Hyperparameter Tuning

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

```
heart=pd.read_csv('Datasets/Heart.csv')
heart.head()
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	N ve
0	70	1	4	130	322	0	2	109	0	2.4	2	
1	67	0	3	115	564	0	2	160	0	1.6	2	
2	57	1	2	124	261	0	0	141	0	0.3	1	

```
heart.tail()
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST
265	52	1	3	172	199	1	0	162	0	0.5	1
266	44	1	2	120	263	0	0	173	0	0.0	1
267	56	0	2	140	294	0	2	153	0	1.3	2

```
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       270 non-null    int64
 1   Sex                       270 non-null    int64
```

```
 2   Chest pain type          270 non-null    int64
 3   BP                       270 non-null    int64
 4   Cholesterol              270 non-null    int64
 5   FBS over 120             270 non-null    int64
 6   EKG results              270 non-null    int64
 7   Max HR                   270 non-null    int64
 8   Exercise angina          270 non-null    int64
 9   ST depression            270 non-null    float64
 10  Slope of ST              270 non-null    int64
 11  Number of vessels fluro  270 non-null    int64
 12  Thallium                 270 non-null    int64
 13  Heart Disease            270 non-null    object
dtypes: float64(1), int64(12), object(1)
memory usage: 29.7+ KB
```

Our target variable is the "Heart Disease" column, but it is an object datatype, so we have to do one-hot encoding to make it a binary variable

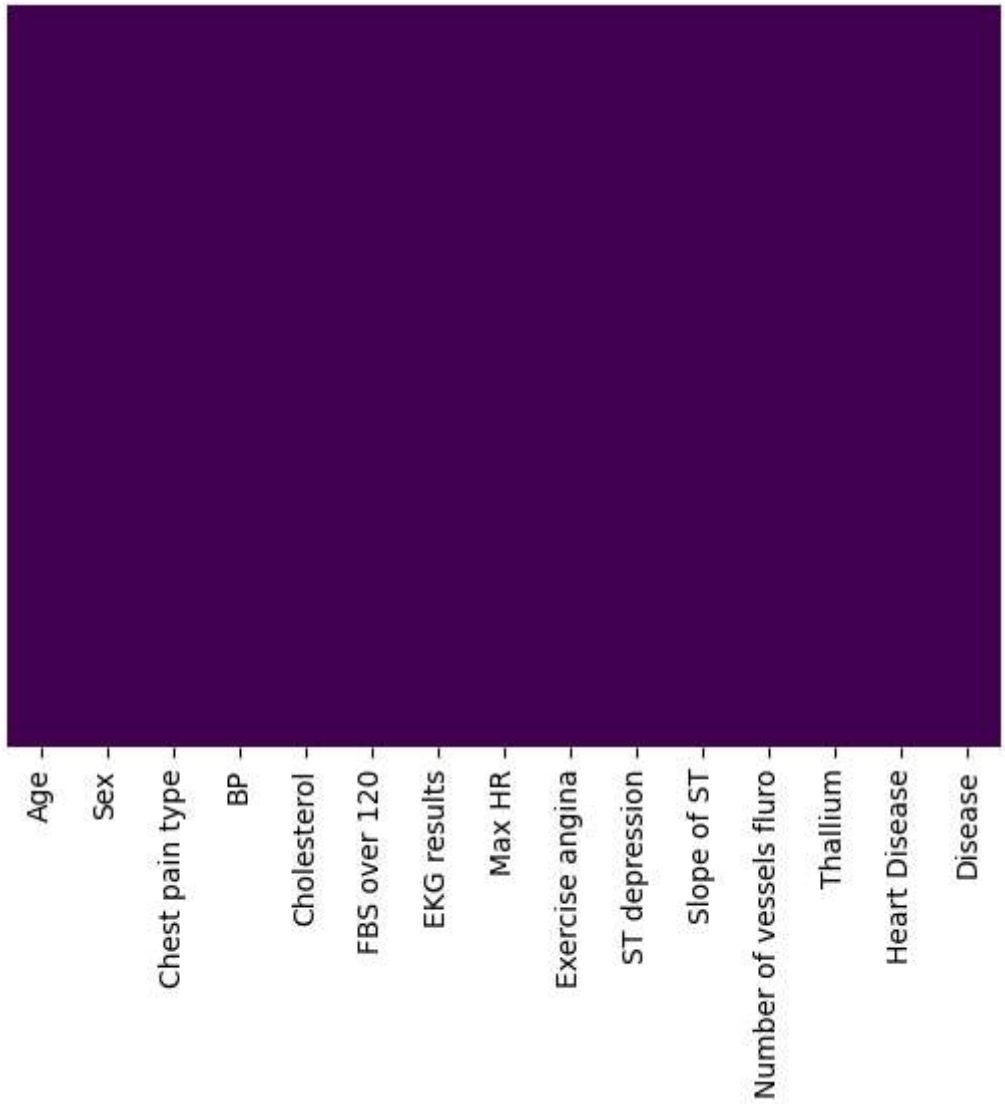
```
#manual encoding
heart['Disease']=heart['Heart Disease'].apply(lambda x: 1 if x == 'Presence' else 0)
```

```
heart.head()
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	N ve
0	70	1	4	130	322	0	2	109	0	2.4	2	
1	67	0	3	115	564	0	2	160	0	1.6	2	
2	57	1	2	124	261	0	0	141	0	0.3	1	
3	64	1	4	128	263	0	0	105	1	0.2	2	

```
sns.heatmap(heart.isnull(), yticklabels = False, cbar = False, cmap = 'viridis')
```

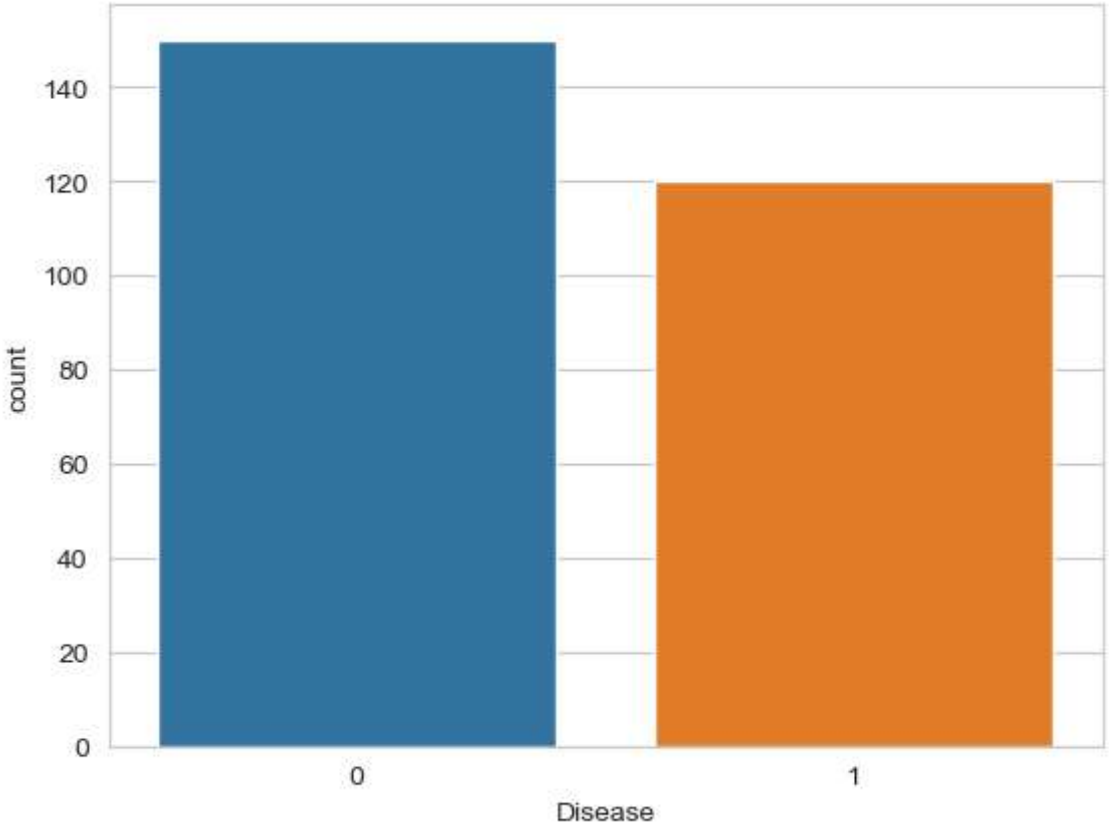
```
<AxesSubplot:>
```



The Dataset is extremely clean

```
sns.set_style('whitegrid')
sns.countplot(x = 'Disease', data = heart)
```

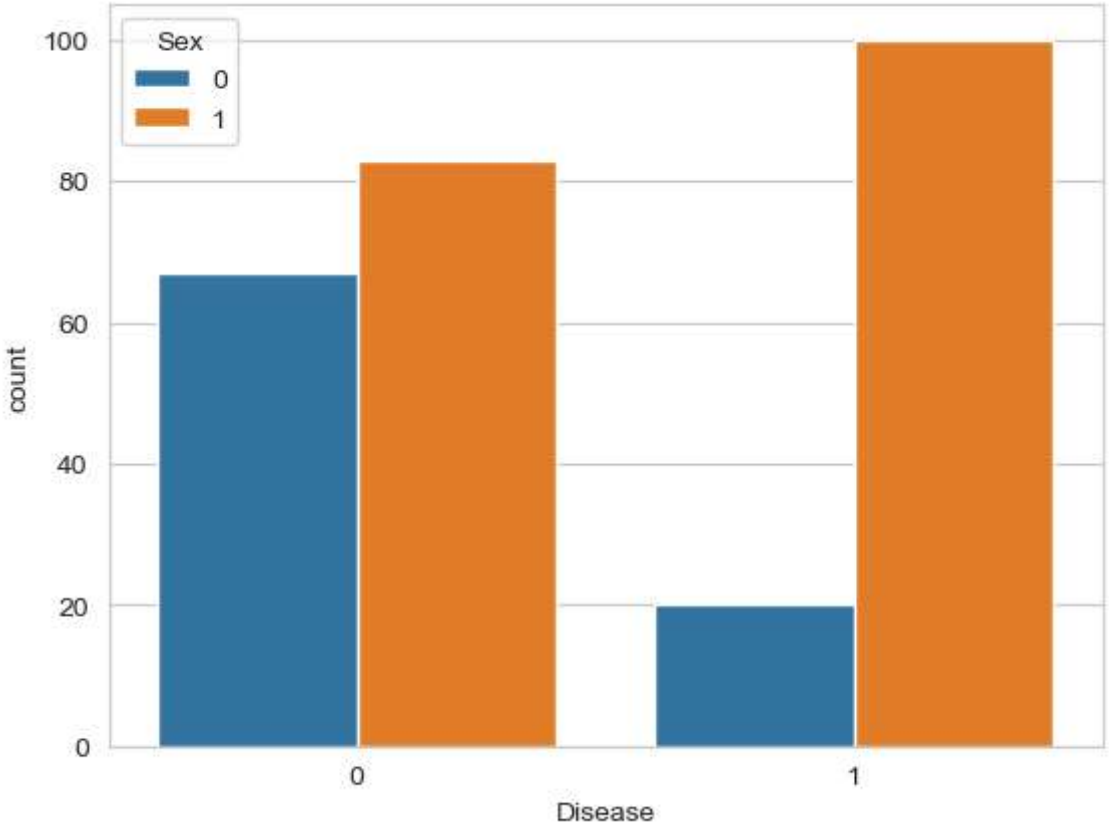
```
<AxesSubplot:xlabel='Disease', ylabel='count'>
```



We can see the number of people who had heart disease and who didn't

```
sns.countplot(x = 'Disease', data = heart,hue='Sex')
```

```
<AxesSubplot:xlabel='Disease', ylabel='count'>
```

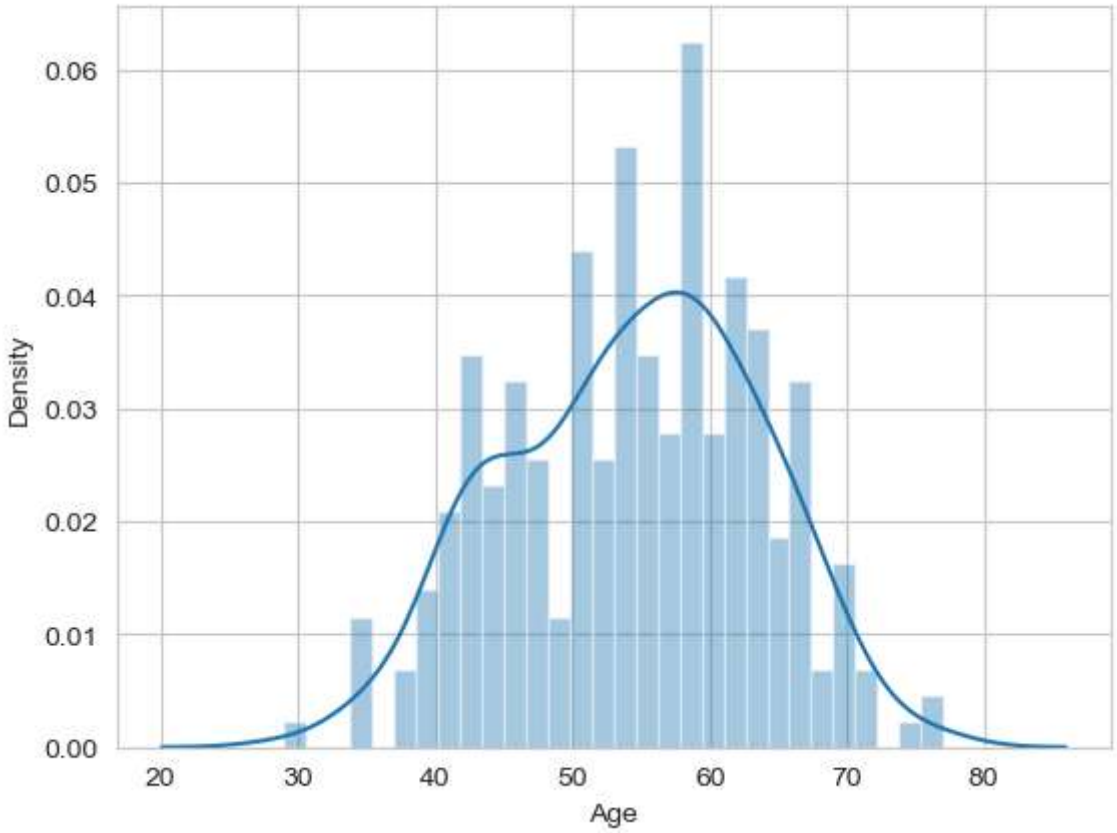


We can see the disease rate as per sex

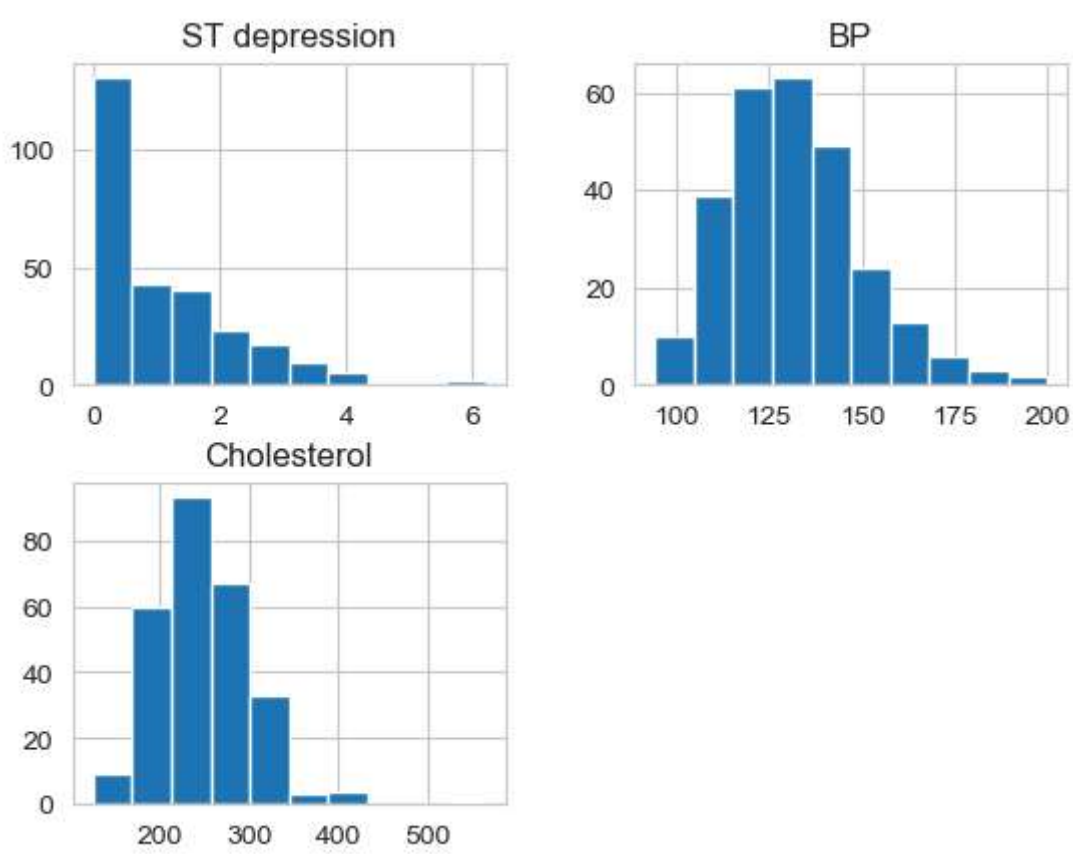
as per this plot Males have a higher rate of heart disease

```
#distribution of Age in the dataset
sns.distplot(heart['Age'].dropna(), kde = True, bins = 30)
```

```
C:\Users\geete\anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='Age', ylabel='Density'>
```

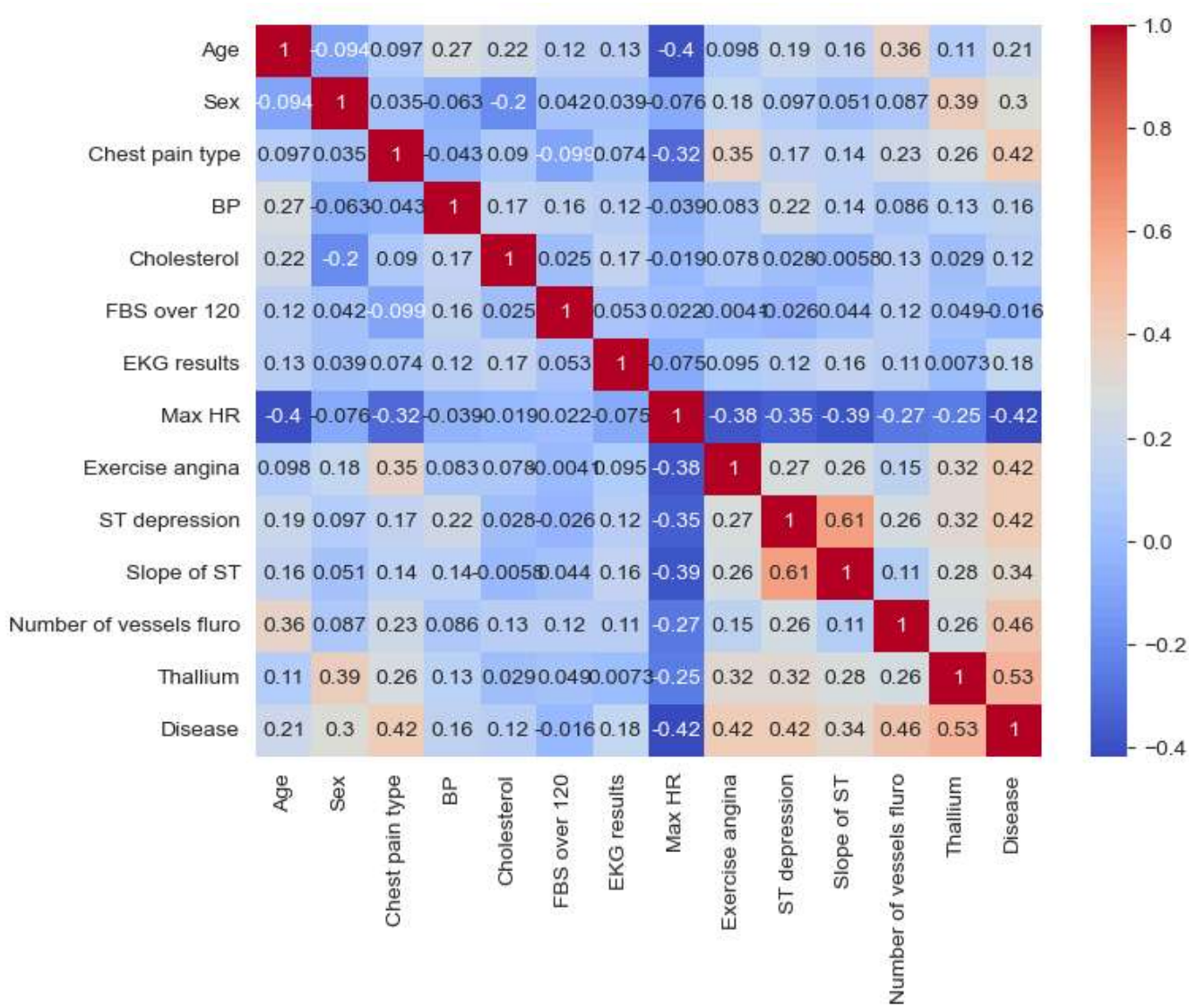



```
heart.hist(['ST depression','BP','Cholesterol'])
plt.show()
```



We don't need to make the data normal because in logistic regression normality of the data is not an assumption

```
plt.figure(figsize=(8, 6))
sns.heatmap(heart.corr(),annot=True,cmap='coolwarm')
plt.show()
```



There is no Multicollinearity in our Data

Logistic Regression Model

```
X=heart.drop(['Disease','Heart Disease'], axis = 1)
y= heart['Disease']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size = 0.3, random_state=101)
```

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
```

```
C:\Users\geete\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: Converg
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
  n_iter_i = _check_optimize_result(
LogisticRegression()
```

```
predictions = logmodel.predict(X_test)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.81      0.95      0.88        41
           1       0.94      0.78      0.85        40

    accuracy                           0.86        81
   macro avg       0.88      0.86      0.86        81
weighted avg       0.88      0.86      0.86        81
```

The Precision for prediction of Absence and Presence of Heart Disease is 81% in the case of absence of disease and 94% in the presence of disease, so out of all the people who had heart disease according to the model's prediction, 94% actually had heart diease out of them.
The sensitivity of the model in the prediction of Absence of disease is 95% and in the presence of disease it is 78%, so 95% people were correctly identified as not having the presence of a disease.
Support is the number of actual accurences, in our testing data, 41 did not actually have a disease and 40 had a disease.
The accuracy is 0.86 so 86% of predictions were correct.
In the above graph AUC is 0.86 which is very close to 1, therefore our model has a good measure of separability

```
train_score = logmodel.score(X_train, y_train)
test_score = logmodel.score(X_test, y_test)
print("Training Score (Accuracy):", train_score)
print("Testing Score (Accuracy):", test_score)
```

```
Training Score (Accuracy): 0.8835978835978836
Testing Score (Accuracy): 0.8641975308641975
```

There is almost no variance in our scores therefore, there is no overfitting

```
oldacc_logreg = round( metrics.accuracy_score(y_test, predictions) * 100, 2 )
print( 'Accuracy of Logistic Regression model : ', oldacc_logreg )
```

```
Accuracy of Logistic Regression model :  86.42
```

```
#for hyper parameter tuning we can do cross validation, since we can tune the folds
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=10, scoring='accuracy')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
acc_logreg = round( metrics.accuracy_score(y_test, predictions) * 100, 2 )
print( 'Accuracy of Logistic Regression model : ', acc_logreg )
```

```
Accuracy of Logistic Regression model :  82.72
```

Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train,y_train)
```

```
GaussianNB()
```

```
y_pred = model.predict(X_test)
```

```
oldacc_nb = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of Gaussian Naive Bayes model : ', acc_nb )
```

```
Accuracy of Gaussian Naive Bayes model :  85.19
```

```
#for hyper parameter tuning we can do cross validation, since we can tune the folds
scores = cross_val_score(model, X, y, cv=8, scoring='accuracy')

model.fit(X_train, y_train)

# Make predictions
```

```
y_pred = model.predict(X_test)

# Calculate accuracy
acc_nb = round(metrics.accuracy_score(y_test, y_pred) * 100, 2)
print('Accuracy of Gaussian Naive Bayes model:', acc_nb)
```

```
Accuracy of Gaussian Naive Bayes model: 85.19
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)


oldacc_dt = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of Decision Tree model : ', acc_dt )
```

```
Accuracy of Decision Tree model :  81.48
```

```
clf = DecisionTreeClassifier()


# Hyperparameter Optimization
parameters = {'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10, 50],
              'min_samples_split': [2, 3, 50, 100],
              'min_samples_leaf': [1, 5, 8, 10]
             }

# Runing the grid search
grid_obj = GridSearchCV(clf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

# Seting the clf to the best combination of parameters
clf = grid_obj.best_estimator_

# Training the model using the training sets
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=50, max_features='log2', min_samples_leaf=8,
                       min_samples_split=50)
```

```
y_pred = clf.predict(X_test)
```

```
acc_dt = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of Decision Tree model : ', acc_dt )
```

```
Accuracy of Decision Tree model :  81.48
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
oldacc_rf = round( metrics.accuracy_score(y_test, y_pred) * 100 , 2 )
print( 'Accuracy of Random Forest model : ', oldacc_rf )
```

```
Accuracy of Random Forest model :  82.72
```

```
rf = RandomForestClassifier()

# Hyperparameter Optimization
parameters = {'n_estimators': [4, 6, 9, 10, 15],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1, 5, 8]
             }

grid_obj = GridSearchCV(rf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

rf = grid_obj.best_estimator_

rf.fit(X_train,y_train)
```

```
RandomForestClassifier(max_depth=5, min_samples_leaf=5, min_samples_split=3,
                       n_estimators=15)
```

```
y_pred = rf.predict(X_test)
```

```
acc_rf = round( metrics.accuracy_score(y_test, y_pred) * 100 , 2 )
print( 'Accuracy of Random Forest model : ', acc_rf )
```

```
Accuracy of Random Forest model :  82.72
```

Support Vector Machine

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from sklearn import svm

svc = svm.SVC()
svc.fit(X_train,y_train)
```

```
SVC()
```

```
y_pred = svc.predict(X_test)
```

```
oldacc_svc = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of SVC model : ', oldacc_svc )
```

```
Accuracy of SVC model :  82.72
```

```
parameters = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]

grid_obj = GridSearchCV(svc, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

svc = grid_obj.best_estimator_

svc.fit(X_train,y_train)
```

```
SVC(C=10, gamma=0.001)
```

```
y_pred = svc.predict(X_test)
```

```
acc_svm = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
print( 'Accuracy of SVM model : ', acc_svm )
```

```
Accuracy of SVM model :  85.19
```

Model Comparison Table

```
models = pd.DataFrame({
    'OldModel': ['Logistic Regression', 'Naive Bayes', 'Decision Tree', 'Random Forest', 'Su
    'Score': [oldacc_logreg, oldacc_nb, oldacc_dt, oldacc_rf, oldacc_svc],
    'New_Score':[acc_logreg, acc_nb, acc_dt, acc_rf, acc_svm]})
models.sort_values(by='Score', ascending=False)
```

	OldModel	Score	New_Score
0	Logistic Regression	86.42	82.72
1	Naive Bayes	85.19	85.19
3	Random Forest	82.72	82.72
4	Support Vector Machines	82.72	85.19
2	Decision Tree	74.07	81.48

From the above table, we notice that **Logistic Regression is best for our data** as it has the highest accuracy of 86.42. in most models, the accuracy increased after hyperparameter tuning, for example, in Decision Tree and in Support Vector Machines, but it didn't increase in Logistic Regression.

+ Code + Text

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.