CS 6375 Machine Learning Project
Heart Disease Prediction

Motivation:

Cardiovascular diseases (CVDs) are number 1 cause of death globally taking an estimated 17.9 million lives each year.

India had the highest number of heart disease deaths last year, followed by China, Russia, the United States and Indonesia.

Identifying those at highest risk of CVDs and ensuring they receive appropriate treatment can prevent premature deaths.

Our main focus is on predicting heart disease of patients who does not have any symptoms i.e., asymptomatic CVDs.

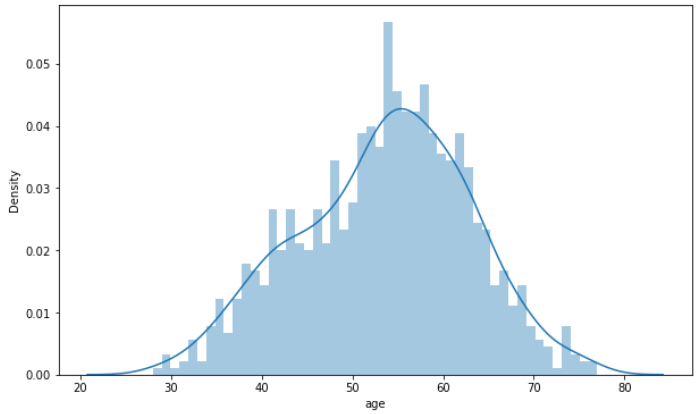
Analysis Of Dataset:

We have downloaded the Heart Disease Dataset from the IEEE Website.

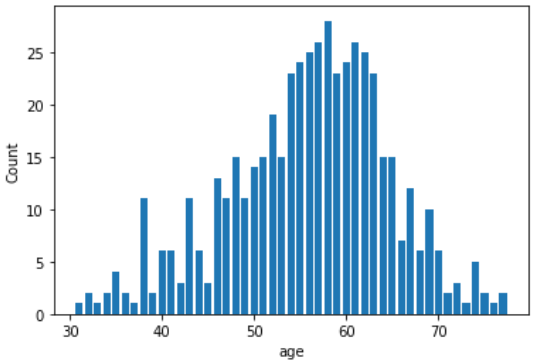
There are 12 Features present in the Dataset. Below Picture describes the Names of the Features and the types of values it contains.

S.No.	Attribute	Unit	Data type
1	age	in years	Numeric
2	sex	1, 0	Binary
3	chest pain type	1,2,3,4	Nominal
4	resting blood pressure	in mm Hg	Numeric
5	serum cholesterol	in mg/dl	Numeric
6	fasting blood sugar	1,0 > 120 mg/dl	Binary
7	resting electrocardiogram results	0,1,2	Nominal
8	maximum heart rate achieved	71–202	Numeric
9	exercise induced angina	0,1	Binary
10	oldpeak =ST	depression	Numeric
11	the slope of the peak exercise ST segment	0,1,2	Nominal
12	class	0,1	Binary

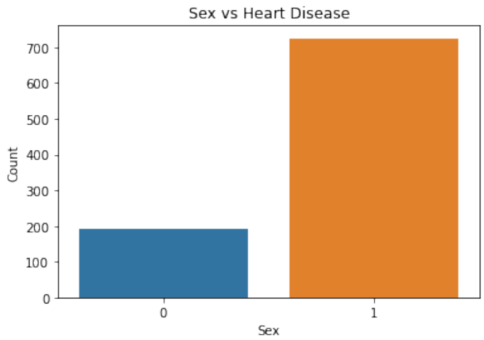
Wide variation of ages but with a dense number of people over 40 in this data.



The Mean age of the people with Heart Attack is 55.



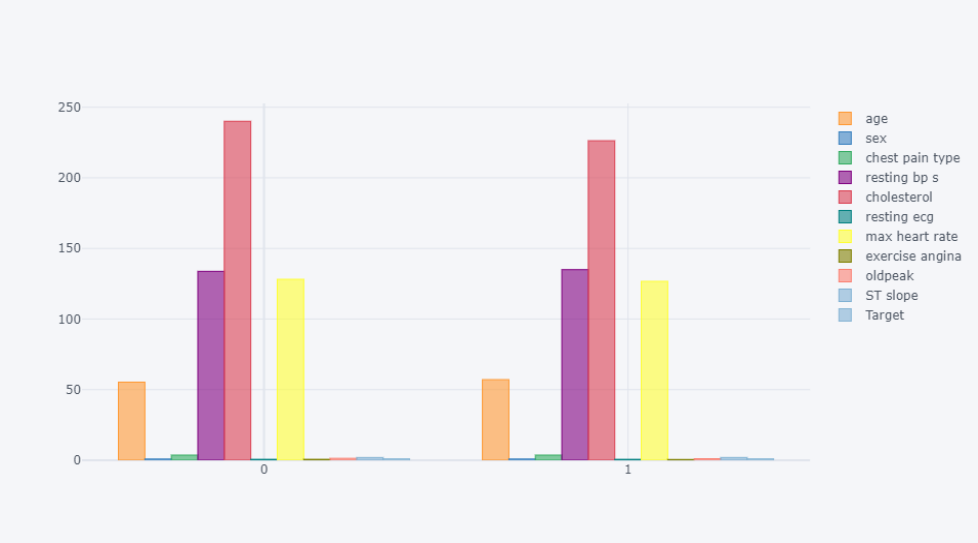
There are more number of Male(1) cases than the Female(0) Cases.



66% of the people having Heart Attack have Normal Fasting Sugar Level, which indicates Diabetes can't be a single Factor for a Heart Attack.

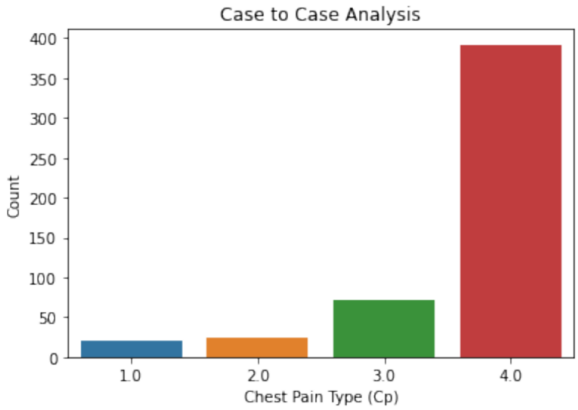
Here, we have grouped the Patients according to Diabetes(Yes or No).

The below Graph shows, people with Normal Fasting Sugar Level also have Asymptomatic Heart Diseases.



Interesting Observation: 77% of the people have Asymptomatic(Chest Pain type=4) Heart Diseases.

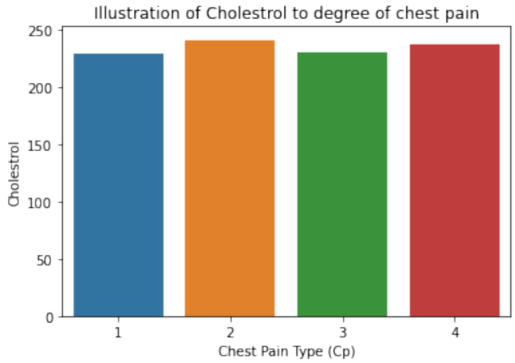
This fact made us curious to analyse the Dataset deeper.

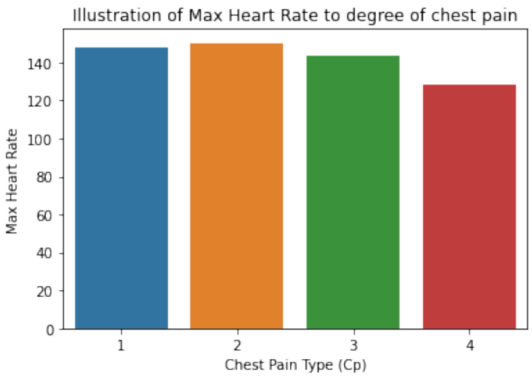


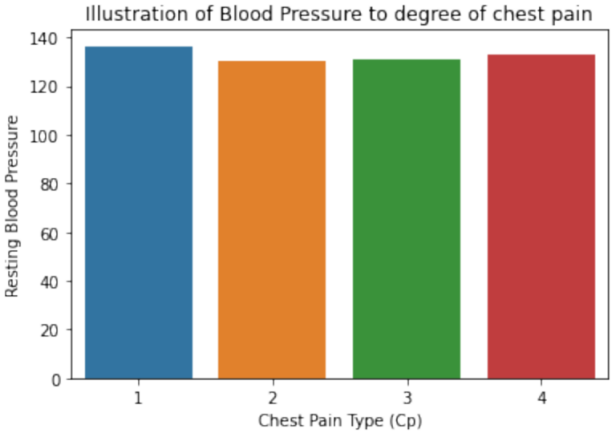
We Observed all the Asymptomatic Cases have some common characteristics such as High Cholesterol Level, High Heart Rate and High Blood Pressure as well.

The Mean Values of those characteristics are:

Cholesterol: 237.14
Resting BP: 133.22
Heart Rate: 128.27







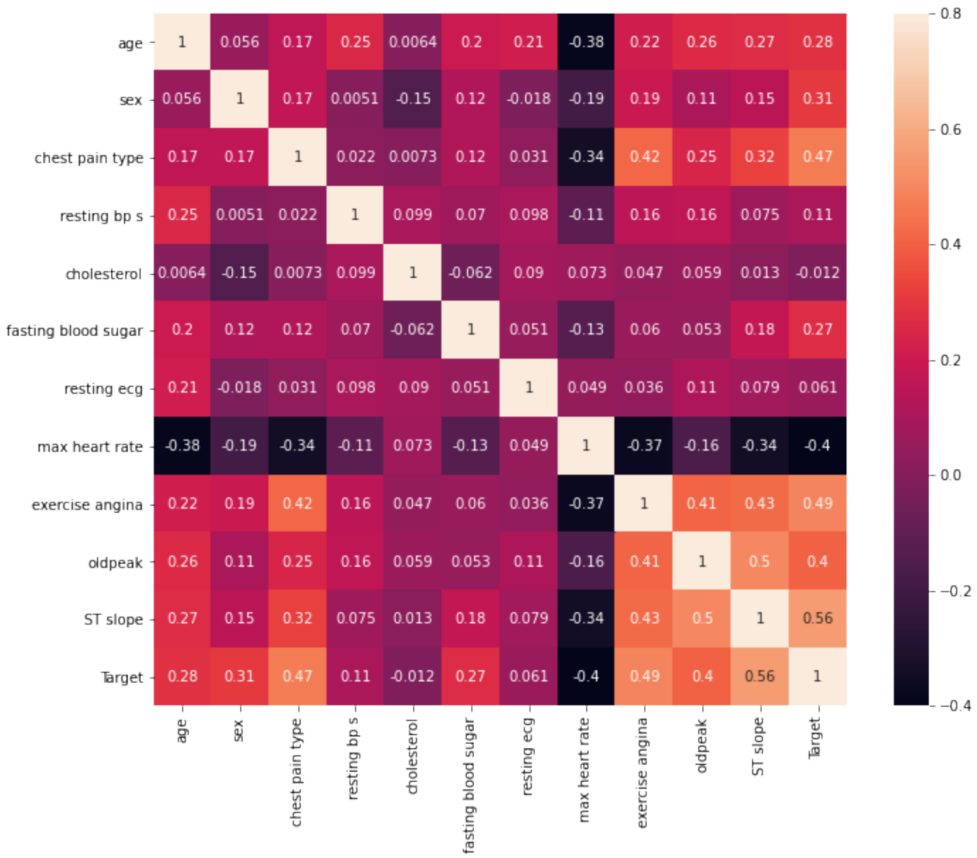
These characteristics often lead to Silent Heart Disease with no Symptoms which is pretty Dangerous.
So, our Goal is to indicate this type of cases and recommend proper medication to these cases in order to avoid premature Deaths.

Evaluation Of the Problem:

In this Project, we have implemented our own version of Gaussian Naive Bayes and KNN.
We did not use Scikit-Learn's Library.

In order to train the model, we have used Spearman Correlation for finding out the Correlation Matrix among the 12 Features including the class Label..

Below Graph shows describes the Correlation Coefficients with respect to each other.



After a thorough Observation , we have found that there is a direct correlation among features Exercise Angina, ST Slope and oldpeak . And the Feature Resting ECG has no effect on the Target feature so we dropped that feature and we observed the accuracy increased by 2%.

Speculation on Train Test Split:

Initially, we used Scikit's train test split in order to train the model and then test it. We allowed the random_state to be 1 as we wanted to pick the repeated samples in order to train the model , in other words we allowed resampling of the data.

We also implemented our own logic for train_test_split function where we have used np.random.choice to pick 80% of the dataset for the training and 20% of the data for the Testing and allowed resampling.

The Model performed approximately the same in both the cases. So, we decided to use the Scikit's Train_test_split function in order to train and test the Model.

We got Accuracy with approx 67% when we used (60,40) train test metric and hence, we decided with (80,20) train test split which provided much better accuracy than this.

A. Naive Bayes:

Here, we have implemented **Gaussian Naive Bayes** as this works well with real-values attributes and our dataset contains continuous values .
Gaussian Naive Bayes : With this version of Naive Bayes, With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution. This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Probabilities of new x values are calculated using the **Gaussian Probability Density Function**.

When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

pdf(x, mean, sd) = (1 / (sqrt(2 * PI) * sd)) * exp(-((x-mean^2)/(2*sd^2)))

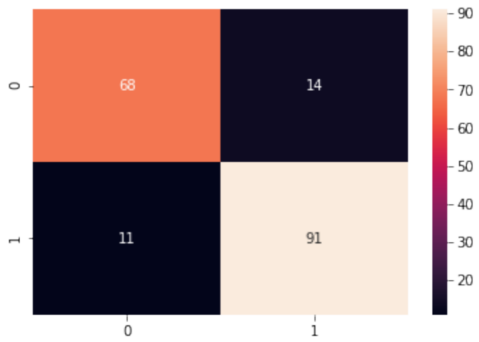
Where pdf(x) is the Gaussian PDF, sqrt() is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp() is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.

As Gaussian Naive Bayes is more robust to outliers , we did not perform the Scaling on our dataset before training the model.

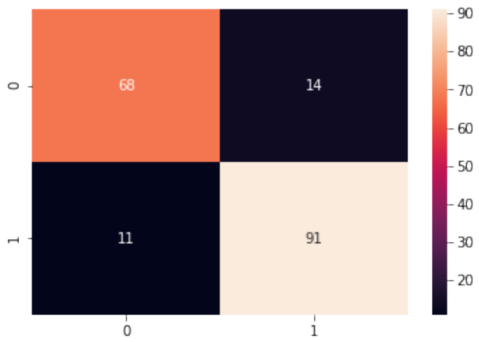
Results(With Comparison to Scikit's Library):

As we can see from the below confusion Matrix, we have a pretty good model with Naive Bayes. There are very less False Positives. To our surprise, our model works pretty much the same way as Scikit's Gaussian Naive Bayes. We were able to create a model which gives pretty much the same Accuracy as Scikit's Gaussian Naive Bayes Model.

Our Algorithms's Confusion Matrix:



Scikit's Confusion Matrix:

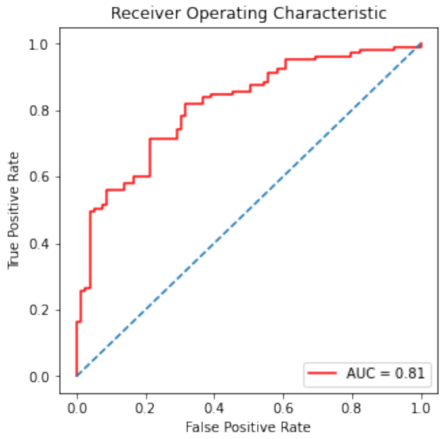


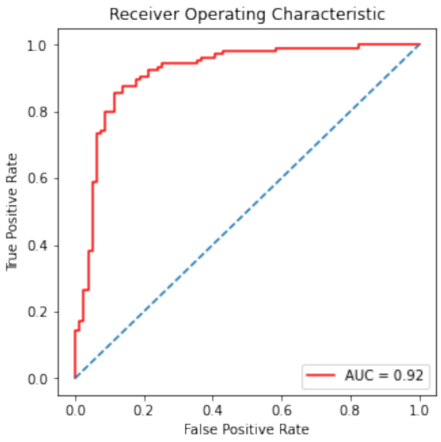
Classification Report:

class label	precision	recall	f1-score	support
0	0.83	0.86	0.84	79
1	0.89	0.87	0.88	105

We got a very good Precision and recall score with approx 86% accuracy. The classification report is exactly the same as Scikit's classification report for this dataset.

ROC-AUC Curve:





In this case, Scikit's ROC Curve is better than us, as the pred_proba function gives better probability for true positives for the samples than our Pred_proba function which we have written manually.

The pred_proba function gives the probability of a sample being predicted to be positive/Negative. And we only need the probabilities of Positives in order to plot the ROC-AUC Curve.

According to the overall result and comparison with Scikit's Gaussian Model , Our model works exactly the same as Scikit's Gaussian Model . This is the case because, Gaussian Naive Bayes is pretty straightforward as it involves the Mean and Standard deviation Calculation, it does not change much from Scikit' Performance.

B. Bagging:

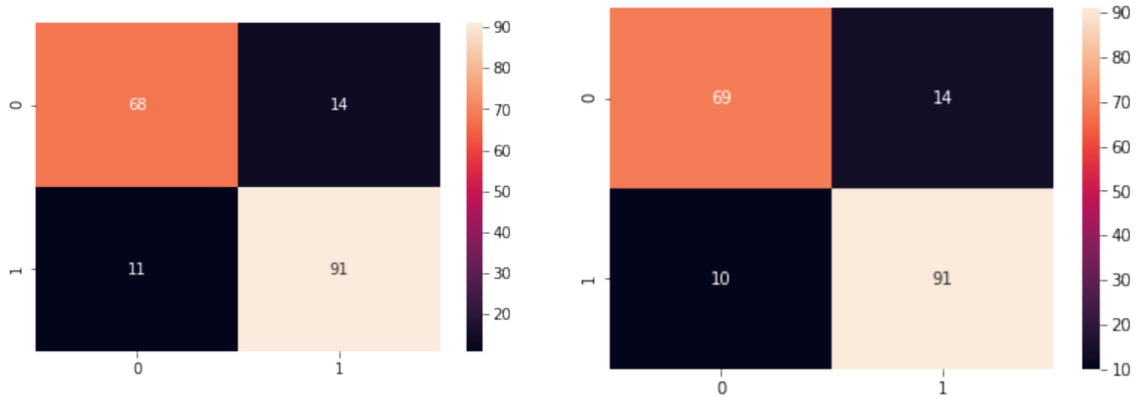
With curiosity to improve the Model Accuracy, we have implemented bagging with Naive Bayes. We have implemented Bag Size around 100 with 8 models and then predict the test dataset and then take the majority vote and decide the final prediction for each data point.

The accuracy that we got varies from range 82% to 87% but the mean is always 85. So, there is not much of an increase in performance after Bagging.

We also implemented Scikit BaggingClassifier with 8 models and max sample 100.

Result(Comparison with Scikit's BaggingClassifier):

Baaging Confusion Matrix: **Scikit's Bagging Classifier**



If we observe the Confusion Matrix, there is not much of a difference but as our Bagging performance varies from 82-87% range, the confusion matrix might not be the same always. Scikit's Bagging classifier always performs better with mean accuracy of 86%, whereas our Bagging performance mean accuracy score is approx 85%.

Classification Report:

Manual Bagging:

class label	precision	recall	f1-score	support
0	0.83	0.86	0.84	79
1	0.89	0.87	0.88	105

Scikit's Bagging classifier

class label	precision	recall	f1-score	support
0	0.85	0.84	0.84	79
1	0.88	0.89	0.88	105

If we see the tables, there are very minute changes in the precision and recall. So, overall the performance between Scikit's Bagging with Gaussian NB as estimators are the same as our Bagging algorithm with 8 models and Bag Size of 100.

C.KNN

We have implemented our own K-Nearest Neighbour algorithm as we have widely distributed data containing both continuous and discrete data. KNN is simpler to implement and performs really well(good accuracy) even when there is no prior knowledge of distribution of data.

Also, as KNN is non-parametric it does not need assumptions regarding the functional form for the problem.

Distance Metrics:

We have used below mentioned distance metrics to measure the performance:

1. Manhattan distance
2. Euclidean distance

Finally, we implemented Minkowski distance so that we can calculate distance metrics for values for $p > 2$.

Minkowski distance is defined as :

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Normalization of data:

Both Manhattan and Euclidean distances work with data which is similar in type. So, we normalized our data from 0 to 1 using Min-Max Normalization.

X' = (X - min(X)) / (Max(X) - Min(X))

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This resulted in a significant increase in accuracy by 6%.

Results with Manhattan distance(p=1): K= 7

Results with Euclidean distance(p=2): K = 11

Results with Minkowski distance with p= K= 13:

Selection of K value:

With our (80,20) train test split, we checked the best values for K from 1 to square.root() of length of the test data i.e 30 in our case.

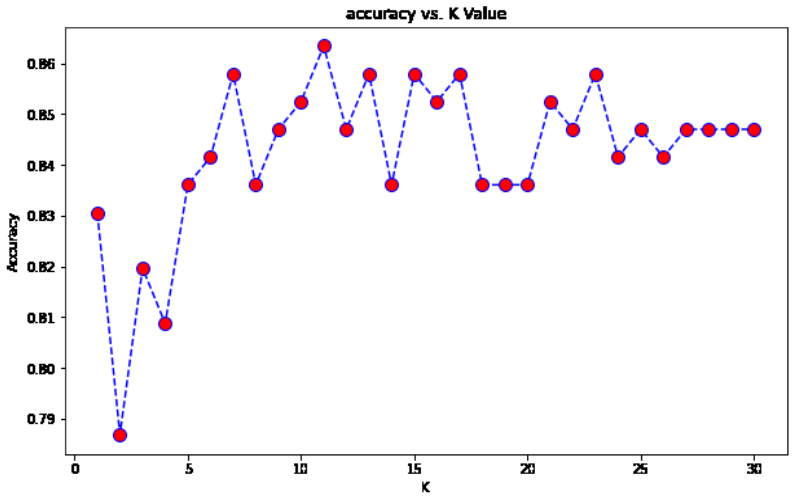
We ran for 50 iterations and found that our K values vary from 10 to 12 so we decided to keep K = 11 because we get the best prediction i.e precision and recall value for K = 11 .

<u>Results(With Comparison to Scikit's Library):</u>

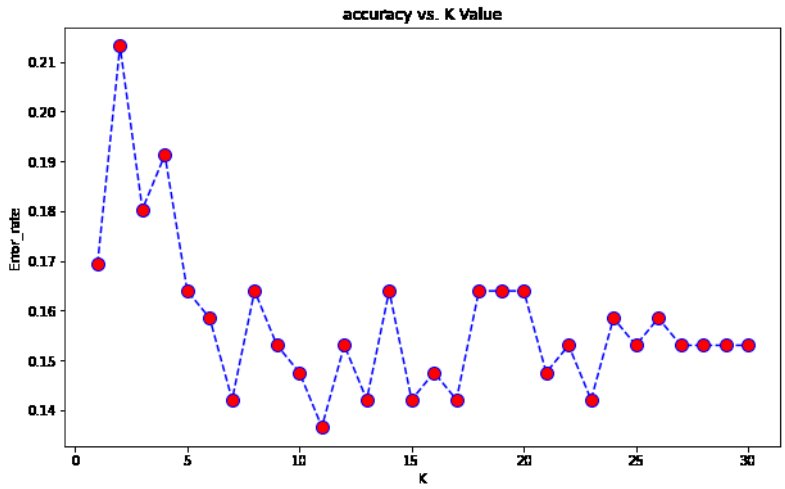
Our model works pretty much the same way as Scikit's KNeighborsClassifier. We achieved pretty much the same Accuracy as Scikit's KNeighborsClassifier. It is because KNN is pretty

straightforward as it involves the distance metric calculation and it does not change much from scikit's performance.

Accuracy vs K Values:



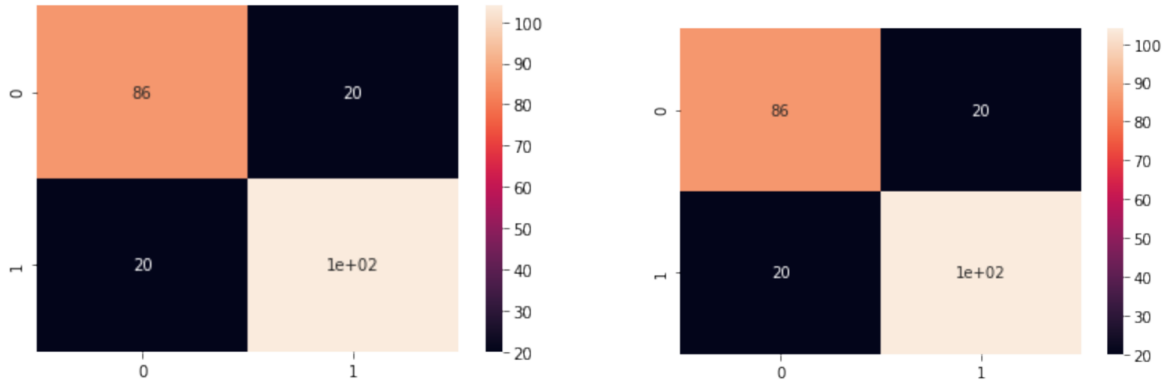
Error_rate vs K values:



We observed that we get best accuracy for K = 11 with minimum error rate with euclidean Distance for our KNN Model which gives us approximately accuracy of 82%

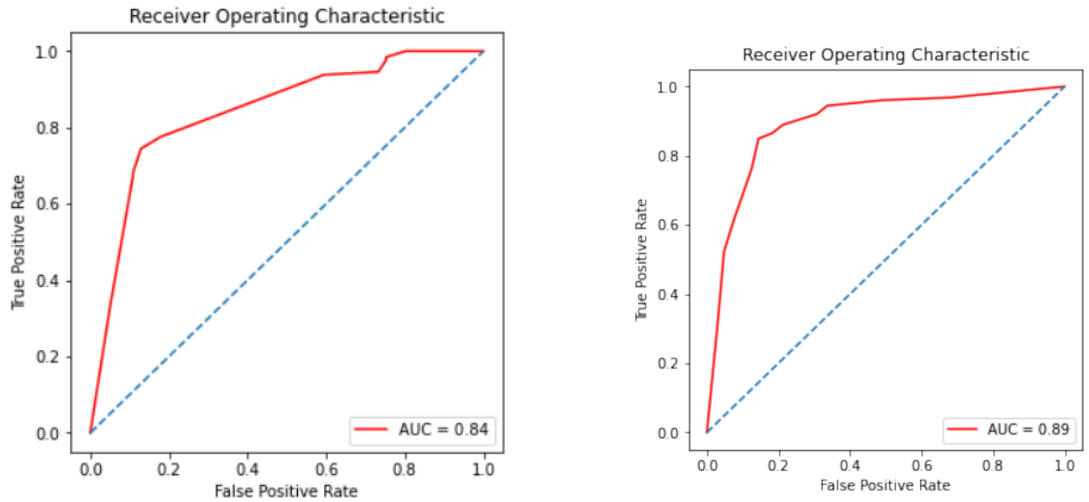
The performance of our KNN Model performs exactly the same as Scikit's KNN Model. Let's see the confusion matrix for the same.

Our KNN CF Matrix: **Scikit's KNN CF Matrix:**



The Values for Confusion matrix is also exactly the same for both of the models. Here True Positive is 102 (1e+02).

Our KNN ROC: **Scikit's ROC Curve:**

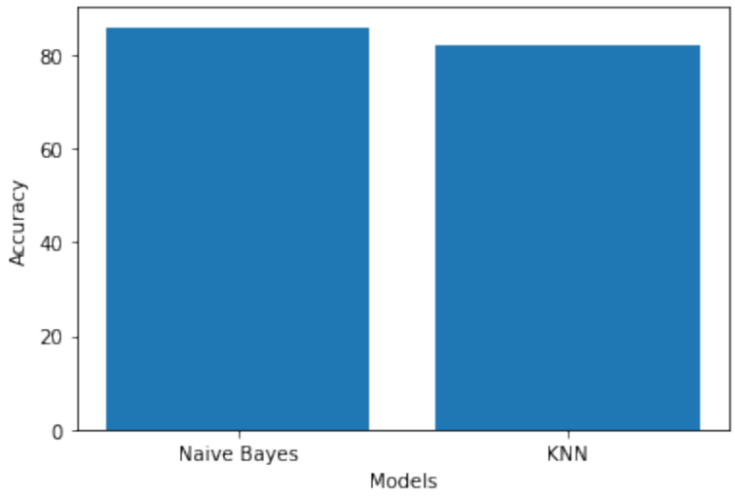


ROC Curve of Scikit is better than our ROC Curve as the probabilities varies slightly with values for Scikits predict probability function with comparison to ours.

Conclusion:

We conclude that both Naive Bayes and KNN performed pretty well with our dataset. Surprisingly, our own implemented KNN and Naive Bayes model worked similar to Scikit's KNN and Gaussian Naive Bayes.

We saw a better performance with the Gaussian Naive Bayes model as compared to KNN.



We conclude that, Our model predicts a better likelihood of a person having Heart Diseases even with Silent Heart Disease i.e Asymptomatic case. This will really help in recommending proper treatment to people with this type of cases beforehand and can save a lot of lives.