

# DRUG CLASSIFICATION

The goal of this activity is to predict the Cholesterol based on sex, BP and Drug, In this activity I will Identify the best model to use in this prediction

- 1. DATA EXPLORATION

```
In [1]: import numpy as np;
import pandas as pd;
import seaborn as sns;
import matplotlib.pyplot as plt;
```

```
In [2]: df = pd.read_csv('drug.csv');
```

```
In [3]: df.head(1000)
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

200 rows × 6 columns

I will check the unique values of each column

```
In [4]: df['BP'].unique()
```

```
Out[4]: array(['HIGH', 'LOW', 'NORMAL'], dtype=object)
```

```
In [5]: df['Cholesterol'].unique()
```

```
Out[5]: array(['HIGH', 'NORMAL'], dtype=object)
```

```
In [6]: df['Drug'].unique()
```

```
Out[6]: array(['DrugY', 'drugC', 'drugX', 'drugA', 'drugB'], dtype=object)
```

## Count the unique values in each feature

```
In [7]: df['Cholesterol'].value_counts()
```

```
Out[7]: HIGH      103  
NORMAL     97  
Name: Cholesterol, dtype: int64
```

```
In [8]: df['Sex'].value_counts()
```

```
Out[8]: M      104  
F      96  
Name: Sex, dtype: int64
```

```
In [9]: df['BP'].value_counts()
```

```
Out[9]: HIGH      77  
LOW       64  
NORMAL    59  
Name: BP, dtype: int64
```

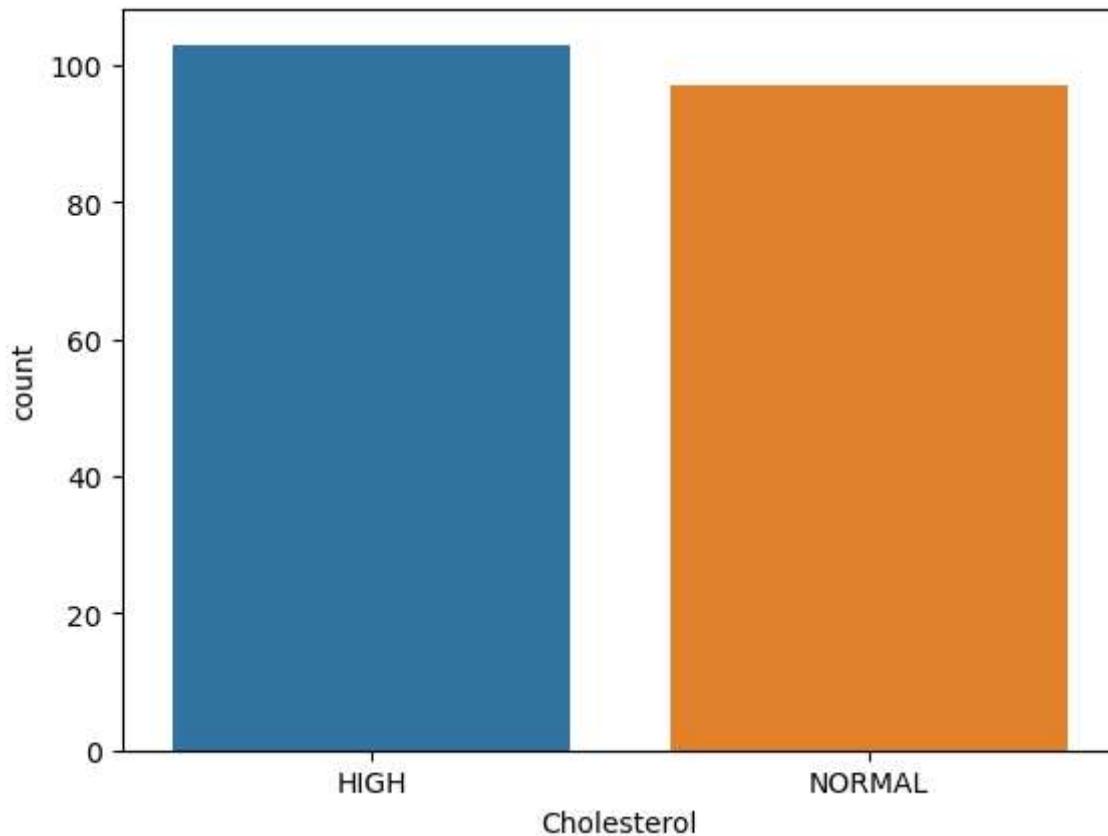
```
In [10]: df['Drug'].value_counts()
```

```
Out[10]: DrugY    91  
drugX    54  
drugA    23  
drugC    16  
drugB    16  
Name: Drug, dtype: int64
```

## Visualization for the count of cholesterol unique values

```
In [11]: sns.countplot(df['Cholesterol']);
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:  
Pass the following variable as a keyword arg: x. From version 0.12, the only valid po-  
sitional argument will be `data`, and passing other arguments without an explicit key  
word will result in an error or misinterpretation.  
warnings.warn(
```



## 2. Data Pre-Processing

I will just Drop some features to have a more clean data

```
In [12]: df = df.drop(['Na_to_K'], axis=1)
```

```
In [13]: df
```

Out[13]:

	Age	Sex	BP	Cholesterol	Drug
0	23	F	HIGH	HIGH	DrugY
1	47	M	LOW	HIGH	drugC
2	47	M	LOW	HIGH	drugC
3	28	F	NORMAL	HIGH	drugX
4	61	F	LOW	HIGH	DrugY
...	...	...	...	...	...
195	56	F	LOW	HIGH	drugC
196	16	M	LOW	HIGH	drugC
197	52	M	NORMAL	HIGH	drugX
198	23	M	NORMAL	NORMAL	drugX
199	40	F	LOW	NORMAL	drugX

200 rows × 5 columns

- 2.1 Data Cleaning
- 2.1.1 Drop Duplicates

In [14]: `#check the shape of the dataframe  
df.shape`

Out[14]: (200, 5)

In [15]: `#Check if there are duplicates  
df.duplicated().sum()`

Out[15]: 12

In [16]: `#drop the duplicated values  
df.drop_duplicates(inplace = True)`

In [17]: `df.duplicated().sum()`

Out[17]: 0

In [18]: `#Check null values  
df.isna().sum()`

Out[18]:

Age	0
Sex	0
BP	0
Cholesterol	0
Drug	0
dtype:	int64

## As we can see, there are no null values in the dataset

```
In [19]: #check data types of features
df.dtypes
```

```
Out[19]: Age          int64
Sex           object
BP            object
Cholesterol  object
Drug          object
dtype: object
```

```
In [20]: # I just drop the age as it is not included in the one-hot encoding
df = df.drop(['Age'], axis=1)
```

Since we do not have any numerical features, we will skip "removing extreme values" step

- 2.2 Converting Categorical Features into numerical features

```
In [21]: #importing Label encoder
from sklearn.preprocessing import LabelEncoder
```

```
In [22]: df.columns
```

```
Out[22]: Index(['Sex', 'BP', 'Cholesterol', 'Drug'], dtype='object')
```

```
In [23]: df2 = df.copy()
df2
```

	Sex	BP	Cholesterol	Drug
0	F	HIGH	HIGH	DrugY
1	M	LOW	HIGH	drugC
3	F	NORMAL	HIGH	drugX
4	F	LOW	HIGH	DrugY
5	F	NORMAL	HIGH	drugX
...	...	...	...	...
195	F	LOW	HIGH	drugC
196	M	LOW	HIGH	drugC
197	M	NORMAL	HIGH	drugX
198	M	NORMAL	NORMAL	drugX
199	F	LOW	NORMAL	drugX

188 rows × 4 columns

```
In [24]: encoder = LabelEncoder();
df3 = df2.copy();
```

```
In [25]: #Loop
for col in df3.columns:
    if(col != 'Cholesterol'):
        df3[col] = encoder.fit_transform(df3[col]);
```

```
In [26]: df3.head(10)
```

Out[26]:

	Sex	BP	Cholesterol	Drug
0	0	0	HIGH	0
1	1	1	HIGH	3
3	0	2	HIGH	4
4	0	1	HIGH	0
5	0	2	HIGH	4
6	0	2	HIGH	0
7	1	1	HIGH	3
8	1	2	HIGH	0
9	1	1	NORMAL	0
10	0	1	HIGH	3

```
In [27]: df3['Cholesterol'] = np.where(df3['Cholesterol'] == 'HIGH', 1,0)
```

```
In [28]: df3
```

Out[28]:

	Sex	BP	Cholesterol	Drug
0	0	0	1	0
1	1	1	1	3
3	0	2	1	4
4	0	1	1	0
5	0	2	1	4
...	...	...	...	...
195	0	1	1	3
196	1	1	1	3
197	1	2	1	4
198	1	2	0	4
199	0	1	0	4

188 rows × 4 columns

In [71]: `df3.dtypes`

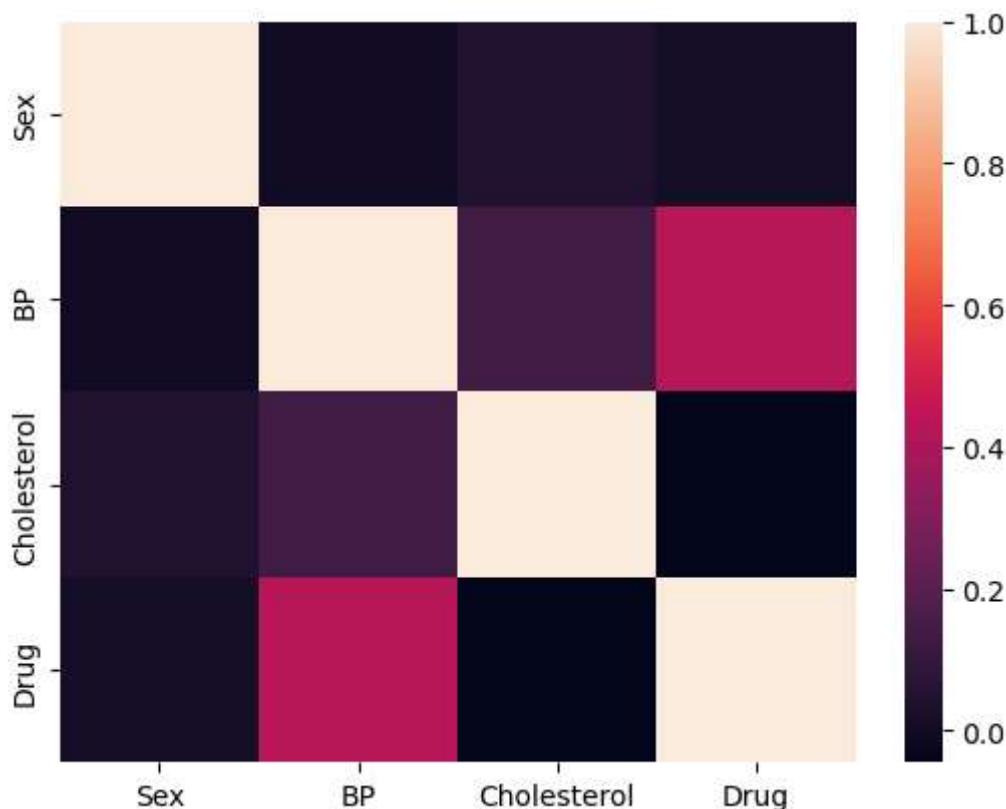
```
Out[71]: Sex      int32
          BP       int32
         Cholesterol  int32
        Drug      int32
       dtype: object
```

The features are now in integer values

We will now select our feature, but lets first check the correlations

- 2.3 Feature Selection

In [72]: `sns.heatmap(df3.corr());`



There are no correlated features in our dataset

## 3. Data Modelling

- 3.1 Split Target variables to feature variables

In [33]: `#Cholesterol is the target var since this is a 0 and 1 values, we can classify this feature`  
`X = df3.drop(['Cholesterol'], axis=1);`  
`y = df3['Cholesterol'];`

- 3.2 Split the dataset to training and tests

