```
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: scikit-learn[alldeps]==0.22.1 in /usr/local/lib/pytho
Requirement already satisfied: typing==3.6.6 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas==0.25.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: mxnet==1.4.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy<1.15.0,>=1.8.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.7/di
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: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
```

```
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 10:17 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','DiagnosisByCriteria'],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','TreatmentGroupBinar',
                     '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()


#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 39 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
AppendixDiameter              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
Peritonitis                   430 non-null float64
AppendixWallLayers            430 non-null float64
TissuePerfusion               430 non-null float64
AppendixOnSono                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
Kokarde                       430 non-null float64
SurroundingTissueReaction     430 non-null float64
PathLymphNodes                430 non-null float64
MesentricLymphadenitis        430 non-null float64
BowelWallThick                430 non-null float64
```

```
Ileus                          430 non-null float64
FecalImpaction                 430 non-null float64
Meteorism                      430 non-null float64
Enteritis                      430 non-null float64
TreatmentGroupBinar            430 non-null int64
PsoasSign                      430 non-null float64
Stool                          430 non-null float64
dtypes: float64(36), int64(3)
memory usage: 131.1 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.TreatmentGroupBinar,df_final[col])
        corr_df.loc[col]=[round(r,3),round(p,3)]

corr_df
```

```
Age
BMI
Height
Weight
AlvaradoScore
PediatricAppendicitisScore
AppendixDiameter
BodyTemp
WBCCount
NeutrophilPerc
CRPEntry
Sex
KetonesInUrine
ErythrocytesInUrine
WBCInUrine
Peritonitis
AppendixWallLayers
TissuePerfusion
AppendixOnSono
MigratoryPain
LowerAbdominalPainRight
ReboundTenderness
CoughingPain
Nausea
AppetiteLoss
Dysuria
FreeFluids
Kokarde
SurroundingTissueReaction
PathLymphNodes
```

```
MesentricLymphadenitis
BowelWallThick
Ileus
FecalImpaction
Meteorism
Enteritis
TreatmentGroupBinar
PsoasSign
Stool
```

	r	p
Age	-0.070	0.148
BMI	-0.088	0.070
Height	-0.070	0.146
Weight	-0.085	0.078
AlvaradoScore	0.410	0.000
PediatricAppendicitisScore	0.332	0.000
AppendixDiameter	0.485	0.000
BodyTemp	0.207	0.000
WBCCount	0.438	0.000
NeutrophilPerc	0.429	0.000
CRPEntry	0.380	0.000
Sex	0.061	0.207
KetonesInUrine	-0.104	0.031
ErythrocytesInUrine	-0.019	0.688
WBCInUrine	0.031	0.520
Peritonitis	-0.760	0.000
AppendixWallLayers	-0.424	0.000
TissuePerfusion	-0.238	0.000
AppendixOnSono	0.243	0.000
MigratoryPain	0.074	0.123
LowerAbdominalPainRight	0.056	0.251
ReboundTenderness	0.157	0.001
CoughingPain	0.102	0.034
Nausea	0.165	0.001
AppetiteLoss	0.085	0.080

AppetiteLoss	[illegible]	[illegible]
Dysuria	-0.031	0.517
FreeFluids	0.184	0.000
Kokarde	0.280	0.000
SurroundingTissueReaction	0.171	0.000
PathLymphNodes	-0.030	0.535
MesentricLymphadenitis	0.106	0.028
BowelWallThick	0.141	0.003
Ileus	0.196	0.000
FecalImpaction	-0.053	0.271
Meteorism	-0.017	0.731
Enteritis	-0.146	0.002
TreatmentGroupBinar	1.000	0.000
PsoasSign	-0.075	0.120
Stool	-0.063	0.194

```
df_final.shape
```

```
(430, 39)
```

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

```
0    265
1    165
Name: TreatmentGroupBinar, dtype: int64
```

1 = yes, 0 = NO

[ ] ↳ *9 cells hidden*

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:  0.9156976744186046
Accuracy score for testing data:  0.8488372093023255
```

```
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
    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: 0.8976470588235295
```

```
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
```

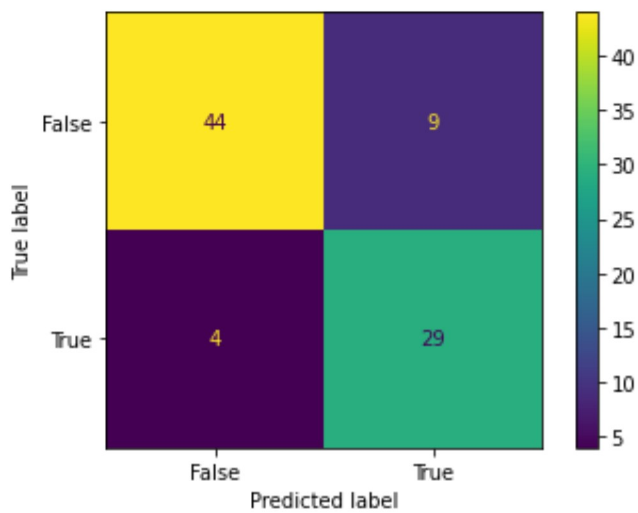
```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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.

