# MACHINE LEARNING PROJECT

## HEART DISEASE PREDICTION

Project Members
WASILA REHMAN -12348
SYED BILAL RIZWAN - 23943

21-05-2022

# Contents

# 1. Introduction

## 1.1 Problem Statement

According to the World Health Organization, every year 12 million deaths occur worldwide due to heart diseases. Heart disease is one of the major causes of mortality among the population of the world.

The major challenge in heart disease is its detection. An early diagnosis of heart disease can help us make lifestyle changes in high-risk patients and reduce the chances of future complications.

There are instruments available which can predict heart disease but either are expensive or not fast enough to calculate chance of heart disease in humans.

That is why our goal in this project is to classify whether a person has heart disease with maximum achievable accuracy. We also aim to detect the top 3 reasons that explain the outcome of our model for a given test case.

## 1.2 Dataset Information

Originally, the dataset come from the CDC and is a major part of the Behavioral Risk Factor Surveillance System (BRFSS), which conducts annual telephone surveys to gather data on the health status of U.S. residents.

This dataset is already cleaned and has 319795 rows × 18 columns.

Following are the features present in this dataset:

1. **HeartDisease:** Respondents that have ever reported having coronary heart disease (CHD) or myocardial infarction (MI).
2. **BMI:** Body Mass Index (BMI).
3. **Smoking:** Have you smoked at least 100 cigarettes in your entire life?
4. **AlcoholDrinking:** Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week
5. **Stroke:** (Ever told) (you had) a stroke?
6. **PhysicalHealth:** Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? (0-30 days).
7. **MentalHealth:** Thinking about your mental health, for how many days during the past 30 days was your mental health not good? (0-30 days).
8. **DiffWalking:** Do you have serious difficulty walking or climbing stairs?
9. **Sex:** Are you male or female?
10. **AgeCategory:** Fourteen-level age category.
11. **Race:** Imputed race/ethnicity value.
12. **Diabetic:** (Ever told) (you had) diabetes?
13. **PhysicalActivity:** Adults who reported doing physical activity or exercise during the past 30 days other than their regular job.
14. **GenHealth:** Would you say that in general your health is...

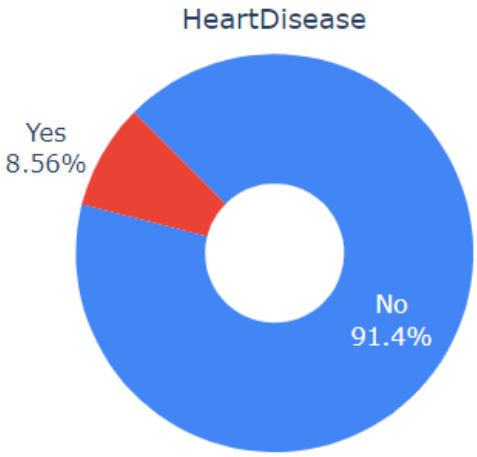
15. **SleepTime:** On average, how many hours of sleep do you get in a 24-hour period?
16. **Asthma:** (Ever told) (you had) asthma?
17. **KidneyDisease:** Not including kidney stones, bladder infection or incontinence, were you ever told you had kidney disease?
18. **SkinCancer:** (Ever told) (you had) skin cancer?

## 1.3 Potential Beneficiaries

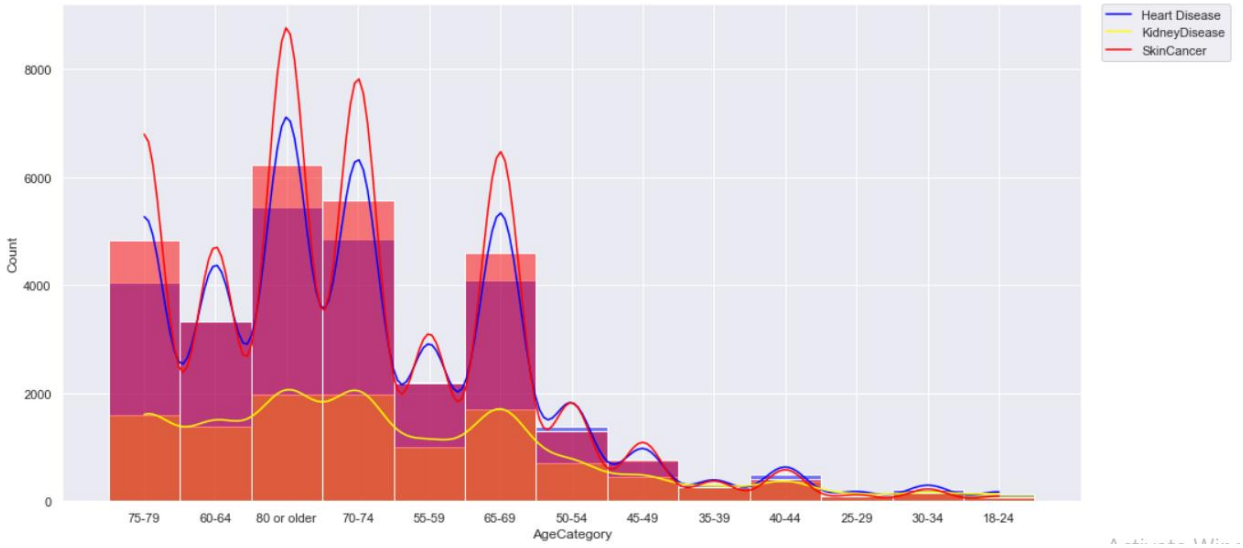
The potential beneficiaries for our project can be doctors, medical research institutes, and patients.

- Since it is not possible for a doctor to consult patients 24/7 and a complete diagnosis can take more than 24 hours as it requires multiple medical tests, getting baseline results early on can help both patients and doctors.
- The hidden patterns that our project uncovers can be used for health diagnosis in medicinal data.
- Medical instrument's developers can also utilize our algorithms to improve the accuracy of their systems and reduce the time taken to provide results.

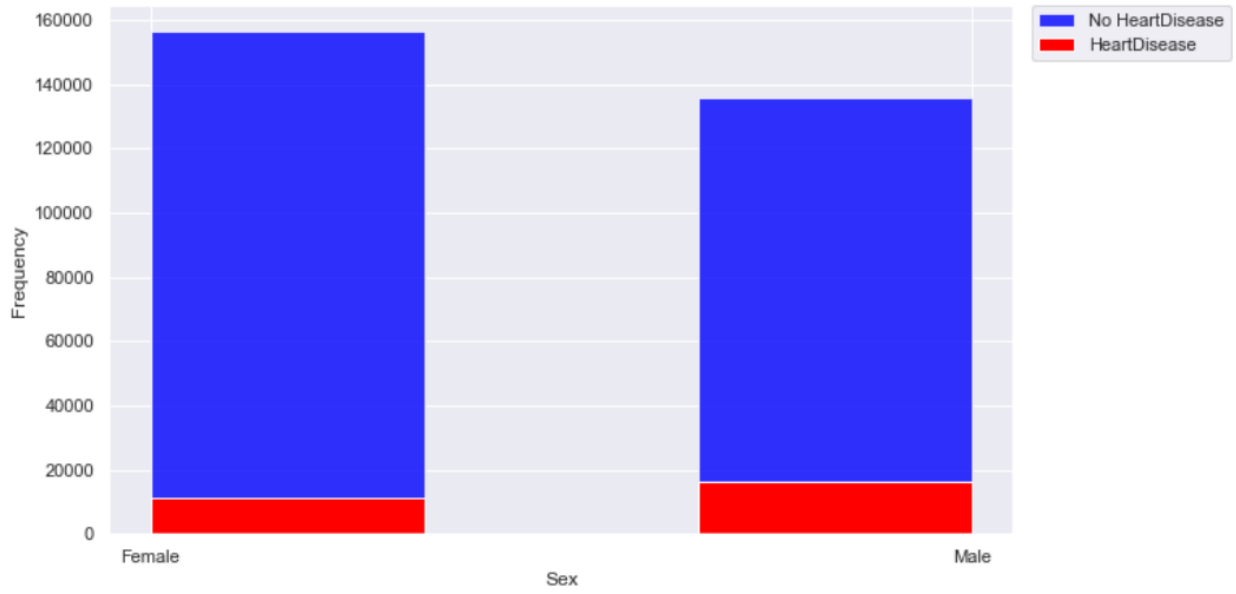
## 2. Exploratory Data Analysis




- The independent variable is highly imbalanced with only 9% of the patients having heart disease.




- There is a common pattern: as age groups increase the occurrence of skin cancer, heart disease, and kidney disease increase.
- Majority of the records in our dataset have age over 60




Probability of Heart Diseases in Males: 0.106

Probability of Heart Diseases in Females: 0.067

- More heart disease patients are Males than Females
- There are more Females in the dataset than Males
- The probability of Males having heart disease is higher than Females

# 3. Pre-Processing

Although the dataset we used was available in its cleanest form we used multiple Pre-processing techniques to ensure the best accuracy.

## 3.1 Removing Null Values

Since there were no null values in the dataset, zero rows were removed.

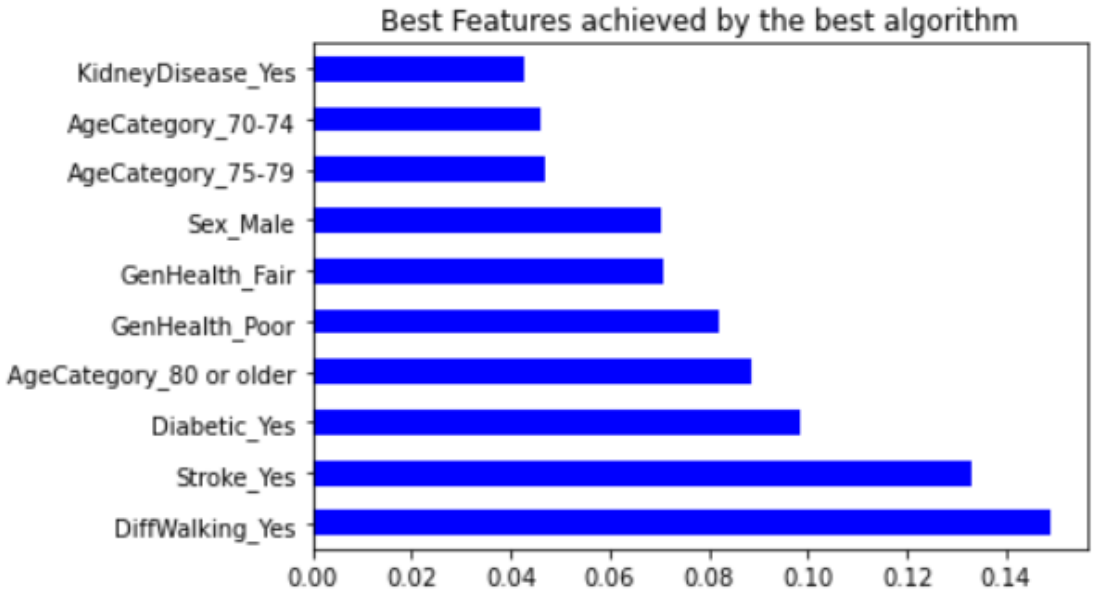
## 3.2 One-Hot Encoding & Scaling

We performed one hot encoding and MinMax scaling to transform the dataset. After encoding our dataset's shape changed to **319795 rows × 37 columns**.

We also transformed the dependent variable into Yes = 1, No = 0 using Categorical Codes.

## 3.3 Feature Reduction

Since gradient boosting gave the best results on baseline models, we tried feature reduction using Gradient Boosting feature importance to get top 18 and 14 features based on their importance.




However, reducing features in the actual dataset decreased our ROC-AUC slightly. Here are the ROC-AUC scores for each case:

- Best ROC-AUC of GB with 18 Features: 0.839
- Best ROC-AUC of GB with 14 Features: 0.841
- **Best ROC-AUC of GB with All Features: 0.8422**

As we can conclude from the confusion matrices below, Feature reduction does not help GB at all. All features are important in our dataset and our ROC AUC of GB with 0.8422 works best.

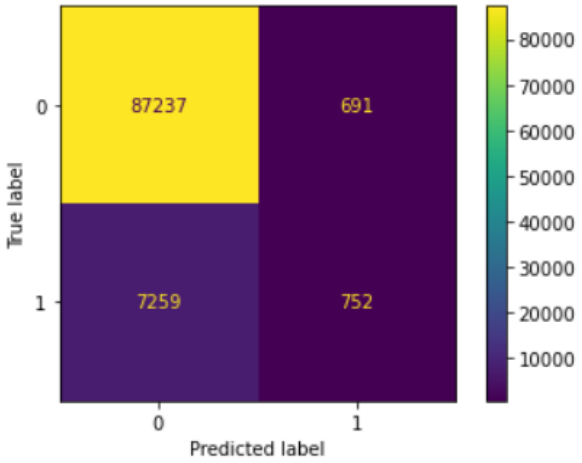



*Figure 1: CONFUSION MATRIX OF GB WITH TOP 18 FEATURES*

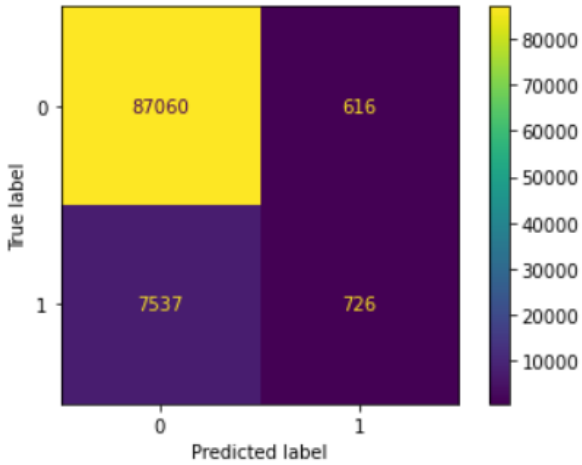



*Figure 2: CONFUSION MATRIX OF GB WITH TOP 14 FEATURES*

## 3.4 Sampling Techniques (SMOTE with Undersampling)

Since our dataset was highly imbalanced, we experimented with multiple sampling techniques to create a uniform dataset. We divided our dataset into Train/test split and then applied SMOTE with various conditions on the train dataset. Since Gradient Boosting was our best-performing model we tested SMOTE's validity on Gradient Boosting models.

- SMOTE with random state = 2; **ROC-AUC = 0.81**
- SMOTE with random state = 2, oversampling = 0.3; **ROC-AUC = 0.834**
- SMOTE with 0.1 Oversampling, 0.5 Undersampling; **ROC-AUC = 0.841**

From the above results, we concluded that smote with 0.1 oversampling, and 0.5 under sampling performed the best. After finalizing this threshold for SMOTE, we then experimented with multiple models to determine the best one.

# 4. Methodology

## 4.1 Evaluation Metrics

Following were the evaluation metrics used in this project:

### 4.1.1 ROC-AUC

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. AUC – ROC is most useful for imbalanced class problem where accuracy is not a good metric which is why this metric was used to identify our best model throughout the project.

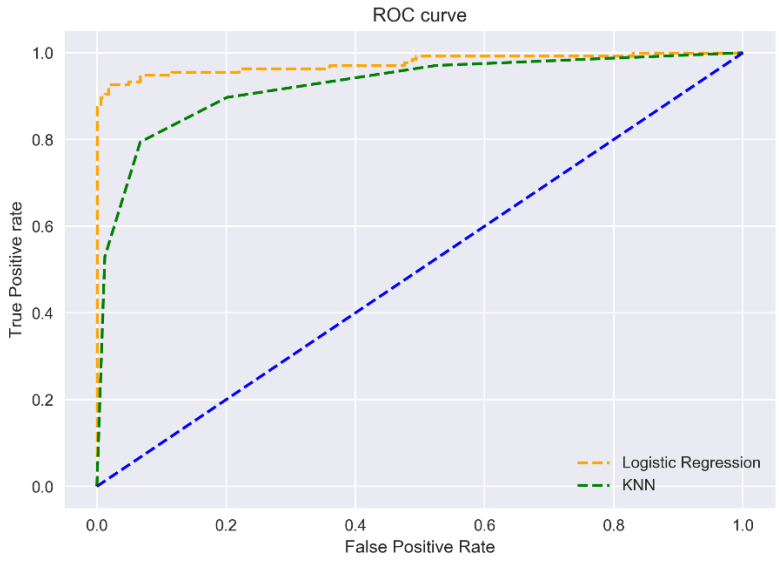



*Figure 3: HYPOTHETICAL EXAMPLE OF ROC-AUC*

Confusion Matrix for our project:

| | | Predicted | |
|---|---|---|---|
| | | 0 = No Heart Disease | 1 = Heart Disease |
| **Actual** | 0 = No Heart Disease | True Negative | False Positive |
| | 1 = Heart Disease | False Negative | True Positive |

$$\text{Recall / TPR} = \frac{TP}{TP+FN}$$

$$\text{Specificity / TNR} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

*Figure 4: CONFUSION MATRIX LABELS FOR THIS PROJECT*

### 4.1.2 Recall/TPR

The Recall is the measure of our model correctly identifying True Positives. Thus, for all the patients who have heart disease, Recall tells us how many we correctly identified as having a heart disease.

In our project, this metric was used as the evaluation criteria to select best probability threshold.

### 4.1.3 Precision

Precision is the ratio between the True Positives and all the Positives. For our problem statement, which would be the measure of patients that we correctly identify having a heart disease out of all the patients having it.

### 4.1.4 Specificity/TNR

The specificity of a test, also referred to as the true negative rate (TNR), is the proportion of samples that are genuinely negative that give a negative result using the test in question.

In our project, the model that gave us the optimum TPR, and TNR was selected as the winner model.

**After finalizing the final model using AUC-ROC, TPR (Recall) was given more weightage than TNR to decide probability threshold. Since in our project, we cannot afford False Negatives as its highly dangerous to patients, whereas False Positives might just increase cost of diagnosis for some patients.**

## 4.2 Creating ROC Curve for Base Cases

The ROC curve helped us in deciding which models we should focus on for the rest of the project.

## 4.3 Model Experimentation

We created models in 4 phases. All our Models were built with **Repeated StratifiedKFold CV** since data was imbalance, and all parameters were selected based on GridSearchCV best estimators.

### 4.3.1 Stage 1 – Models on Sample Scaled Dataset

Next, we started creating models on complete dataset, but it took us more than 20 minutes to run a basic KNN model.

**So, we decided to train our models on a sample of dataset (0.35) first as it takes less time.** The sample consisted of 8.6% of heart disease patients so it was fairly imbalance and a true representative of our dataset.

Models created on sample scaled dataset:

- KNN Classifier
- Gradient Boosting Classifier
- Radom Forest Classifier

### 4.3.2 Stage 2 – Models on Full Dataset

After we decided the best parameters of models in stage 1, we implemented them again on the full dataset.

Models built on full dataset:

- KNN Classifier
- Gradient Boosting Classifier

- Radom Forest Classifier

### 4.3.3 Stage 3 – Models with SMOTE (Train/Test)

Our models on full dataset with best hyperparameters had high ROC-AUC scores but the Recall was extremely poor. That is why we used SMOTE balancing on train dataset and then tested on the Validation dataset. We experimented with multiple SMOTE thresholds as shown in Pre-processing.

Models built in this stage had better Recall with a slight compromise on ROC-AUC score.

Models built on full dataset with SMOTE (train/test split):

- KNN Classifier
- Gradient Boosting Classifier
- Radom Forest Classifier
- Logistic Regression
- Decision Tree Classifier
- XGB Classifier
- Stacking Classifier
- Voting Classifier

### 4.3.4 Stage 4 – Models with Probability Threshold

After training models on SMOTE and seeing significant improvement in Recall, we experimented with Probability thresholds on our best model derived from stage 3. But this time we did not use SMOTE on the dataset as it was reducing our ROC-AUC slightly.

Using probability threshold, without SMOTE resulted in the best ROC-AUC, with good Recall and accuracy scores. There were only two models tried in this stage because it was established earlier from results that Voting Classifier got the best results. Sequential Model on neural network was also tried to see whether it could beat our voting classifier.

Models tried in this stage:

- Neural Networks
- Voting Classifier

## 4.4 Interpretability through LIME

We used LIME Interpreter (Local Surrogate Model) to provide the top positively correlated features with heart disease prediction. Whenever a prediction was made using our best model, lime would give top 3 reasons to why it thinks the model predicted that particular outcome. Lime model used Multiple Linear Regression with Lasso Path feature selection.

## 4.5 Deployment on Streamlit

Finally, the best performing model was deployed on Streamlit. A pickle file was made for our best trained model and Streamlit app script was written in a separate .py file.

# 5. Results

## 5.1 Best Model

After experimenting with around 100+ machine learning models, the best ROC-AUC score was obtained on Voting Classifier consisting of 12 base models. The evaluation metric used was ROC-AUC, Recall, and specificity. Where more weight was given to TPR or Recall. Here is a step-by-step code review for the best model:

**Importing Important Libraries ¶**

```
from sklearn.linear_model import  LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, VotingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_auc_score,  confusion_matrix, ConfusionMatrixDisplay
import numpy as np
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import warnings
warnings.filterwarnings('ignore')

%load_ext autotime
```

After importing libraries and loading the dataset, pre-processing was done. We did not apply SMOTE or scaled the dataset in this model.

```
full_df["HeartDisease"] = pd.Categorical(full_df["HeartDisease"]).codes
X_full = full_df.drop(columns = ['HeartDisease'])
y_full = full_df[['HeartDisease']]
x_onehot_full = pd.get_dummies(X_full, drop_first = True)
x_onehot_full.columns = x_onehot_full.columns.str.replace(' ','_')
x_onehot_full.columns = x_onehot_full.columns.str.replace('-','_')
x_onehot_full.columns = x_onehot_full.columns.str.replace(')','_')
x_onehot_full.columns = x_onehot_full.columns.str.replace('(','_')
x_onehot_full.columns = x_onehot_full.columns.str.replace(',','_')
```

After pre-processing, we fit our Winner Voting Classifier model on the whole dataset in deployment stage.

The Voting Classifier uses 12 models with 10 of them being Gradient Boosting Classifiers, and the rest being XGB Classifier and Logistic Regression. We experimented with Learning_rate, n_estimators, and max_depth parameters in GB Classifiers. We used soft voting as we had an imbalanced class problem so hard voting automatically assumes a probability threshold of 0.5. Soft voting averaged out the probabilities rather than final outcomes.

**Fitting the model**

```
lr = LogisticRegression(class_weight= 'balanced', max_iter = 500)
clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)
clf2 =  GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)
clf3 =  GradientBoostingClassifier(n_estimators = 150 , max_depth = 5, learning_rate = 0.05)
clf4 =  GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)
clf5 = GradientBoostingClassifier(n_estimators = 350 , max_depth = 5, learning_rate = 0.05)
clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)
clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)
clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)
clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)
clf10 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.20)
clf11 = LogisticRegression(class_weight= 'balanced')
clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200)
vc = VotingClassifier(estimators = [('gb1', clf1), ('gb2', clf2), ('gb3', clf3),('gb4', clf4),('gb5', clf5),('gb6', clf6), ('gb7'
                      voting = 'soft', n_jobs = 10, verbose = True)

vc.fit(x_onehot_full, y_full)
```

We further evaluated our best model in deployment stage to get test ROC scores. **In the test stage we have applied probability threshold of 0.101 on predictions.** This threshold was selected after rigorous testing on multiple thresholds.

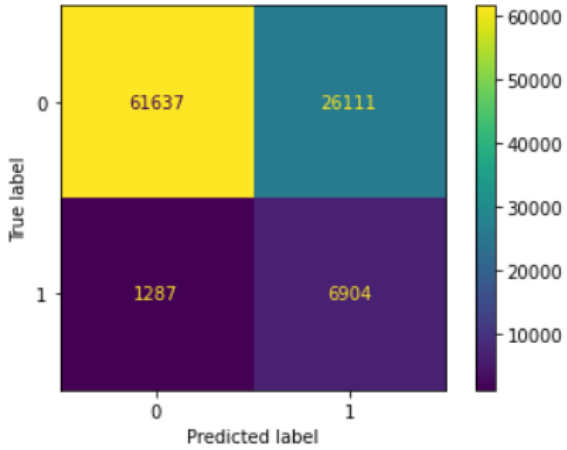
**Evaluating the model (Optional)**

```
x_test_full = pd.read_csv('x_test_full.csv')
y_test_full = pd.read_csv('y_test_full.csv')

md_probs = vc.predict_proba(x_test_full)
md_probs = md_probs[:,1]
md_auc = roc_auc_score(y_test_full, md_probs)
print('Test AUC-ROC of the model is:  %.3f % ', md_auc)

predictions = (md_probs >= 0.101).astype(bool)
confusion = confusion_matrix(y_test_full, predictions)

display = ConfusionMatrixDisplay(confusion_matrix = confusion)
display.plot()
plt.grid(False)
plt.show()
print("TPR or Recall: ", (confusion[1][1]/(confusion[1][0] +confusion[1][1]))*100)
print("precision: ", (confusion[1][1]/(confusion[0][1] +confusion[1][1]))*100)
print("Accuracy: ", ((confusion[0][0]+confusion[1][1])/(confusion[0][0]+confusion[0][1]+confusion[1][0]+confusion[1][1]))*100)
print("TNR or Specifity: ", (confusion[0][0]/(confusion[0][1] +confusion[0][0]))*100)
```

```
Test AUC-ROC of the model is:  %.3f %  0.8535277110705034
```

**The test ROC AUC score we got was 0.8442 which is the highest, we achieved through out this project.**

Here is the confusion matrix of our best model. This model gives us the optimum TPR and TNR for our case. We chose this model with a higher TPR because we cannot compromise on Recall in our project as it could be fatal for patients. However, this approach made us compromise on overall accuracy and specificity (TNR) but in real life, it would just mean an increase in additional testing costs associated with false positives. The test AUC-ROC in screenshot is different here because the model is trained on entire dataset however, we achieved 0.8442 by separating out the test set.

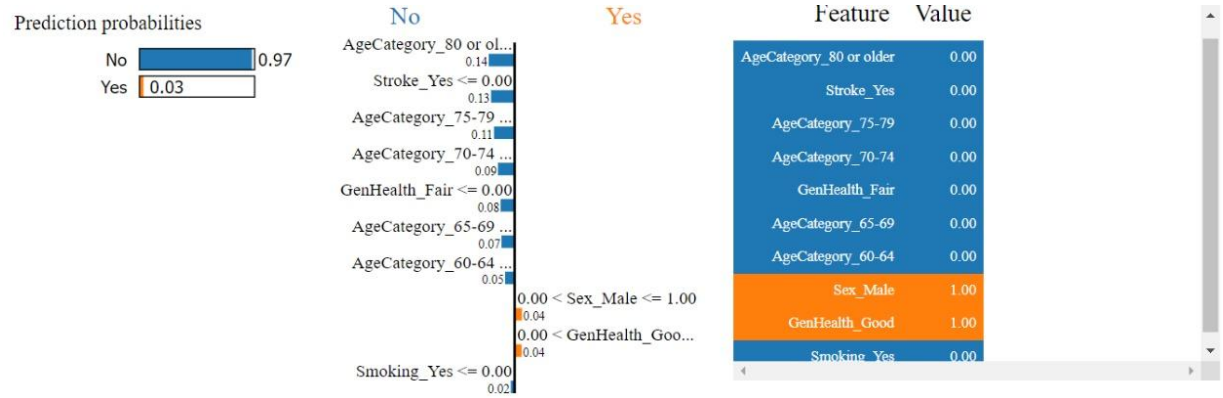



```
TPR or Recall:  84.28763276767184
precision:  20.911706799939424
Accuracy:  71.44227060944975
TNR or Specifity:  70.24319642612937
```

A TPR of 84.3% means for all the patients with heart disease, 84.3% of them were classified correctly by our model. A specificity of 70.2 means our model classified patients with no heart diseases 70.2% of the times correctly. Overall accuracy of prediction of our model is around 71.44%.

### 5.1.1 Interpreting with Lime

LIME was used to interpret the individual predictions given by the model.




```
ime: 2.77 s (started: 2022-05-20 05:05:52 +05:00)
```

LIME provides the top contributing factors that result into a given contribution. For the first test record, LIME gives this test record 97% probability of NOT HAVING A HEART DISEASE because:

- Test record does not belong to any age category that is 60+
- This test record has had no stroke in the past (Stroke_Yes = 0)

Similarly, LIME has given this test record 3% probability of HAVING A HEART DISEASE based on these parameters:

- This test record has a general good health (GenHealth_Good = 1)
- This test record is a male (Sex_Male = 1)

## 5.2 Second Best Model

Other than the best model, the other two models that performed very well were also Voting Classifier Models.

Our second-best model had more XGB Classifier base models than the first one. However, we can see that the ROC-AUC decreased in doing so.

```
clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)
clf2 =  GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)
clf3 =  GradientBoostingClassifier(n_estimators = 150 , max_depth = 5, learning_rate = 0.05)
clf4 =  GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)
clf5 = GradientBoostingClassifier(n_estimators = 350 , max_depth = 5, learning_rate = 0.05)
clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)
clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)
clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)
clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)
clf10 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.20)
clf11 = LogisticRegression(class_weight= 'balanced')
clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200)
clf13 = XGBClassifier( random_state=2, learning_rate = 0.07, n_estimators= 200)
clf15 = XGBClassifier( random_state=2, learning_rate = 0.03, n_estimators= 200)
clf16 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 150)
clf18 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 250)



model = vc = VotingClassifier(estimators = [('gb1', clf1), ('gb2', clf2), ('gb3', clf3),
                                            ('gb4', clf4),('gb5', clf5),('gb6', clf6),
                                            ('gb7', clf7), ('gb8', clf8),('gb9', clf9),
                                            ('gb10', clf10), ('lr1', clf11),('xgb1', clf12),
                                            ('xgb2', clf13),('xgb3', clf15),
                                            ('xgb4', clf16),('xgb5', clf18)
                                            ],
                      voting = 'soft', n_jobs = -1, verbose = True)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Train ROC AUC: 0.916
Test AUC-ROC %.3f %  0.844091224302452
```

## 5.3 Third Best Model

Our third best model was produced by removing some GB Classifier models from the second-best models. Doing so, resulted in the least ROC scores. Which means presence of GB classifiers increases our score.

```
clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)
clf2 =  GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)
clf4 =  GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)
clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)
clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)
clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)
clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)

clf11 = LogisticRegression(class_weight= 'balanced')
clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200)
clf13 = XGBClassifier( random_state=2, learning_rate = 0.07, n_estimators= 200)
clf15 = XGBClassifier( random_state=2, learning_rate = 0.03, n_estimators= 200)
clf16 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 150)
clf18 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 250)



model = vc = VotingClassifier(estimators = [('gb1', clf1), ('gb2', clf2),
                                            ('gb4', clf4),('gb6', clf6),
                                            ('gb7', clf7), ('gb8', clf8),('gb9', clf9),
                                            ('lr1', clf11),('xgb1', clf12),
                                            ('xgb2', clf13),('xgb3', clf15),
                                            ('xgb4', clf16),('xgb5', clf18)
                                            ],
                      voting = 'soft', n_jobs = -1, verbose = True)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Train ROC AUC: 0.916
Test AUC-ROC %.3f %  0.8440633144991229
```

## 5.4 All Results

This table summarizes the best scores obtained on each ML Algorithm in every stage of Model experimentation:

| Model | Experimentation Stage | Parameters | ROC-AUC | Recall | Precision |
|---|---|---|---|---|---|
| Base Case Models | Stage 0 - Base Case Models | Decision Tree Classifier | 0.589 | - | - |
| | | Logistic Regression | 0.8425 | - | - |
| | | K-Nearest Neighbors Classifier | 0.7965 | - | - |
| | | Random Forest Classifier | 0.792 | - | - |
| | | Gradient Boosting Classifier | 0.8427 | - | - |
| | | Naive Bayes Categorical | 0.655 | - | - |
| | | Gaussian Naive Bayes | 0.644 | - | - |
| | | Mixed Naive Bayes with Gaussian NB | 0.805 | - | - |
| | | Mixed Naive Bayes with Decision Tree Classifier | 0.578 | - | - |
| KNN Classifier | Stage 1 - on sample scaled dataset | n_neighbors = 400 | 0.828 | - | - |
| | Stage 2 - on Full dataset | n_neighbors = 410 | 0.8325 | 1 | 73 |
| | Stage 3 - Full dataset + SMOTE | n_neighbors = 410 | 0.83087 | 48.66 | 31.74 |
| | Stage 3 - Full dataset + SMOTE | n_neighbors = 300 | 0.83088 | 50.78 | 30.85 |
| | Stage 3 - Full dataset + SMOTE | n_neighbors = 100 | 0.83004 | 55.23 | 29.09 |
| Gradient Boosting Classifier | Stage 1 - on sample scaled dataset | n_estimators = 800, learning_rate = 0.05 | 0.842 | - | - |
| | Stage 2 - on Full dataset | n_est = 250, learning_rate = 0.05, depth = 5 | 0.843 | 9 | 54 |
| | Stage 3 - Full dataset + SMOTE | n_est = 300, learning_rate = 0.05, depth = 4 | 0.8424 | 58.99 | 29.72 |
| | Stage 3 - Full dataset + SMOTE | n_est = 250, learning_rate = 0.05, depth = 5 | 0.8424 | 59.58 | 29.27 |
| | Stage 3 - Full dataset + SMOTE | n_est = 250, learning_rate = 0.05, depth = 5 | 0.8424 | 59.58 | 29.27 |
| Random Forest Classifier | Stage 1 - on sample scaled dataset | max_depth=20, max_features=5, min_samples_leaf =100, min_samples_split=50, n_est =1000 | 0.838 | - | - |
| | Stage 2 - on Full dataset | max_depth=20, max_features=5, min_samples_leaf =15, min_samples_split=15, n_est =1000 | 0.8408 | 3.3 | 61 |
| | Stage 3 - Full dataset + SMOTE | max_depth=20, max_features=5, min_samples_leaf =100, min_samples_split=50, n_est =1000 | 0.8361 | 54.59 | 30.62 |
| | Stage 3 - Full dataset + SMOTE | max_depth=100, max_features=sqrt, min_samples_leaf =50, min_samples_split=50, n_est =700 | 0.8387 | 56.76 | 29.82 |
| | Stage 3 - Full dataset + SMOTE | max_depth=20, max_features=5, min_samples_leaf =15, min_samples_split=15, n_est =1000 | 0.8329 | 63.09 | 26.53 |
| Decision Tree Classifier | Stage 3 - Full dataset + SMOTE | min_samples_split = 4 | 0.6771 | 51.81 | 19.13 |
| | Stage 3 - Full dataset + SMOTE | max_features = sqrt | 0.6593 | 53.66 | 18.58 |
| | Stage 3 - Full dataset + SMOTE | min_samples_split = 2 | 0.6581 | 53.98 | 18.31 |
| | Stage 3 - Full dataset + SMOTE | min_samples_split = 6 | 0.69 | 52.17 | 19.46 |

| Model | Experimentation Stage | Parameters | ROC-AUC | Recall | Precision |
|---|---|---|---|---|---|
| XGB Classifier | Stage 2 - on Full dataset | n_est = 200, learning_rate = 0.05 | 0.842 | - | - |
| | Stage 3 - Full dataset + SMOTE | n_est = 800, learning_rate = 0.1 | 0.8324 | 59.13 | 28.27 |
| | Stage 3 - Full dataset + SMOTE | max_depth = none | 0.8347 | 59.02 | 28.61 |
| | Stage 3 - Full dataset + SMOTE | n_est = 400, learning_rate = 0.1 | 0.8373 | 59.54 | 29.18 |
| | Stage 3 - Full dataset + SMOTE | n_est = 100, learning_rate = 0.1 | 0.842 | 59.38 | 29.62 |
| Logistic Regression | Stage 2 - on Full dataset | class_weight= balanced, max_iter = 500 | 0.8418 | - | - |
| | Stage 3 - Full dataset + SMOTE | class_weight= balanced, max_iter = 500 | 0.8418 | 78.13 | 22.42 |
| Stacking Classifier | Stage 3 - Full dataset + SMOTE | clf6 = LR(class_weight= 'balanced')<br>clf3 = XGB( random_state=2, verbosity=1, learning_rate = 0.05, n_estimators = 1000, n_jobs = -1) | 0.8428 | 61.43 | 28.58 |
| | Stage 3 - Full dataset + SMOTE | clf1 = LR(class_weight= 'balanced', max_iter = 500)<br>clf2 = XGB( random_state=2, verbosity=1, learning_rate = 0.05, n_estimators = 1000, n_jobs = -1)<br>clf3 = RF(n_estimators = 1000 , max_depth = 20, max_features = 5, min_samples_leaf = 15, min_samples_split = 15) | 0.8435 | 61.64 | 28.59 |
| | Stage 3 - Full dataset + SMOTE | clf1 = RF(n_estimators = 1000 , max_depth = 20, max_features = 5, min_samples_leaf = 15, min_samples_split = 15)<br>clf2 = XGB( random_state=2, learning_rate = 0.1, n_est = 1000)<br>clf9 = RF(n_estimators = 1000 , max_depth = 20, max_features = 5, min_samples_leaf = 13, min_samples_split = 13)<br>clf10 = RF(n_estimators = 1000 , max_depth = 20, max_features = 5, min_samples_leaf = 17, min_samples_split = 17)<br>clf11 = LR(class_weight= 'balanced', max_iter = 500) | 0.8438 | 61.78 | 28.67 |

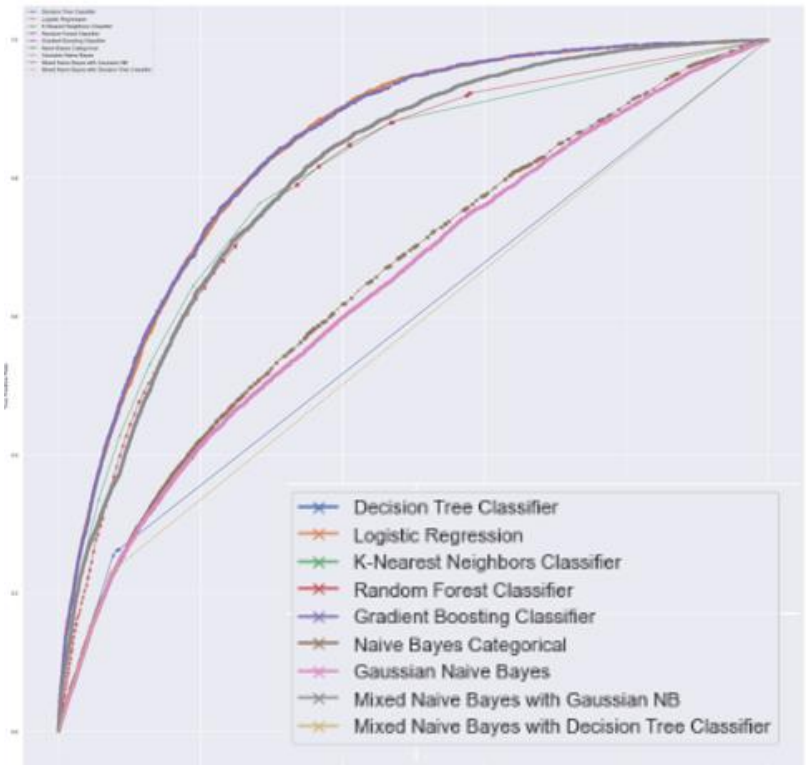
| Model | Experimentation Stage | Parameters | ROC-AUC | Recall | Precision |
|---|---|---|---|---|---|
| Voting Classifier | Stage 3 - Full dataset + SMOTE | clf1 = LogisticRegression(class_weight= 'balanced', max_iter = 500)<br>clf2 = XGBClassifier( random_state=2, verbosity=1, learning_rate = 0.05, n_estimators = 1000, n_jobs = -1)<br>clf3 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05) | 0.8428 | 66.56 | 26.98 |
| | Stage 3 - Full dataset + SMOTE | LR(class_weight= 'balanced', max_iter = 500)<br>XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators = 1000) | 0.8424 | 69.89 | 25.43 |
| | **Stage 4 - on Full dataset with probability threshold (ULTIMATE BEST MODEL)** | **clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)**<br>**clf2 = GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)**<br>**clf3 = GradientBoostingClassifier(n_estimators = 150 , max_depth = 5, learning_rate = 0.05)**<br>**clf4 = GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)**<br>**clf5 = GradientBoostingClassifier(n_estimators = 350 , max_depth = 5, learning_rate = 0.05)**<br>**clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)**<br>**clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)**<br>**clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)**<br>**clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)**<br>**clf10 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.20)**<br>**clf11 = LogisticRegression(class_weight= 'balanced')**<br>**clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200)** | **0.8442** | **83.12** | **20.64** |
| | Stage 2 - on Full dataset (SECOND BEST MODEL) | clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)<br>clf2 = GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)<br>clf4 = GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)<br>clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)<br>clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)<br>clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)<br>clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)<br>clf11 = LogisticRegression(class_weight= 'balanced')<br>clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200) | 0.8442 | - | - |

| Model | Experimentation Stage | Parameters | ROC-AUC | Recall | Precision |
|---|---|---|---|---|---|
| Voting Classifier | Stage 2 - on Full dataset (THIRD BEST MODEL) | clf1 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.05)<br>clf2 = GradientBoostingClassifier(n_estimators = 200 , max_depth = 5, learning_rate = 0.05)<br>clf4 = GradientBoostingClassifier(n_estimators = 300 , max_depth = 5, learning_rate = 0.05)<br>clf6 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 4, learning_rate = 0.05)<br>clf7 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 6, learning_rate = 0.05)<br>clf8 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.10)<br>clf9 = GradientBoostingClassifier(n_estimators = 250 , max_depth = 5, learning_rate = 0.15)<br>clf11 = LogisticRegression(class_weight= 'balanced')<br>clf12 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 200)<br>clf13 = XGBClassifier( random_state=2, learning_rate = 0.07, n_estimators= 200)<br>clf15 = XGBClassifier( random_state=2, learning_rate = 0.03, n_estimators= 200)<br>clf16 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 150)<br>clf18 = XGBClassifier( random_state=2, learning_rate = 0.05, n_estimators= 250) | 0.844 | - | - |
| Neural Network | Stage 3 - Full dataset + SMOTE | Dense(units=37, activation='relu'),<br>Dense(units=26, activation='relu'),<br>Dense(units=1), 200 epochs | 0.8407 | 0 | Nan |
| | Stage 3 - Full dataset + SMOTE | Dense(units=30, activation='relu'),<br>Dense(units=12, activation='relu'),<br>Dense(units=1), 500 epochs | 0.8377 | 0 | Nan |
| | Stage 2 - on Full dataset | Dense(units=37, activation='relu'),<br>Dense(units=20, activation='relu'),<br>Dense(units=1, activation = 'sigmoid'), 500 epochs | 0.8379 | 88.98 | 17.21 |
| | Stage 2 - on Full dataset | Dense(units=37, activation='relu'),<br>Dense(units=26, activation='relu'),<br>Dense(units=1, activation = 'sigmoid'), 500 epochs | 0.836 | 88.34 | 17.7 |

# 6. Project Experimentation Analysis

The first step was to perform pre-processing. As explained before, we performed one-hot encoding and MinMax Scalar transformation on the entire dataset. Although min-max scaling was not needed for most of these algorithms except logistic regression, naïve bayes and KNN, we still did it on our entire project to keep consistency.

Next step was to build base case models using ROC Curve. As can be seen from the graph below, Gradient Boosting Classifier and Logistic Regression performed the best. Results for Base Cases are given in **this table.**




Our dataset being imbalanced was one of the key problems we faced, and we had to resort to numerous techniques to eradicate the problem. After performing base cases we trained multiple models on sample dataset to figure out the best parameters using **GridSearchCV** (stage 1). Then models with best selected hyperparameters were trained on the full dataset using **RepeatedStratifiedKfold** (stage 2)**.** After training on full dataset, we figured our Recall was very poor. So we decided to use SMOTE with undersampling techniques to balance the dataset (stage 3). Doing so, improved our Recall and Precision but reduced our ROC-AUC slightly. In the final stage (stage 4) we applied probability threshold on the test dataset without using SMOTE to get the best Recall and ROC-AUC scores. It is important to note that Naïve Bayes was not included in further stages as it has no parameters to play with.

Here is a summary of our learnings from each model in every subsequent stage:

## 6.1 K-NearestNeighbors Classifier

On the sample dataset multiple N_neighbors were tried through GridSearchCV on a range of 20 – 1000. KNN produced best ROC-AUC scores when K = 350+. After extensive search for n_neighbors using 35% of dataset, K = 410 was found to be working the best on sample dataset.

Next we tested on multiple K values on full scaled dataset using GridsearchCV. After a lot of hyperparameter tweaking, K range between 350-420 gave approximately the same results with 0.0001 difference. Best K value is decided to be 410 with an ROC-AUC of 0.8325 as it was given by grid search.

However, the Recall in Stage 2 was around 1% which was extremely poor. That's why we applied SMOTE on train dataset and then tested KNN on validation set.

In Stage 3, the KNN Classifier continued to provide best results on K = 410. However, the Recall increased to 49%, and ROC-AUC decreased to 0.830. This proved that balancing the data through SMOTE can actually be very helpful in improving Recall while compromising a little on ROC-AUC scores. The probability threshold for all the models tried with KNN was 0.5.

## 6.2 DecisionTree Classifier

Since our base model score for Decision Tree was not particularly good, we did not perform exhaustive testing on it. We only implemented DT in Stage 3 of experimentation on full dataset with SMOTE.

The parameters we experimented with were min_samples_split (2-6), and max_features (sqrt). By increasing the min_samples_split, ROC-AUC scores increased by 1%, as it reduced overfitting. However, adding max_features did not improve the scores at all.

As expected, the best DT resulted in a score of only 0.69 which is why it was not explored further.

## 6.3 Logistic Regression

Logistic Regression gave the second-best ROC scores on base model (0.842). We evaluated LR on full dataset with class_weights = balanced (Stage 2).

Doing so, the model automatically assigned the class weights inversely proportional to their respective frequencies. This could be the reason LR resulted in high scores in stage 2, 0.8418.

In stage 3, we applied SMOTE and ran the same model again which resulted in consistent ROC scores and the best Recall of 78.13%.

## 6.4 Random Forest Classifier

In stage 1, we performed an exhaustive search on n_estimators parameter within a range of 100 to 1000. The ROC scores increased from 0.792 (with 100 estimators) to 0.811 (with 1000 estimators). We chose 1000 estimators to get accurate results.

Next max_depth was checked on a range of 10-50, along with max_features from (3-sqrt). The performance decreased as max_depth went 20+. As per convention max_features in RF should be less than Sqrt of total features. That is why the max_features selected by grid search were 5 to reduce overfitting.

We got best ROC-score of 0.838 on depth=20, features=5, estimators=1000.

In stage 2, the same parameters performed best along with min_samples_split/leaf = 15 each. This reduced overfitting and increased ROC-AUC to 0.84.

In stage 3, parameters selected from stage2 gave us ROC of 0.83, with the best Recall of 63% on SMOTE dataset.

## 6.5 Gradient Boosting Classifier

Gradient Boosting Algorithm was our best base model giving an ROC-AUC of **0.843.** This model was the most extensively explored model throughout the project.

In stage 1 with sampled dataset, several parameters were tried and Stratified KFold validation with grid search was used several times to find the optimum parameters. It was identified that increasing max_depth decreased AUC-ROC score. Increasing the learning rate from 0.01 to 0.05 increased our score however the score started decreasing beyond that. Furthermore, increasing number of estimators from 50 till 250 increased our AUC-ROC score however, it became stagnant with minimal decrease in score till 1000 estimators.

In stage 2 with full dataset, more optimization was done around the best parameters identified from stage 1 and our grid search result gave the best parameters as learning rate 0.05, n_estimators 1000 and max_depth 3. It is imperative to note that as max depth or learning rate is increased, the optimum n_estimators would decrease. However, at this point the recall was terrible.

In stage 3 with SMOTE, extensive search of parameters were tried again around the best parameters from stage 2 and after tweaking of n_estimators, it was realized that the best number of estimators do not change too much when estimators are increased from 200-1000. So, 250 estimators came out as the optimum one for us. With Smote strategy significantly improved our recall as expected.

Lastly, it was decided to pick the best model of Gradient boosting with the highest ROC which turned out to have 250 estimators, 0.05 learning rate and 5 max depths.

## 6.6 XGB Classifier

XG boosting classifier performed as well as other top performing algorithms such as Gradient boosting and Logistic regression.

XG boost was directly tried with and without SMOTE i.e., stage 2 and 3. XGB with smote was run extensively for different parameters. It was seen that increasing learning rate up to a point increased AUC-ROC and increasing number of estimators decreased AUC-ROC. Hence, in stage 1 the learning rate was kept at 0.1 and estimators as 100 which gave a score of **0.842**. Changing from with SMOTE and without did not affect this algorithm too much.

## 6.7 Neural Networks

Neural Network was tried in the latter half of the project with SMOTE and without SMOTE. Sequential model was used with different layers and the algorithm performed quite decently. However, it was not gradient boosting or XGBoost. The best neural network configuration was achieved with SMOTE and 3 layers with an AUC-ROC of **0.8407**. Neural network was not explored as extensively since we had already found the best models and it neural network was not able to reach them.

## 6.8 Stacking Classifier

After identifying the best base models with their parameters, stacking was used to ensemble the models together. Several strategies were tried using Random Forest, Gradient boosting, XG boost and logistic regression, the best AUC-ROC of 0.8439 was achieved with 3 Random Forest, 1 XGboost and 1 Logistic regression base models. It was noticed that addition of random forest or gradient boosting improved scores showing that these two algorithms are best base models to work with. Stacking was tried with SMOTE only as voting algorithm already beat stacking in SMOTE stage so it was certain that it would beat it in non-smote stage as well.

## 6.9 Voting Classifier

This algorithm performed gave the best AUC-ROC score and belongs to our top 3 models. After identifying the best base models, several variations of base model were tried for this model With SMOTE and without SMOTE based on the best identified base models. After using a combination of best random forest, gradient boosting, XGboost and logistic algorithms, it was known that random forest did not help in getting the best AUC-ROC, so it was removed. Afterwards, several tries were done in Stage 3 and Stage 2 and the best model turned out to be the ones that included several variations of Gradient boosting, 1 XGBoost and 1 logistic regression model. However, best model formation had a lot of ingenuity to it. We used our best gradient boosting model and created 10 different variations of it by slightly changing one parameter at a time. The result achieved worked well for us and helped us identify our best model having an AUC-ROC of **0.8422**. It was certain afterwards that increased gradient boosting's weightage in Voting Classifier increased our score.

## 6.10 Optimum Probability Threshold

After our best model was identified as voting classifier, we started exploring the threshold probability at which we wanted to deploy our model. It was seen that as threshold probability increased, the number of false negatives increased, and number of false positives decreased. However, in our case, as mentioned earlier, we are predicting heart disease, we would like our false negative rate to be extremely low as giving wrong diagnosis could be deadlier for a patient. On the other hand, a few false positives would just incur additional testing costs. But, in achieving a higher recall rate, our accuracy was taking a major hit due to imbalanced data. So, after rigorous search, we decided to use a probability threshold of 0.101 as it gave us a recall of 83% with a decent accuracy of 71.3% as it seems the best compromise.

## 6.11 Interpretability and Deployment

In medical field, it is extremely important to have a model which can be interpreted as doctors cannot rely on model's prediction alone regardless of how powerful the model is. The imperativeness of this led to the use of LIME with Multiple Linear Regression and a Lasso feature selector to explain the model's predictions. The use of LIME became clearer after the model was deployed on Streamlit. In addition to the model's prediction of whether a person has heart disease or not, the model also gave reasons to why it thinks so. We made the Streamlit application in such a way that only top 3 reasons ordered by probability were displayed to the user. Deployment of the LIME powered model completed our project and tied it together beautifully.

# 7. Conclusion

Going forward, we would like to take this project further by trying newer models such as support vector machine and try to make AUC-ROC even better. However, recent feedback about our app from professional doctors in the field suggested that if we can create a more mature app with a better model using more features that doctors normally use to predict heart disease such as heart score, cholesterol etc., then we can take this project to a production stage and equip hospitals with a great tool to preliminary diagnose a heart disease.