```
In [34]: #importing model selection for training and test
from sklearn.model_selection import train_test_split
```

```
In [35]: X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42, train_size=.6)
#stratify - I use stratify so that the features have the same proportion.
```

```
In [36]: #see values of train and test
print(X.shape)
print(X_train.shape)
print(X_test.shape)
```

```
(188, 3)
(112, 3)
(76, 3)
```

### 3.3 Choose The Best Model

- 3.3.1 Naive Bayes

```
In [37]: #gaussian naive bayes
from sklearn.naive_bayes import GaussianNB
```

```
In [38]: #for evaluation

from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import classification_report # so that we will not compute the ac
```

```
In [39]: def evaluate_model(model):
    print('Classification Report\n',classification_report(y_test, model.predict(X_test)))
    print('Confusion Matrix \n',confusion_matrix(y_test, model.predict(X_test)), labels)
    print('\nTraining Accuracy Score: ', model.score(X_train, y_train))
    print('Test Accuracy Score: ', model.score(X_test, y_test))
    print('\n ROC Curve')
    plot_roc_curve(model,X_test,y_test);
```

```
In [40]: gnb = GaussianNB()
gnb.fit(X_train, y_train);
gnb.score(X_test, y_test)
```

```
Out[40]: 0.4868421052631579
```

```
In [41]: evaluate_model(gnb)
```

## Classification Report

	precision	recall	f1-score	support
HIGH	0.47	0.41	0.43	37
NORMAL	0.50	0.56	0.53	39
accuracy			0.49	76
macro avg	0.48	0.48	0.48	76
weighted avg	0.48	0.49	0.48	76

## Confusion Matrix

```
[[15 22]
 [17 22]]
```

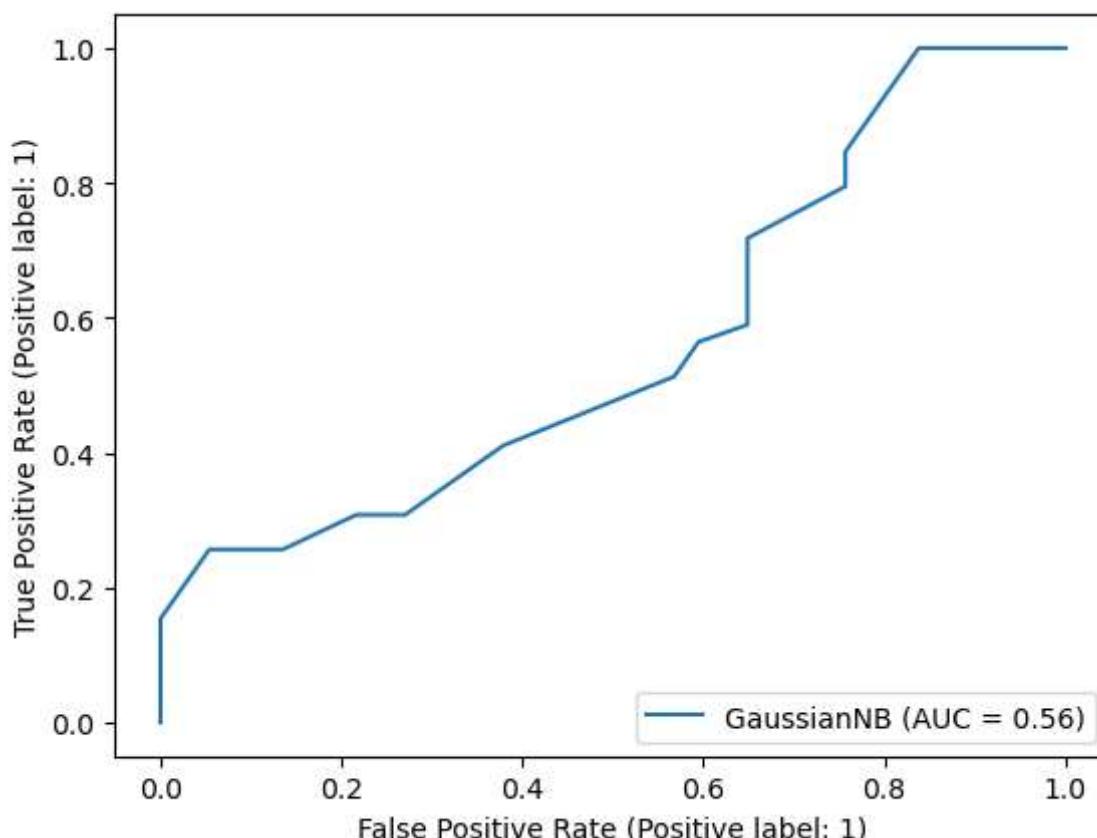
Training Accuracy Score: 0.5357142857142857

Test Accuracy Score: 0.4868421052631579

## ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.

warnings.warn(msg, category=FutureWarning)
```



- 3.3.2 Bernoulli Naive Bayes

```
In [42]: from sklearn.naive_bayes import BernoulliNB
```

```
In [43]: bnb = BernoulliNB()
bnb.fit(X_train, y_train);
```

```
bnb.score(X_test, y_test)
```

Out[43]: 0.5526315789473685

In [44]: evaluate\_model(bnb)

