```
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.
Collecting scikit-learn[alldeps]==0.22.1
  Using cached scikit_learn-0.22.1-cp37-cp37m-manylinux1_x86_64.whl (7.0 MB)
Requirement already satisfied: mxnet==1.4.0 in /usr/local/lib/python3.7/dist-package
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: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy<1.15.0,>=1.8.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
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: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.0.2
    Uninstalling scikit-learn-1.0.2:
      Successfully uninstalled scikit-learn-1.0.2
ERROR: pip's dependency resolver does not currently take into account all the packag
yellowbrick 1.5 requires numpy>=1.16.0, but you have numpy 1.14.6 which is incompati
yellowbrick 1.5 requires scikit-learn>=1.0.0, but you have scikit-learn 0.22.1 which
librosa 0.8.1 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatibl
kapre 0.3.7 requires numpy>=1.18.5, but you have numpy 1.14.6 which is incompatible.
imbalanced-learn 0.8.1 requires scikit-learn>=0.24, but you have scikit-learn 0.22.1
Successfully installed scikit-learn-0.22.1
WARNING: The following packages were previously imported in this runtime:
  [sklearn]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
import datawig
```

✓ 0s completed at 7:21 AM

```
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
 ...       ...        ...     ...     ...     ...            ...
425  12.147844  22.292563    male   166.5    61.8              5
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)

# Ultrasound
df.drop(['AppendixOnSono','AppendixDiameter','AppendixWallLayers','Kokarde','TissuePerfusi
        'BowelWallThick','Ileus','Enteritis'],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','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 30 columns):
Age                              430 non-null float64
BMI                              430 non-null float64
Height                           430 non-null float64
Weight                           430 non-null float64
AlvaradoScore                    430 non-null float64
PediatricAppendicitisScore       430 non-null float64
BodyTemp                         430 non-null float64
WBCCount                         430 non-null float64
```

```
NeutrophilPerc              430 non-null float64
CRPEntry                    430 non-null float64
Sex                         430 non-null float64
KetonesInUrine              430 non-null float64
ErythrocytesInUrine         430 non-null float64
WBCInUrine                  430 non-null float64
Peritonitis                 430 non-null float64
MigratoryPain               430 non-null int64
LowerAbdominalPainRight     430 non-null float64
ReboundTenderness           430 non-null float64
CoughingPain                430 non-null float64
Nausea                      430 non-null int64
AppetiteLoss                430 non-null float64
Dysuria                     430 non-null float64
FreeFluids                  430 non-null float64
PathLymphNodes              430 non-null float64
MesentricLymphadenitis      430 non-null float64
FecalImpaction              430 non-null float64
Meteorism                   430 non-null float64
AppendicitisComplications   430 non-null int64
PsoasSign                   430 non-null float64
Stool                       430 non-null float64
dtypes: float64(27), int64(3)
memory usage: 100.9 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
BodyTemp
WBCCount
NeutrophilPerc
CRPEntry
Sex
KetonesInUrine
ErythrocytesInUrine
WBCInUrine
Peritonitis
```

```
MigratoryPain
LowerAbdominalPainRight
ReboundTenderness
CoughingPain
Nausea
AppetiteLoss
Dysuria
FreeFluids
PathLymphNodes
MesentricLymphadenitis
FecalImpaction
Meteorism
AppendicitisComplications
PsoasSign
Stool
```

	r	p
Age	-0.098	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
BodyTemp	0.285	0.000
WBCCount	0.327	0.000
NeutrophilPerc	0.257	0.000
CRPEntry	0.616	0.000
Sex	-0.020	0.677
KetonesInUrine	-0.107	0.026
ErythrocytesInUrine	-0.189	0.000
WBCInUrine	-0.048	0.316
Peritonitis	-0.458	0.000
MigratoryPain	0.065	0.177
LowerAbdominalPainRight	-0.061	0.205
ReboundTenderness	0.069	0.152
CoughingPain	0.053	0.277
Nausea	0.207	0.000
AppetiteLoss	0.145	0.003

Dysuria	0.013	0.792
FreeFluids	0.112	0.021
PathLymphNodes	-0.040	0.403
MesentricLymphadenitis	0.006	0.901
FecalImpaction	0.049	0.311
Meteorism	0.013	0.794
AppendicitisComplications	1.000	0.000
PsoasSign	-0.084	0.082
Stool	-0.112	0.021

```
df_final.shape
```

```
(430, 30)
```

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

```
0    379
1     51
Name: AppendicitisComplications, dtype: int64
```

1 = yes, 0 = NO

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

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

```
(379, 30)
(51, 30)
```

```
#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, 29)
(344, 29)
(86, 29)
```

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

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

SMOTE techniques

```
import platform; print(platform.platform())
import sys; print("Python", sys.version)
import numpy; print("NumPy", numpy.__version__)
import scipy; print("SciPy", scipy.__version__)
import sklearn; print("Scikit-Learn", sklearn.__version__)
```

```
Linux-5.10.133+-x86_64-with-Ubuntu-18.04-bionic
Python 3.7.15 (default, Oct 12 2022, 19:14:55)
[GCC 7.5.0]
NumPy 1.14.6
SciPy 1.5.4
Scikit-Learn 1.0.2
```

```
pip install -U scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-package
Collecting scikit-learn
  Using cached scikit_learn-1.0.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packag
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.22.1
    Uninstalling scikit-learn-0.22.1:
      Successfully uninstalled scikit-learn-0.22.1
ERROR: pip's dependency resolver does not currently take into account all the packag
```

```
yellowbrick 1.5 requires numpy>=1.16.0, but you have numpy 1.14.6 which is incompati
librosa 0.8.1 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatibl
kapre 0.3.7 requires numpy>=1.18.5, but you have numpy 1.14.6 which is incompatible.
datawig 0.2.0 requires scikit-learn[alldeps]==0.22.1, but you have scikit-learn 1.0.
Successfully installed scikit-learn-1.0.2
WARNING: The following packages were previously imported in this runtime:
  [sklearn]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
from imblearn.over_sampling import SMOTE


smt = SMOTE()
X_train, Y_train = smt.fit_resample(X_train, Y_train)
X_test, Y_test = smt.fit_resample(X_test, Y_test)


print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(Y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(Y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(Y_train == 0)))
```

```
After OverSampling, the shape of train_X: (606, 29)
After OverSampling, the shape of train_y: (606,)

After OverSampling, counts of label '1': 303
After OverSampling, counts of label '0': 303
```

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(n_estimators=10, random_state=1)
```

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

```
0.9966996699669967
0.9210526315789473
```

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:818: 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()
```

```
# 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:  0.9026402640264026
Accuracy score for testing data:  0.9539473684210527
```

```
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:818: 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:818: Conver
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: 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:818: 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:818: 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:818: 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:818: 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:818: 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:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

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

```
Avg accuracy: 0.888032786885246
```

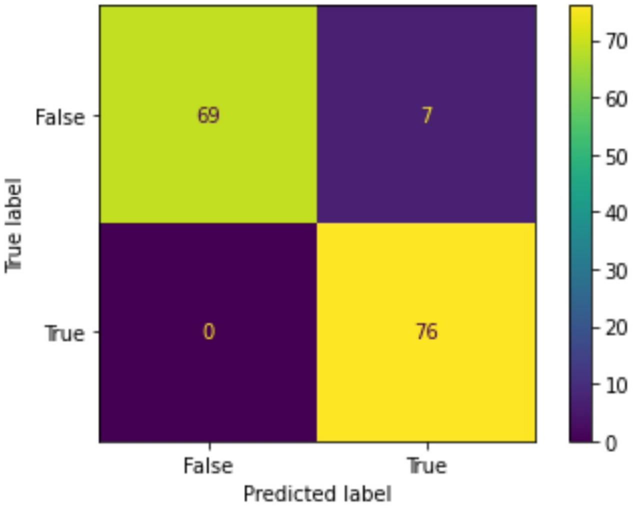
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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:818: 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:818: 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:818: 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:818: 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:818: 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:818: 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:818: 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:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

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

```
    Avg accuracy: 0.9737500000000001
```

```
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))
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)
```

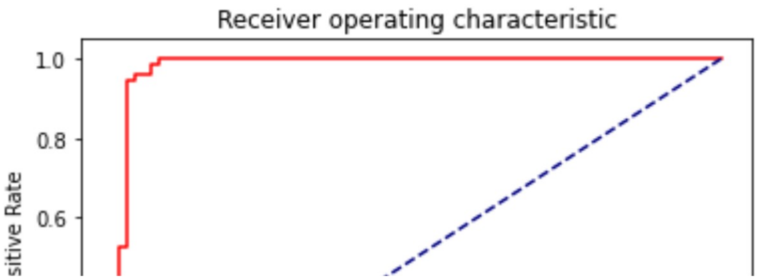
```
    Sensitivity:  1.0
    specificity:  0.9078947368421053
    PPV:  0.9156626506024096
    NPV:  1.0
```

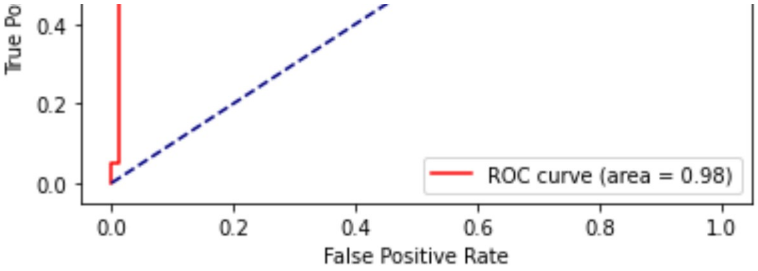
```
# AUROC and AUPR value
from sklearn.metrics import auc, roc_curve, precision_recall_curve

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: 0.9793975069252078
    AUPR: 0.9542077436147269
```

```
# 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
```

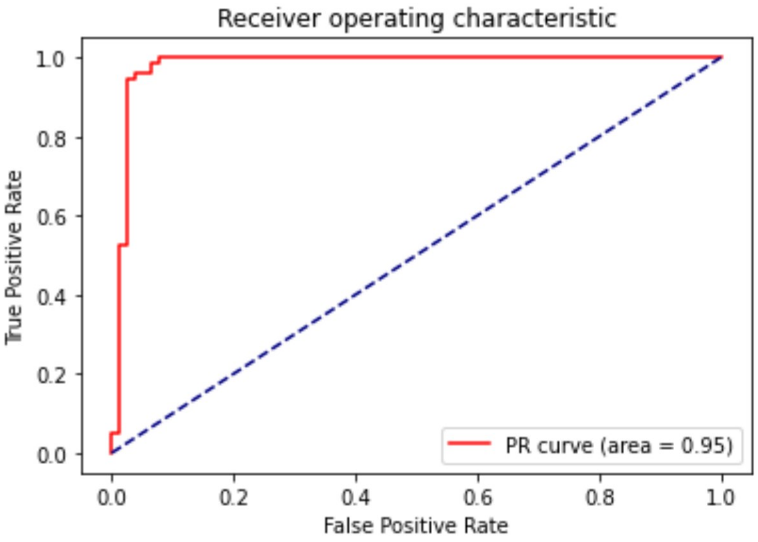
```
    <function matplotlib.pyplot.show(*args, **kw)>
```





```
# 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(n_estimators=10, random_state=1)
```

```
# 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_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:  0.9966996699669967
     Accuracy score for testing data:  0.9210526315789473
```

```
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([0.86885246, 0.93442623, 0.96721311, 0.83606557, 0.86885246,
            0.91803279, 0.98333333, 1.        , 0.96666667, 1.        ])
```

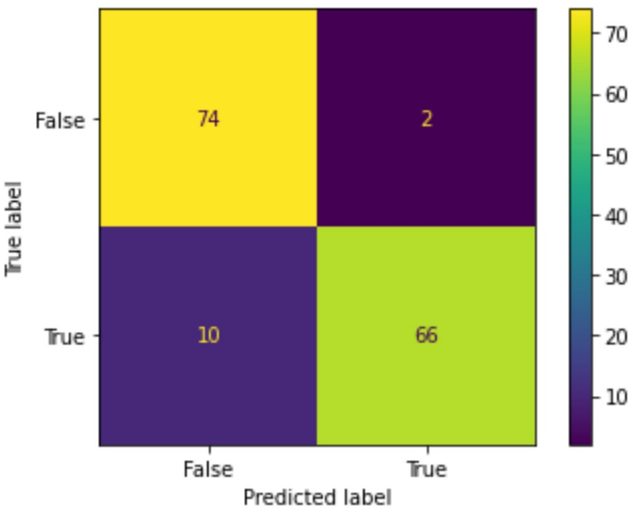
```
print("Avg accuracy: {}".format(result.mean()))

     Avg accuracy: 0.9343442622950819
```

```
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([0.9375    , 0.9375    , 0.86666667, 0.93333333, 1.        ,
            1.        , 1.        , 1.        , 1.        , 1.        ])
```

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

     Avg accuracy: 0.9675
```

```
# 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)
```

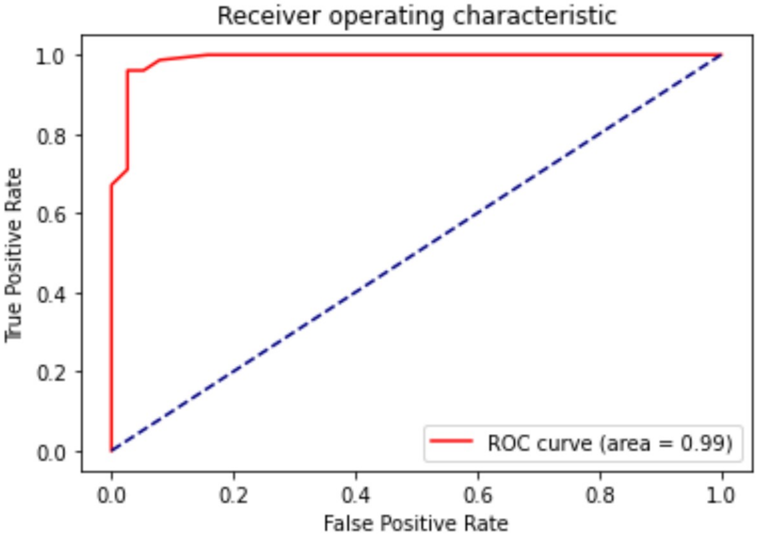
```
Sensitivity:  0.868421052631579
specificity:  0.9736842105263158
PPV:  0.9705882352941176
NPV:  0.8809523809523809
```

```
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)
print('AUPR: ',area)
```

```
AUROC: 0.989612188365651
AUPR: 0.9885283786560877
```

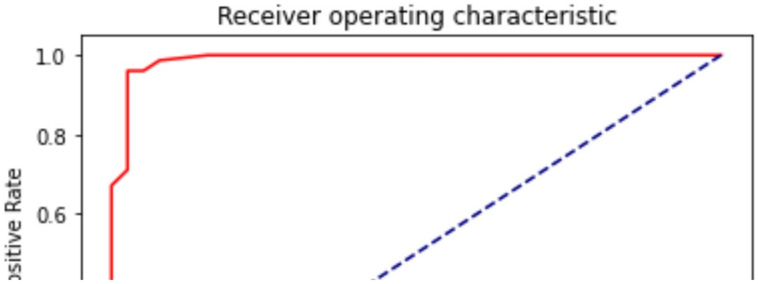
```
# AURoc graph

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
```

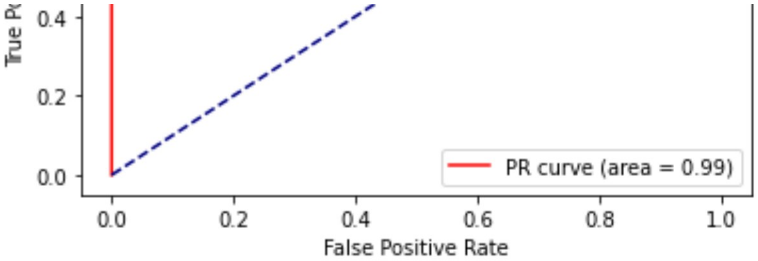
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# 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()
```

```
# 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:  0.8552631578947368
```

```
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([0.83606557, 0.90163934, 0.95081967, 0.86885246, 0.86885246,
            0.95081967, 0.98333333, 0.98333333, 0.98333333, 0.98333333])
```

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

```
     Avg accuracy: 0.9310382513661202
```

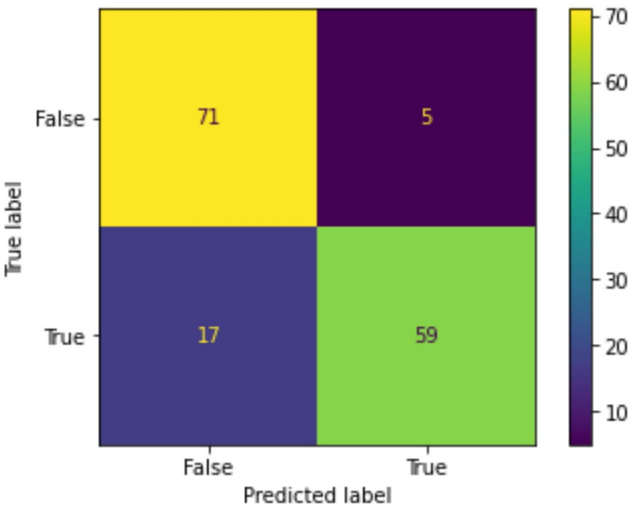
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
     array([0.875     , 0.875     , 0.93333333, 1.        , 1.        ,
            0.93333333, 0.86666667, 1.        , 1.        , 1.        ])
```

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

```
     Avg accuracy: 0.9483333333333335
```

```
# 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))
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)
```

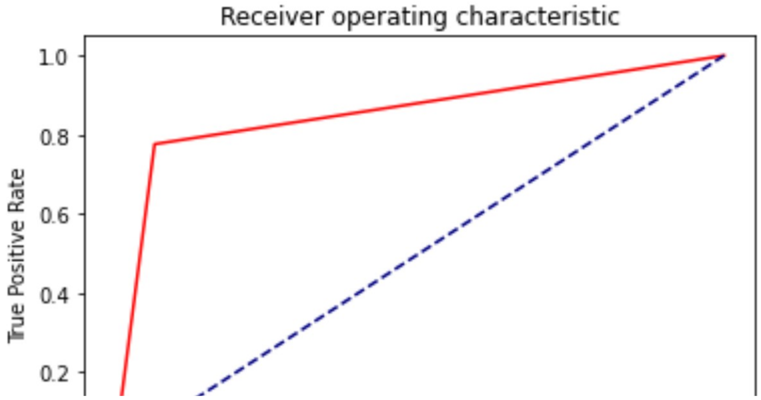
```
    Sensitivity:  0.7763157894736842
    specificity:  0.9342105263157895
    PPV:  0.921875
    NPV:  0.8068181818181818
```

```
# 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: 0.8552631578947368
    AUPR: 0.905016447368421
```

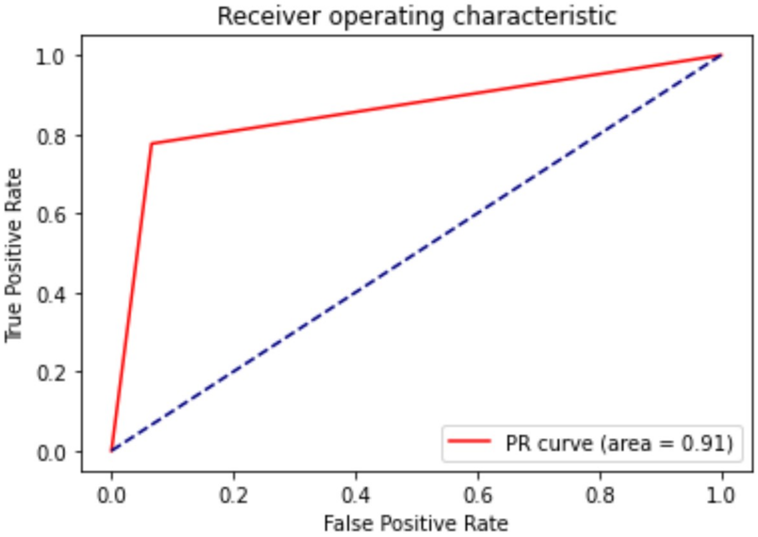
```
# AURoc graph

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(n_estimators=10, random_state=1)
```

```
# 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)
```

```
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:  0.9554455445544554
     Accuracy score for testing data:  0.9539473684210527
```

```
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([0.80327869, 0.91803279, 0.93442623, 0.80327869, 0.8852459 ,
            0.91803279, 0.98333333, 1.        , 0.96666667, 1.        ])
```

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

```
     Avg accuracy: 0.9212295081967212
```

```
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
```

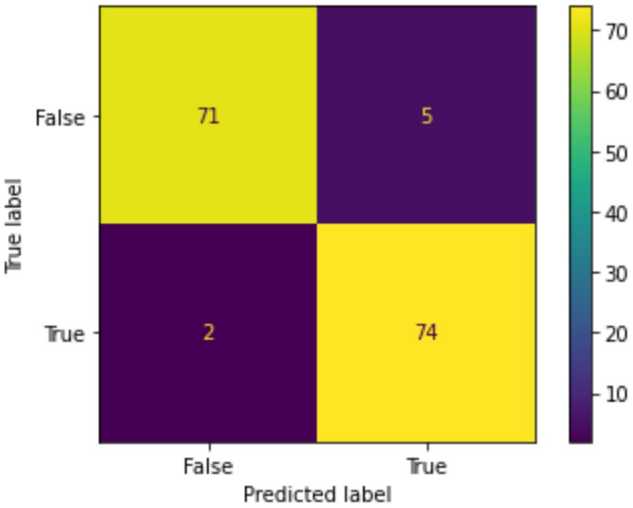
```
     array([0.875     , 0.875     , 0.86666667, 0.93333333, 0.93333333,
            0.93333333, 0.86666667, 1.        , 1.        , 1.        ])
```

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

```
     Avg accuracy: 0.9283333333333333
```

```
# 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:  0.9736842105263158
     specificity:  0.9342105263157895
     PPV:  0.9367088607594937
     NPV:  0.9726027397260274
```

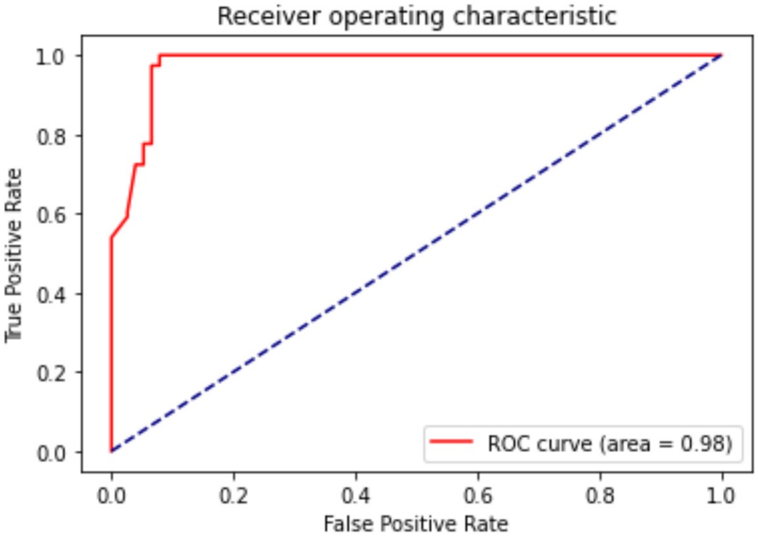
```
# AUROC and AUPR value
y_predictProb = gdb.predict_proba(X_test)

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: 0.977233379501385
```

```
AUPR: 0.9734737939061837
```

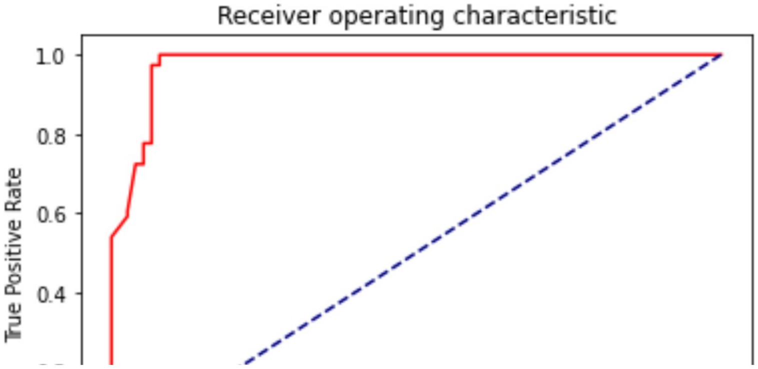
```
# AURoc graph

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
```

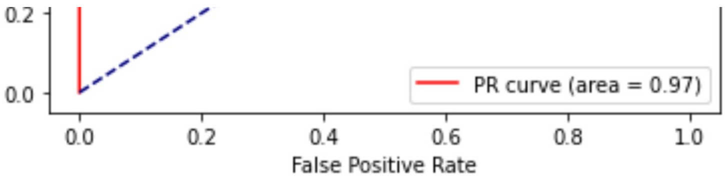
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# 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(min_samples_split=2, n_estimators=10, random_state=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:  0.9504950495049505
     Accuracy score for testing data:  0.9473684210526315
```

```
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([0.85245902, 0.91803279, 0.93442623, 0.83606557, 0.8852459 ,
            0.91803279, 0.98333333, 1.        , 0.96666667, 1.        ])
```

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

```
     Avg accuracy: 0.9294262295081968
```

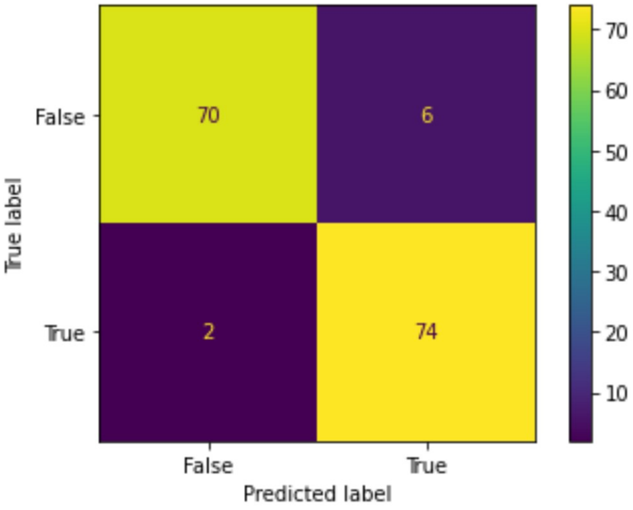
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
     array([0.9375    , 0.875     , 0.93333333, 0.93333333, 0.93333333,
            0.93333333, 0.93333333, 1.        , 1.        , 1.        ])
```

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

```
     Avg accuracy: 0.9479166666666666
```

```
# 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)
```

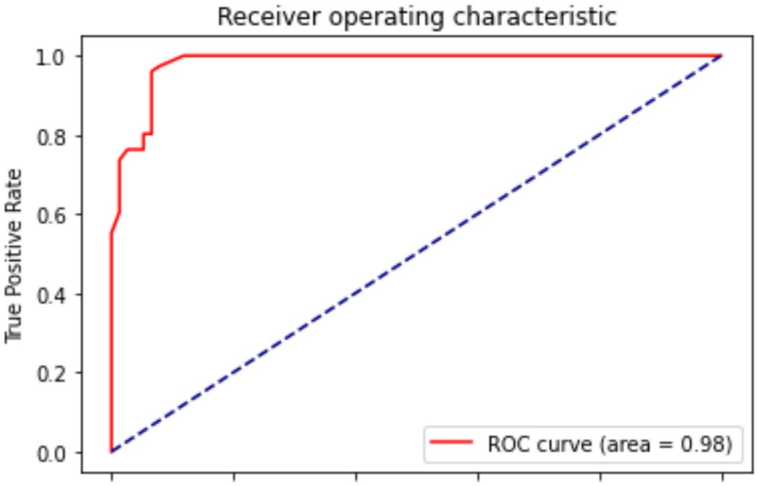
```
    Sensitivity:  0.9736842105263158
    specificity:  0.9210526315789473
    PPV:  0.925
    NPV:  0.9722222222222222
```

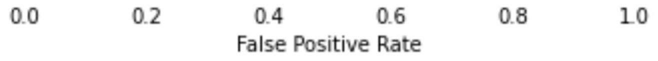
```
# AUROC and AUPR value
y_predictProb = xgb_clf.predict_proba(X_test)

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: 0.9813885041551247
    AUPR: 0.9793919865598656
```

```
# 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
```

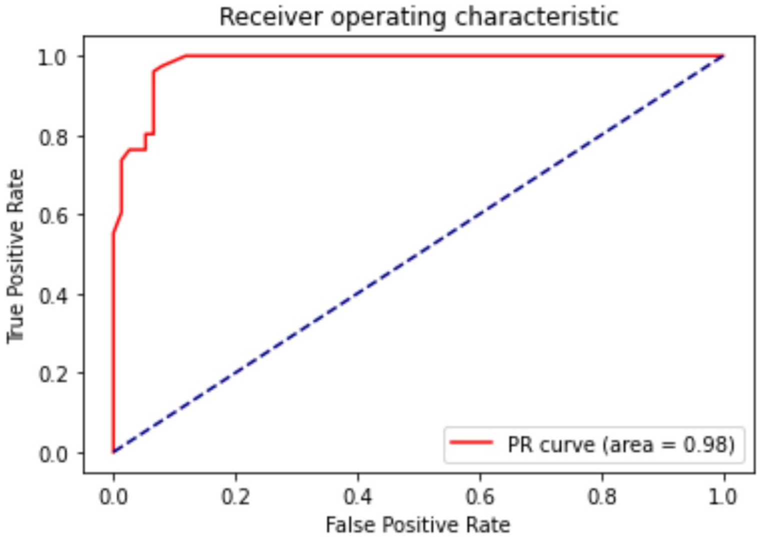
```
    <function matplotlib.pyplot.show(*args, **kw)>
```





```
# 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()
```

```
# 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:  0.7887788778877888
     Accuracy score for testing data:  0.875
```

```
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([0.81967213, 0.85245902, 0.90163934, 0.85245902, 0.86885246,
            0.80327869, 0.65      , 0.61666667, 0.63333333, 0.75      ])
```

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

```
     Avg accuracy: 0.7748360655737704
```

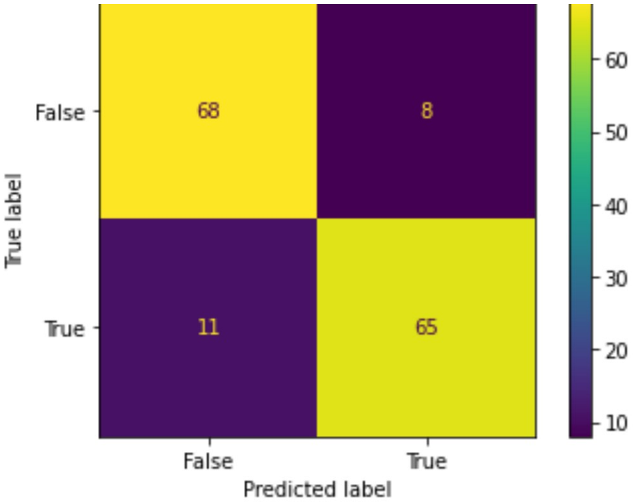
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
     array([0.8125    , 1.        , 0.93333333, 1.        , 0.8       ,
            0.93333333, 0.66666667, 0.86666667, 0.8       , 1.        ])
```

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

```
     Avg accuracy: 0.88125
```

```
# 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:  0.8552631578947368
specificity:  0.8947368421052632
PPV:  0.8904109589041096
NPV:  0.8607594936708861
```

```
# 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-121-289267775586> in <module>
      1 # AUROC and AUPR value
      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)
    799         if not self.probability:
    800             raise AttributeError(
--> 801                 "predict_proba is not available when  probability=False"
    802             )
    803         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()
```

```
# 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:  0.8762376237623762
     Accuracy score for testing data:  0.9671052631578947
```

```
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([0.81967213, 0.83606557, 0.86885246, 0.78688525, 0.91803279,
            0.86885246, 0.9       , 0.93333333, 0.86666667, 0.9       ])
```

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

```
     Avg accuracy: 0.8698360655737705
```

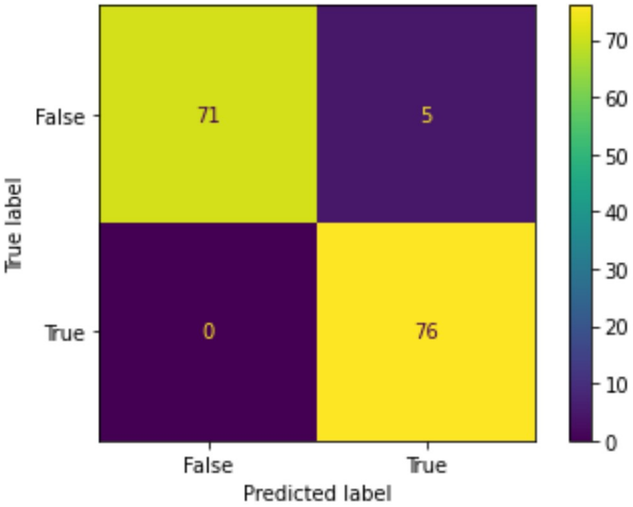
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
     array([0.9375    , 0.9375    , 1.        , 1.        , 0.86666667,
            0.93333333, 1.        , 1.        , 1.        , 0.86666667])
```

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

```
     Avg accuracy: 0.9541666666666668
```

```
# 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:  1.0
     specificity:  0.9342105263157895
     PPV:  0.9382716049382716
     NPV:  1.0
```

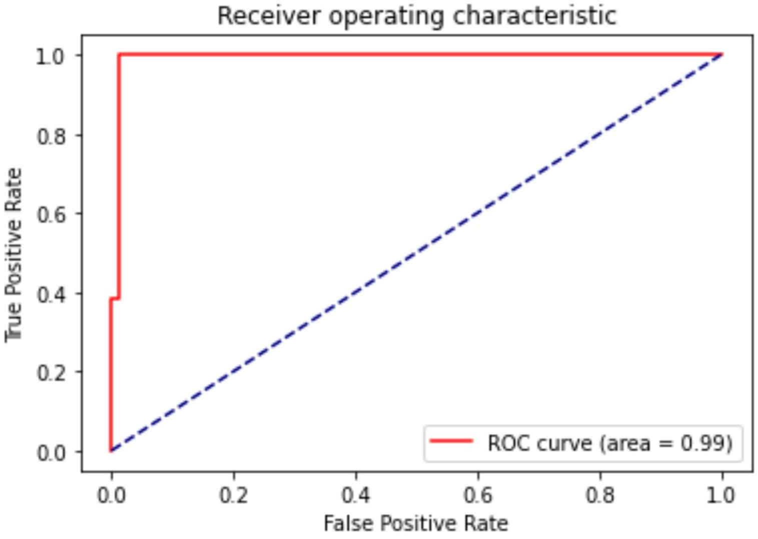
```
# AUROC and AUPR value
y_predictProb = gnb.predict_proba(X_test)

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: 0.9918628808864266
AUPR: 0.9875962293746098
```

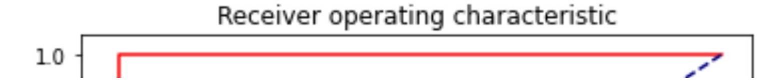
```
# AURoc graph

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

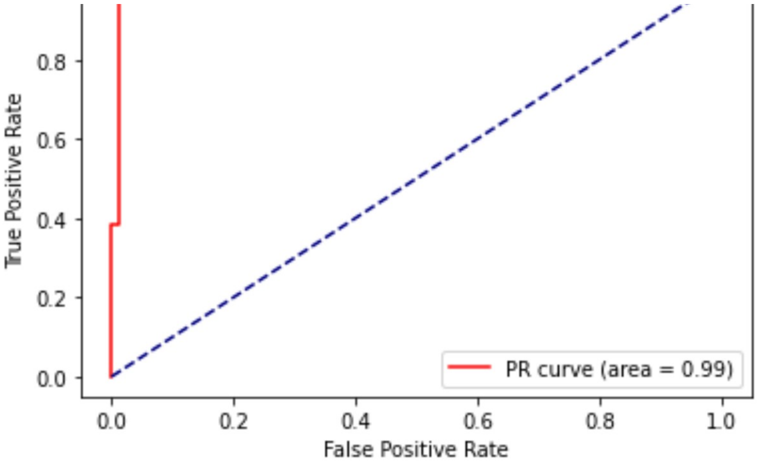
```
<function matplotlib.pyplot.show(*args, **kw)>
```



```
# AUPR graph

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

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




True Positive Rate
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
PR curve (area = 0.99)