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
```

```
/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: 0.8277777777777777
```

```
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))
```

```
npv = (TN / float(TN + FN))

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

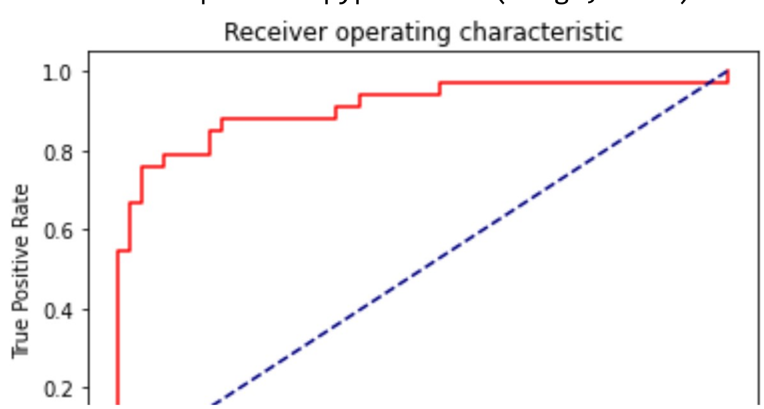
```
Sensitivity:  0.8787878787878788
specificity:  0.8301886792452831
PPV:  0.7631578947368421
NPV:  0.9166666666666666
```

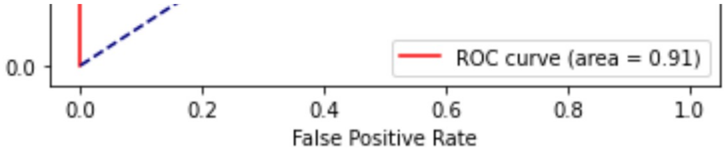
```
# AUROC and AUPR value
from sklearn.metrics import auc, roc_curve, precision_recall_curve

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: 0.9085191538021727
AUPR: 0.9045489361357539
```

```
# 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
```

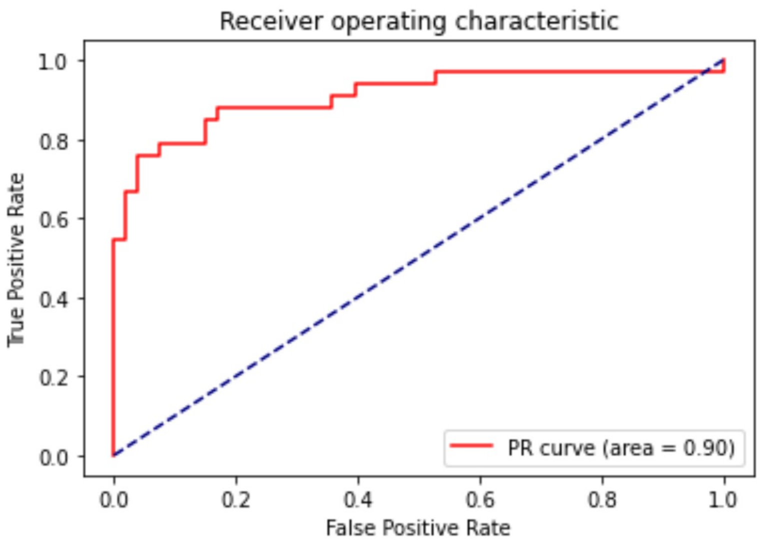
```
<function matplotlib.pyplot.show(*args, **kw)>
```






```
# 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:  0.9941860465116279
Accuracy score for testing data:  0.8488372093023255
```

```
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([0.88571429, 0.94285714, 0.88571429, 0.94285714, 0.73529412,
       0.82352941, 0.88235294, 0.82352941, 0.94117647, 0.79411765])
```

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

```
Avg accuracy: 0.8657142857142859
```

```
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
```

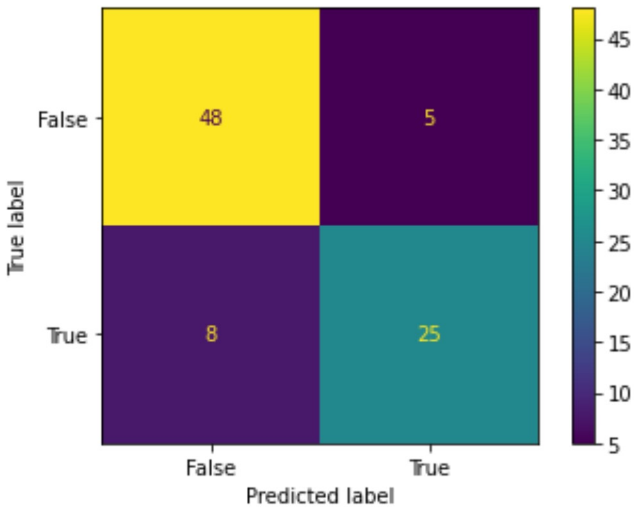
```
array([0.88888889, 0.66666667, 0.77777778, 0.55555556, 0.66666667,
       0.88888889, 0.875     , 0.75      , 0.75      , 0.75      ])
```

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

```
Avg accuracy: 0.7569444444444444
```

```
# make predictions
```

```
# 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:  0.7575757575757576
specificity:  0.9056603773584906
PPV:  0.8333333333333334
NPV:  0.8571428571428571
```

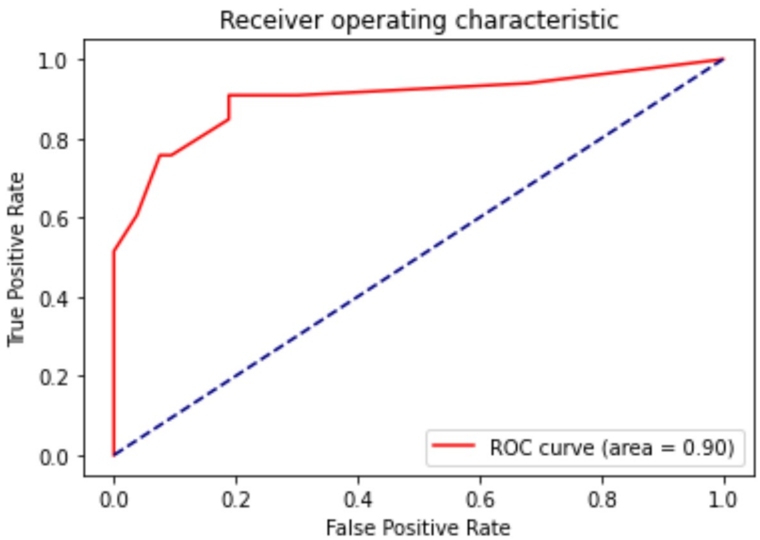
```
y_predictProb = forest.predict_proba(X_test)

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: 0.8996569468267581
AUPR: 0.8950751227519758
```

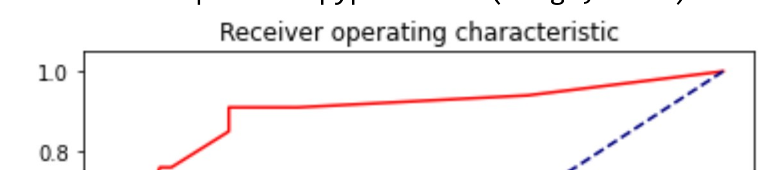
```
# AURoc graph

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
```

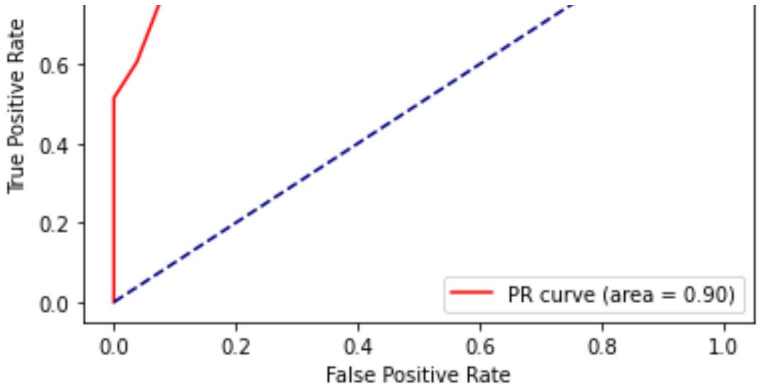
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
# 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:  0.8372093023255814
```

```
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([0.77142857, 0.94285714, 0.82857143, 0.91428571, 0.85294118,
       0.76470588, 0.82352941, 0.79411765, 0.91176471, 0.91176471])
```

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

```
Avg accuracy: 0.8515966386554622
```

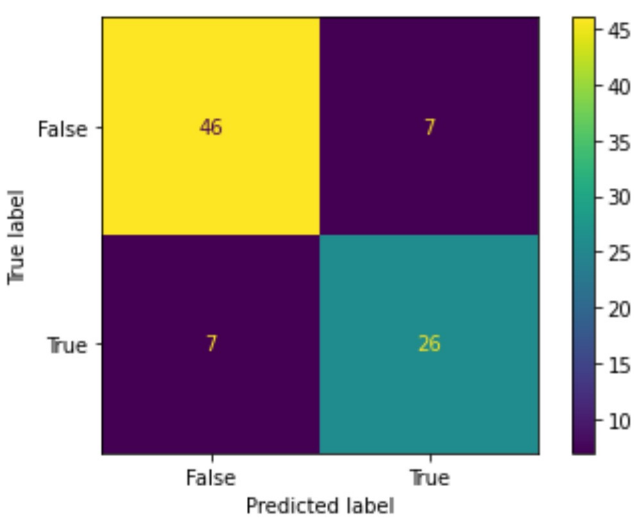
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
array([1.        , 0.77777778, 0.77777778, 0.88888889, 0.88888889,
       0.77777778, 0.625     , 0.875     , 0.625     , 0.75      ])
```

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

```
Avg accuracy: 0.7986111111111111
```

```
# make predictions
predicted = dclf.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)
```

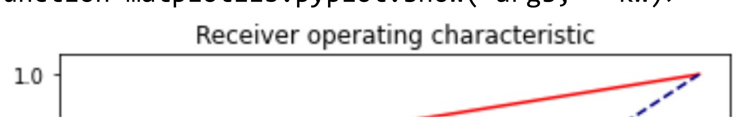
```
Sensitivity:  0.7878787878787878
specificity:  0.8679245283018868
PPV:  0.7878787878787878
NPV:  0.8679245283018868
```

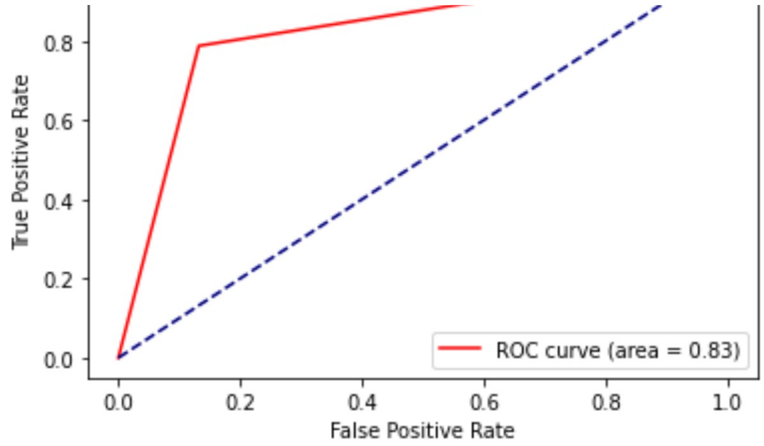
```
# AUROC and AUPR value
y_predictProb = dclf.predict_proba(X_test)

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: 0.8279016580903373
AUPR: 0.8285764622973926
```

```
# 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
```

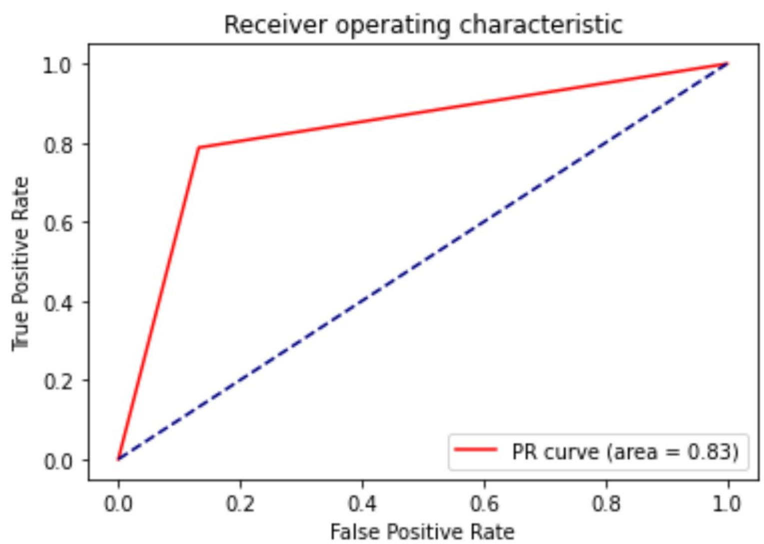
```
<function matplotlib.pyplot.show(*args, **kw)>
```






```
# 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 Boost

```
#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,
```

```
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:  0.936046511627907
Accuracy score for testing data:  0.8604651162790697
```

```
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([0.88571429, 0.94285714, 0.91428571, 0.94285714, 0.88235294,
       0.82352941, 0.85294118, 0.82352941, 0.94117647, 0.88235294])
```

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

```
Avg accuracy: 0.8891596638655462
```

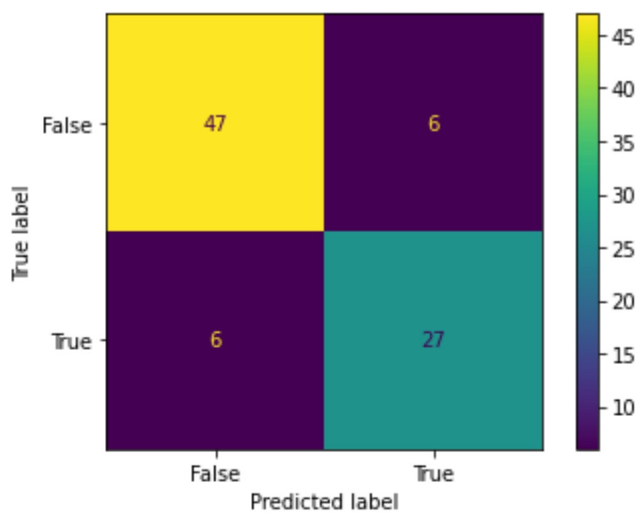
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
array([1.        , 0.66666667, 0.88888889, 0.88888889, 0.88888889,
       0.77777778, 0.75      , 0.875     , 0.75      , 0.875     ])
```

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

```
Avg accuracy: 0.836111111111111
```

```
# 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:  0.8181818181818182
specificity:  0.8867924528301887
```

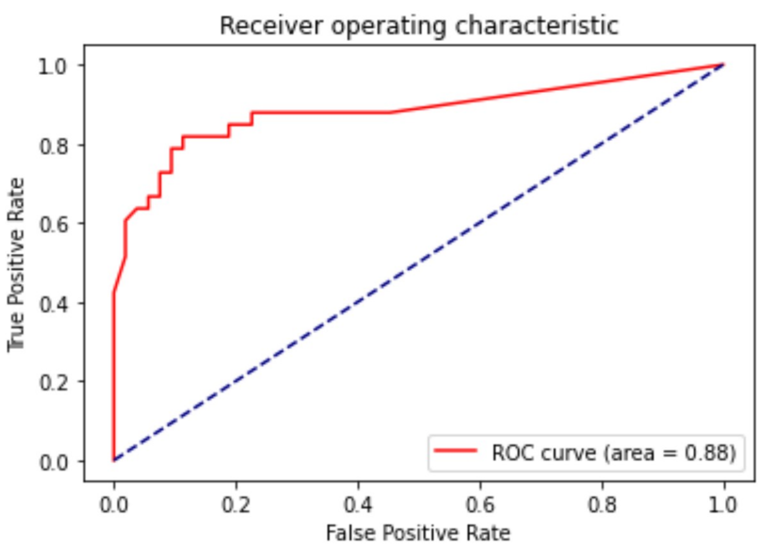
```
PPV:  0.8181818181818182
NPV:  0.8867924528301887
```

```
# 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: 0.8805031446540881
AUPR: 0.8805018826634343
```

```
# 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
```

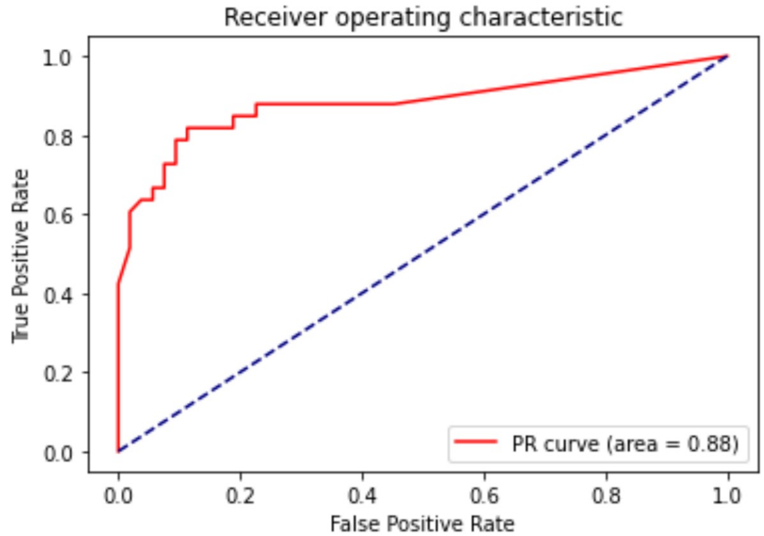
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
# 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)>
```



XGBoost

```
#using XGBClassifier
from xgboost import XGBClassifier
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:  0.936046511627907
Accuracy score for testing data:  0.8488372093023255
```

```
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([0.91428571, 1.        , 0.91428571, 0.94285714, 0.88235294,
       0.82352941, 0.82352941, 0.79411765, 0.97058824, 0.88235294])
```

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

```
Avg accuracy: 0.8947899159663866
```

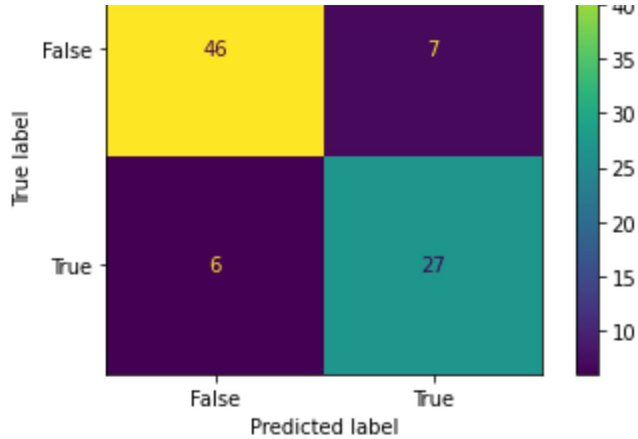
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

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

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

```
Avg accuracy: 0.8375
```

```
# 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:  0.8181818181818182
specificity:  0.8679245283018868
PPV:  0.7941176470588235
NPV:  0.8846153846153846
```

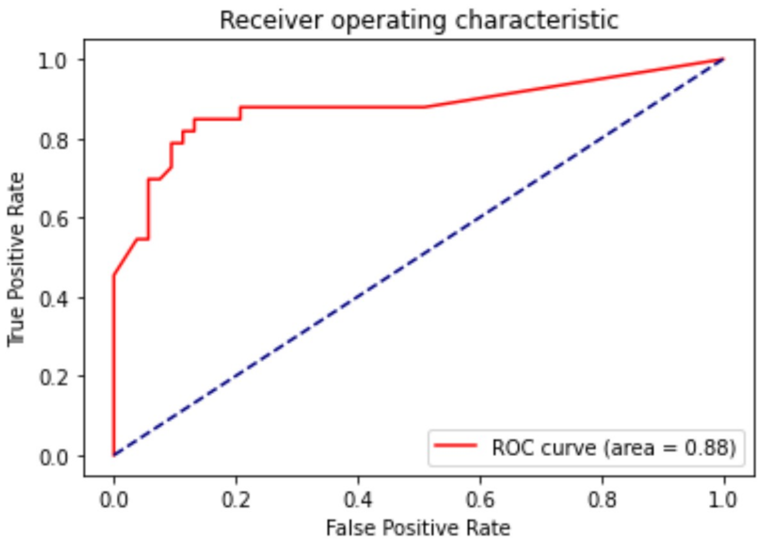
```
# AUROC and AUPR value
y_predictProb = xgb_clf.predict_proba(X_test)

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: 0.876214979988565
AUPR: 0.8742991485236645
```

```
# AURoc graph
```

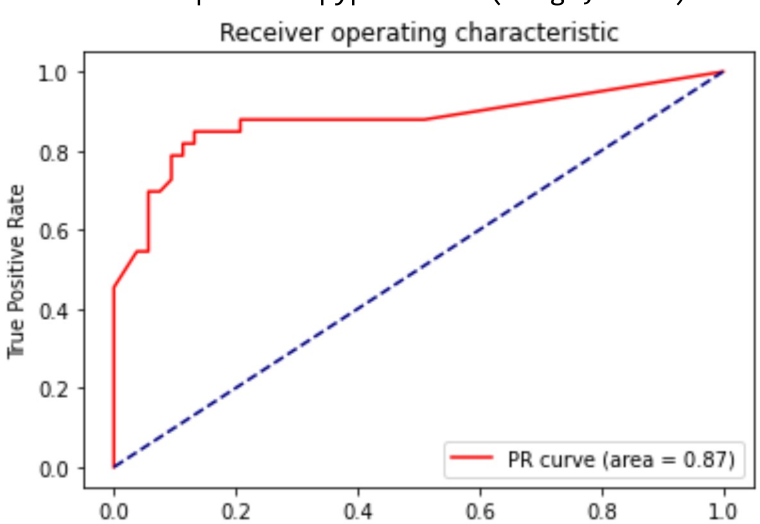
```
plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

```
<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)>
```



False Positive Rate

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:  0.7063953488372093
Accuracy score for testing data:  0.7093023255813954
```

```
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([0.77142857, 0.8       , 0.74285714, 0.6       , 0.64705882,
       0.61764706, 0.70588235, 0.73529412, 0.61764706, 0.70588235])
```

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

```
Avg accuracy: 0.6943697478991597
```

```
from sklearn.model_selection import cross_val_score
```

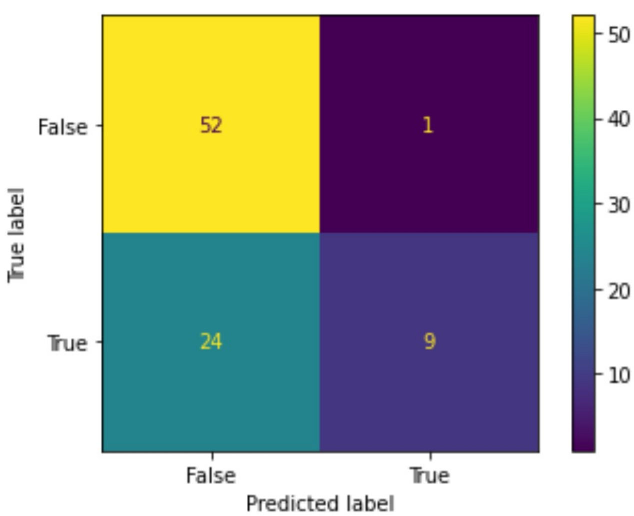
```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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, 0.77777778, 0.66666667, 0.44444444, 0.55555556,
           0.55555556, 0.875     , 0.875     , 0.5       , 0.75      ])
```

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

```
    Avg accuracy: 0.6777777777777778
```

```
# 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()
```



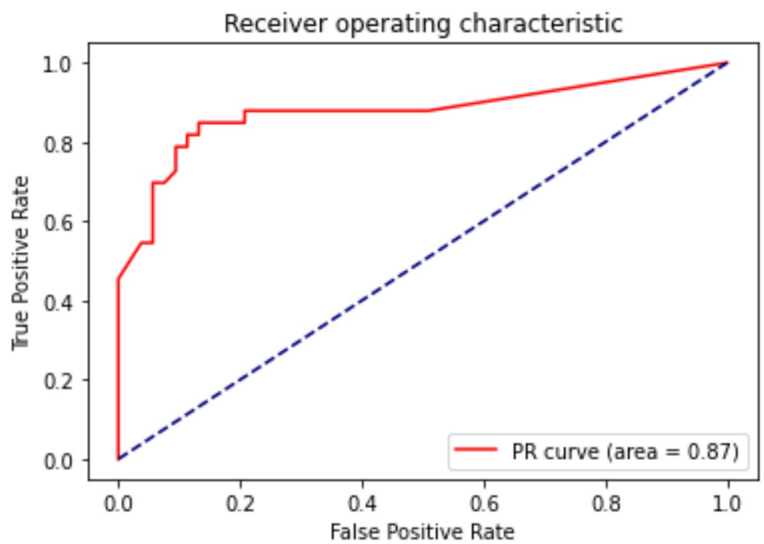

```
TN = confusion_matrix[0][0]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]
FP = confusion_matrix[0][1]

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:  0.2727272727272727
specificity:  0.9811320754716981
PPV:  0.9
NPV:  0.6842105263157895
```

```
# 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)>
```




```
# 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)
```

```
AttributeError                            Traceback (most recent call last)
<ipython-input-117-289267775586> 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:  0.8924418604651163
Accuracy score for testing data:  0.8837209302325582
```

```
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_train, Y_train, cv = kf)
result
```

```
array([0.94285714, 0.94285714, 0.94285714, 0.8       , 0.82352941,
       0.85294118, 0.94117647, 0.82352941, 0.88235294, 0.82352941])
```

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

```
Avg accuracy: 0.8775630252100841
```

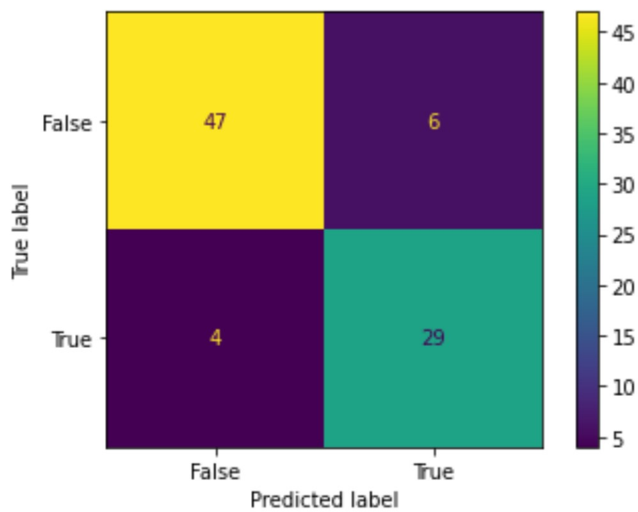
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
array([0.44444444, 0.77777778, 0.55555556, 0.88888889, 0.88888889,
       0.77777778, 0.875     , 0.5       , 0.625     , 0.625     ])
```

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

```
Avg accuracy: 0.6958333333333334
```

```
# 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:  0.8787878787878788
specificity:  0.8867924528301887
PPV:  0.8285714285714286
NPV:  0.9215686274509803
```

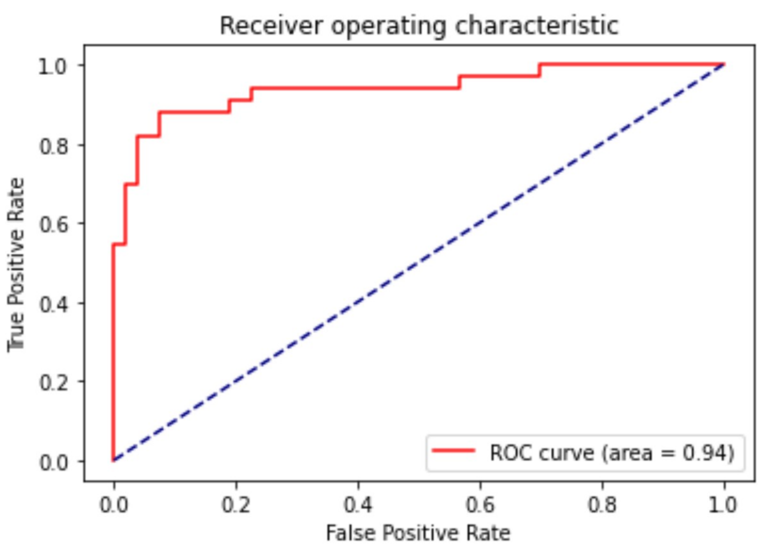
```
# AUROC and AUPR value
y_predictProb = gnb.predict_proba(X_test)
```

```
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)
```

```
AUROC: 0.9371069182389938
AUPR: 0.9291949217652111
```

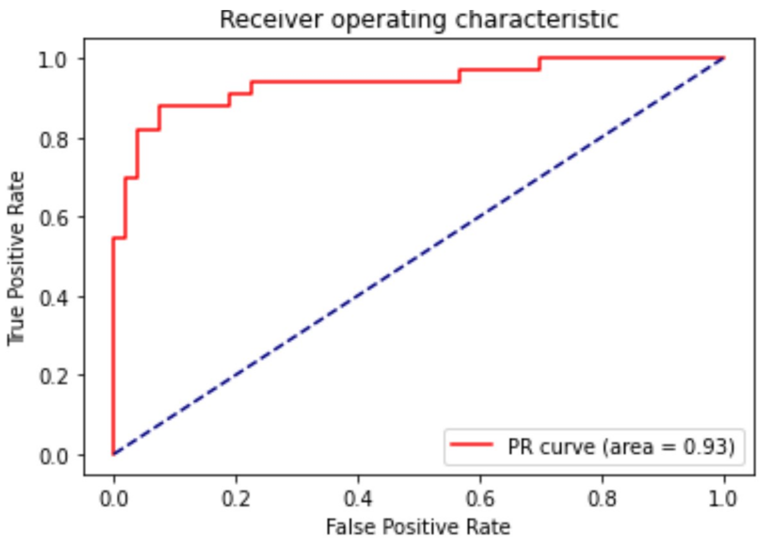
```
# AURoc graph

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)>
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
PR curve (area = 0.93)
0.0
0.2
0.4
0.6
0.8
1.0