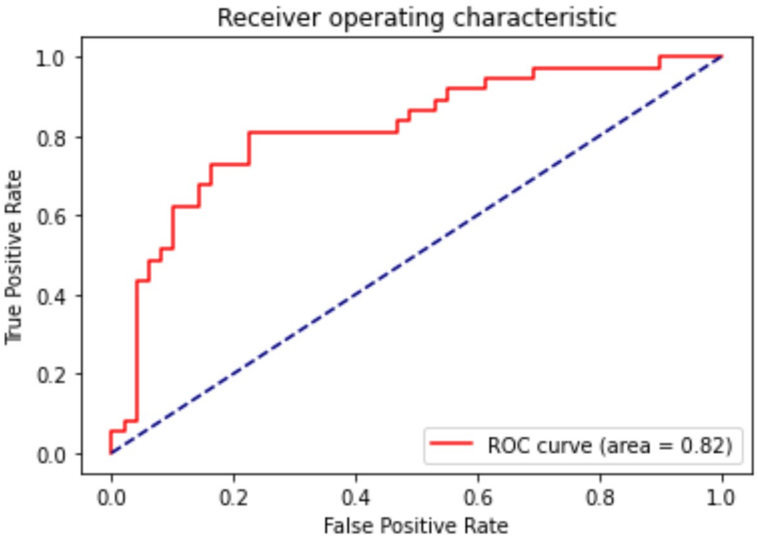
```
Sensitivity:  0.7297297297297297
specificity:  0.8367346938775511
PPV:  0.7714285714285715
NPV:  0.803921568627451
```

```
# AUROC and AUPR value
from sklearn.metrics import auc, roc_curve, precision_recall_curve

y_predictProb = model.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
AUROC: 0.819084390512962
AUPR: 0.7534701327211375
```

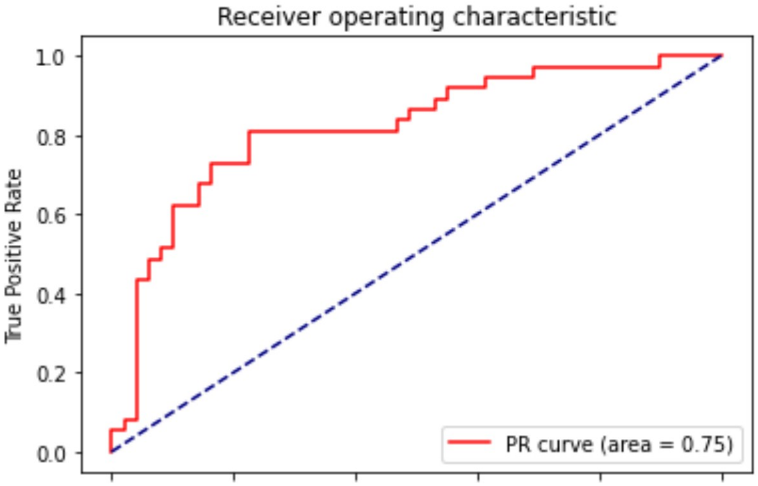
```
# AURoc graph
# AUROC graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

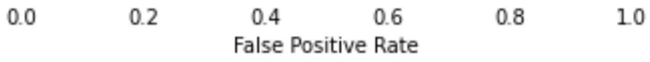
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```





Random Forest

```
# model training Using random forest
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(random_state = 1, n_estimators = 10, min_samples_split = 2
forest.fit(X_train, Y_train)
```

```
     RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                            criterion='gini', max_depth=None, max_features='auto',
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=10,
                            n_jobs=None, oob_score=False, random_state=1, verbose=0,
                            warm_start=False)
```

```
# accuracy score for training data and testing data
X_train_prediction=forest.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=forest.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)
```

```
print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)
```

```
     Accuracy score for training data:  0.997093023255814
     Accuracy score for testing data:  0.8023255813953488
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(forest , X_train, Y_train, cv = kf)
result
```

```
     array([0.74285714, 0.65714286, 0.57142857, 0.71428571, 0.67647059,
            0.70588235, 0.61764706, 0.73529412, 0.67647059, 0.58823529])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.6685714285714286
```

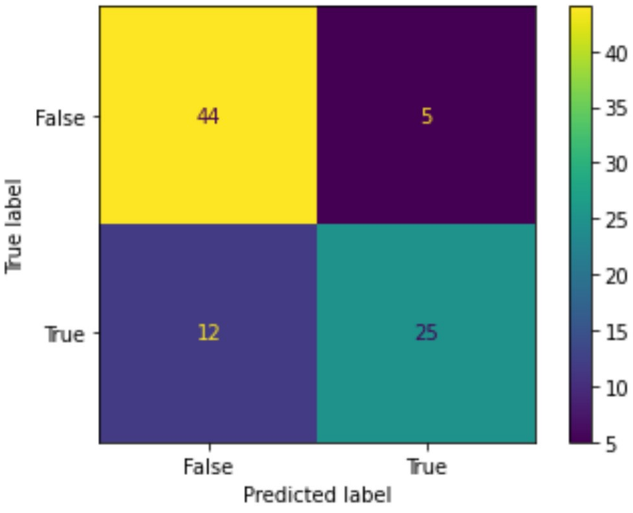
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(forest , X_test, Y_test, cv = kf)
result
```

```
     array([0.88888889, 0.66666667, 0.77777778, 0.66666667, 1.        ,
            0.66666667, 0.875     , 0.625     , 0.5       , 0.5       ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.7166666666666666
```

```
# make predictions
predicted = forest.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```



```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
```

```
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

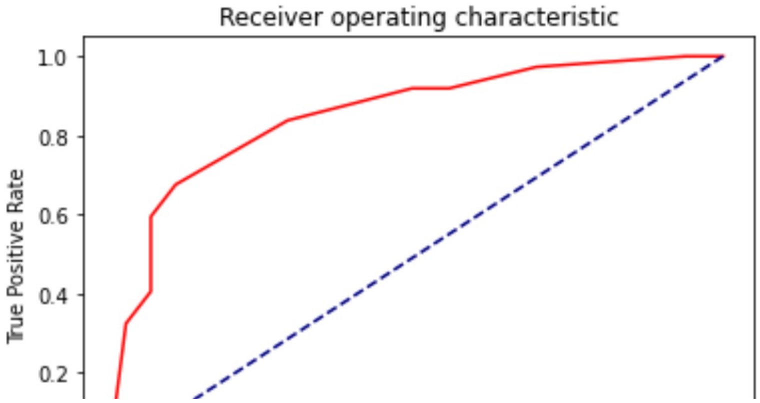
```
Sensitivity:  0.6756756756756757
specificity:  0.8979591836734694
PPV:  0.8333333333333334
NPV:  0.7857142857142857
```

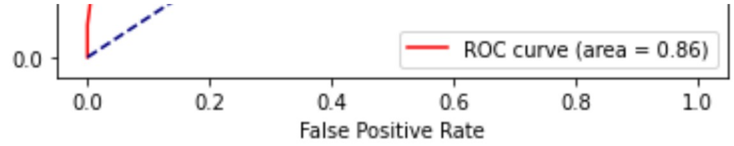
```
y_predictProb = forest.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
AUROC: 0.8574186431329289
AUPR: 0.8347604253197229
```

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

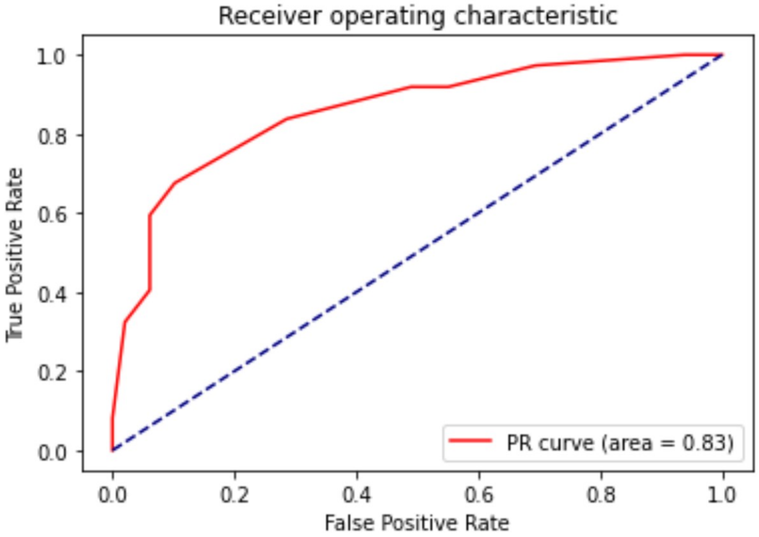
```
<function matplotlib.pyplot.show(*args, **kw)>
```





```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



Decision Tree

```
# using decisin tree
from sklearn.tree import DecisionTreeClassifier
dclf = DecisionTreeClassifier()
dclf.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# accuracy score for training data and testing data
```

```
X_train_prediction=dclf.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=dclf.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)


print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)
```

```
     Accuracy score for training data:  1.0
     Accuracy score for testing data:  0.6976744186046512
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(dclf , X_train, Y_train, cv = kf)
result
```

```
     array([0.57142857, 0.57142857, 0.48571429, 0.54285714, 0.64705882,
            0.52941176, 0.58823529, 0.64705882, 0.64705882, 0.52941176])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.5759663865546218
```

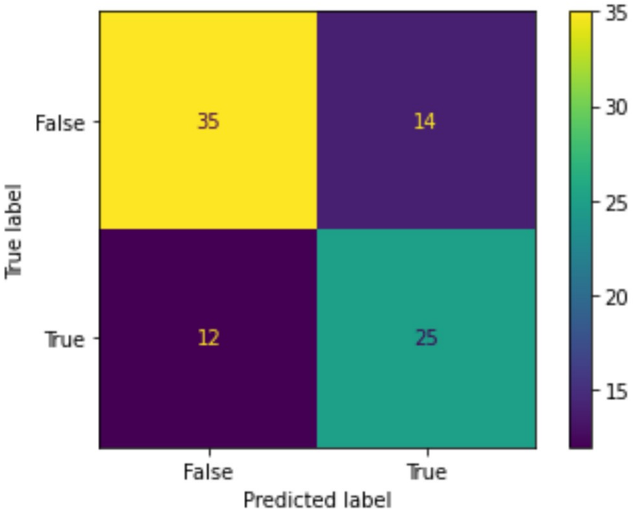
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(dclf , X_test, Y_test, cv = kf)
result
```

```
     array([0.55555556, 0.66666667, 0.66666667, 0.55555556, 0.66666667,
            0.66666667, 0.5       , 0.5       , 0.625     , 0.75      ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.6152777777777778
```

```
# make predictions
predicted = dclf.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```



```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

```
Sensitivity:  0.6756756756756757
specificity:  0.7142857142857143
PPV:  0.6410256410256411
NPV:  0.7446808510638298
```

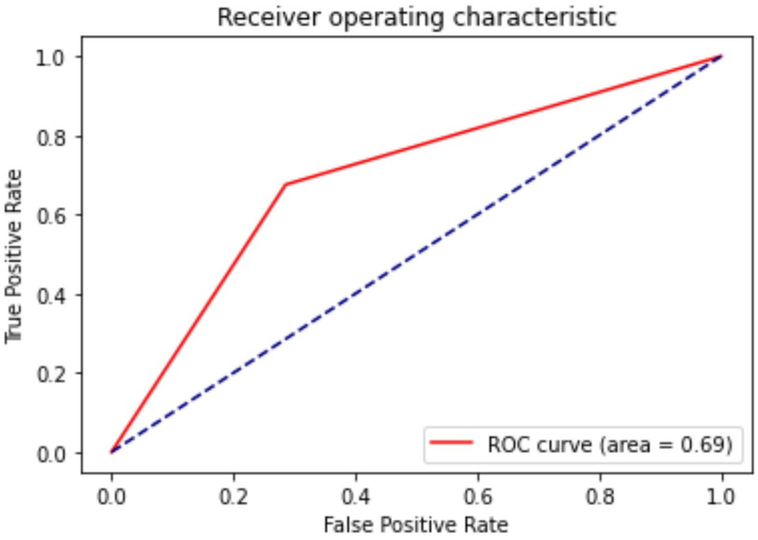
```
# AUROC and AUPR value
y_predictProb = dclf.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)
```

```
print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
     AUROC: 0.694980694980695
     AUPR: 0.7281181002111234
```

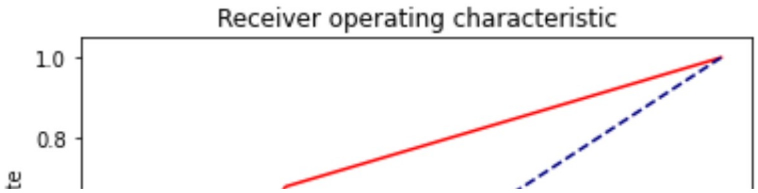
```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

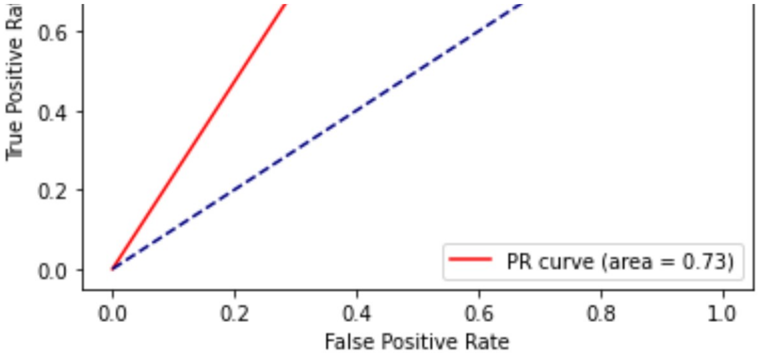
```
     <function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
     <function matplotlib.pyplot.show(*args, **kw)>
```





Gradient Bosst

```
#using GradientBoost
from sklearn.ensemble import GradientBoostingClassifier
gdb = GradientBoostingClassifier(random_state = 1, n_estimators = 10, min_samples_split =
gdb.fit(X_train,Y_train)
```

```
     GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                                learning_rate=0.1, loss='deviance', max_depth=3,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=10,
                                n_iter_no_change=None, presort='deprecated',
                                random_state=1, subsample=1.0, tol=0.0001,
                                validation_fraction=0.1, verbose=0,
                                warm_start=False)
```

```
# accuracy score for training data and testing data
X_train_prediction=gdb.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=gdb.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)


print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)
```

```
     Accuracy score for training data:  0.8459302325581395
     Accuracy score for testing data:  0.7209302325581395
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
```

```
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(gdb, X_train, Y_train, cv = kf)
result
```

```
     array([0.68571429, 0.71428571, 0.6       , 0.74285714, 0.76470588,
            0.70588235, 0.64705882, 0.79411765, 0.79411765, 0.61764706])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.7066386554621849
```

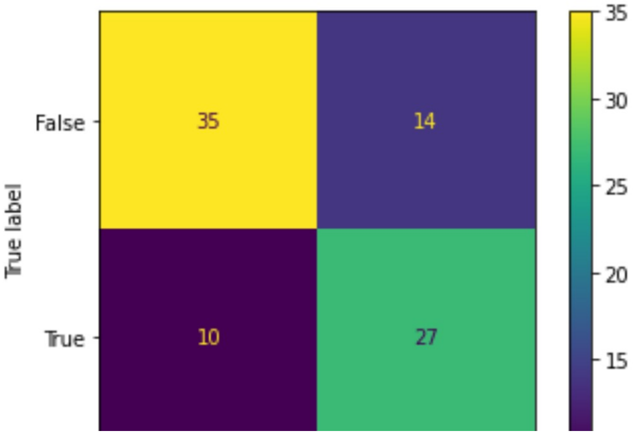
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(gdb, X_test, Y_test, cv = kf)
result
```

```
     array([0.77777778, 0.77777778, 0.66666667, 0.66666667, 0.88888889,
            0.66666667, 0.625     , 0.625     , 0.625     , 0.625     ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.6944444444444444
```

```
# make predictions
predicted = gdb.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```



```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

```
Sensitivity:  0.7297297297297297
specificity:  0.7142857142857143
PPV:  0.6585365853658537
NPV:  0.7777777777777778
```

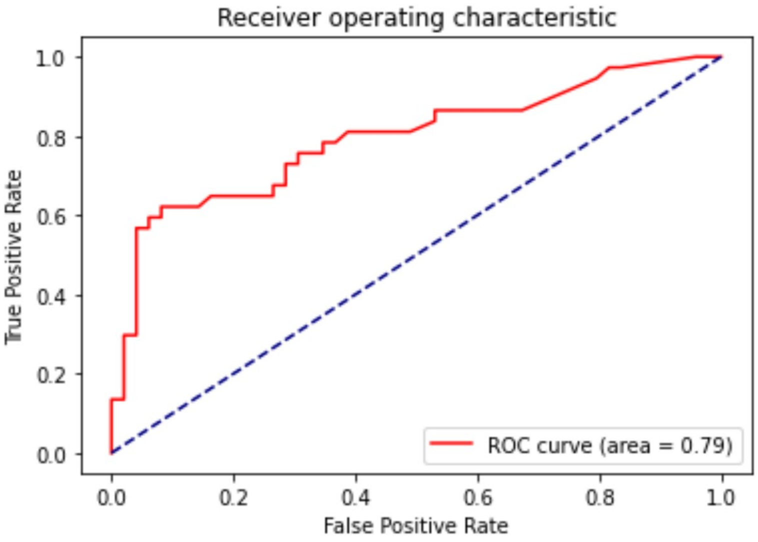
```
# AUROC and AUPR value
y_predictProb = gdb.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
AUROC: 0.7934362934362934
AUPR: 0.7824173325929256
```

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

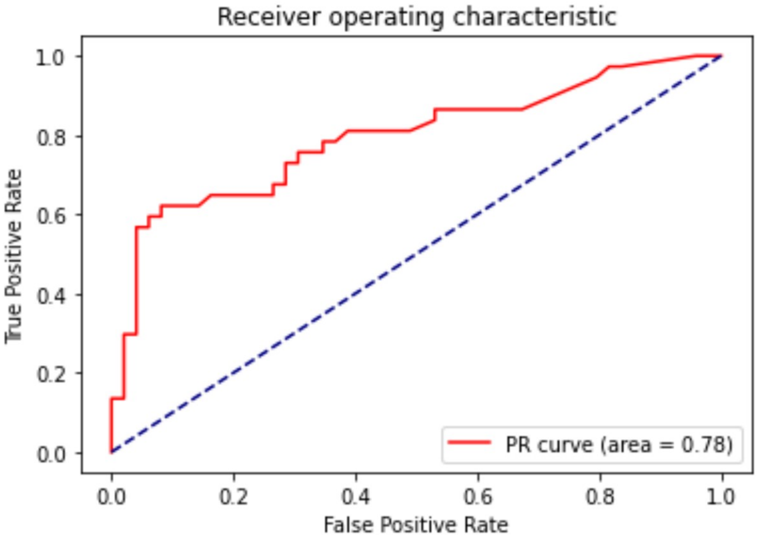
```
<function matplotlib.pyplot.show(*args, **kw)>
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
    <function matplotlib.pyplot.show(*args, **kw)>
```



XGBoost

```
#using XGBClassifier
from xgboost import XGBClassifier
xgb_clf = XGBClassifier(random_state = 1, n_estimators = 10, min_samples_split = 2)
xgb_clf = XGBClassifier(random_state = 1, n_estimators = 10, min_samples_split = 2)
xgb_clf.fit(X_train, Y_train)

     XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                   colsample_bynode=1, colsample_bytree=1, gamma=0,
                   learning_rate=0.1, max_delta_step=0, max_depth=3,
                   min_child_weight=1, min_samples_split=2, missing=None,
                   n_estimators=10, n_jobs=1, nthread=None,
                   objective='binary:logistic', random_state=1, reg_alpha=0,
                   reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
                   subsample=1, verbosity=1)
```

```
# accuracy score for training data and testing data
X_train_prediction=xgb_clf.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=xgb_clf.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)
```

```
print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)

     Accuracy score for training data:  0.8343023255813954
     Accuracy score for testing data:  0.7674418604651163
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(xgb_clf, X_train, Y_train, cv = kf)
result

     array([0.68571429, 0.71428571, 0.6       , 0.77142857, 0.79411765,
            0.82352941, 0.61764706, 0.82352941, 0.76470588, 0.52941176])
```

```
print("Avg accuracy: {}".format(result.mean()))

     Avg accuracy: 0.712436974789916
```

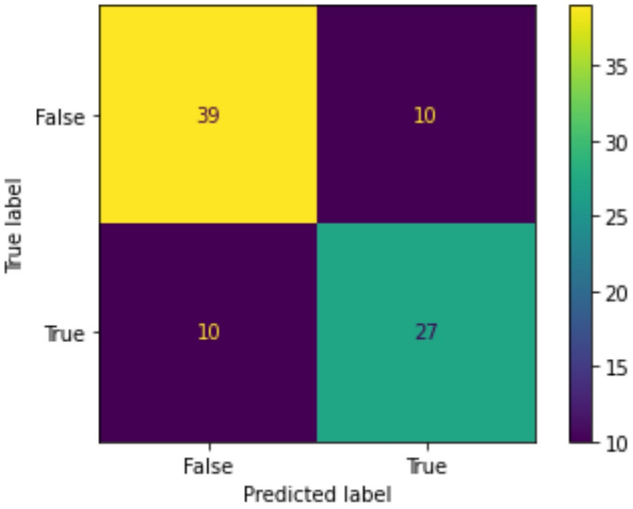
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(xgb_clf, X_test, Y_test, cv = kf)
result
```

```
array([0.66666667, 0.77777778, 0.55555556, 0.77777778, 0.88888889,
       0.66666667, 0.625     , 0.5       , 0.625     , 0.625     ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.6708333333333333
```

```
# make predictions
predicted = xgb_clf.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```



```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

```
Sensitivity:  0.7297297297297297
specificity:  0.7959183673469388
```

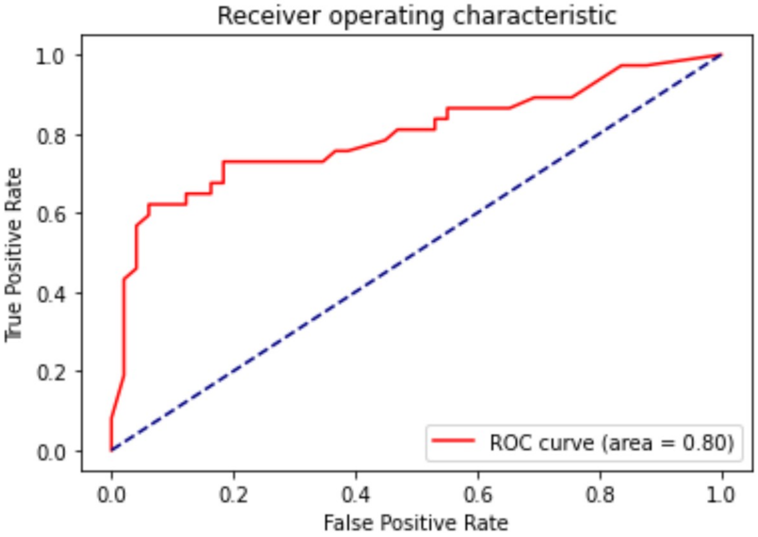
```
     PPV:  0.7297297297297297
     NPV:  0.7959183673469388
```

```
# AUROC and AUPR value
y_predictProb = xgb_clf.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
```

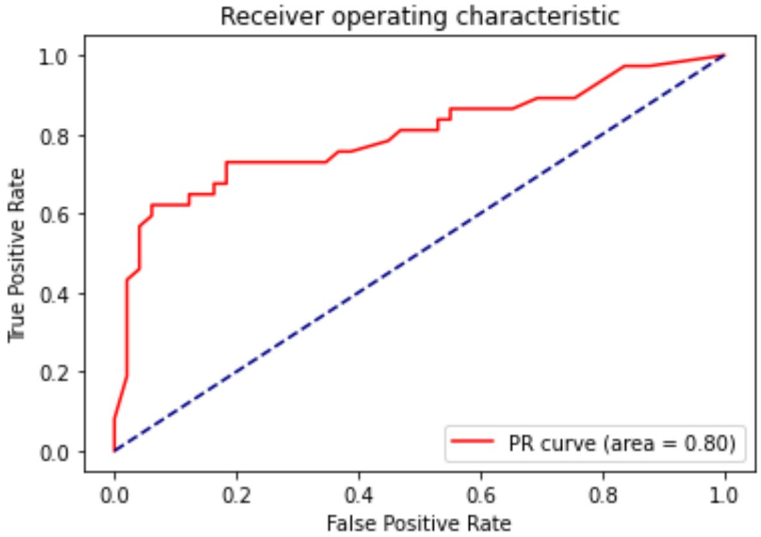
```
     AUROC: 0.7972972972972974
     AUPR: 0.7981290381782874
```

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
     <function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show

     <function matplotlib.pyplot.show(*args, **kw)>
```



Support Vector

```
#using support vector
from sklearn import svm
sv_clf = svm.SVC()
sv_clf.fit(X_train, Y_train)

     SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
         decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
         max_iter=-1, probability=False, random_state=None, shrinking=True,
         tol=0.001, verbose=False)
```

```
# accuracy score for training data and testing data
X_train_prediction=sv_clf.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)

X_test_prediction=sv_clf.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)
```

```
print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)

     Accuracy score for training data:  0.7209302325581395
     Accuracy score for testing data:  0.7790697674418605
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(sv_clf , X_train, Y_train, cv = kf)
result
```

```
     array([0.74285714, 0.68571429, 0.71428571, 0.71428571, 0.76470588,
            0.73529412, 0.61764706, 0.82352941, 0.73529412, 0.61764706])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.7151260504201682
```

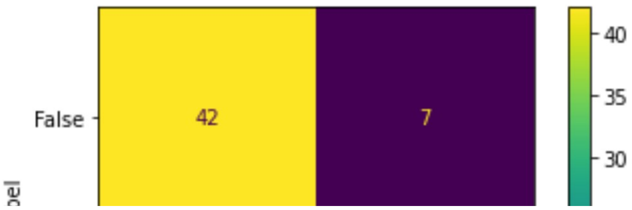
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(sv_clf , X_test, Y_test, cv = kf)
result
```

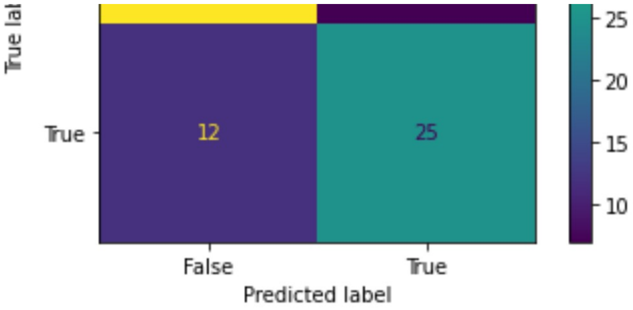
```
     array([0.77777778, 0.88888889, 0.77777778, 0.77777778, 0.66666667,
            0.66666667, 0.875     , 0.75      , 0.625     , 0.625     ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
     Avg accuracy: 0.7430555555555556
```

```
# make predictions
predicted = sv_clf.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```





```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

```
Sensitivity:  0.6756756756756757
specificity:  0.8571428571428571
PPV:  0.78125
NPV:  0.7777777777777778
```

```
# AUROC and AUPR value
y_predictProb = sv_clf.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-114-289267775586> in <module>
      1 # AUROC and AUPR value
----> 2 y_predictProb = sv_clf.predict_proba(X_test)
      3 
      4 fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
      5 roc_auc = auc(fpr, tpr)
```

1 frames

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py in _check_proba(self)
    601     def _check_proba(self):
    602         if not self.probability:
--> 603             raise AttributeError("predict_proba is not available when "
    604                                  " probability=False")
    605         if self._impl not in ('c_svc', 'nu_svc'):

AttributeError: predict_proba is not available when  probability=False
```

SEARCH STACK OVERFLOW

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

Gausian Naive Bayes

```
#using Naive Bayesian

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
# accuracy score for training data and testing data
X_train_prediction=gnb.predict(X_train)
X_training_accuracy=accuracy_score(X_train_prediction,Y_train)
```

```
X_test_prediction=gnb.predict(X_test)
X_testing_accuracy=accuracy_score(X_test_prediction,Y_test)


print('Accuracy score for training data: ',X_training_accuracy)
print('Accuracy score for testing data: ',X_testing_accuracy)
```

```
Accuracy score for training data:  0.7296511627906976
Accuracy score for testing data:  0.7558139534883721
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(gnb , X, Y, cv = kf)
result
```

```
array([0.60465116, 0.51162791, 0.6744186 , 0.72093023, 0.81395349,
       0.8372093 , 0.6744186 , 0.93023256, 0.60465116, 0.34883721])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.6720930232558139
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

k = 10
kf = KFold(n_splits=k, random_state=None)
result = cross_val_score(gnb , X_test, Y_test, cv = kf)
result
```

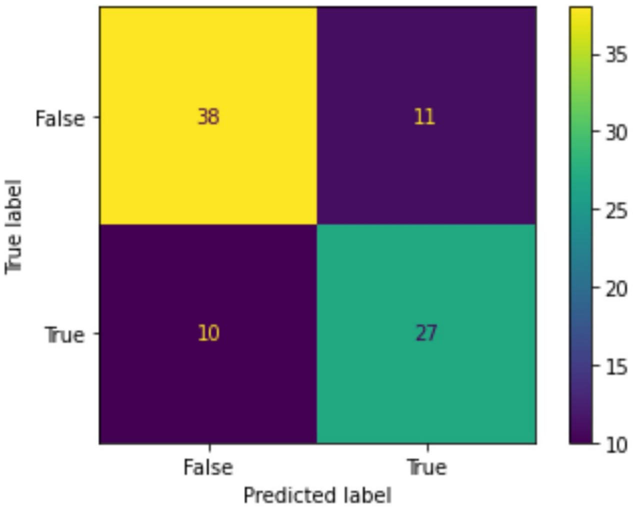
```
array([0.77777778, 0.66666667, 0.77777778, 0.44444444, 0.88888889,
       0.88888889, 0.875     , 0.625     , 0.75      , 0.625     ])
```

```
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.7319444444444445
```

```
# make predictions
predicted = gnb.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)
```

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_l
cm_display.plot()
plt.show()
```

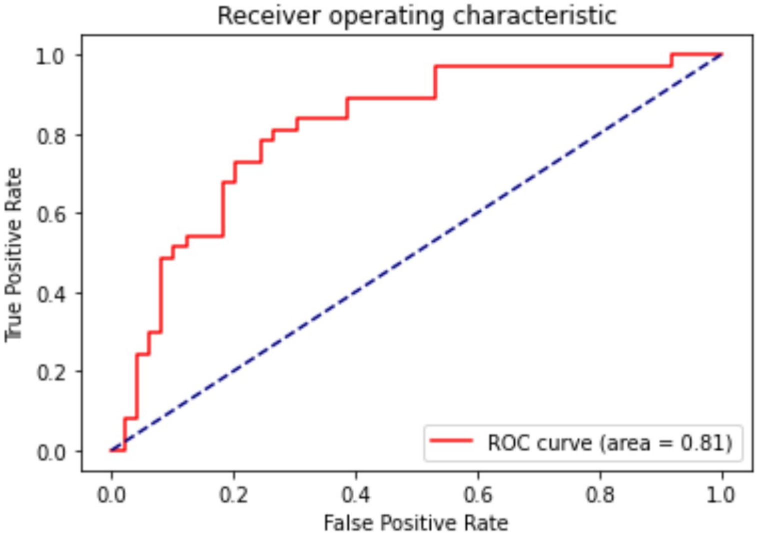


```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

sensitivity = (TP / float(TP + FN))
specificity = (TN / float(TN + FP))
ppv = (TP / float(TP + FP))
npv = (TN / float(TN + FN))

print("Sensitivity: ",sensitivity)
print("specificity: ",specificity)
print("PPV: ",ppv)
print("NPV: ",npv)
```

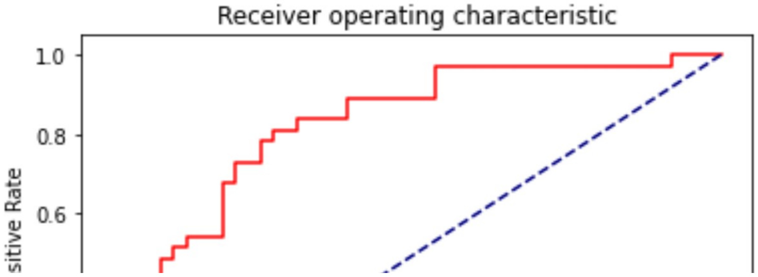
```
Sensitivity:  0.7297297297297297
specificity:  0.7755102040816326
PPV:  0.7105263157894737
NPV:  0.7916666666666666
```

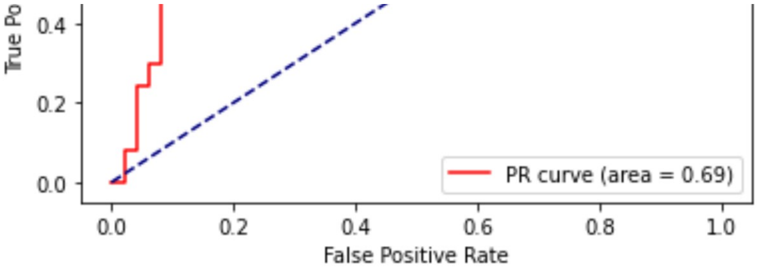
```
# AUROC and AUPR value
y_predictProb = gnb.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(Y_test, y_predictProb[::,1])
roc_auc = auc(fpr, tpr)

precision, recall, thresholds = precision_recall_curve(Y_test, y_predictProb[::,1])
area = auc(recall, precision)

print("AUROC:",roc_auc)
print("AUPR:",area)
print( AUPR: ,area)

     AUROC: 0.8135686707115278
     AUPR: 0.6947340845215407
```

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show

     <function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

plt.plot(fpr, tpr, color='red', label='PR curve (area = %0.2f)' % area)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show

     <function matplotlib.pyplot.show(*args, **kw)>
```




True Po
0.4
0.2
0.0
PR curve (area = 0.69)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate