

```

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 (0.1.0)
Requirement already satisfied: typing==3.6.6 in /usr/local/lib/python3.7/dist-packages (3.6.6)
Requirement already satisfied: pandas==0.25.3 in /usr/local/lib/python3.7/dist-packages (0.25.3)
Requirement already satisfied: mxnet==1.4.0 in /usr/local/lib/python3.7/dist-packages (1.4.0)
Requirement already satisfied: scikit-learn[alldeps]==0.22.1 in /usr/local/lib/python3.7/dist-packages (0.22.1)
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (0.8.1)
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.7/dist-packages (2.25.1)
Requirement already satisfied: numpy<1.15.0,>=1.8.2 in /usr/local/lib/python3.7/dist-packages (1.19.5)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.7/dist-packages (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (2018.9)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (0.14.0)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-packages (1.5.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (1.16.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (2018.11.29)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (1.25.11)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (3.0.4)

```

```

import datawig

```

```

path = "/content/app_data.csv"

```

```

df = pd.read_csv(path)
df

```

	Age	BMI	Sex	Height	Weight	AlvaradoScore	PediatricAppendicitis
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 9:48 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(['DiagnosisByCriteria','TreatmentGroupBinar'],axis=1,inplace=True)
```

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

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

```
#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','AppendicitisComplications',
                          '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
AppendicitisComplications 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.AppendicitisComplications,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

```

MesentericLymphadenitis
 BowelWallThick
 Ileus
 FecalImpaction
 Meteorism
 Enteritis
 AppendicitisComplications
 PsoasSign
 Stool

	r	p
Age	-0.099	0.041
BMI	-0.069	0.153
Height	-0.084	0.082
Weight	-0.071	0.144
AlvaradoScore	0.279	0.000
PediatricAppendicitisScore	0.255	0.000
AppendixDiameter	0.312	0.000
BodyTemp	0.285	0.000
WBCCount	0.327	0.000

```
df_final.shape
```

```
(430, 39)
```

Sex	-0.020	0.677
------------	--------	-------

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

```
0    379
```

```
1     51
```

```
Name: AppendicitisComplications, dtype: int64
```

Peritonitis	-0.458	0.000
--------------------	--------	-------

1 = yes, 0 = NO

```
no = df_final[df_final.AppendicitisComplications==0]
```

```
yes = df_final[df_final.AppendicitisComplications==1]
```

LowerAbdominalPainRight	0.061	0.205
--------------------------------	-------	-------

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

```
print(yes.shape)
```

```
(379, 39)
```

```
(51, 39)
```

Anastatal ...	0.145	0.000
----------------------	-------	-------

```
#spliting the data for training and testing

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

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, 38)
(344, 38)
(86, 38)

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

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

N_estimator_Random Forest classifier

```
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)

model_score2 = forest.score(X_test, Y_test)
model_score1 = forest.score(X_train, Y_train)
print(model_score1)
print(model_score2)

0.9854651162790697
0.9186046511627907
```

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:  0.9273255813953488
Accuracy score for testing data:  0.9302325581395349
```

```
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: 0.892436974789916

```
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
 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.

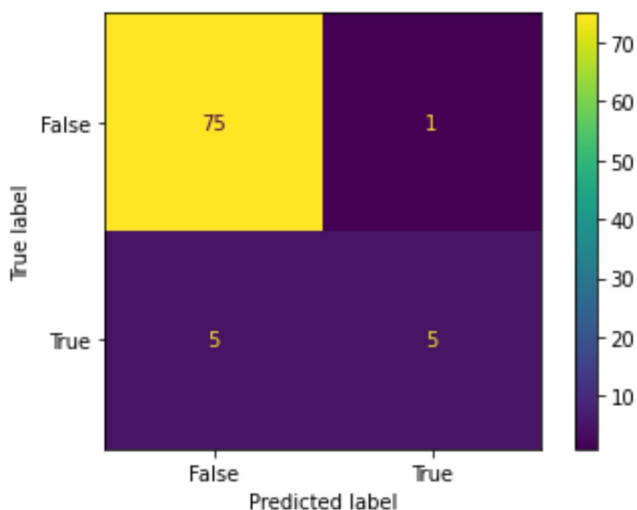
```
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.9319444444444445
```

```
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:  0.5
    specificity:  0.9868421052631579
    PPV:  0.8333333333333334
    NPV:  0.9375

# 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)

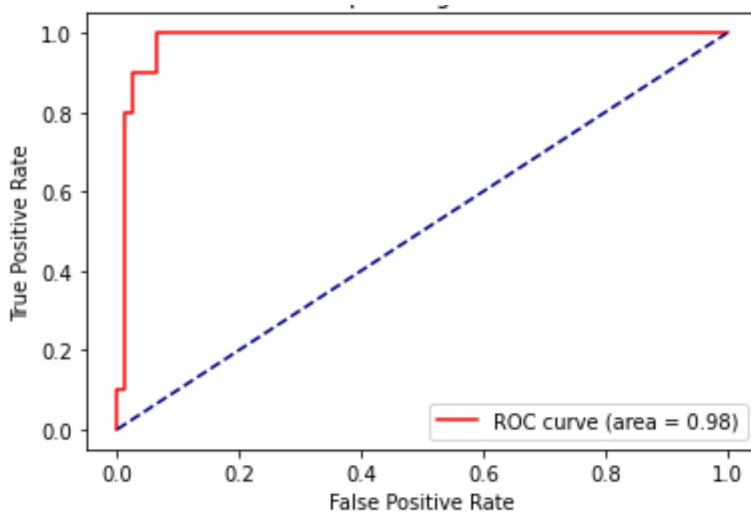
print("AUROC:",roc_auc)
print("AUPR:",area)

    AUROC: 0.9815789473684211
    AUPR: 0.7940440115440115

# 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
```

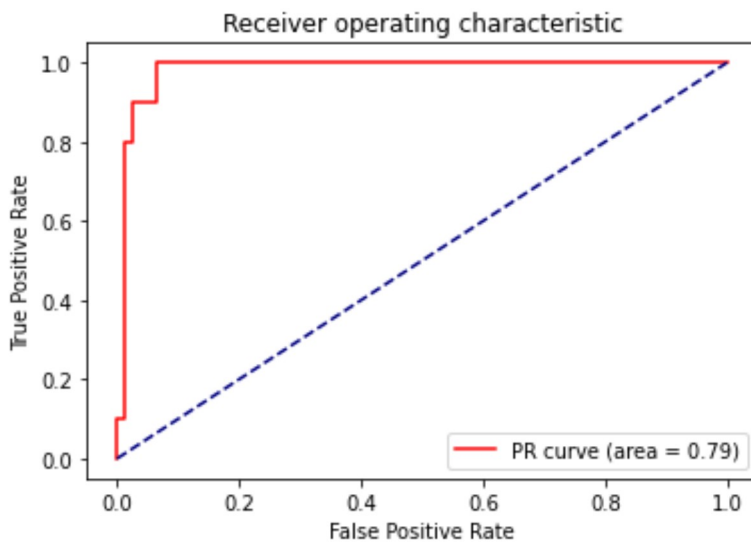
```
<function matplotlib.pyplot.show(*args, **kw)>
    Receiver operating characteristic
```



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)
```

```
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:  0.9854651162790697
Accuracy score for testing data:  0.9186046511627907
```

```
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([0.94285714, 0.85714286, 0.88571429, 0.82857143, 0.94117647,
       0.82352941, 0.91176471, 0.85294118, 0.94117647, 0.88235294])
```

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

```
Avg accuracy: 0.8867226890756303
```

```
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([1.          , 0.77777778, 1.          , 0.77777778, 0.77777778,
       1.          , 1.          , 1.          , 0.875        , 1.          ])
```

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

```
Avg accuracy: 0.9208333333333332
```

```
# make predictions
```

```
predicted = forest.predict(X_test)
```

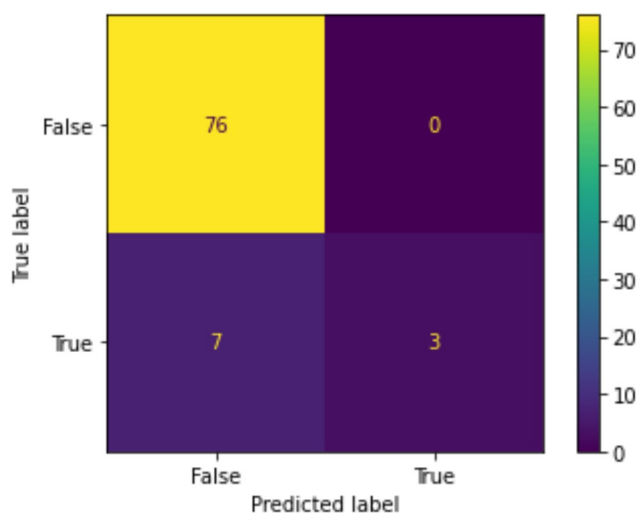
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)
```

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_1
```

```
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: 0.3
```

```
specificity: 1.0
```

```
ppv: 1.0
```

```
PPV: 1.0
NPV: 0.9156626506024096
```

```
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)

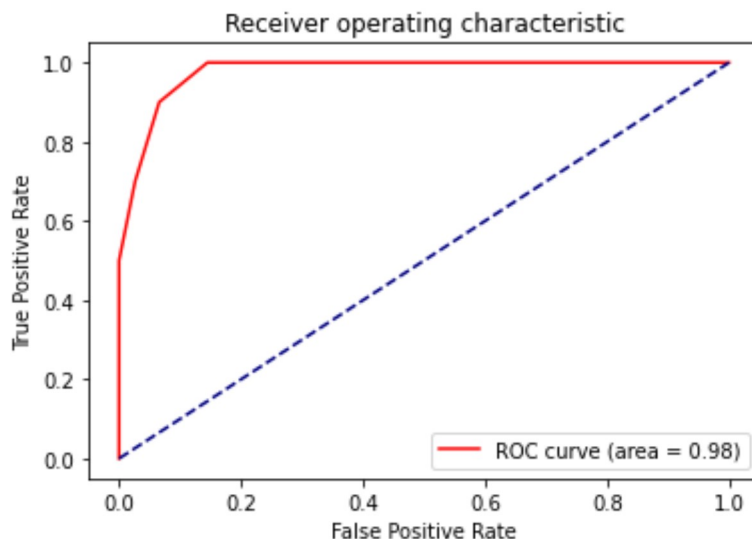
print("AUROC:",roc_auc)
print("AUPR:",area)

AUROC: 0.9776315789473684
AUPR: 0.8757936507936507
```

```
# 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)>
```



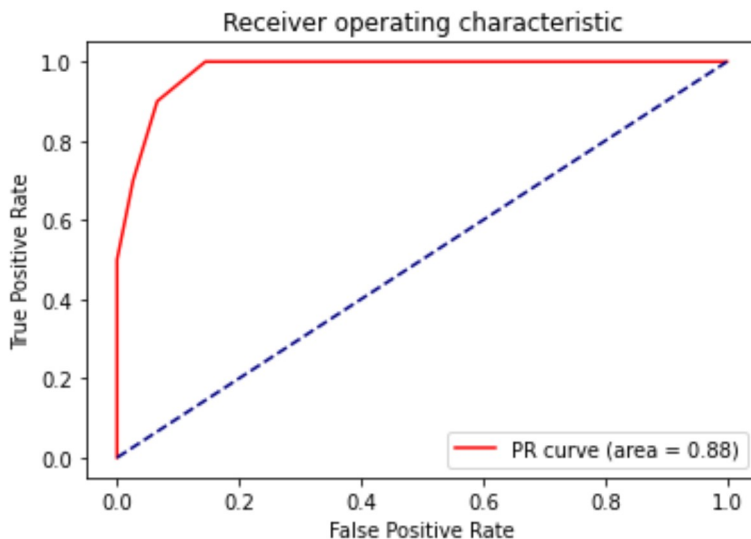
```
# AUPR graph
```

```
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.xlabel('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:  0.9186046511627907
```

```
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([0.85714286, 0.82857143, 0.85714286, 0.8          , 0.82352941,
       0.79411765, 0.82352941, 0.76470588, 0.91176471, 0.85294118])

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

Avg accuracy: 0.8313445378151261
```

```
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([0.88888889, 0.88888889, 0.77777778, 0.88888889, 0.88888889,
       1.          , 1.          , 1.          , 0.875      , 1.          ])

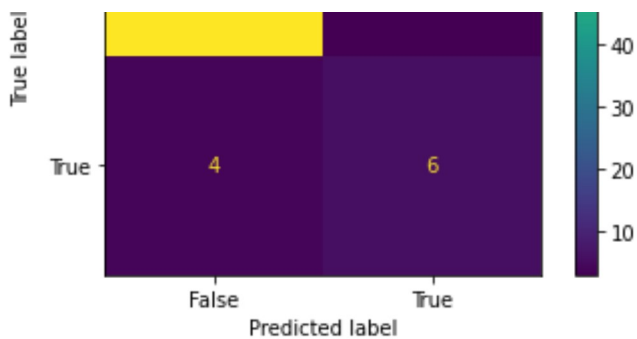
print("Avg accuracy: {}".format(result.mean()))

Avg accuracy: 0.9208333333333332
```

```
# 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_1
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: 0.6
specificity: 0.9605263157894737
PPV: 0.6666666666666666
NPV: 0.948051948051948
```

```
# 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)
```

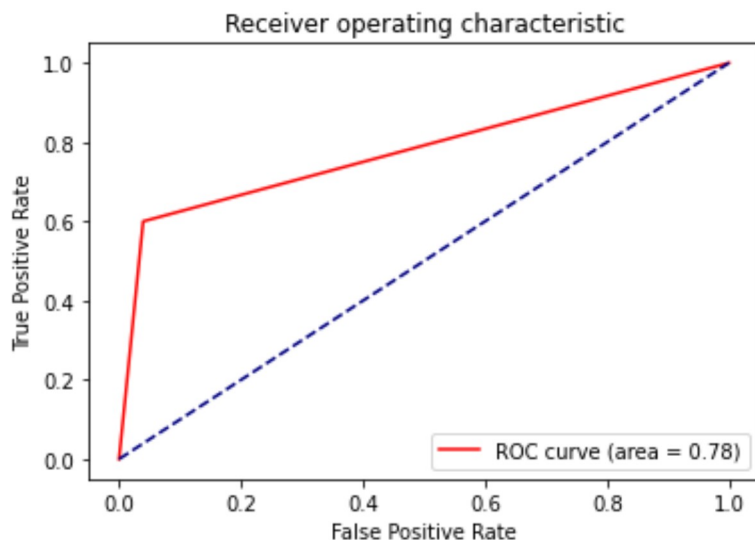
```
AUROC: 0.7802631578947369
AUPR: 0.6565891472868217
```

```
# 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
```

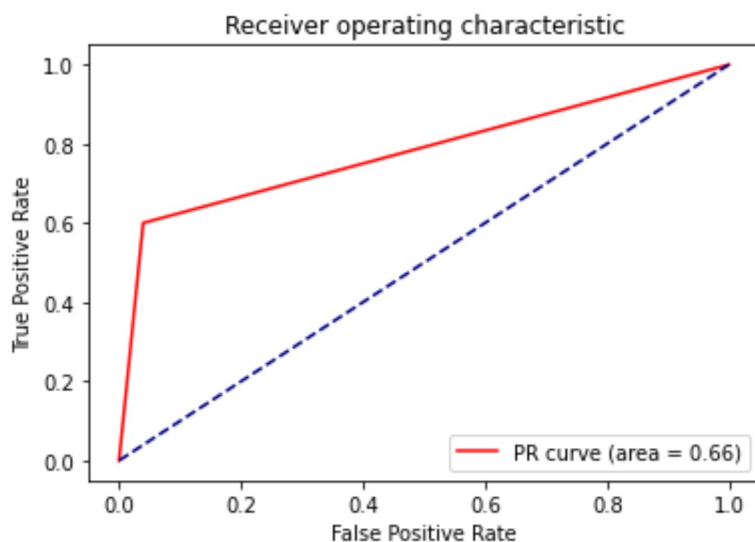
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# 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:  0.9563953488372093
    Accuracy score for testing data:  0.9186046511627907

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([0.94285714, 0.88571429, 0.91428571, 0.82857143, 0.91176471,
           0.85294118, 0.82352941, 0.88235294, 0.97058824, 0.91176471])

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

    Avg accuracy: 0.892436974789916
```

```

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([0.88888889, 0.88888889, 1.          , 0.88888889, 0.88888889,
        1.          , 1.          , 1.          , 0.875      , 1.          ])

```

```

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

```

```

Avg accuracy: 0.9430555555555555

```

```

# make predictions

```

```

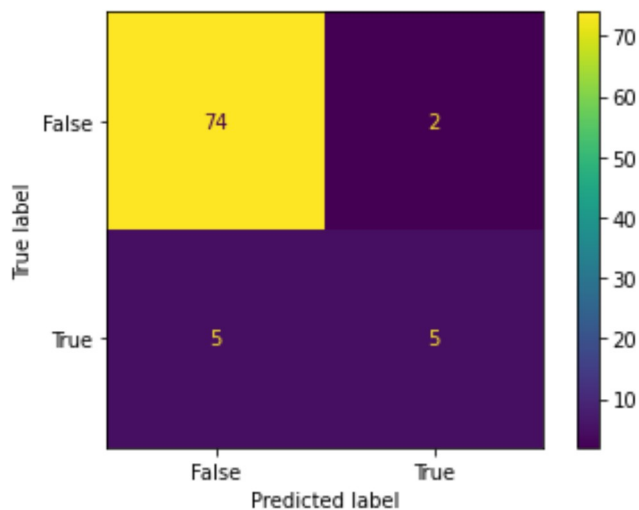
predicted = gdb.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix = metrics.confusion_matrix(Y_test,predicted)

```

```

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels=[
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))

```

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

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

```
Sensitivity:  0.5
specificity:  0.9736842105263158
PPV:  0.7142857142857143
NPV:  0.9367088607594937
```

```
# 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)
```

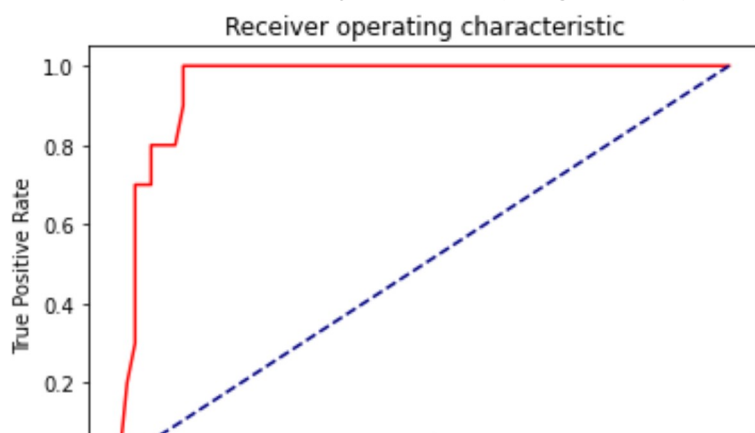
```
print("AUROC:",roc_auc)
print("AUPR:",area)
```

```
AUROC: 0.9605263157894737
AUPR: 0.6845212630506748
```

```
# 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)>
```

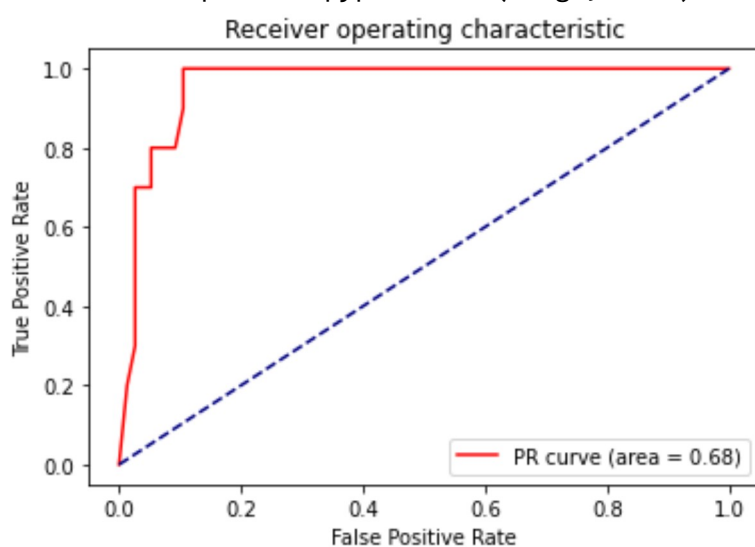




```
# AUPR graph
```

```
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)>
```



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:  0.9534883720930233
    Accuracy score for testing data:  0.9186046511627907

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, 0.91428571, 0.94285714, 0.85714286, 0.91176471,
           0.91176471, 0.88235294, 0.94117647, 0.97058824, 0.91176471])

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

    Avg accuracy: 0.915798319327731

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([0.88888889, 1.          , 1.          , 0.88888889, 0.88888889,
           1.          , 1.          , 1.          , 0.875       , 1.          ])

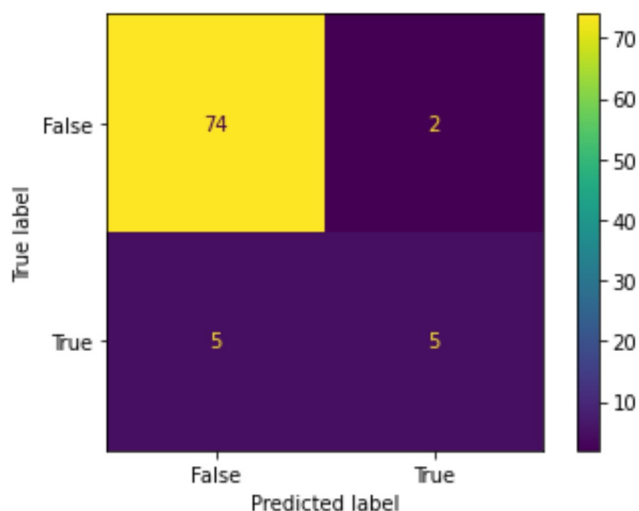
print("Avg accuracy: {}".format(result.mean()))

    Avg accuracy: 0.9541666666666666

# make predictions
predicted = xgb_clf.predict(Y_test)
```

```
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_1
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:  0.5
specificity:  0.9736842105263158
PPV:  0.7142857142857143
NPV:  0.9367088607594937
```

```
# 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: 0.9539473684210527
```

```
AUPR: 0.6617540792540793
```

```
# 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')
```

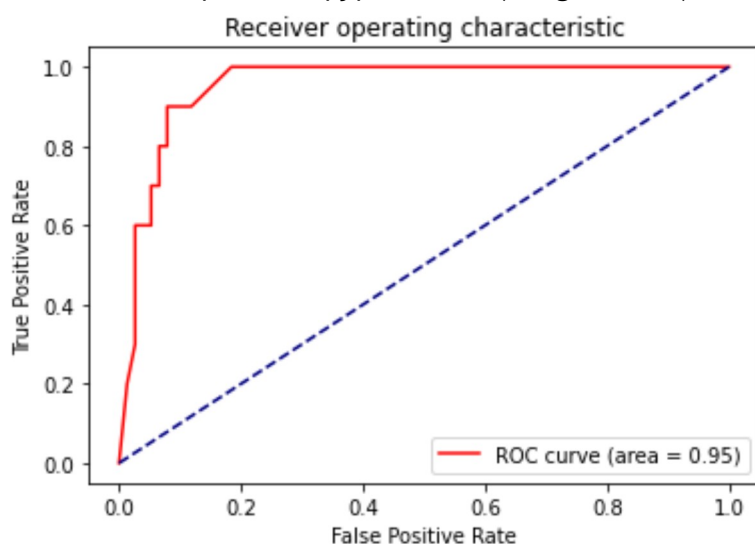
```
plt.ylabel('True Positive Rate')
```

```
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')
```

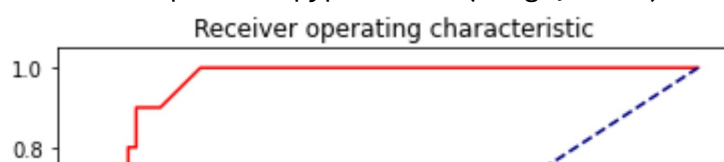
```
plt.ylabel('True Positive Rate')
```

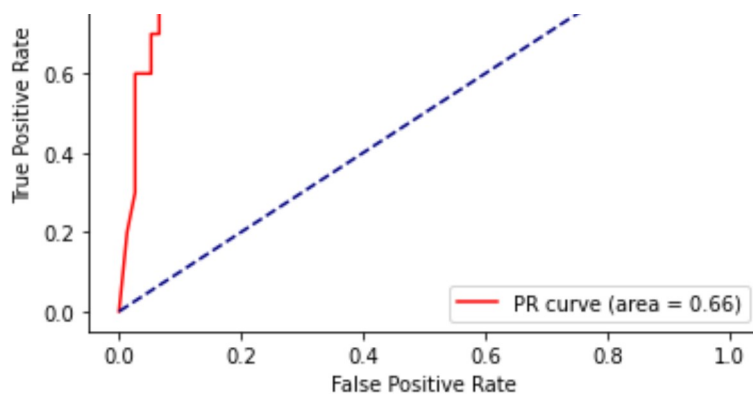
```
plt.title('Receiver operating characteristic')
```

```
plt.legend(loc="lower right")
```

```
plt.show
```

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





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:  0.9186046511627907
Accuracy score for testing data:  0.9069767441860465

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([0.91428571, 0.91428571, 0.91428571, 0.88571429, 0.91176471,
       0.88235294, 0.91176471, 0.91176471, 0.97058824, 0.94117647])
```

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

```
Avg accuracy: 0.915798319327731
```

```
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
```

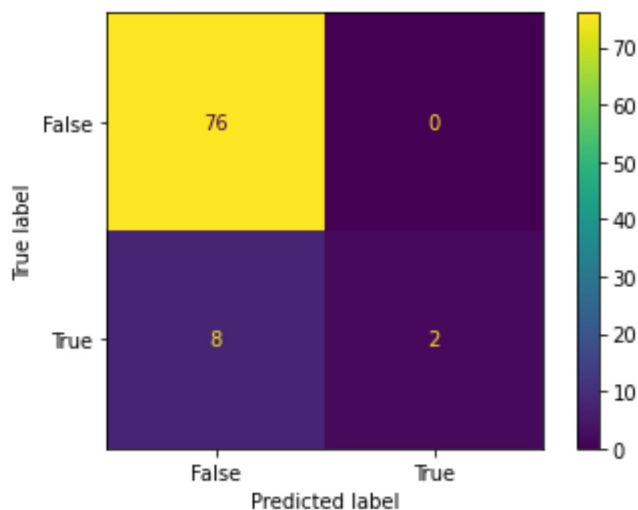
```
array([0.88888889, 1.          , 1.          , 0.88888889, 0.88888889,
       1.          , 1.          , 1.          , 0.875       , 0.875       ])
```

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

```
Avg accuracy: 0.9416666666666667
```

```
# 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_labels=[
True, False])
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:  0.2
specificity:  1.0
PPV:  1.0
NPV:  0.9047619047619048

# 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-116-289267775586> 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
```

Gaussian 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:  0.7994186046511628
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(gnb , X_train, Y_train, cv = kf)
result

array([0.77142857, 0.71428571, 0.71428571, 0.94285714, 0.79411765,
       0.76470588, 0.85294118, 0.79411765, 0.85294118, 0.76470588])

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

Avg accuracy: 0.7966386554621848
```

```
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

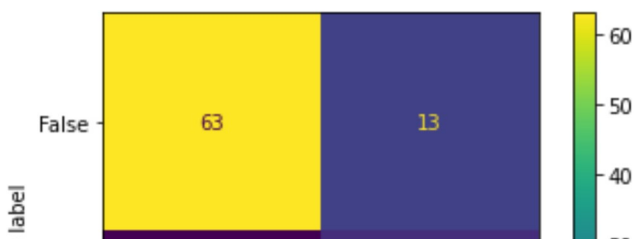
array([0.77777778, 0.77777778, 0.55555556, 0.77777778, 0.55555556,
       0.88888889, 1.          , 0.875       , 0.75        , 0.875       ])

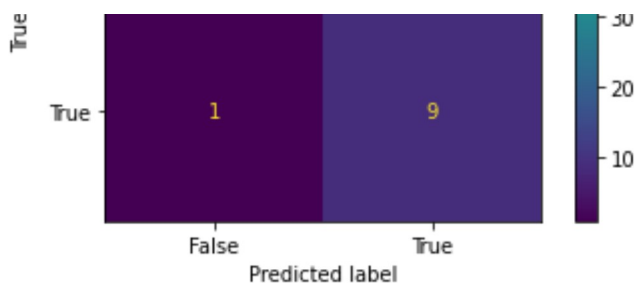
print("Avg accuracy: {}".format(result.mean()))

Avg accuracy: 0.7833333333333334
```

```
# 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_1
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:  0.9
specificity:  0.8289473684210527
PPV:  0.4090909090909091
NPV:  0.984375
```

```
# 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)
```

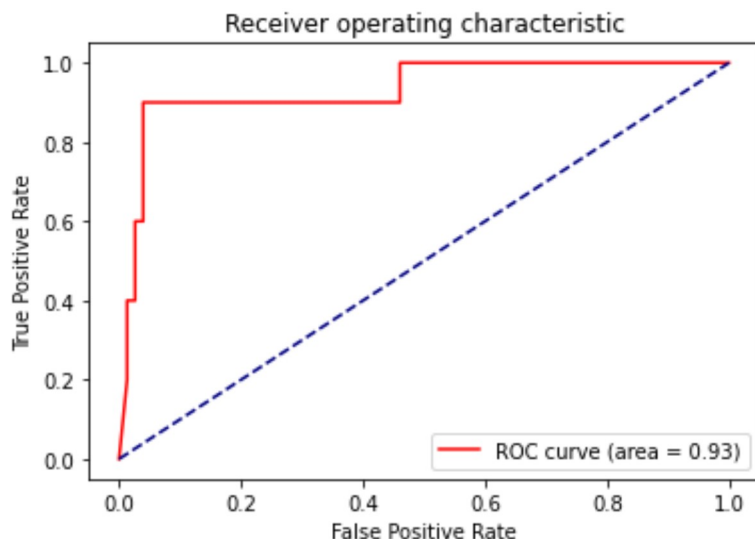
```
AUROC: 0.9328947368421053
AUPR: 0.6921608946608946
```

```
# 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.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % area)
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

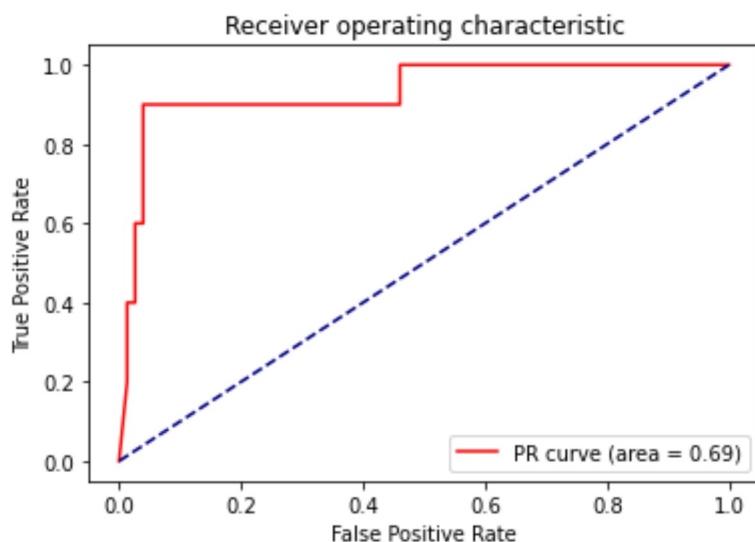
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# 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)>
```



[Colab paid products](#) - [Cancel contracts here](#)