Classification Report				
	precision	recall	f1-score	support
HIGH	0.56	0.38	0.45	37
NORMAL	0.55	0.72	0.62	39
accuracy			0.55	76
macro avg	0.55	0.55	0.54	76
weighted avg	0.55	0.55	0.54	76

Confusion Matrix

```
[[14 23]
 [11 28]]
```

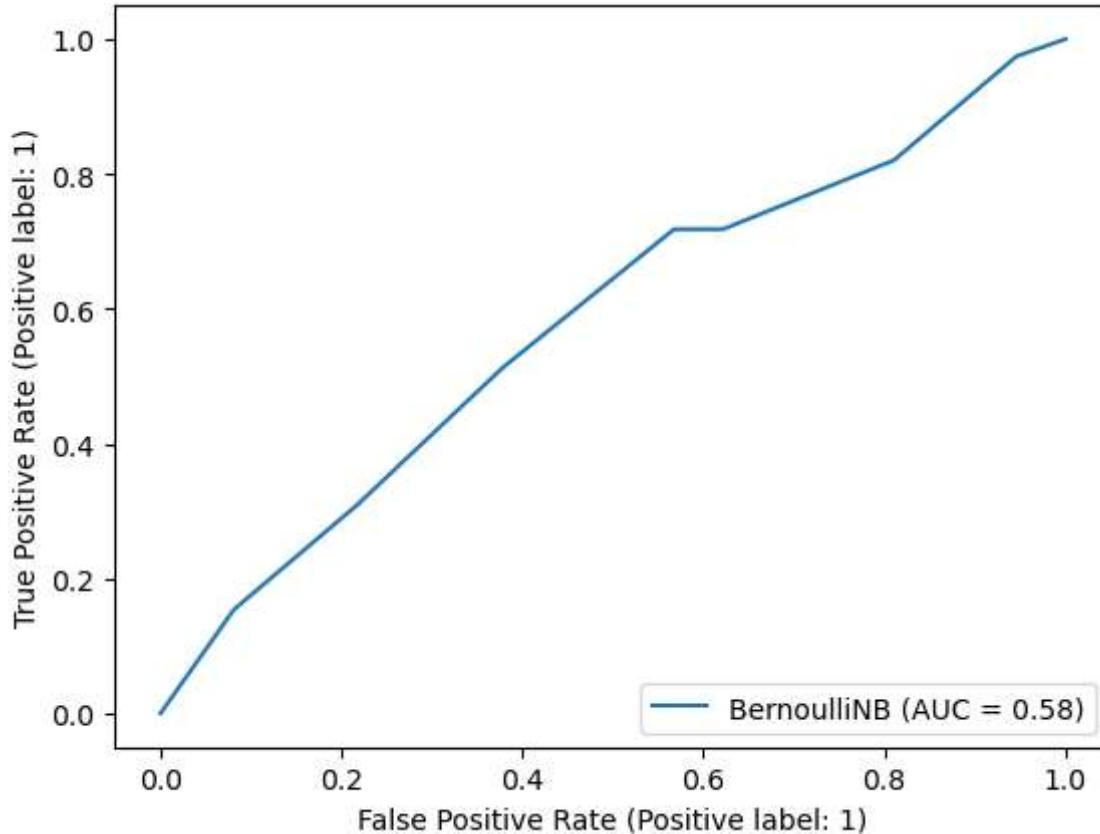
Training Accuracy Score: 0.5178571428571429

Test Accuracy Score: 0.5526315789473685

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.

warnings.warn(msg, category=FutureWarning)
```



- 3.3.3 Decision Tree

```
In [45]: from sklearn.tree import DecisionTreeClassifier
```

```
In [46]: dt = DecisionTreeClassifier()
dt.fit(X_train, y_train);
dt.score(X_test, y_test)
```

```
Out[46]: 0.631578947368421
```

```
In [47]: evaluate_model(dt)
```

Classification Report				
	precision	recall	f1-score	support
HIGH	0.62	0.65	0.63	37
NORMAL	0.65	0.62	0.63	39
accuracy			0.63	76
macro avg	0.63	0.63	0.63	76
weighted avg	0.63	0.63	0.63	76

Confusion Matrix

```
[[24 13]
 [15 24]]
```

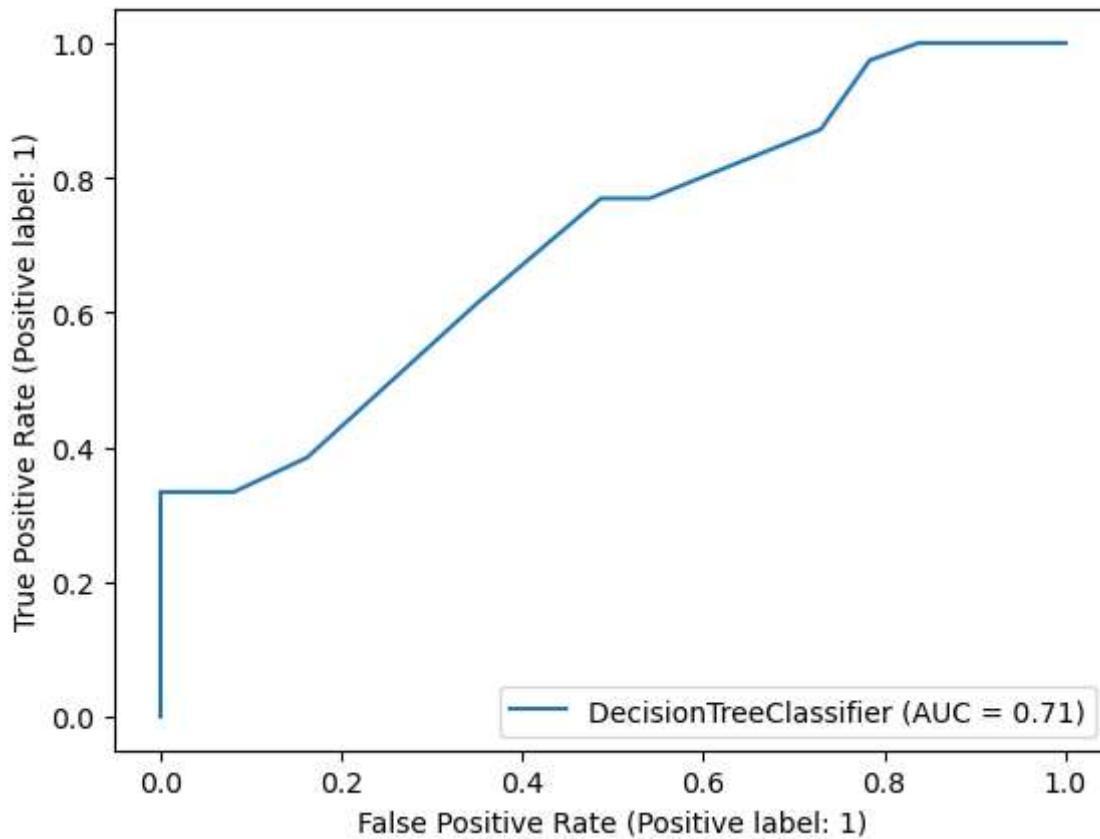
Training Accuracy Score: 0.6517857142857143

Test Accuracy Score: 0.631578947368421

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.

warnings.warn(msg, category=FutureWarning)
```



- 3.3.4 Random Forest

```
In [48]: from sklearn.ensemble import RandomForestClassifier
```

```
In [49]: rf = RandomForestClassifier()
rf.fit(X_train, y_train);
rf.score(X_test, y_test)
```

```
Out[49]: 0.6447368421052632
```

```
In [50]: evaluate_model(rf)
```

	precision	recall	f1-score	support
HIGH	0.68	0.51	0.58	37
NORMAL	0.62	0.77	0.69	39
accuracy			0.64	76
macro avg	0.65	0.64	0.64	76
weighted avg	0.65	0.64	0.64	76

Confusion Matrix

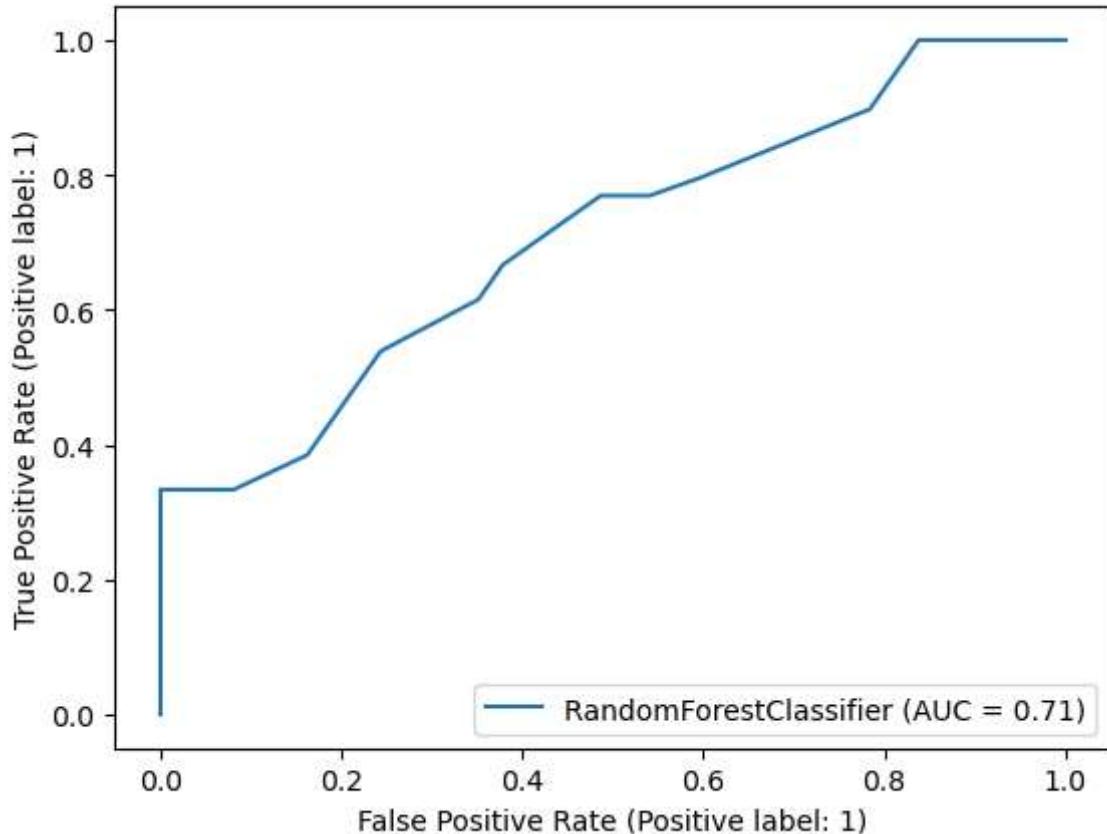
```
[[19 18]
 [ 9 30]]
```

Training Accuracy Score: 0.6517857142857143

Test Accuracy Score: 0.6447368421052632

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
warnings.warn(msg, category=FutureWarning)
```



- 3.3.5 K-Nearest Neighbors

```
In [51]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [52]: knn = KNeighborsClassifier()
knn.fit(X_train, y_train);
knn.score(X_test, y_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
    mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
Out[52]: 0.5789473684210527
```

```
In [53]: evaluate_model(knn)
```

## Classification Report

	precision	recall	f1-score	support
HIGH	0.56	0.65	0.60	37
NORMAL	0.61	0.51	0.56	39
accuracy			0.58	76
macro avg	0.58	0.58	0.58	76
weighted avg	0.58	0.58	0.58	76

## Confusion Matrix

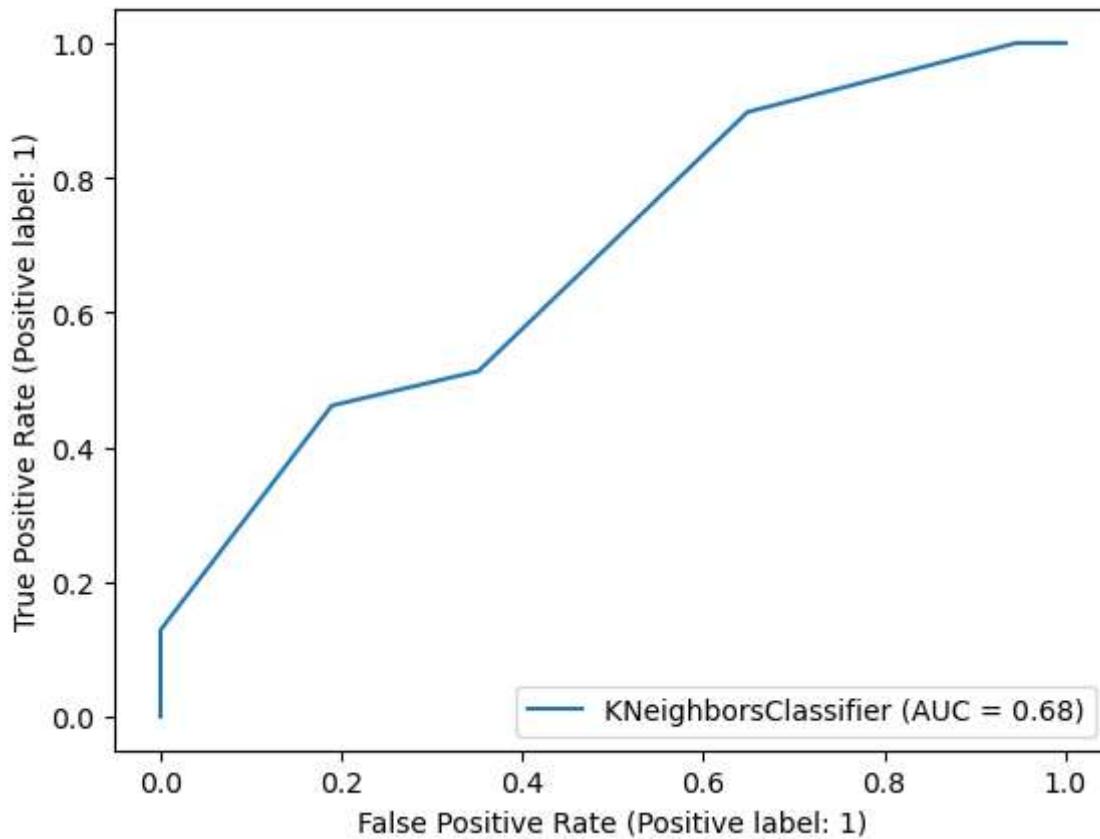
```
[[24 13]
 [19 20]]
```

Training Accuracy Score: 0.6339285714285714

Test Accuracy Score: 0.5789473684210527

## ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
    mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
    mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
    mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
    mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
    warnings.warn(msg, category=FutureWarning)
```



- 3.3.5 Logistic Regression

```
In [54]: from sklearn.linear_model import LogisticRegression
```

```
In [55]: lr = LogisticRegression()
lr.fit(X_train, y_train);
lr.score(X_test, y_test)
```

```
Out[55]: 0.4605263157894737
```

```
In [56]: evaluate_model(lr)
```

Classification Report				
	precision	recall	f1-score	support
HIGH	0.44	0.43	0.44	37
NORMAL	0.47	0.49	0.48	39
accuracy			0.46	76
macro avg	0.46	0.46	0.46	76
weighted avg	0.46	0.46	0.46	76

Confusion Matrix

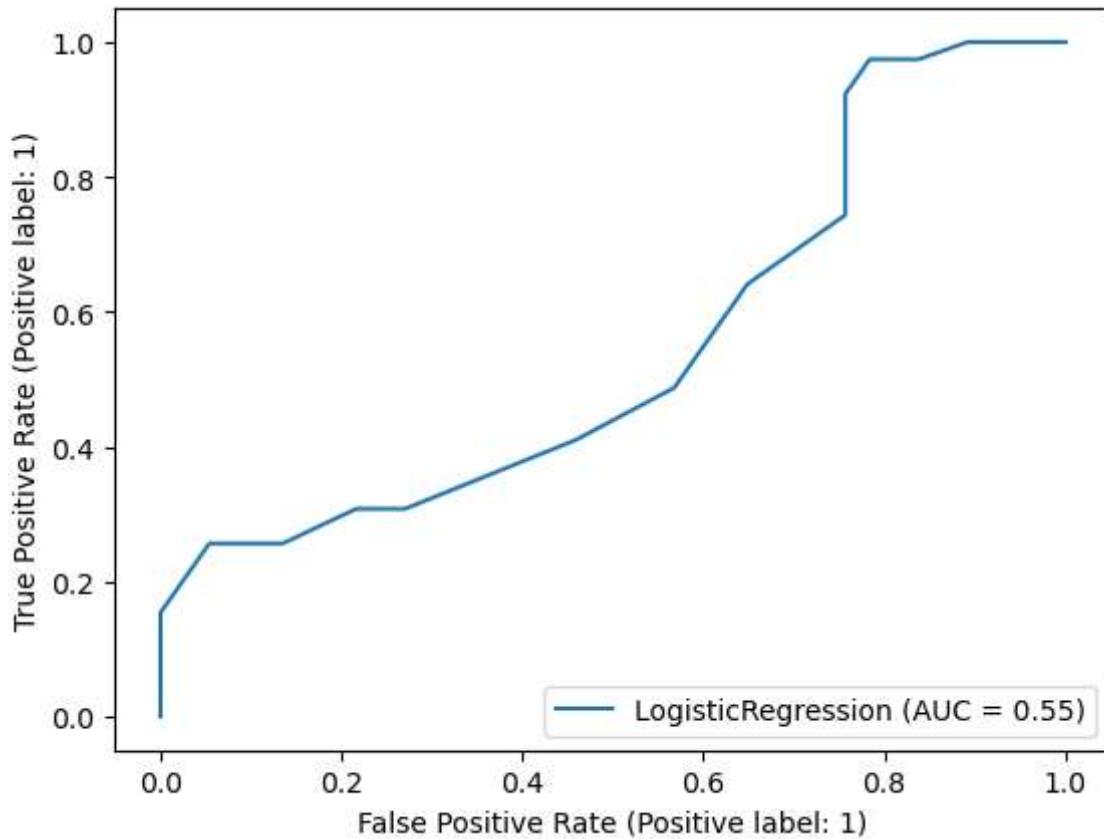
```
[[16 21]
 [20 19]]
```

Training Accuracy Score: 0.5178571428571429

Test Accuracy Score: 0.4605263157894737

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
warnings.warn(msg, category=FutureWarning)
```



## 4. Hyperparameter Tuning / Cross Validation

To tune our model into a better model

- 4.1.1 Decision Tree Classifier Hyperparameter Tuning

```
In [57]: from sklearn.model_selection import RandomizedSearchCV
```

```
In [58]: params = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [5, 10, 15, 20]
}

rsearch = RandomizedSearchCV(DecisionTreeClassifier(), params, n_iter = 15, cv = 10)
rsearch.fit(X_train, y_train)
```

```
Out[58]: RandomizedSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_iter=15,
                           param_distributions={'criterion': ['gini', 'entropy'],
                                                'max_depth': [5, 10, 15, 20],
                                                'splitter': ['best', 'random']})
```

```
In [59]: rsearch.best_params_
```

```
Out[59]: {'splitter': 'best', 'max_depth': 5, 'criterion': 'gini'}
```

```
In [60]: dt_tuned = DecisionTreeClassifier(  
        splitter ='random',  
        max_depth = 5,  
        criterion = 'entropy'  
)
```

```
In [61]: dt_tuned.fit(X_train, y_train)
```

```
Out[61]: DecisionTreeClassifier(criterion='entropy', max_depth=5, splitter='random')
```

```
In [62]: evaluate_model(dt_tuned)
```

Classification Report				
	precision	recall	f1-score	support
HIGH	0.62	0.65	0.63	37
NORMAL	0.65	0.62	0.63	39
accuracy			0.63	76
macro avg	0.63	0.63	0.63	76
weighted avg	0.63	0.63	0.63	76

Confusion Matrix

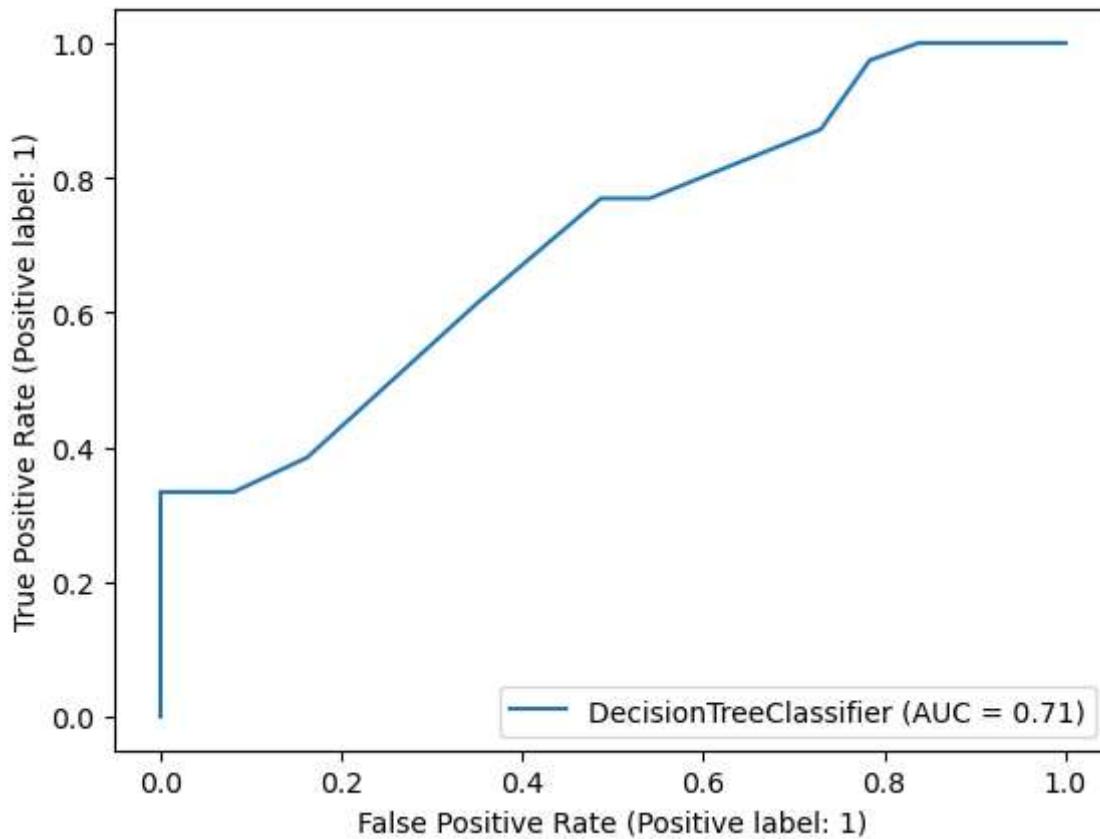
```
[[24 13]  
 [15 24]]
```

Training Accuracy Score: 0.6517857142857143

Test Accuracy Score: 0.631578947368421

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:  
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is depre  
cated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklear  
n.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDisplay.f  
rom_estimator`.  
warnings.warn(msg, category=FutureWarning)
```



- 4.4.2 Random Forest Classifier Hyperparameter Tuning

```
In [63]: rf.get_params()
```

```
Out[63]: {'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_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
In [64]: params = {
```

```
    'criterion': ['gini', 'entropy'],
    'n_estimators': [5, 10, 15, 20, 30, 50, 100],
    'max_depth': [5, 10, 15, 20]
}
```

```
rsearch_rf = RandomizedSearchCV(RandomForestClassifier(), params, n_iter = 15, cv = 10)
rsearch_rf.fit(X_train, y_train)
```

```
Out[64]: RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(), n_iter=15,
                           param_distributions={'criterion': ['gini', 'entropy'],
                           'max_depth': [5, 10, 15, 20],
                           'n_estimators': [5, 10, 15, 20, 30, 50,
                           100]})
```

```
In [65]: rsearch_rf.best_params_
```

```
Out[65]: {'n_estimators': 100, 'max_depth': 10, 'criterion': 'entropy'}
```

```
In [66]: rf_tuned = RandomForestClassifier(
            n_estimators = 15,
            max_depth = 20,
            criterion = 'entropy'
        )
```

```
In [67]: rf_tuned.fit(X_train, y_train)
```

```
Out[67]: RandomForestClassifier(criterion='entropy', max_depth=20, n_estimators=15)
```

```
In [68]: evaluate_model(rf_tuned)
```

Classification Report		precision	recall	f1-score	support
HIGH		0.63	0.73	0.68	37
NORMAL		0.70	0.59	0.64	39
accuracy				0.66	76
macro avg		0.66	0.66	0.66	76
weighted avg		0.66	0.66	0.66	76

Confusion Matrix

```
[[27 10]
 [16 23]]
```

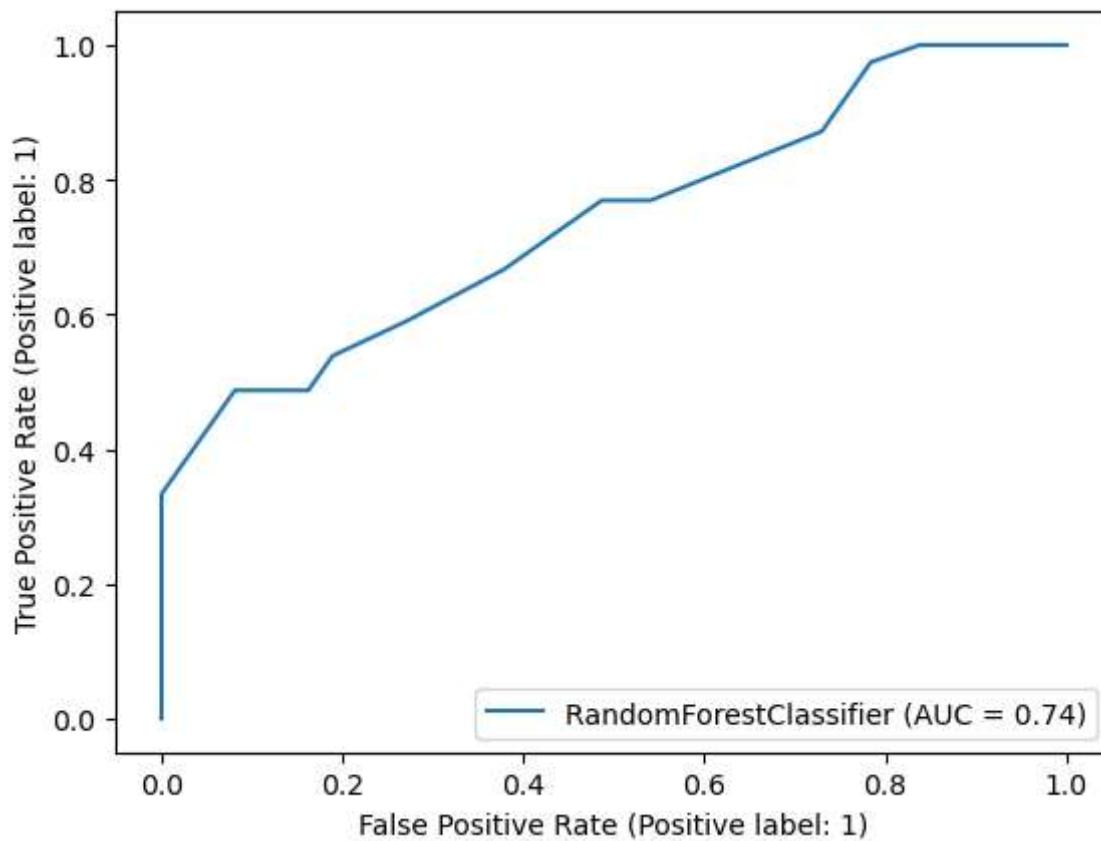
Training Accuracy Score: 0.6428571428571429

Test Accuracy Score: 0.6578947368421053

ROC Curve

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.

warnings.warn(msg, category=FutureWarning)
```



## 5. Choosing a model / conclusion

- Decision Tree Classifier is my model selection since it has the highest AUC which is 71% and a close value of training and test accuracy, which is 65% and 63%. I think that this is the best model.

In [ ]: