```
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/
Collecting datawig
  Downloading datawig-0.2.0.tar.gz (61 kB)
     |████████████████████████████████| 61 kB 24 kB/s
Collecting scikit-learn[alldeps]==0.22.1
  Downloading scikit_learn-0.22.1-cp37-cp37m-manylinux1_x86_64.whl (7.0 MB)
     |████████████████████████████████| 7.0 MB 3.3 MB/s
Collecting typing==3.6.6
  Downloading typing-3.6.6-py3-none-any.whl (25 kB)
Collecting pandas==0.25.3
  Downloading pandas-0.25.3-cp37-cp37m-manylinux1_x86_64.whl (10.4 MB)
     |████████████████████████████████| 10.4 MB 31.5 MB/s
Collecting mxnet==1.4.0
  Downloading mxnet-1.4.0-py2.py3-none-manylinux1_x86_64.whl (29.6 MB)
     |████████████████████████████████| 29.6 MB 1.5 MB/s
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.7/dist-pac
Collecting graphviz<0.9.0,>=0.8.1
  Downloading graphviz-0.8.4-py2.py3-none-any.whl (16 kB)
Collecting numpy<1.15.0,>=1.8.2
  Downloading numpy-1.14.6-cp37-cp37m-manylinux1_x86_64.whl (13.8 MB)
     |████████████████████████████████| 13.8 MB 31.2 MB/s
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Collecting scipy>=0.17.0
  Downloading scipy-1.7.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl
     |████████████████████████████████| 38.2 MB 1.3 MB/s
  Downloading scipy-1.7.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (28.
     |████████████████████████████████| 28.5 MB 1.3 MB/s
  Downloading scipy-1.7.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (28.
     |████████████████████████████████| 28.5 MB 49.2 MB/s
  Downloading scipy-1.6.3-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
     |████████████████████████████████| 27.4 MB 1.4 MB/s
  Downloading scipy-1.6.2-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
     |████████████████████████████████| 27.4 MB 1.4 MB/s
```

✓ 52s completed at 7:27 PM

```
  Downloading scipy-1.6.0-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
     |████████████████████████████████| 27.4 MB 1.3 MB/s
  Downloading scipy-1.5.4-cp37-cp37m-manylinux1_x86_64.whl (25.9 MB)
     |████████████████████████████████| 25.9 MB 1.4 MB/s
Building wheels for collected packages: datawig
  Building wheel for datawig (setup.py) ... done
  Created wheel for datawig: filename=datawig-0.2.0-py3-none-any.whl size=72679 sha2
  Stored in directory: /root/.cache/pip/wheels/23/44/aa/12cf6e868f0d71e3c4e577963300
Successfully built datawig
Installing collected packages: numpy, scipy, scikit-learn, graphviz, typing, pandas,
  Attempting uninstall: numpy
    Found existing installation: numpy 1.21.6
    Uninstalling numpy-1.21.6:
      Successfully uninstalled numpy-1.21.6
  Attempting uninstall: scipy
    Found existing installation: scipy 1.7.3
    Uninstalling scipy-1.7.3:
      Successfully uninstalled scipy-1.7.3
  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
  Attempting uninstall: graphviz
    Found existing installation: graphviz 0.10.1
    Uninstalling graphviz-0.10.1:
      Successfully uninstalled graphviz-0.10.1
  Attempting uninstall: pandas
    Found existing installation: pandas 1.3.5
    Uninstalling pandas-1.3.5:
      Successfully uninstalled pandas-1.3.5
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
xarray 0.20.2 requires numpy>=1.18, but you have numpy 1.14.6 which is incompatible.
xarray 0.20.2 requires pandas>=1.1, but you have pandas 0.25.3 which is incompatible
xarray-einstats 0.2.2 requires numpy>=1.21, but you have numpy 1.14.6 which is incom
tifffile 2021.11.2 requires numpy>=1.15.1, but you have numpy 1.14.6 which is incomp
thinc 8.1.5 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatible.
tensorflow 2.9.2 requires numpy>=1.20, but you have numpy 1.14.6 which is incompatib
tables 3.7.0 requires numpy>=1.19.0, but you have numpy 1.14.6 which is incompatible
statsmodels 0.12.2 requires numpy>=1.15, but you have numpy 1.14.6 which is incompat
spacy 3.4.2 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatible.
seaborn 0.11.2 requires numpy>=1.15, but you have numpy 1.14.6 which is incompatible
scikit-image 0.18.3 requires numpy>=1.16.5, but you have numpy 1.14.6 which is incom
resampy 0.4.2 requires numpy>=1.17, but you have numpy 1.14.6 which is incompatible.
pywavelets 1.3.0 requires numpy>=1.17.3, but you have numpy 1.14.6 which is incompat
pymc 4.1.4 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatible.
pyerfa 2.0.0.1 requires numpy>=1.17, but you have numpy 1.14.6 which is incompatible
pyarrow 6.0.1 requires numpy>=1.16.6, but you have numpy 1.14.6 which is incompatibl
prophet 1.1.1 requires numpy>=1.15.4, but you have numpy 1.14.6 which is incompatibl
prophet 1.1.1 requires pandas>=1.0.4, but you have pandas 0.25.3 which is incompatib
plotnine 0.8.0 requires numpy>=1.19.0, but you have numpy 1.14.6 which is incompatib
```

```
plotnine 0.8.0 requires pandas>=1.1.0, but you have pandas 0.25.3 which is incompati
numba 0.56.3 requires numpy<1.24,>=1.18, but you have numpy 1.14.6 which is incompat
mizani 0.7.3 requires pandas>=1.1.0, but you have pandas 0.25.3 which is incompatibl
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.
jaxlib 0.3.22+cuda11.cudnn805 requires numpy>=1.20, but you have numpy 1.14.6 which
jax 0.3.23 requires numpy>=1.20, but you have numpy 1.14.6 which is incompatible.
imgaug 0.4.0 requires numpy>=1.15, 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
httpstan 4.6.1 requires numpy<2.0,>=1.16, but you have numpy 1.14.6 which is incompa
gym 0.25.2 requires numpy>=1.18.0, but you have numpy 1.14.6 which is incompatible.
google-colab 1.0.0 requires pandas>=1.1.0, but you have pandas 0.25.3 which is incom
cvxpy 1.2.1 requires numpy>=1.15, but you have numpy 1.14.6 which is incompatible.
cmdstanpy 1.0.8 requires numpy>=1.21, but you have numpy 1.14.6 which is incompatibl
blis 0.7.9 requires numpy>=1.15.0, but you have numpy 1.14.6 which is incompatible.
astropy 4.3.1 requires numpy>=1.17, but you have numpy 1.14.6 which is incompatible.
aesara 2.7.9 requires numpy>=1.17.0, but you have numpy 1.14.6 which is incompatible
aeppl 0.0.33 requires numpy>=1.18.1, but you have numpy 1.14.6 which is incompatible
Successfully installed datawig-0.2.0 graphviz-0.8.4 mxnet-1.4.0 numpy-1.14.6 pandas-
WARNING: The following packages were previously imported in this runtime:
  [numpy,typing]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
import datawig
```

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

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

	Age	BMI	Sex	Height	Weight	AlvaradoScore	PediatricAppendiciti
0	12.531143	16.494601	male	159.0	41.7	7	
1	12.410678	12.595222	female	152.0	29.1	8	
2	10.537988	15.991247	male	133.5	28.5	3	
3	10.425736	16.185025	male	146.0	34.5	4	
4	13.270363	20.449137	female	164.0	55.0	2	
...	...	...	...	...	...	...	
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(['AppendicitisComplications','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','DiagnosisByCriteria',
                     'PsoasSign','Stool'],axis=1).copy()
```

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

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

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

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

```
#perform label Encoding for categorical data

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

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

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

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

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

```
# Datawig imputation

from datawig import SimpleImputer
```

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

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

```
df_dw_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 15 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
dtypes: float64(15)
memory usage: 50.5 KB
```

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

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

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

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

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

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

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

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 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
```

```
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
DiagnosisByCriteria           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.DiagnosisByCriteria,df_final[col])
        corr_df.loc[col]=[round(r,3),round(p,3)]

corr_df
```

```
Age
BMI
Height
Weight
AlvaradoScore
PediatricAppendicitisScore
BodyTemp
WBCCount
NeutrophilPerc
CRPEntry
Sex
KetonesInUrine
ErythrocytesInUrine
WBCInUrine
Peritonitis
MigratoryPain
LowerAbdominalPainRight
ReboundTenderness
CoughingPain
Nausea
AppetiteLoss
Dysuria
```

```
Dysuria
FreeFluids
PathLymphNodes
MesentricLymphadenitis
FecalImpaction
Meteorism
DiagnosisByCriteria
PsoasSign
Stool
```

	r	p
Age	0.073	0.129
BMI	0.109	0.024
Height	0.050	0.301
Weight	0.094	0.051
AlvaradoScore	-0.439	0.000
PediatricAppendicitisScore	-0.373	0.000
BodyTemp	-0.196	0.000
WBCCount	-0.412	0.000
NeutrophilPerc	-0.446	0.000
CRPEntry	-0.265	0.000
Sex	-0.102	0.034
KetonesInUrine	0.091	0.058
ErythrocytesInUrine	0.041	0.397
WBCInUrine	-0.076	0.116
Peritonitis	0.529	0.000
MigratoryPain	-0.141	0.003
LowerAbdominalPainRight	-0.067	0.166
ReboundTenderness	-0.158	0.001
CoughingPain	-0.144	0.003
Nausea	-0.138	0.004
AppetiteLoss	-0.067	0.164
Dysuria	0.098	0.043
FreeFluids	-0.191	0.000
PathLymphNodes	0.018	0.709
MesentricLymphadenitis	-0.047	0.327

FecalImpaction	0.038	0.426
Meteorism	0.064	0.186
DiagnosisByCriteria	1.000	0.000
PsoasSign	0.080	0.097
Stool	0.071	0.144

```
df_final.shape
```

```
(430, 30)
```

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

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

1 = yes, 0 = NO

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

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

```
(246, 30)
(184, 30)
```

```
#spliting the data for training and testing

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

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

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

```
(430, 29)
(344, 29)
```

```
(86, 29)
```

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

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

Logistic Regression

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

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

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

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

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


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

```
Accuracy score for training data:  0.7703488372093024
Accuracy score for testing data:  0.8255813953488372
```

```
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: 0.7301680672268908
```

```
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
```

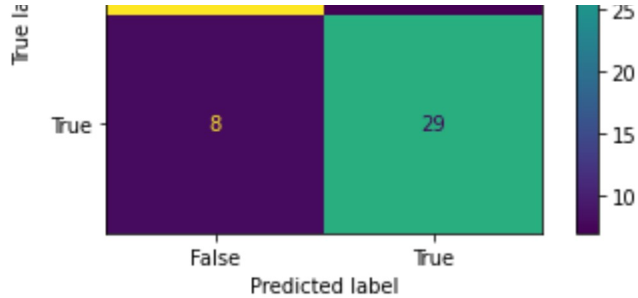
```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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
```

```
        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: 0.8125
```

```
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:  0.7837837837837838
specificity:  0.8571428571428571
PPV:  0.8055555555555556
NPV:  0.84
```

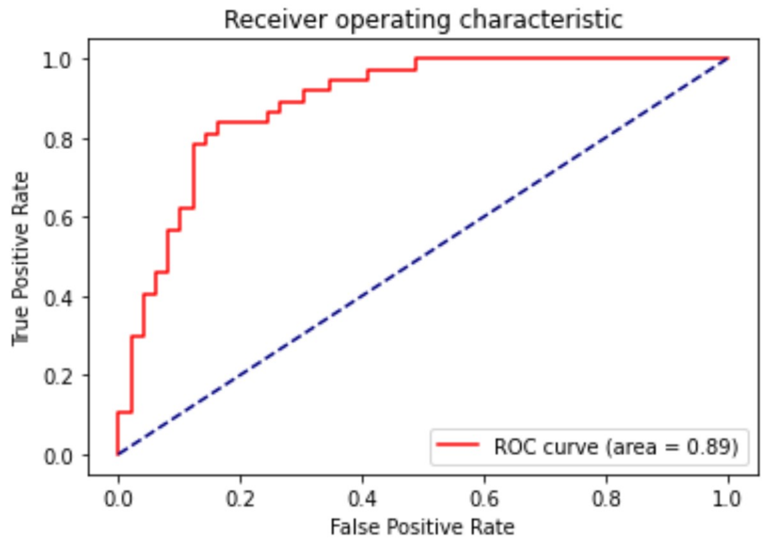
```
# AUROC and AUPR value
from sklearn.metrics import auc, roc_curve, precision_recall_curve

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: 0.8902371759514617
AUPR: 0.8301087020654656
```

```
# AURoc graph

plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % roc_auc)
```

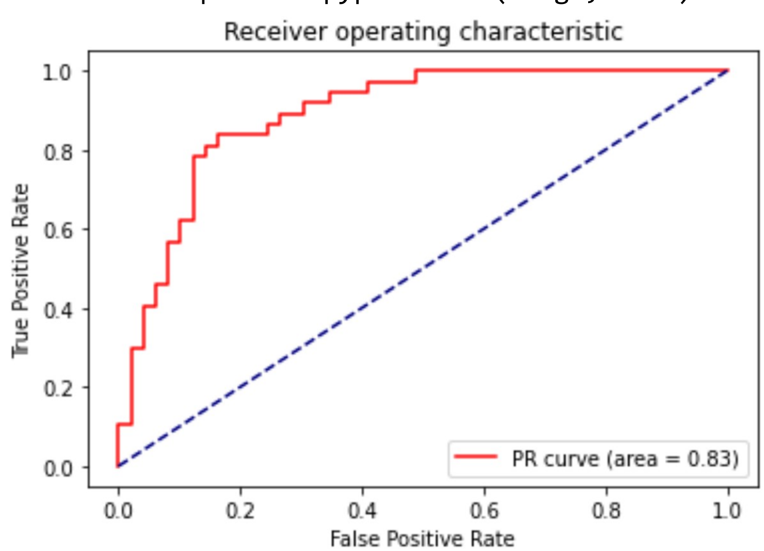
```
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show
```

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




```
# AUPR graph

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

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



Random Forest

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

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

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

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


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

```
Accuracy score for training data:  0.9941860465116279
Accuracy score for testing data:  0.8372093023255814
```

```
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(forest , X_train, Y_train, cv = kf)
result
```

```
array([0.82857143, 0.77142857, 0.48571429, 0.71428571, 0.88235294,
       0.73529412, 0.70588235, 0.73529412, 0.64705882, 0.52941176])
```

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

```
Avg accuracy: 0.703529411764706
```

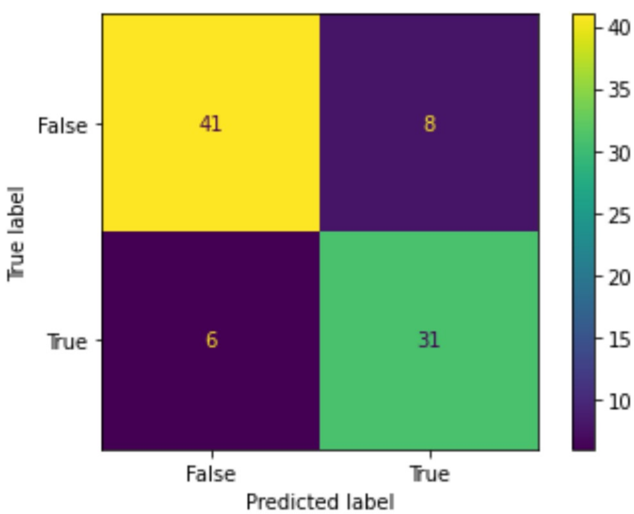
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

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

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

```
Avg accuracy: 0.775
```

```
# 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)
```

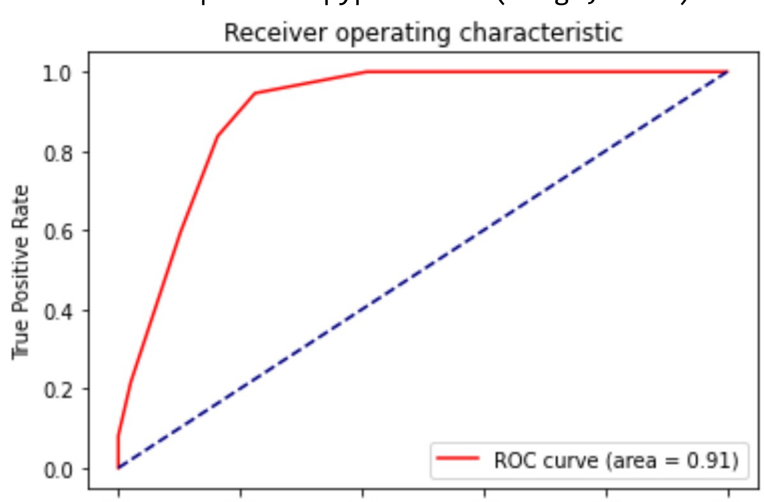
```
Sensitivity:  0.8378378378378378
specificity:  0.8367346938775511
PPV:  0.7948717948717948
NPV:  0.8723404255319149
```

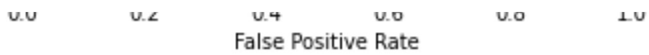
```
y_predictProb = forest.predict_proba(X_test)

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: 0.9051296194153337
AUPR: 0.8455022528249074
```

```
# AURoc graph

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

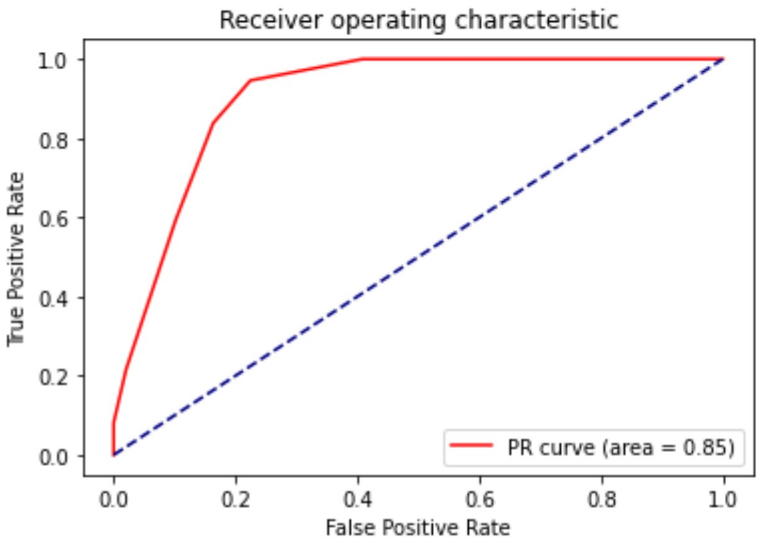
```
<function matplotlib.pyplot.show(*args, **kw)>
```






```
# AUPR graph

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

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



Decision Tree

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

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

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

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


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

```
Accuracy score for training data:  1.0
Accuracy score for testing data:  0.7558139534883721
```

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

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

```
array([0.74285714, 0.57142857, 0.6       , 0.65714286, 0.67647059,
       0.85294118, 0.67647059, 0.58823529, 0.70588235, 0.67647059])
```

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

```
Avg accuracy: 0.6747899159663866
```

```
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
```

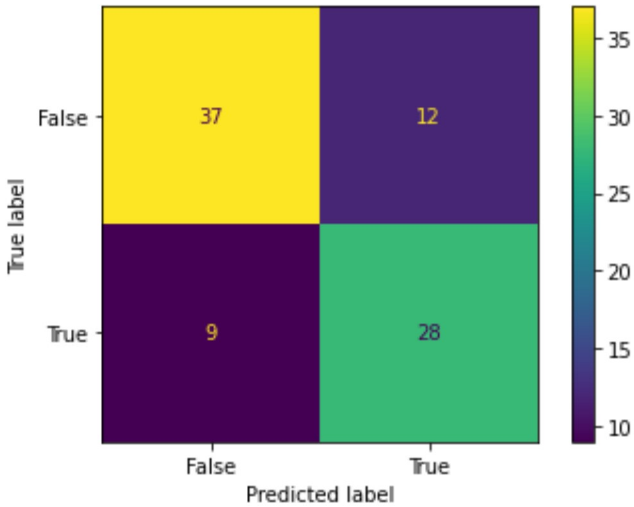
```
array([0.77777778, 0.88888889, 0.66666667, 0.77777778, 0.88888889,
       0.88888889, 0.875     , 0.625     , 0.625     , 0.625     ])
```

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

```
Avg accuracy: 0.7638888888888888
```

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

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




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

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

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

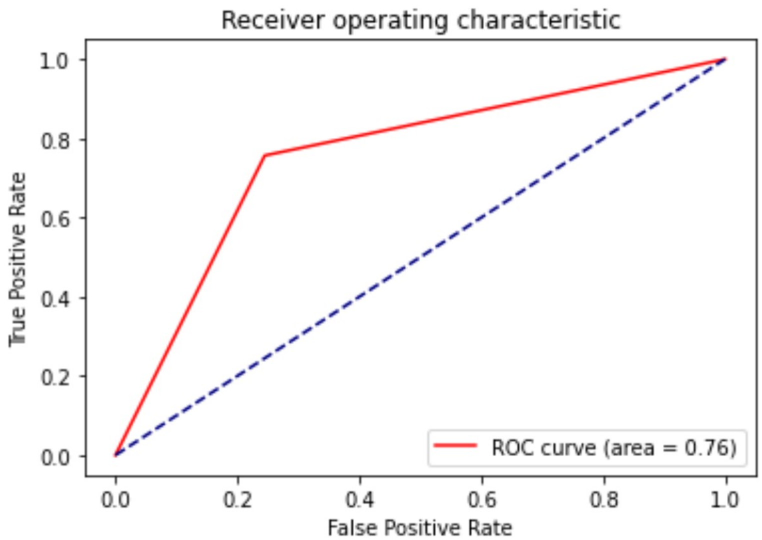
```
Sensitivity:  0.7567567567567568
specificity:  0.7551020408163265
PPV:  0.7
NPV:  0.8043478260869565
```

```
# 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: 0.7559293987865416
AUPR: 0.7807039597737272
```

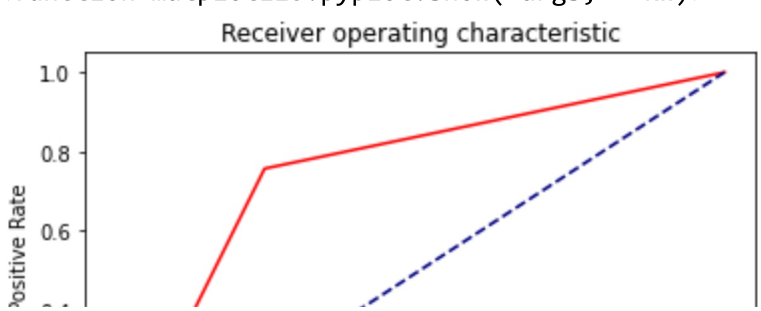
```
# AURoc graph

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
```

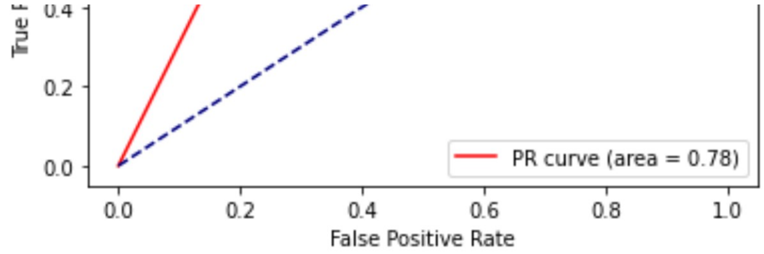
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
# 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:  0.8197674418604651
Accuracy score for testing data:  0.7790697674418605
```

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

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

```
array([0.77142857, 0.77142857, 0.62857143, 0.8       , 0.70588235,
       0.82352941, 0.73529412, 0.79411765, 0.73529412, 0.67647059])
```

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

```
Avg accuracy: 0.7442016806722689
```

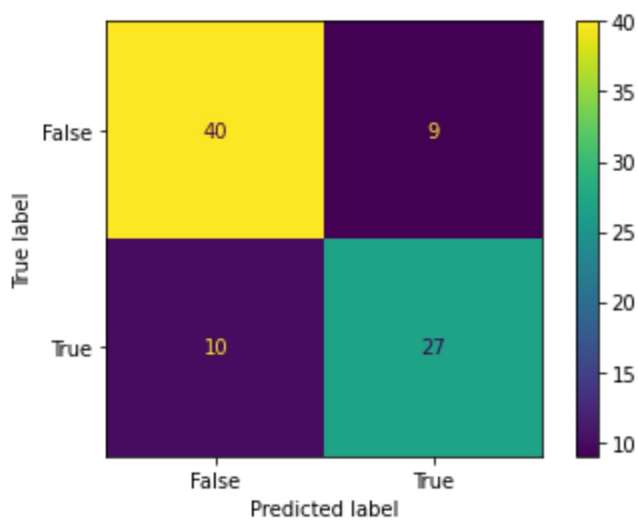
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

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

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

```
Avg accuracy: 0.7527777777777777
```

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

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



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

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

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

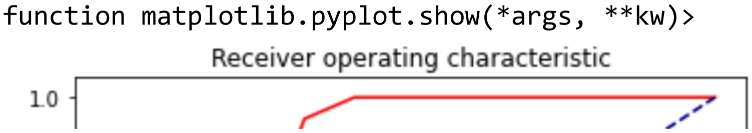
```
Sensitivity:  0.7297297297297297
specificity:  0.8163265306122449
PPV:  0.75
NPV:  0.8
```

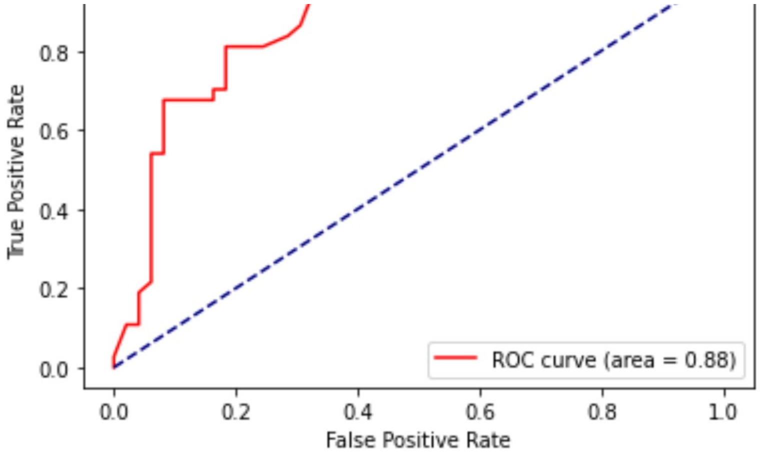
```
# AUROC and AUPR value
y_predictProb = gdb.predict_proba(X_test)

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: 0.8786541643684501
AUPR: 0.788681500940104
```

```
# 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
```

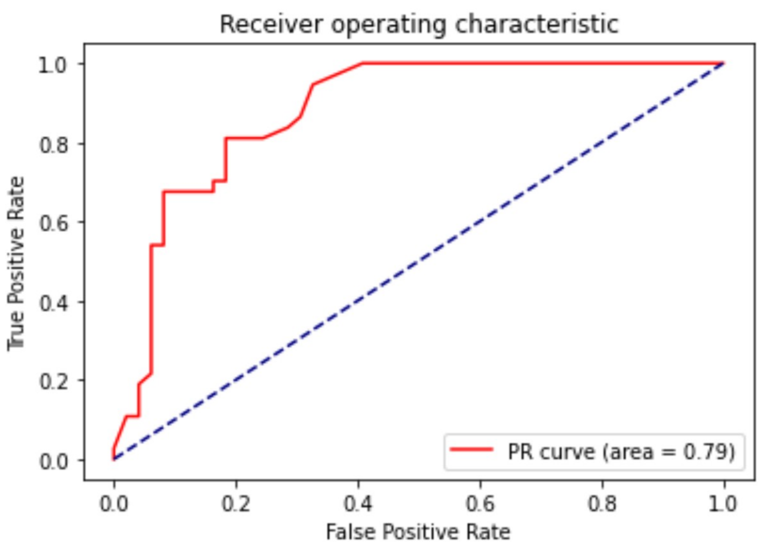
```
<function matplotlib.pyplot.show(*args, **kw)>
```






```
# 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:  0.8284883720930233
Accuracy score for testing data:  0.7674418604651163
```

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

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

```
array([0.77142857, 0.77142857, 0.6       , 0.8       , 0.79411765,
       0.76470588, 0.73529412, 0.79411765, 0.73529412, 0.67647059])
```

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

```
Avg accuracy: 0.7442857142857143
```

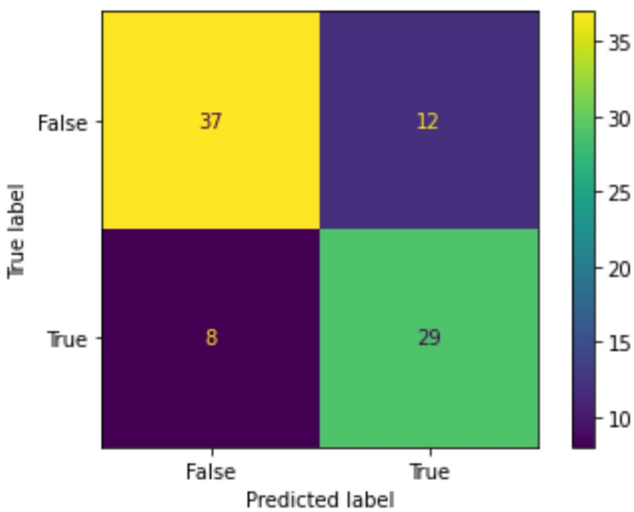
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

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

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

```
Avg accuracy: 0.798611111111111
```

```
# 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)
```

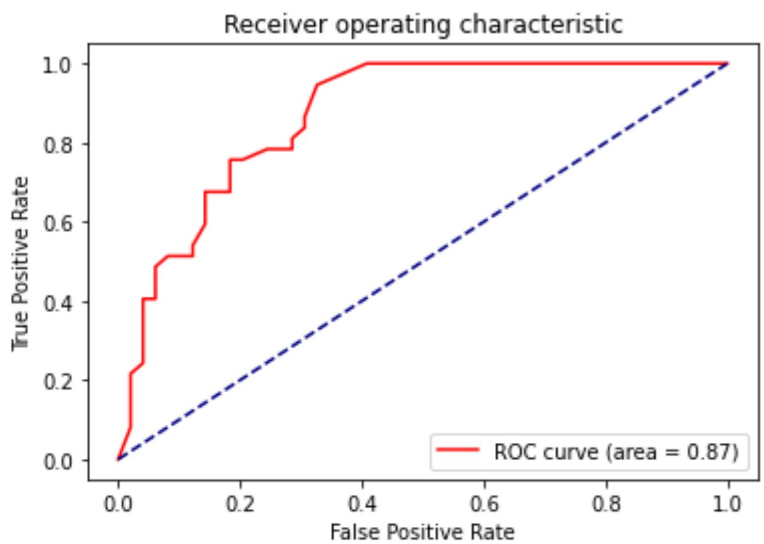
```
Sensitivity:  0.7837837837837838
specificity:  0.7551020408163265
PPV:  0.7073170731707317
NPV:  0.8222222222222222
```

```
# 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: 0.8695532266960839
AUPR: 0.787357788478683
```

```
# 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
```

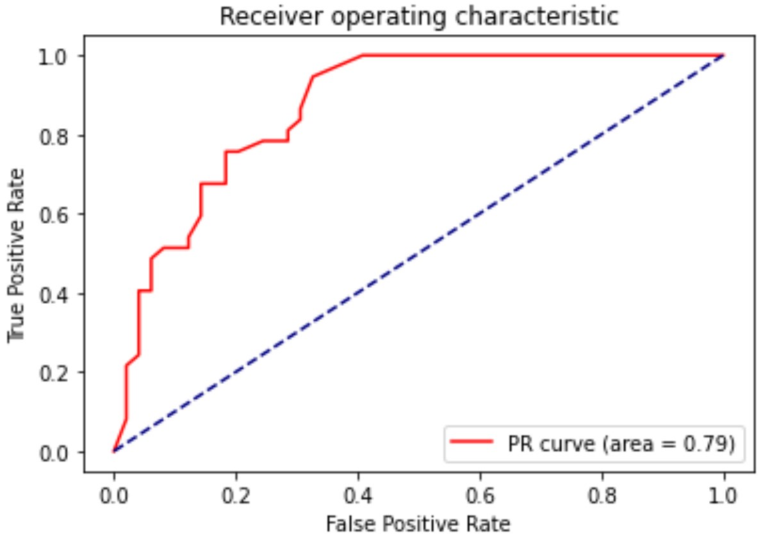
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
# 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:  0.7238372093023255
Accuracy score for testing data:  0.7790697674418605
```

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

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

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

```
array([0.74285714, 0.74285714, 0.71428571, 0.71428571, 0.73529412,
       0.73529412, 0.61764706, 0.82352941, 0.73529412, 0.64705882])
```

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

```
Avg accuracy: 0.7208403361344539
```

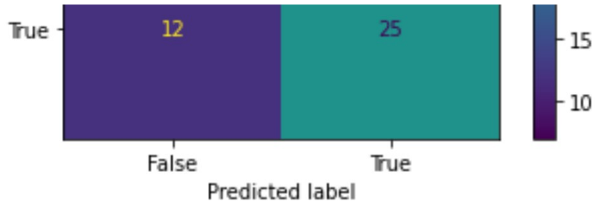
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

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

```
array([0.77777778, 0.88888889, 0.77777778, 0.77777778, 0.66666667,
       0.55555556, 0.875     , 0.75      , 0.625     , 0.625     ])
```

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

```
Avg accuracy: 0.7319444444444445
```

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

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




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

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

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

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

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

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

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

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

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

1 frames

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py in _check_proba(self)
    601     def _check_proba(self):
```

```
    601     def _check_proba(self):
    602         if not self.probability:
--> 603             raise AttributeError("predict_proba is not available when "
    604                                  " probability=False")
    605         if self._impl not in ('c_svc', 'nu_svc'):

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

SEARCH STACK OVERFLOW

```
# AURoc graph

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

```
# AUPR graph

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

Gausian Naive Bayes

```
#using Naive Bayesian

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

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

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

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

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

```
Accuracy score for training data:  0.752906976744186
Accuracy score for testing data:  0.8255813953488372
```

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

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

```
array([0.53488372, 0.41860465, 0.76744186, 0.88372093, 0.86046512,
       0.81395349, 0.76744186, 0.95348837, 0.60465116, 0.39534884])
```

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

```
Avg accuracy: 0.7
```

```
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
```

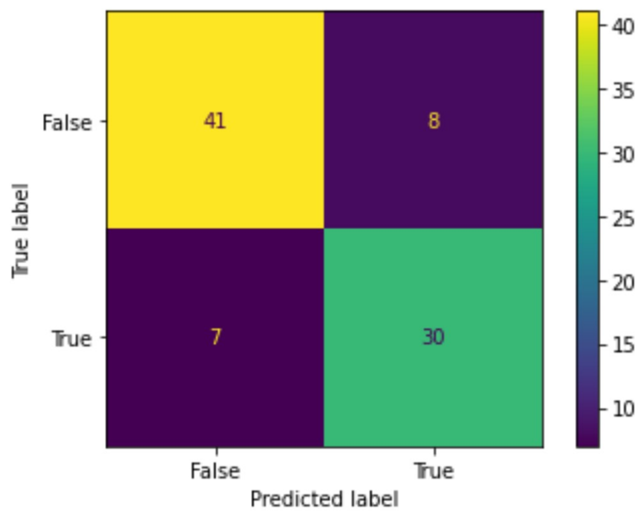
```
array([0.77777778, 0.88888889, 1.        , 0.77777778, 0.88888889,
       1.        , 0.625     , 0.75      , 1.        , 0.625     ])
```

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

```
Avg accuracy: 0.8333333333333333
```

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

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

```
plt.show()
```




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

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

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

```
Sensitivity:  0.8108108108108109
specificity:  0.8367346938775511
PPV:  0.7894736842105263
NPV:  0.8541666666666666
```

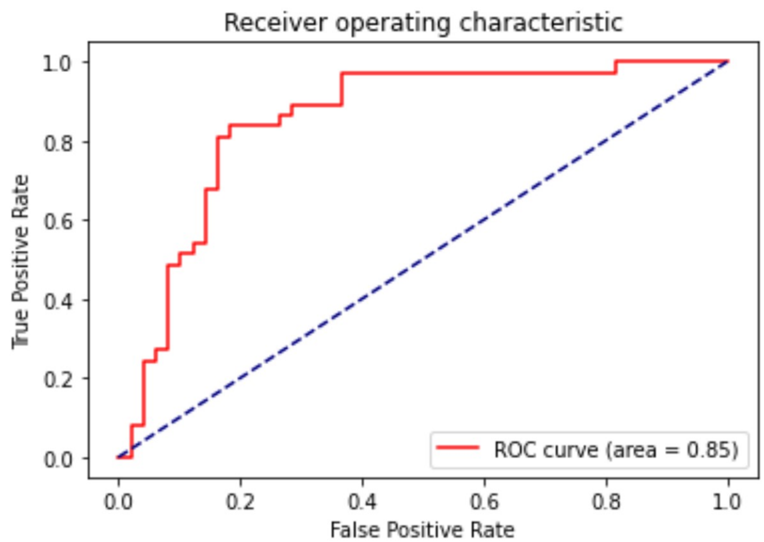
```
# AUROC and AUPR value
y_predictProb = gnb.predict_proba(X_test)

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: 0.8532818532818534
```

```
AUPR: 0.7235303550478562
```

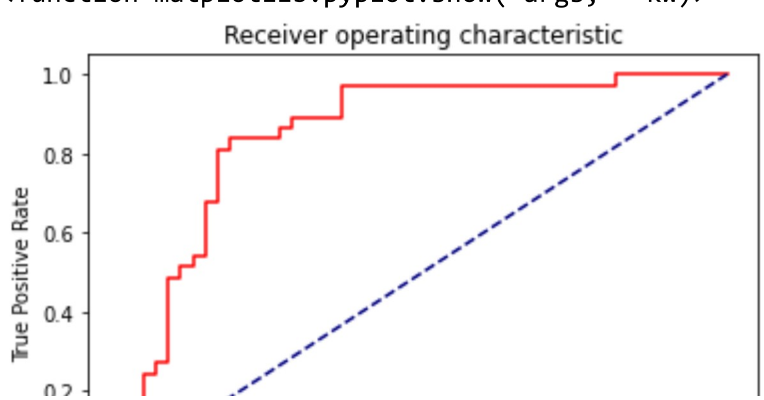
```
# AURoc graph

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
```

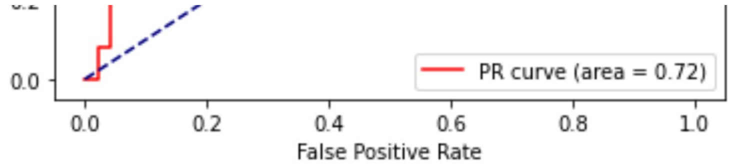
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
# 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)>
```




0.0
PR curve (area = 0.72)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate