Applied Data Intelligence

Project Name: Disease X Detection Dataset

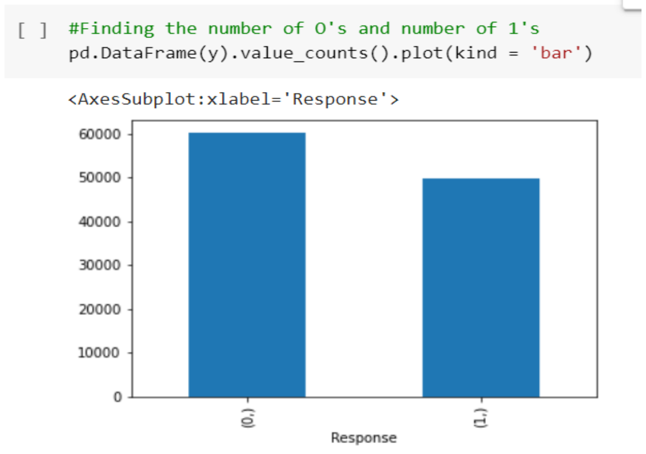
Project Description

This is a classification problem which takes the dataset of disease detection dataset. The dataset contains 90 predictor variables and 1 response variable with a total number of samples of 110,000. The project's objective is to find the best fit model which gives a better accuracy score and satisfactory confusion matrix.

Data Visualization

The given dataset has been observed and tried to find some insights from them. Below are a few key features in the dataset.

1. The dataset has no null values. By running the **X.isnull().sum()** function, we found that there are no columns which has a null value. This eliminates the abnormality or some noise in the dataset.
2. The dataset has 90 predictor variables which are labelled as X1,X2,X3….X90 and has a response variable as 'Response.'
3. The count for 'Response' column shows that there are more 0's than 1's in the dataset. The below figure 1 shows the bar plot of the distribution of the classes in Response column.

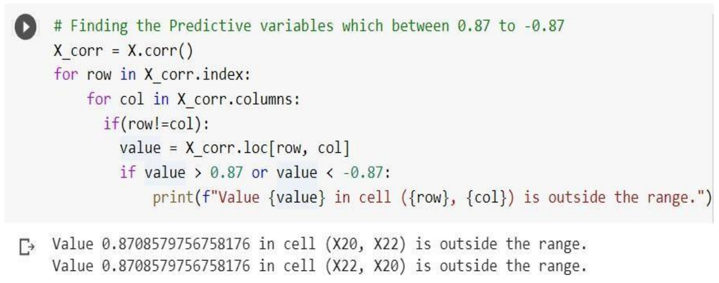


```
#Finding the number of 0's and number of 1's
pd.DataFrame(y).value_counts().plot(kind = 'bar')
```

```
<AxesSubplot:xlabel='Response'>
```

Fig 1. Response variable count distribution.

4. The correlation matrix is drawn for the input variable and tried to find if there are any variables which are related to each other. After running **X.corr()** we got 90 x 90 matrix and used for loop to sort the predictor variables which are in the range near **-0.87 to 0.87**. After running the automation code to find the related variables, we found that 22 and 20 have a slightly linear relation. Below is the loop figure 2&3 for selecting the variables in the above range and the correlation matrix.



```
# Finding the Predictive variables which between 0.87 to -0.87
X_corr = X.corr()
for row in X_corr.index:
    for col in X_corr.columns:
      if(row!=col):
        value = X_corr.loc[row, col]
        if value > 0.87 or value < -0.87:
            print(f"Value {value} in cell ({row}, {col}) is outside the range.")
```

```
Value 0.8708579756758176 in cell (X20, X22) is outside the range.
Value 0.8708579756758176 in cell (X22, X20) is outside the range.
```

Fig 2. Code for correlation matrix to find out of range detection.

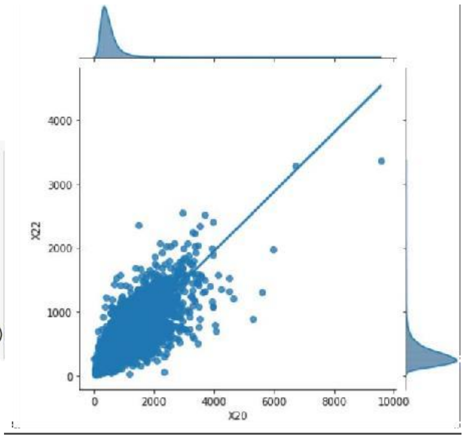
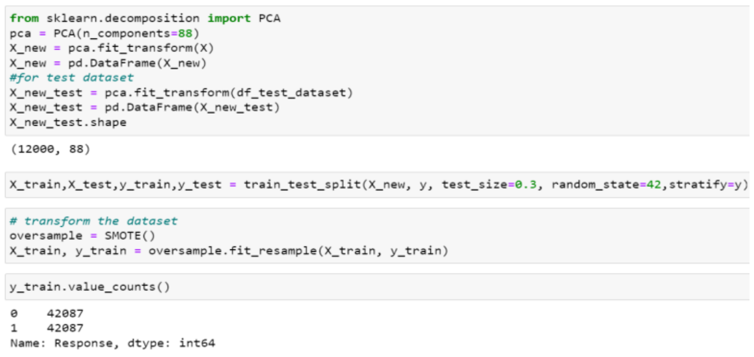


Fig 3. Scatter Plot between X20 and X22.

Data Preprocessing and Data Splitting

1. Data Preprocessing:

After running the correlation matrix, we found that there is no relation between any of the predictor variables. So, applying Principal Component Analysis (PCA) with n_components of 88 is used and the X predictors are standardized with the Standard Scaler function. For upscaling the data, we applied SMOTE to the training dataset to even out the y_train values.



```
from sklearn.decomposition import PCA
pca = PCA(n_components=88)
X_new = pca.fit_transform(X)
X_new = pd.DataFrame(X_new)
#for test dataset
X_new_test = pca.fit_transform(df_test_dataset)
X_new_test = pd.DataFrame(X_new_test)
X_new_test.shape
```

```
(12000, 88)
```

```
X_train,X_test,y_train,y_test = train_test_split(X_new, y, test_size=0.3, random_state=42,stratify=y)
```

```
# transform the dataset
oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

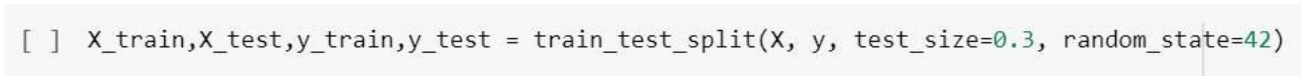
```
y_train.value_counts()
```

```
0    42087
1    42087
Name: Response, dtype: int64
```

Fig 4. Standard Scaling with PCA.

2. Data Splitting:

The dataset can be split into training and testing dataset with a percentage split of 30%, where 30% goes to the testing dataset. The splitting is done as shown in the figure below.



```
[ ]  X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

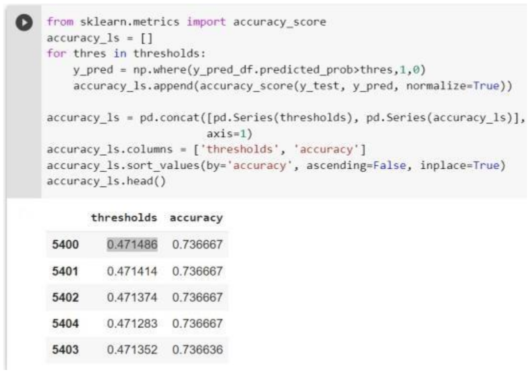
Description on Model Selection

The given dataset is a classification dataset and we have tried Logistic Regression and Decision Tree to the X_train and y_train dataset. The model which gives a better accuracy score and gives out a good confusion matrix among the two models is chosen.

1) Logistic Regression:

After splitting the dataset to training and testing dataset, the logistic regression is run using the training dataset. The predicted probability is obtained from the test dataset. The various thresholds for the predicted probability are obtained from roc curve and the threshold with the best accuracy is obtained and used for predicted output.

- With a threshold of **0.471486** has the best accuracy for the model. This was obtained by running a for loop by appending the respective accuracies to a data-frame as shown in the figure4 below.



```
from sklearn.metrics import accuracy_score
accuracy_ls = []
for thres in thresholds:
    y_pred = np.where(y_pred_df.predicted_prob>thres,1,0)
    accuracy_ls.append(accuracy_score(y_test, y_pred, normalize=True))

accuracy_ls = pd.concat([pd.Series(thresholds), pd.Series(accuracy_ls)],
                        axis=1)
accuracy_ls.columns = ['thresholds', 'accuracy']
accuracy_ls.sort_values(by='accuracy', ascending=False, inplace=True)
accuracy_ls.head()
```

```
      thresholds  accuracy
5400    0.471486  0.736667
5401    0.471414  0.736667
5402    0.471374  0.736667
5404    0.471283  0.736667
5403    0.471352  0.736636
```

Fig 5. Code to find the best threshold with best accuracy.

Performance Metrics:

After running the model, the performance is measured by knowing the accuracy and confusion matrix of the model. For the thresholds like 0.471 and 0.79 are used to get these metrics.

The confusion matrix with this threshold is shown in the figure 6.

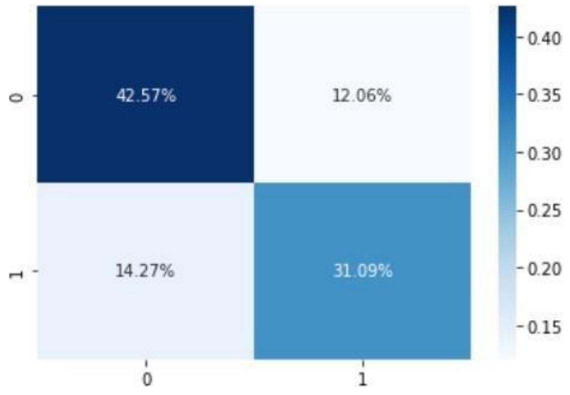


Fig 6. Heatmap showing the confusion matrix for 0.471.

Considering the (0,0) and (0,1) which are True Positives and False Negatives of the confusion matrix as the key features for our classification dataset. As we can observe from the confusion matrix there is good share between the True Positive and True Negative. The accuracy for this threshold is **73.4%.**

- With the threshold shown from the ROC_curve which is **0.797.** The confusion matrix obtained for this case is shown in the figure 7 below.

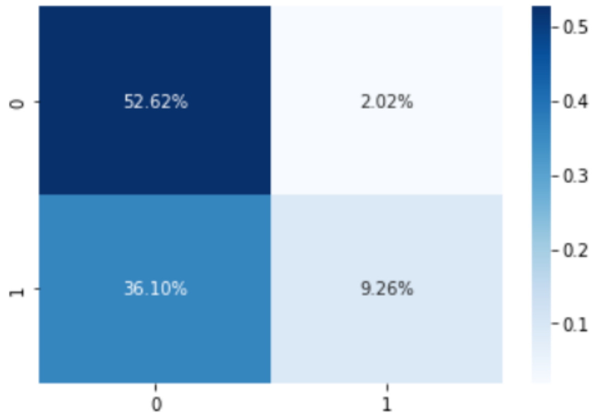


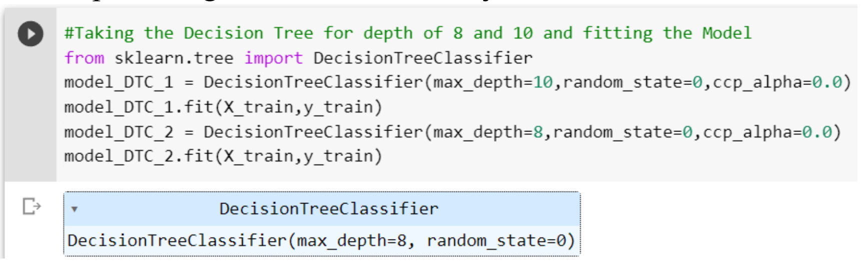
Fig 7. Heatmap showing the confusion matrix for 0.797.

As we can observe in the confusion matrix, we see that the False Negative are more which is not a good way for detection of disease. With this threshold, the model will predict that the disease is notpresent but on the other hand the disease is there in the body. The accuracy for this threshold is **61.87%.**

So, it can be concluded that using threshold of **0.471486** is better than **0.797** as in the confusion matrix the False Negative is lower for 0.471486.

2) Decision Tree Model:

The decision tree model is also used for this dataset. The depth for this is compared between 8 and 10. Itis found that with depth ‘10’ gives the best accuracy.



```
#Taking the Decision Tree for depth of 8 and 10 and fitting the Model
from sklearn.tree import DecisionTreeClassifier
model_DTC_1 = DecisionTreeClassifier(max_depth=10,random_state=0,ccp_alpha=0.0)
model_DTC_1.fit(X_train,y_train)
model_DTC_2 = DecisionTreeClassifier(max_depth=8,random_state=0,ccp_alpha=0.0)
model_DTC_2.fit(X_train,y_train)
```

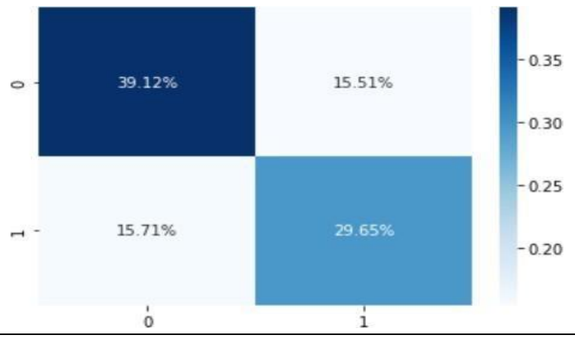
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=8, random_state=0)
```

Fig 8. Code to run the Decision Tree Model with two depths.

Performance Metrics:

The depth for the decision model was taken as 8 and 10. The comparison is done with these two thresholds to get the best suited threshold.

- For the threshold of 8 and 10, the confusion matrix is as below,



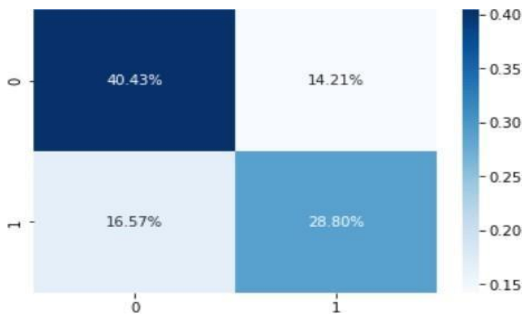
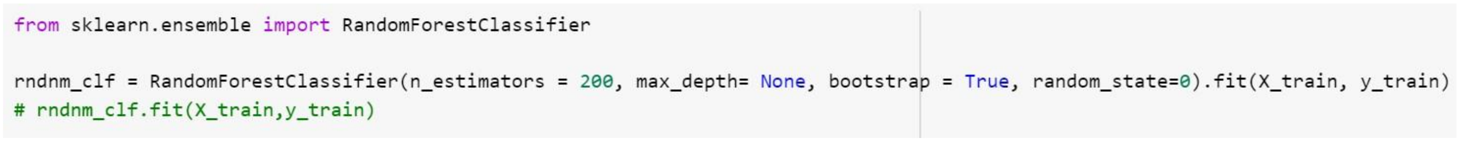


Fig9. Confusion Matrix with depth of 8. Fig10. Confusion Matrix with depth of 10.

From the above figure, we can observe that there is slight difference in the True Positive, but the overall accuracy doesn’t change much. Taking the False Negative as the best parameter ofdifferentiation. We can go with the threshold of 10 as the best suited for the model which gives accuracy of **68.77%.**

3) Random Forest Algorithm:

The Random Forest Algorithm is used for this dataset. The number of estimators used for the algorithm are 200 with bootstrap as True and a random state of 0. The algorithm is fitted and prediction with X_test is made.



```
from sklearn.ensemble import RandomForestClassifier

rndnm_clf = RandomForestClassifier(n_estimators = 200, max_depth= None, bootstrap = True, random_state=0).fit(X_train, y_train)
# rndnm_clf.fit(X_train,y_train)
```

Fig 11. Random Forest Algorithm model creation with the estimators.

Performance Matrix:

The confusion matrix for the random forest algorithm gives a good True Positive and a lower False Positive when compared to the previous models with an accuracy of **74.46%.**

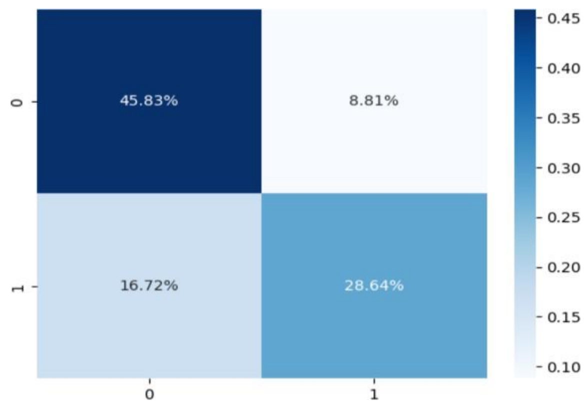
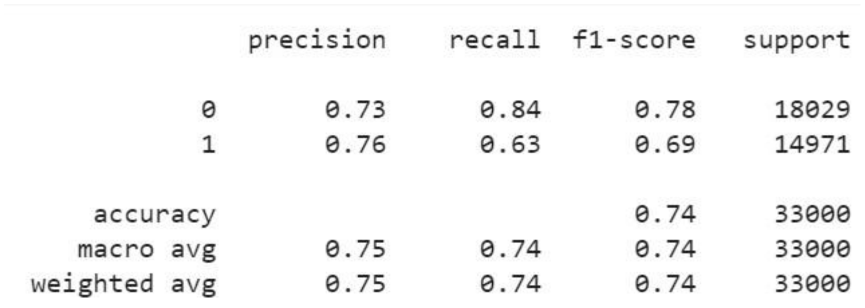


Fig 12. Confusion Matrix for Random Forest.

The below is the classification report for the random forest algorithm. It gives detailed values on the precision, recall and f1-score.



```
              precision    recall  f1-score   support

           0       0.73      0.84      0.78     18029
           1       0.76      0.63      0.69     14971

    accuracy                           0.74     33000
   macro avg       0.75      0.74      0.74     33000
weighted avg       0.75      0.74      0.74     33000
```

Fig 13. Classification Report for Random Forest.

4) Bagging, Adaptive Boosting, Gradient Boosting Model:

The dataset is run in Bagging, Adaptive Boosting and Gradient Boosting. Among the three models, both Adaptive Boosting and Gradient Boosting gave better results on confusion matrix.

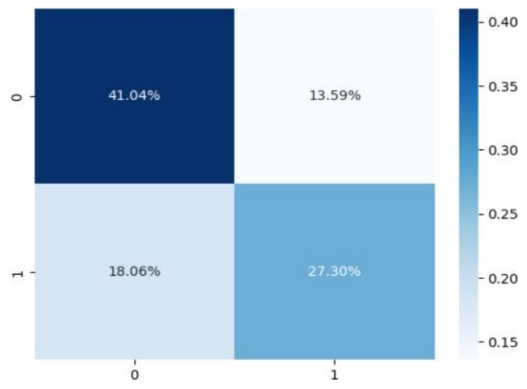


Fig14. Confusion Matrix for Bagging Model

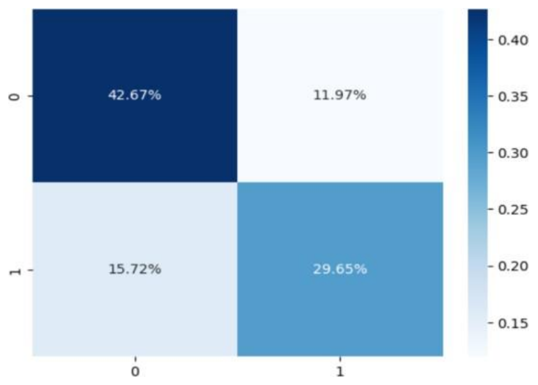


Fig 15. Confusion Matrix for Adaptive Boosting Model

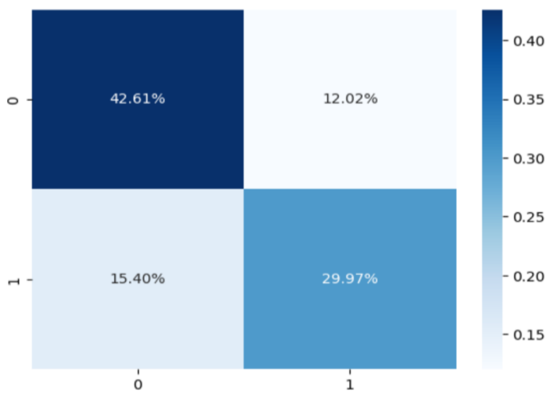
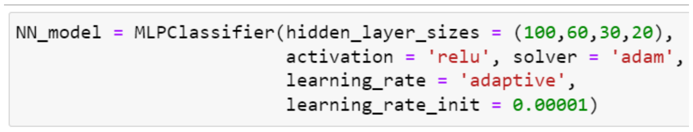


Fig 16. Confusion Matrix for Gradient Boosting Model.

There isn't much difference on the accuracy of the three models. Compared to the Logistic Regression model the accuracy of these Boosting models was less.

5) **MLP Classifier:**

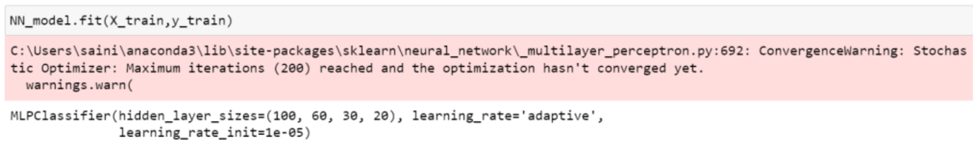
The dataset is run on an MLP Classifier model with tuning the parameters with multiple hidden layers which makes it a deep learning model. The parameter for the hidden layer size is (100,60,30,20). This requires a higher computational power for which we used our local machine to fit the model. The activation function we used is 'relu', solver is 'adam' and learning_rate which is 'adaptive'.



```
NN_model = MLPClassifier(hidden_layer_sizes = (100,60,30,20),
                         activation = 'relu', solver = 'adam',
                         learning_rate = 'adaptive',
                         learning_rate_init = 0.00001)
```

Fig 17. MLP Classifier model with multiple layers.

The model is fitted using the X_train and y_train dataset which was derived from the data splitting.



```
NN_model.fit(X_train,y_train)

C:\Users\saini\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochas
tic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(

MLPClassifier(hidden_layer_sizes=(100, 60, 30, 20), learning_rate='adaptive',
              learning_rate_init=1e-05)
```

Fig 18. MLP Classifier model fit with multiple layers.

Performance Metrics:

The confusion matrix for this model gives us the best accuracy and a desired confusion matrix with a lower false negative %age value of **11.38%.** The accuracy for the MLP Classifier model is **75.26%.** The below figure gives details on the confusion matrix.

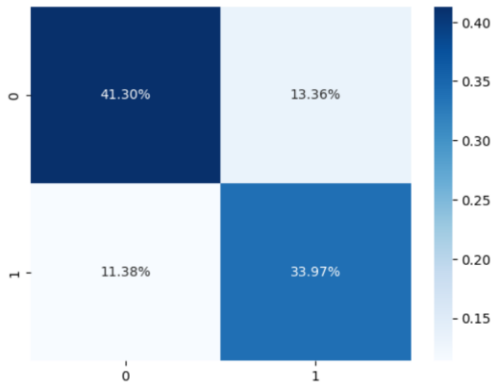
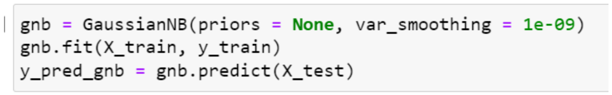


Fig 19. MLP Classifier model Confusion Matrix.

The above figure shows the ROC curve for the MLP classifier model. The ROC score of 0.75, with some fine tunning on the threshold of the ROC, the model can be performed at a good rate.

6) Naïve Bayes Theorem:

The dataset is run on a Naïve bayes model, the required metrics and the evaluation of the model is done. For this model, the var_smoothing of 1e-09 is given and the model is fitted.



```
gnb = GaussianNB(priors = None, var_smoothing = 1e-09)
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
```

Fig 20. Naïve Bayes Theorem Model fitting.

The y_pred_gnb is found using the predict for the X_test of the model which is further used for the performance of the model.

Performance Metrics:

We ran the accuracy score, confusion matrix, ROC_curve and the classification report for the predicted dataset. The accuracy score for this model is **52.66%** which is not an appreciable accuracy. This means that when a new dataset goes into the model then the chances of it getting predicted is just 50% which is like a coin toss.

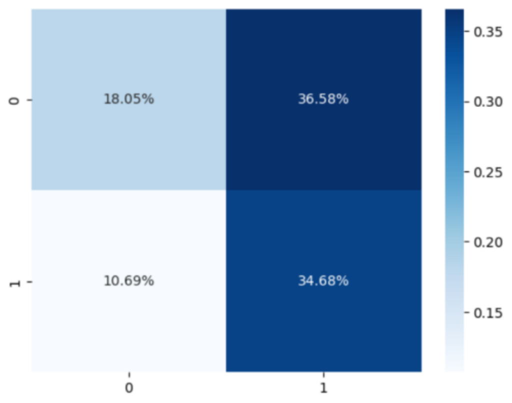
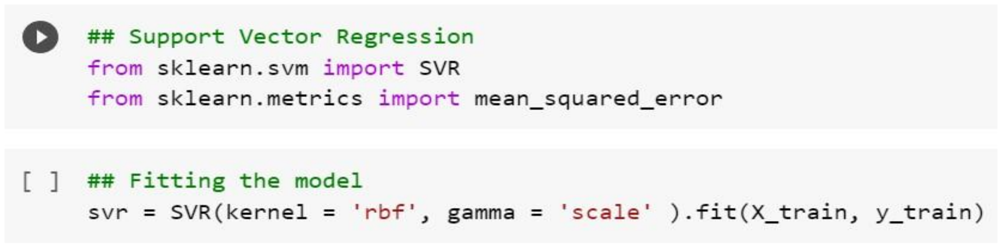


Fig 21.Naive Theorem confusion matrix.

The above figure shows the confusion matrix for the Gaussian Naïve theorem. We can observe that the false negative for the model is 10.69% which is significantly low but the accuracy is not really good. So, choosing this model for the dataset is not acceptable and can not go forward with this.

7) Support Vector Regression Model:

The dataset is run on a Support Vector Machine model with tuning parameters of kernels as rbf and gamma with ‘scale’ parameter. These are run on a grid search with SVR() as the model.



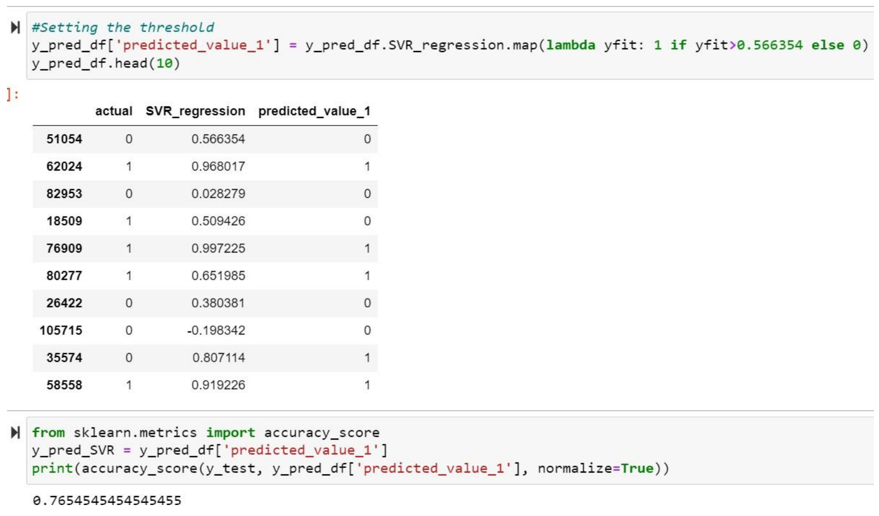
```
## Support Vector Regression
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

```
[ ]  ## Fitting the model
     svr = SVR(kernel = 'rbf', gamma = 'scale' ).fit(X_train, y_train)
```

Fig 22. Support Vector Regression Model Creation

The SVR model then fit with the X_test data. The test data fit is a continuous dataset which needs to be converted to a distinct dataset with 0’s and 1’s. The conversion is done by setting a threshold on the regression output.

By using trial and error method, we found that 0.56 is giving a desired output with better accuracy of 76.54% which is highest compared to other models.



```
#Setting the threshold
y_pred_df['predicted_value_1'] = y_pred_df.SVR_regression.map(lambda yfit: 1 if yfit>0.566354 else 0)
y_pred_df.head(10)
```

	actual	SVR_regression	predicted_value_1
51054	0	0.566354	0
62024	1	0.968017	1
82953	0	0.028279	0
18509	1	0.509426	0
76909	1	0.997225	1
80277	1	0.651985	1
26422	0	0.380381	0
105715	0	-0.198342	0
35574	0	0.807114	1
58558	1	0.919226	1

```
from sklearn.metrics import accuracy_score
y_pred_SVR = y_pred_df['predicted_value_1']
print(accuracy_score(y_test, y_pred_df['predicted_value_1'], normalize=True))
```

```
0.7654545454545455
```

Fig 23. SVR with the threshold value.

Performance metrics:

By running the confusion for the above predicted output, we can see that the False Negative for the model is higher compared to our previous best model (i.e., Logistic Regression). We can understand by this that even with a higher accuracy for this model, the confusion matrix is not the expected output which was needed. The false negative in the confusion matrix indicates that the disease is not detected but in actual the disease is present. The FN is the value which needs to be lower to get a desired output.

```
<AxesSubplot:>
```

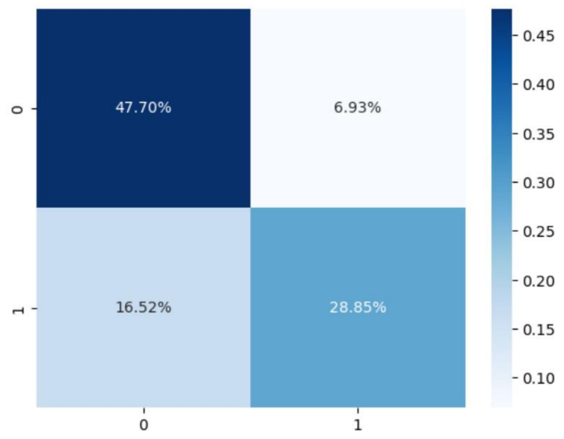


Fig 24. The confusion matrix for SVR.

<u>**ROC Curve:**</u>

The ROC Curve is taken for model which gives out the best confusion matrix and better accuracy.

The ROC curve is drawn for the Logistic Regression to get the best suited threshold for the predicted probability of the response. The ROC curve or the area under the curve is **0.80** which is in figure 10 below.

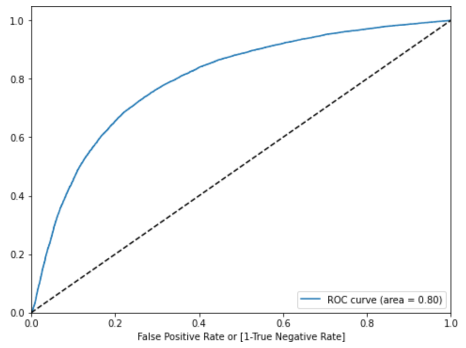


Fig 25. ROC Curve for Logistic Regression.

The ROC curve for Support Vector Regression Model which gives out the best suited threshold and auc_score for the predicted values of the model. The AUC_score for the model is **0.75**.

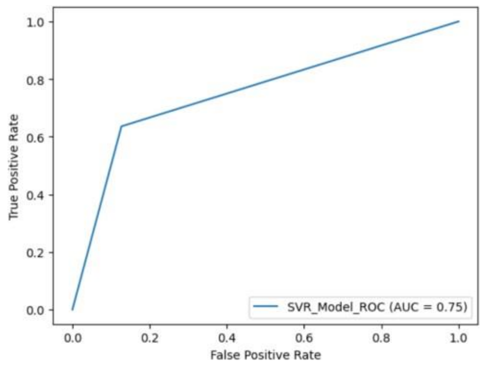


Fig 26. ROC Curve for SVR Model

The ROC curve for Random Forest Model which gives out the best suited threshold and auc_score for the predicted values of the model. The AUC_score for the model is **0.72**.

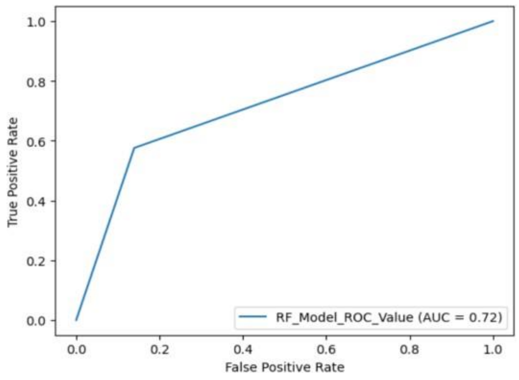


Fig 27. The ROC Curve for Random Forest Model.

The ROC_curve for the Naïve theorem model is as above which is **0.55.**

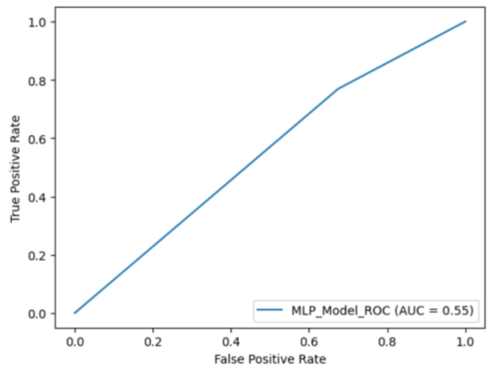


Fig 28.Naive Theorem ROC_Curve.

As we can see in the above figure that the false negative has been reduced compared to other models and with a good accuracy of 75.26%.

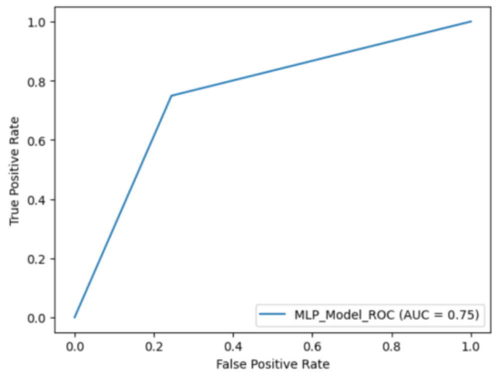


Fig 29. MLP Classifier model with ROC_Curve.

Lessons Learned:

- The selection of the model plays a crucial role in modelling the given dataset.
- Data visualization is important for further processing of the given dataset. As we found that our dataset is clear with no noise, we didn't perform any techniques on them. Still, we performed SMOTE to the dataset.
- Accuracy is not the only parameter to conclude the model selection. As we can observe from our analysis, we used the confusion matrix as an important metric to eliminate the models. We focused on the False negatives to be reduced and select the model which gives the better percentage on that.

Conclusion:

In conclusion we can say that by trying different models into the dataset, we understood the functionality of the models and the parameters of it. We can see that MLP Classifier and Support Vector Regression have given out good confusion matrix and accuracy score. The result is better than Logistic Regression confusion matrix from other deliverables for MLP Classifier as the dataset is more inclined towards non-linear model.