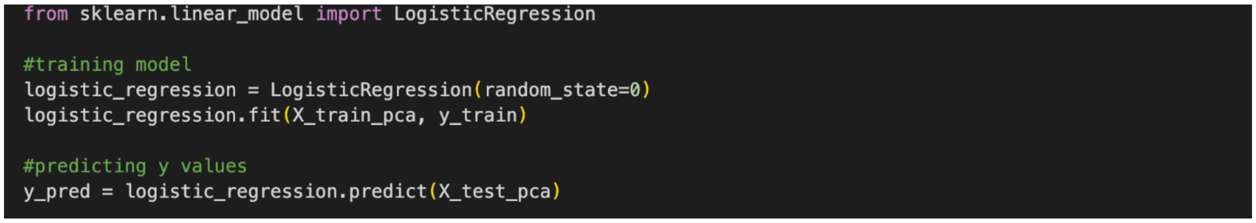
CSE 587 -B Project Phase – 2

Report

Logistic Regression:

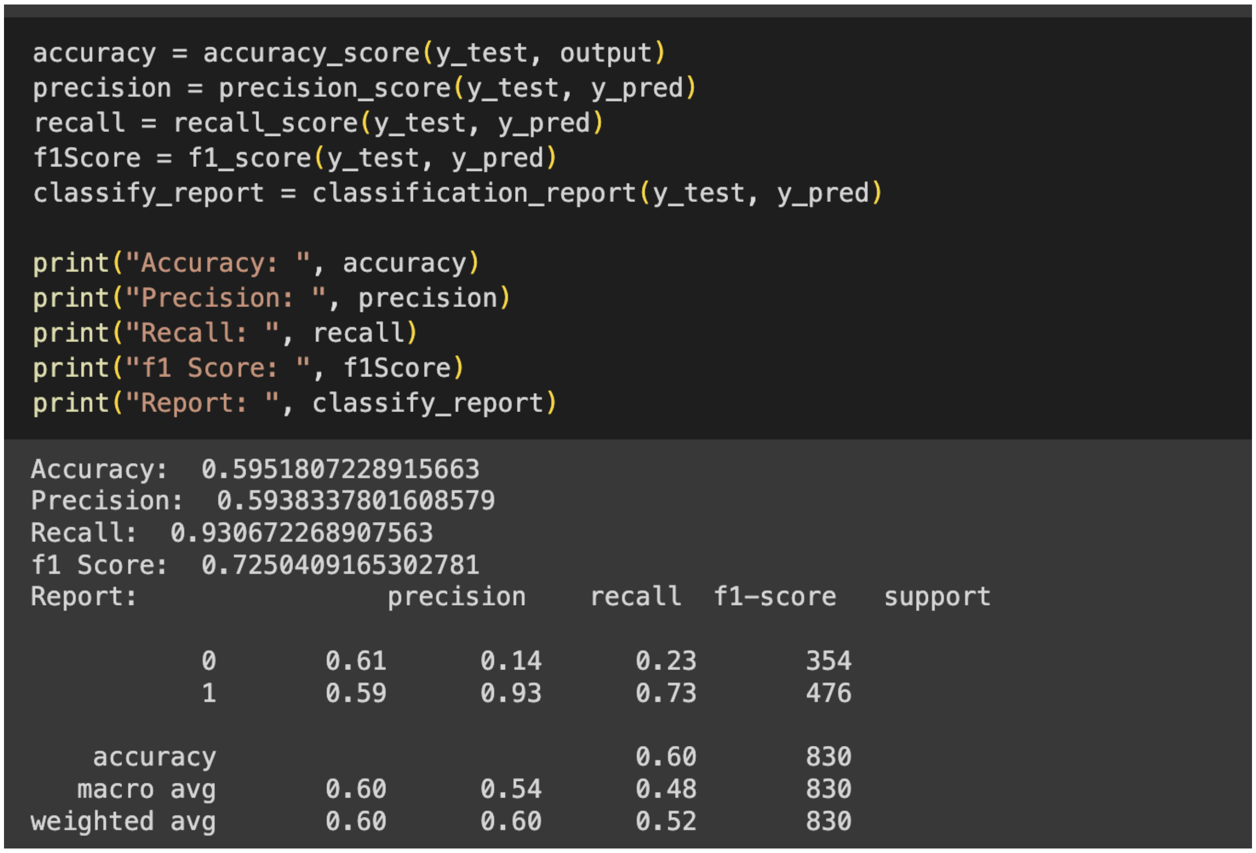
We chose logistic regression for our problem statement because logistic regression model is more efficient and even with low computational resources, it can handle large datasets. When compared to more complex models, this model is less prone to overfitting. Logistic regression is a type of model which is very easy to implement, interpret the values and efficient to train. Logistic regression helps in identifying the features that are important in predicting the results or outcome. This model can also handle many features and scalable which makes it apt for situation where feature number is high. We can also use logistic regression even if the dataset we have is small and this model provides better results even with limited data. With all these advantages, we chose logistic regression to train the data and predict the values for our problem statement.

For the process of training the model, we have first initialized the algorithm and just ran it without any optimization techniques as shown below:



```
from sklearn.linear_model import LogisticRegression

#training model
logistic_regression = LogisticRegression(random_state=0)
logistic_regression.fit(X_train_pca, y_train)

#predicting y values
y_pred = logistic_regression.predict(X_test_pca)
```

After training the model and predicting the values, we have calculated all the performance metrics such as accuracy, precision, recall, etc.



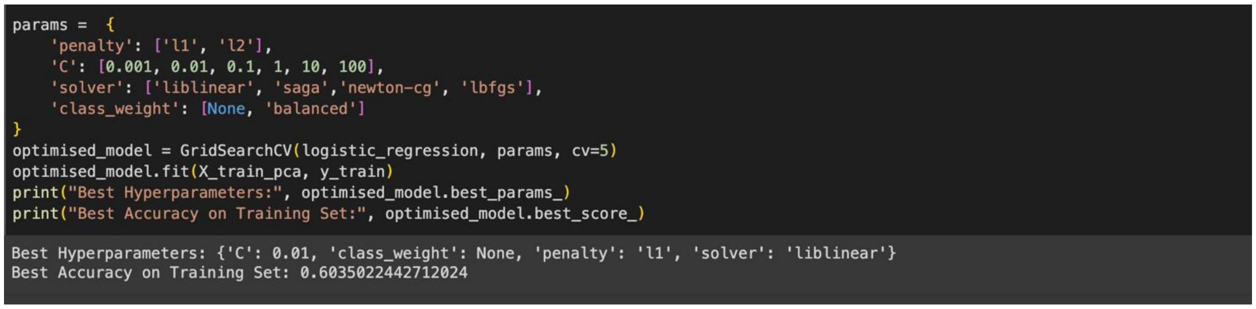
```
accuracy = accuracy_score(y_test, output)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1Score = f1_score(y_test, y_pred)
classify_report = classification_report(y_test, y_pred)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

```
Accuracy:  0.5951807228915663
Precision:  0.5938337801608579
Recall:  0.930672268907563
f1 Score:  0.7250409165302781
Report:                 precision    recall  f1-score   support

           0       0.61      0.14      0.23       354
           1       0.59      0.93      0.73       476

    accuracy                           0.60       830
   macro avg       0.60      0.54      0.48       830
weighted avg       0.60      0.60      0.52       830
```

Accuracy is generally defined as the measure of overall correctness of the model used. The accuracy achieved by the logistic regression model is 59.5 percent, precision value achieved is 59.3 percent, recall value is 93.06 percent and other values are also recorded.

Now, to fine-tune the model and train it more effectively we applied the grid search method provided to the model where the parameters are penalty, solver, class weight and C where penalty means L1 and L2 regularization, solver means the algorithm that we use during the optimization problem.

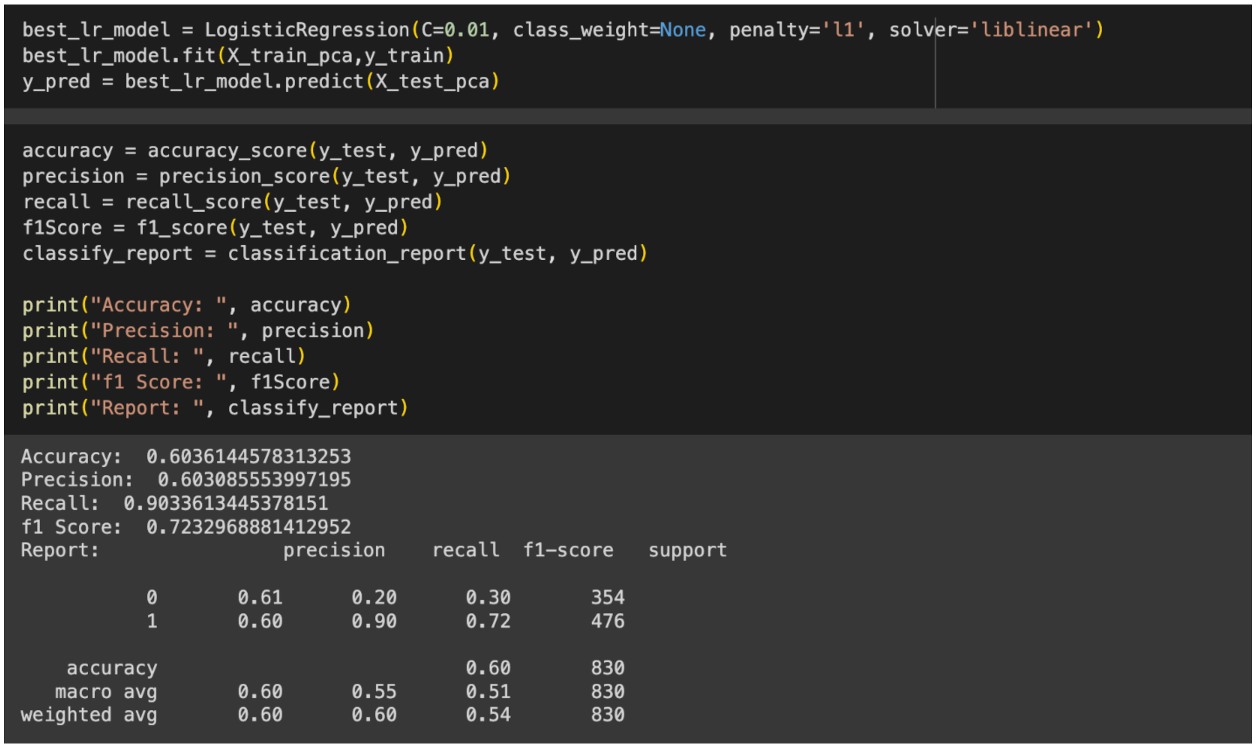
After applying the grid search, we got the following values as the best parameters for my model,



```
params =  {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga','newton-cg', 'lbfgs'],
    'class_weight': [None, 'balanced']
}
optimised_model = GridSearchCV(logistic_regression, params, cv=5)
optimised_model.fit(X_train_pca, y_train)
print("Best Hyperparameters:", optimised_model.best_params_)
print("Best Accuracy on Training Set:", optimised_model.best_score_)
```

```
Best Hyperparameters: {'C': 0.01, 'class_weight': None, 'penalty': 'l1', 'solver': 'liblinear'}
Best Accuracy on Training Set: 0.6035022442712024
```

Now, keeping these parameters and training the model again, we have tested this model on the test data and got the following results:


```
best_lr_model = LogisticRegression(C=0.01, class_weight=None, penalty='l1', solver='liblinear')
best_lr_model.fit(X_train_pca,y_train)
y_pred = best_lr_model.predict(X_test_pca)
```

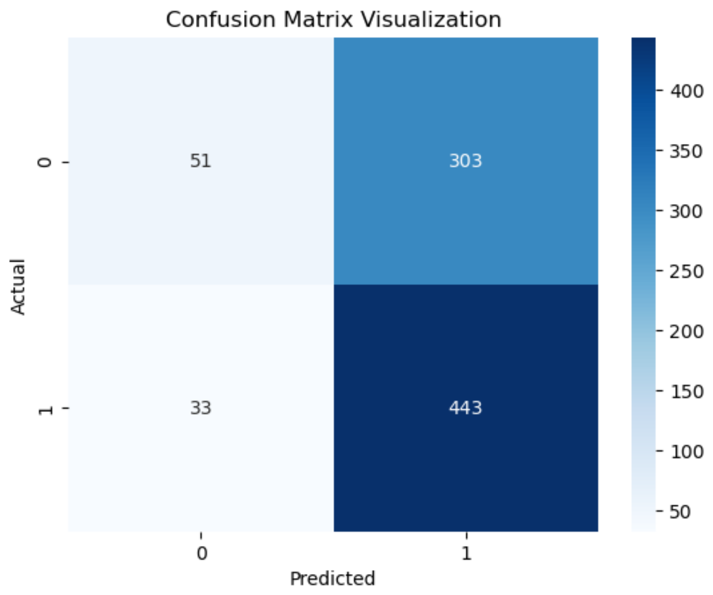
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1Score = f1_score(y_test, y_pred)
classify_report = classification_report(y_test, y_pred)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

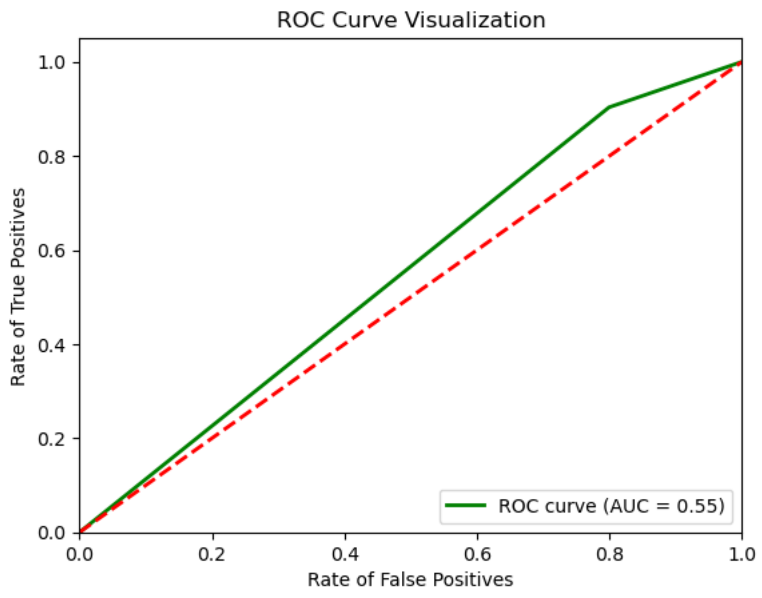
```
Accuracy:  0.6036144578313253
Precision:  0.603085553997195
Recall:  0.9033613445378151
f1 Score:  0.7232968881412952
Report:                precision    recall  f1-score   support

           0       0.61      0.20      0.30       354
           1       0.60      0.90      0.72       476

    accuracy                           0.60       830
   macro avg       0.60      0.55      0.51       830
weighted avg       0.60      0.60      0.54       830
```

<u>Analysis:</u>

From the above snippet, we can see that after applying the grid search method, the accuracy of the model increased from 59.5 percent to 60.36 percent. Now we have trained the model completely. As we can see, the accuracy of the model is 60.36 percent which means the model predicted 60.36 percent of the values correctly. The precision value is 60.30 percent which means the ratio of the positive outcomes i.e., the ratio of true positives to the sum of true positives and false negatives. Recall, which is the count of true cases in the model is about 90.3 percent, and finally the f1 score which is the harmonic mean of precision and recall values is 72.3 percent. The f1 score indicated that there is a good balance between the recall and precision values.

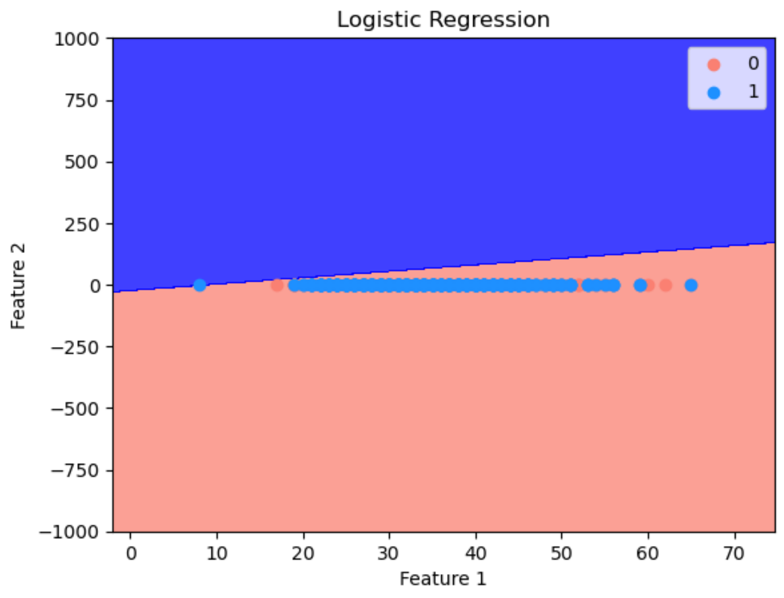
<u>Confusion matrix and ROC curve snippets are also attached:</u>

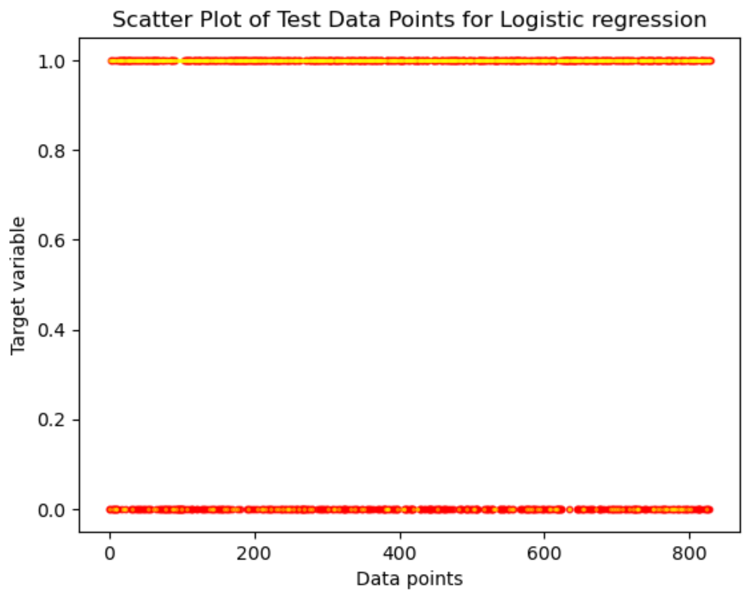




From the confusion matrix, we can understand that the model predicted the data equally which means half of them are true predictions and half of them are false.

Decision Boundary and Scatter Plot graphs are also attached:





As we can see from the above snippets, the blue points in the orange region are the wrongly predicted labels and vice versa.

<u>According to the problem statement:</u>

As we can see, by using the logistic regression model, we have achieved an accuracy of 60.3 percent. This means the model correctly predicts 60.3% that whether a person will take mental health treatment or not. Coming to recall, the percentage was 90.3 which tells us that the model captured the data of the individuals who are going to take mental health treatment. For the precision, the model has achieved 60.30% meaning that a person will take mental health treatment and it is correct for this amount of percentage. Finally, coming to the f1 score, the harmonic means of precision and recall value is 72.3%. This indicates that there is a good balance between the recall and precision values.

To conclude, we have used Logistic Regression for our problem statement to analyze the dataset, get the results and required graphs. We have also used hyperparameters and optimized the performance of the model. With all the values of recall, accuracy and precision, the model could have done better in terms of providing the accuracy. More feature and model modifications may be needed to improve the model's performance.

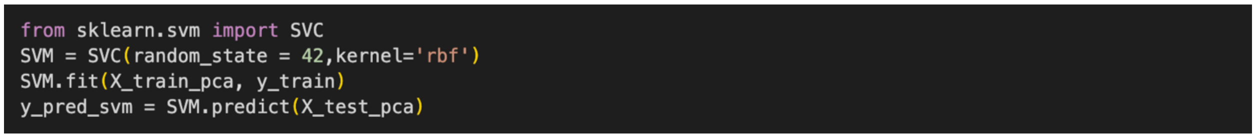
<u>Issues with this model:</u>

Some of the issues with the logistic regression model is that the model is sensitive to outliers. The coefficients and the predictions can be influenced by outliers thereby reducing the accuracy of the model. And tuning the hyperparameters can also impact performance. There is also a chance that model may not perform as expected due to presence of imbalanced classes in the dataset. Feature engineering and relevant feature selection is also very important to improve the accuracy of the model. The model performance can be increased by addressing all these problems that includes data exploration, feature analysis, and hyperparameter adjustment.

<u>Support Vector Machine:</u>

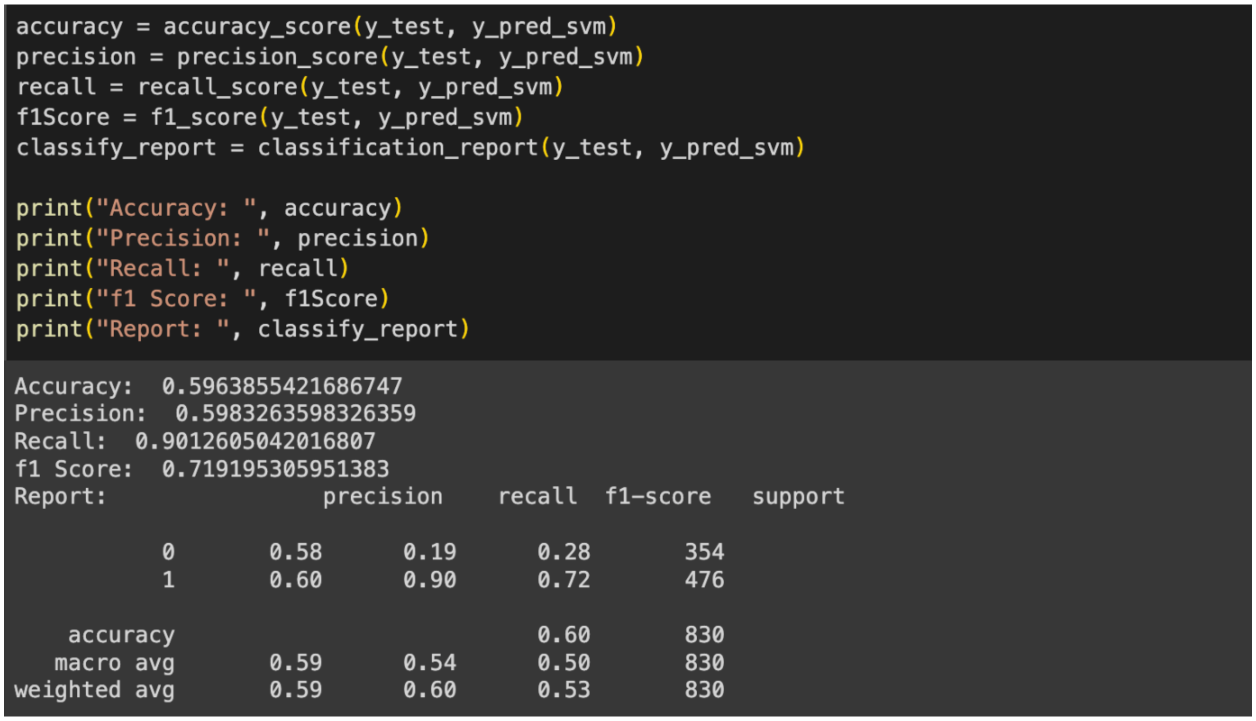
Support Vector Machines (SVM) is chosen for our problem statement because for both classification and regression tasks, SVM can be used. SVM is flexible so it can be applied to solve many problems. One of the reasons we chose SVM is that it is less prone to overfitting. Regularization can also be applied on SVM which makes the model to deliver better accuracy and other parameters. They also can handle high-dimensional data and used in many applications like image and text classification. We chose support vector machines model to train the data and predict the values for our problem statement since it has many advantages.

In the process of training the model, the algorithm is first initialized, and we just ran it without any optimization techniques as shown below:



```
from sklearn.svm import SVC
SVM = SVC(random_state = 42,kernel='rbf')
SVM.fit(X_train_pca, y_train)
y_pred_svm = SVM.predict(X_test_pca)
```

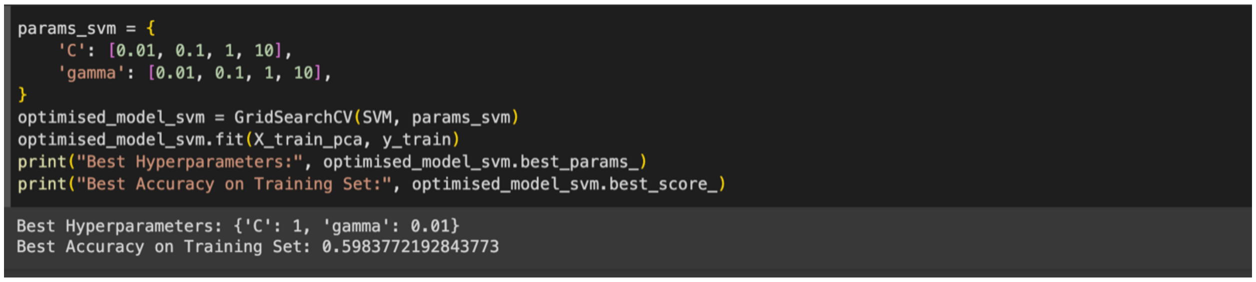
We have used sklearn library which is a popular machine learning library for our model training. With all the parameters set (random_state and kernel), we are training the model to get the patterns and make predictions. The performance metrics were obtained as follows:



```
accuracy = accuracy_score(y_test, y_pred_svm)
precision = precision_score(y_test, y_pred_svm)
recall = recall_score(y_test, y_pred_svm)
f1Score = f1_score(y_test, y_pred_svm)
classify_report = classification_report(y_test, y_pred_svm)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

```
Accuracy:  0.5963855421686747
Precision:  0.5983263598326359
Recall:  0.9012605042016807
f1 Score:  0.719195305951383
Report:                precision    recall  f1-score   support

           0       0.58      0.19      0.28       354
           1       0.60      0.90      0.72       476

    accuracy                           0.60       830
   macro avg       0.59      0.54      0.50       830
weighted avg       0.59      0.60      0.53       830
```

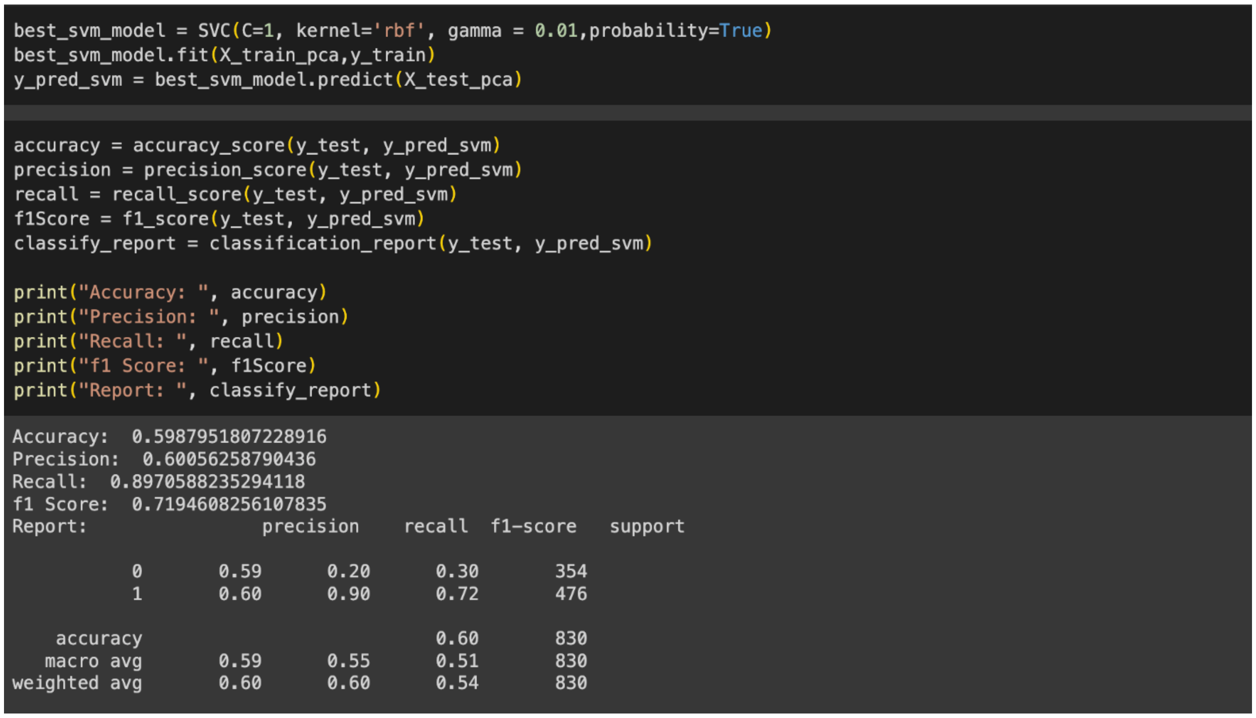
To train the model more effectively and fine-tune it, we have applied the grid search method provided to the model where the parameters are C and gamma where C is the parameter for regularization and gamma defines the influence of an example training model.



```
params_svm = {
    'C': [0.01, 0.1, 1, 10],
    'gamma': [0.01, 0.1, 1, 10],
}
optimised_model_svm = GridSearchCV(SVM, params_svm)
optimised_model_svm.fit(X_train_pca, y_train)
print("Best Hyperparameters:", optimised_model_svm.best_params_)
print("Best Accuracy on Training Set:", optimised_model_svm.best_score_)
```

```
Best Hyperparameters: {'C': 1, 'gamma': 0.01}
Best Accuracy on Training Set: 0.5983772192843773
```

As we can see from the above snippet, these are the best values for parameters for our model. So, we use these parameter values and train the dataset:



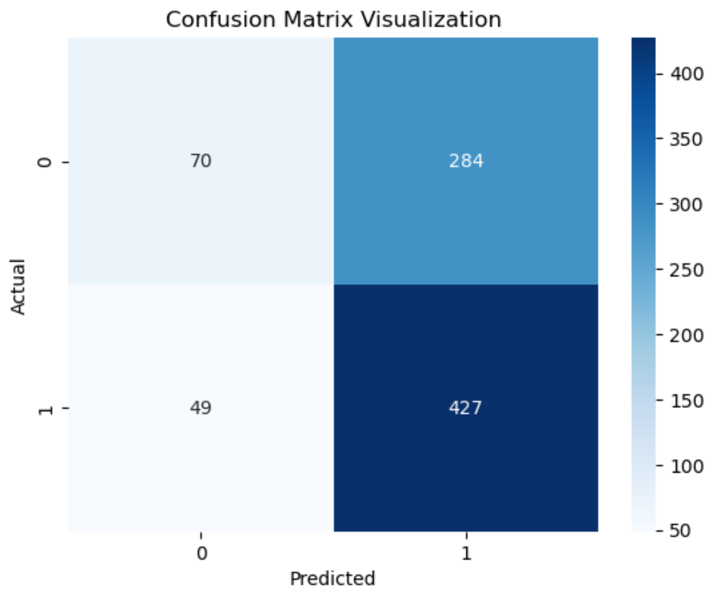
```
best_svm_model = SVC(C=1, kernel='rbf', gamma = 0.01,probability=True)
best_svm_model.fit(X_train_pca,y_train)
y_pred_svm = best_svm_model.predict(X_test_pca)
```

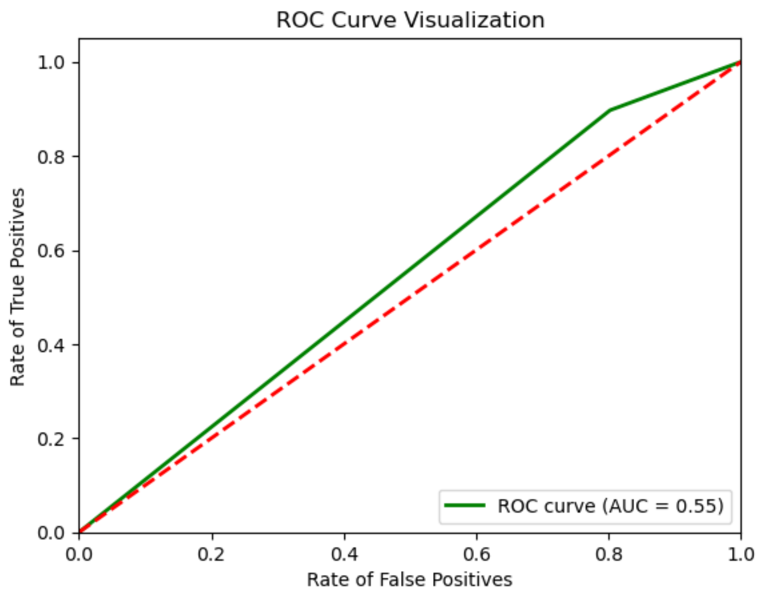
```
accuracy = accuracy_score(y_test, y_pred_svm)
precision = precision_score(y_test, y_pred_svm)
recall = recall_score(y_test, y_pred_svm)
f1Score = f1_score(y_test, y_pred_svm)
classify_report = classification_report(y_test, y_pred_svm)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

```
Accuracy:  0.5987951807228916
Precision:  0.60056258790436
Recall:  0.8970588235294118
f1 Score:  0.7194608256107835
Report:                precision    recall  f1-score   support

           0       0.59      0.20      0.30       354
           1       0.60      0.90      0.72       476

    accuracy                           0.60       830
   macro avg       0.59      0.55      0.51       830
weighted avg       0.60      0.60      0.54       830
```

We can see that the accuracy of the model is increased when the hyperparameters are used. The accuracy of the model is 59.8% whereas the precision and recall values are 60% and 89.7% respectively. The accuracy is 59.8% meaning that the prediction made by model correctly tells us that whether a person will take mental health treatment or not 59% of the time.

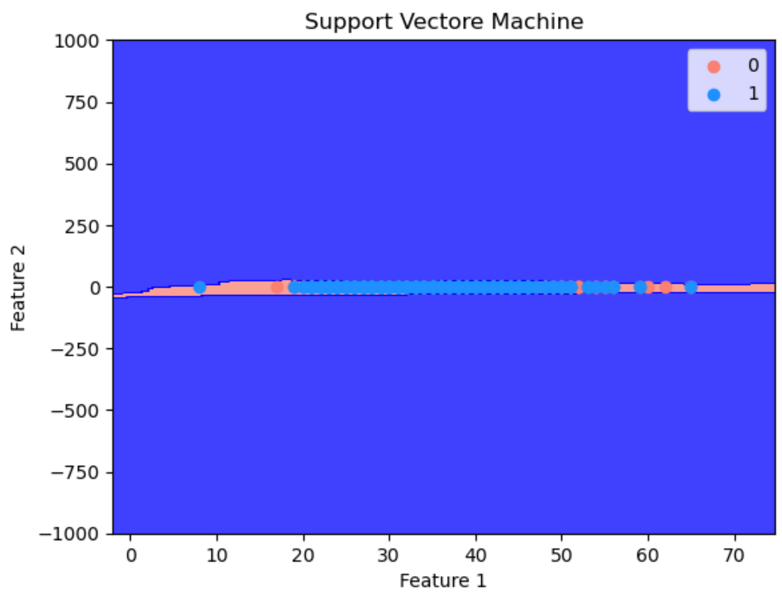
<u>The confusion matrix and ROC curve of the model are also attached:</u>

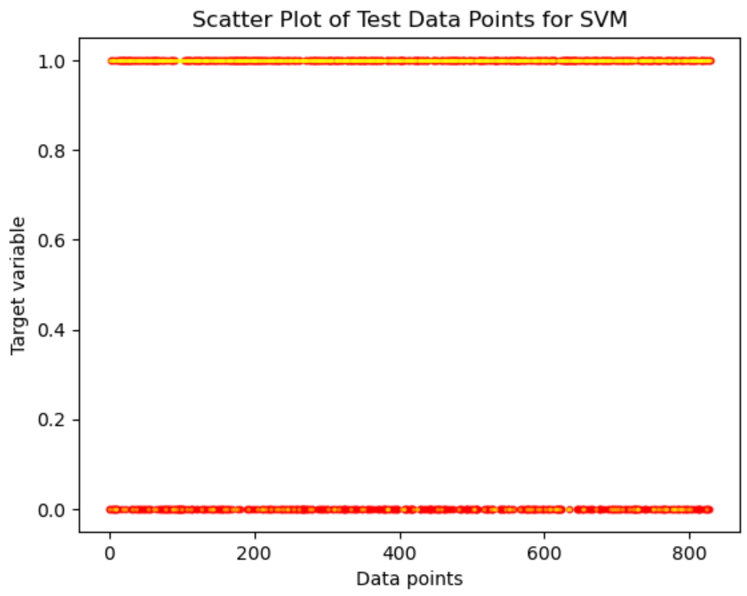




As you can see here, there are 427+70 true to true predictions, 284+49 are predicted incorrectly. From the confusion matrix, we can understand that the model predicted the data equally which means half of them are true predictions and half of them are false.

The decision boundary and scatter plot of the model are also attached to check how many points are misrepresented by the model: As we can see from the graphs, the blue point in the orange area is miscalculated by our model and vice versa.





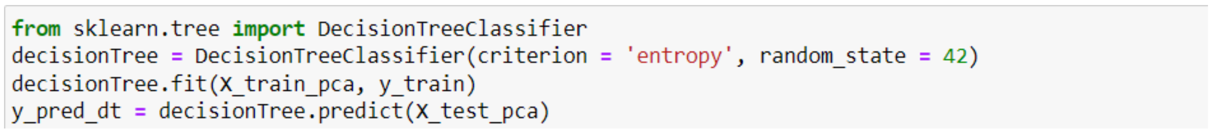
Issues with the SVM model:

One of the main reasons for the less accuracy of the model is that SVMs are sensitive to scaling the features which impacts the performance of the model. Presence of imbalanced classes in the dataset may also affect the performance of the model. Sometimes the noise that is present in the dataset or any other irrelevant features also impact the performance of the model. By addressing all these problems that includes data exploration, feature analysis, and hyperparameter adjustment, the model performance can be increased.

Decision Tree

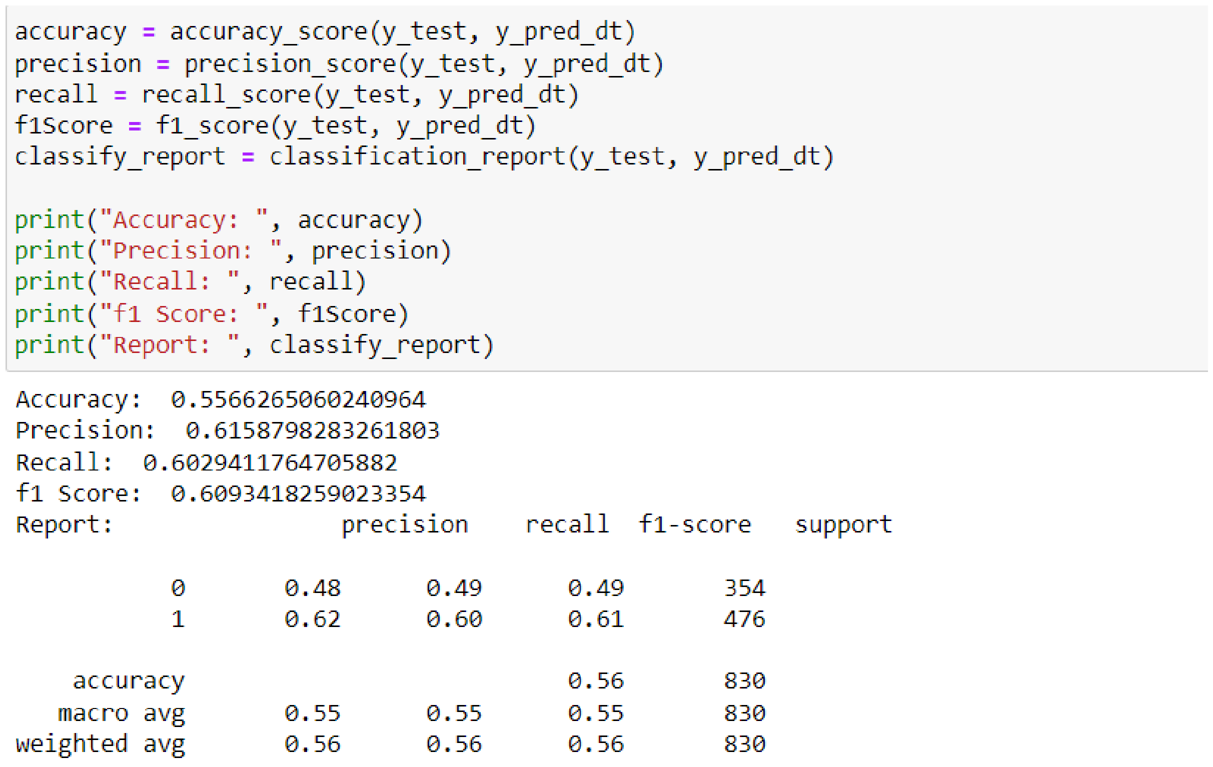
We have chosen Decision tree as it favours our binary classification problem because it's simpler to implement and it has the capacity to handle non-linear relationships. Decision trees prioritizes features and are robust to outliers. They also rapidly classify new records and perform feature selection by ignoring irrelevant ones to make predictions. Decision trees uses a hierarchical approach, which starts from the top with a single root and branches out to leaf nodes. It uses recursive partitioning to repeatedly split the data into different nodes. The decision tree organizes the data into regions according to the density. The resultant decision tree provides clear predictions in distinct categories.

We first implemented the model without using any optimization techniques as below:



```
from sklearn.tree import DecisionTreeClassifier
decisionTree = DecisionTreeClassifier(criterion = 'entropy', random_state = 42)
decisionTree.fit(X_train_pca, y_train)
y_pred_dt = decisionTree.predict(X_test_pca)
```

For our basic model implementation, below is the demonstration of the performance metrics:



```
accuracy = accuracy_score(y_test, y_pred_dt)
precision = precision_score(y_test, y_pred_dt)
recall = recall_score(y_test, y_pred_dt)
f1Score = f1_score(y_test, y_pred_dt)
classify_report = classification_report(y_test, y_pred_dt)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

```
Accuracy:  0.5566265060240964
Precision:  0.6158798283261803
Recall:  0.6029411764705882
f1 Score:  0.6093418259023354
Report:                precision    recall  f1-score   support

           0       0.48      0.49      0.49       354
           1       0.62      0.60      0.61       476

    accuracy                           0.56       830
   macro avg       0.55      0.55      0.55       830
weighted avg       0.56      0.56      0.56       830
```

Optimization-

To tune the hyperparameters, we use gridsearch to train the model with all the hyperparameter combinations on the training dataset and finds the best hyperparameters that yield best performance. Below are the best hyperparameters returned from the gridsearch after cross-validation:

```
Best Hyperparameters: {'class_weight': None, 'criterion': 'gini', 'max_features': 'sqrt', 'min_samples_leaf': 4, 'splitter': 'r
andom'}
```

```
best_decisionTree = DecisionTreeClassifier(criterion = 'gini', random_state = 42, class_weight=None,
                                           min_samples_leaf=4,splitter='random',max_features='sqrt')
best_decisionTree.fit(X_train_pca, y_train)
y_pred_dt = best_decisionTree.predict(X_test_pca)
```

The performance metrics after running test data on our updated model are as shown:

```
accuracy = accuracy_score(y_test, y_pred_dt)
precision = precision_score(y_test, y_pred_dt)
recall = recall_score(y_test, y_pred_dt)
f1Score = f1_score(y_test, y_pred_dt)
classify_report = classification_report(y_test, y_pred_dt)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("f1 Score: ", f1Score)
print("Report: ", classify_report)
```

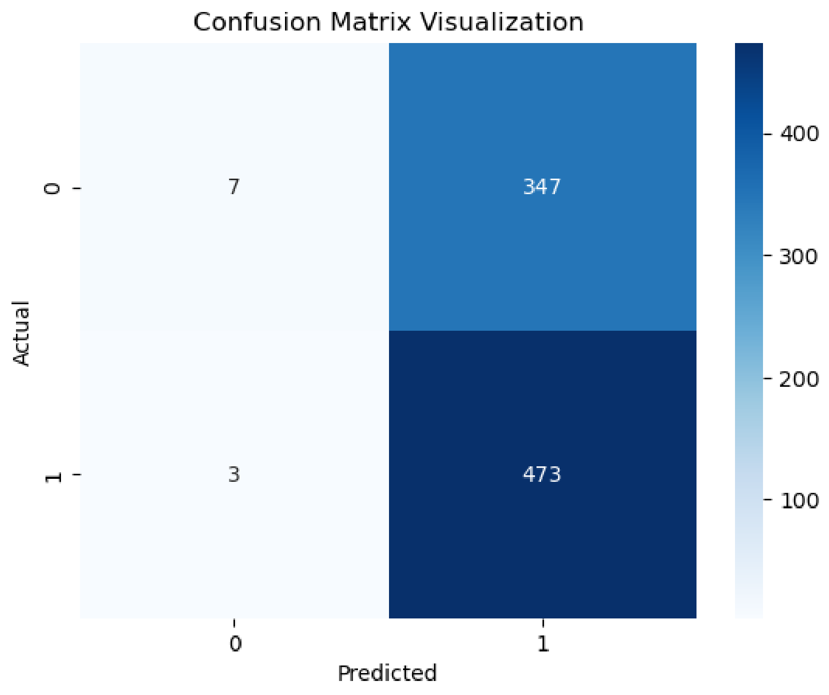
```
Accuracy:  0.5783132530120482
Precision:  0.5768292682926829
Recall:  0.9936974789915967
f1 Score:  0.7299382716049383
Report:                precision    recall  f1-score   support

           0       0.70      0.02      0.04       354
           1       0.58      0.99      0.73       476

    accuracy                           0.58       830
   macro avg       0.64      0.51      0.38       830
weighted avg       0.63      0.58      0.44       830
```

As observed from the above metrics, we can say that post optimizing our model, the test accuracy has increased from 55.6 to 57.8

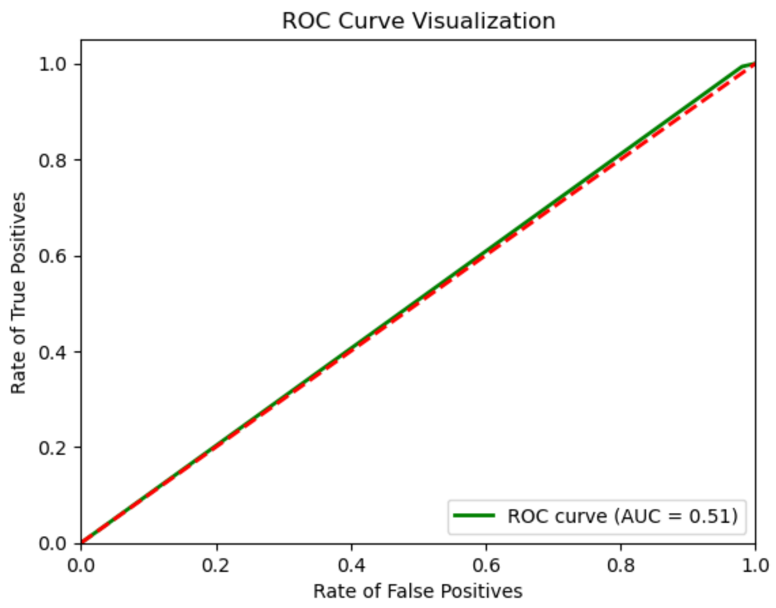
The Accuracy of the model is 57.8 which means the model can correctly predict where a person will be taking treatment for mental health or not 57.8% of the time.

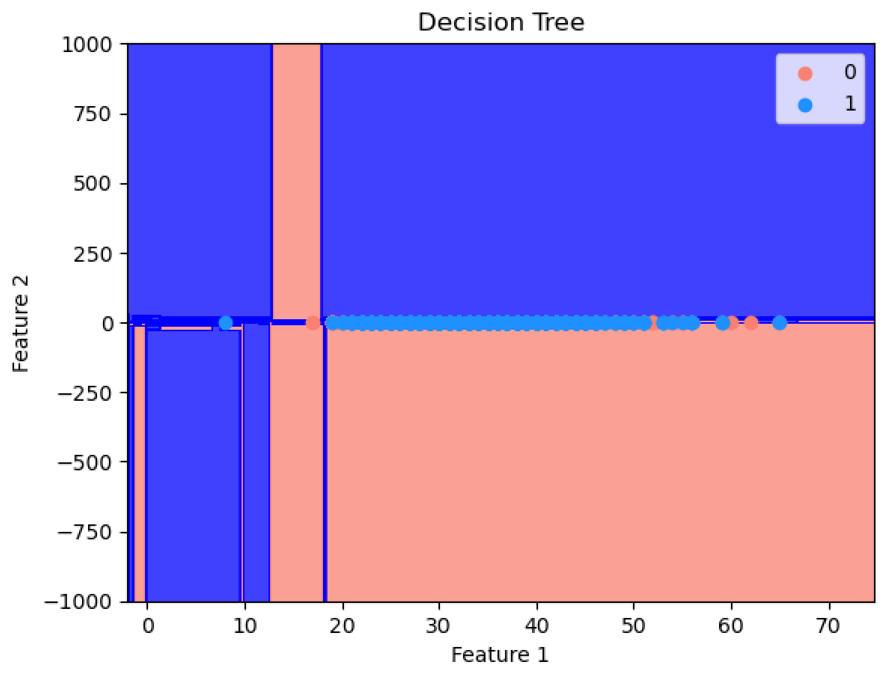
A recall of 99.3% suggests that the model is pretty good at determining if individuals are actually going to take mental health treatment.

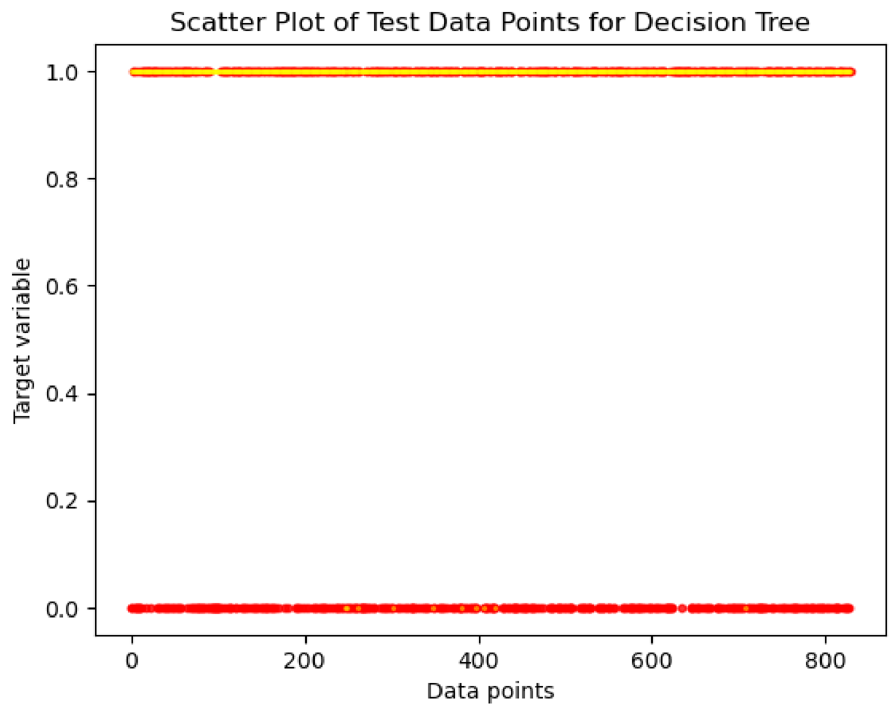
A precision of 57.6% means that when the model predicts that a person will take mental health treatment, it is correct about 57.6% of the time. And the F1 score value of 72.99% suggests a good amount of balance between precision and recall.



As visualized in the above confusion matrix, we can see that the number of times the model correctly predicted the negative classes and positive classes are 473+7 times. The incorrect classifications i.e, the false positives and false negatives are 3+347 .




Decision Tree
Feature 1
Feature 2
0
1
1000
750
500
250
0
−250
−500
−750
−1000
0
10
20
30
40
50
60
70


Scatter Plot of Test Data Points for Decision Tree
Target variable
Data points
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800

As you can see from these pictures, here the blue points in the orange boundary are the miscalculated points by our model and vice versa. Similarly from the scatter plot you can observe the same thing between red and yellow colours. Hence the demonstration of the test data.

Problems with Decision tree:

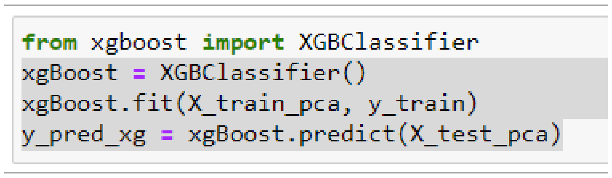
To improve model's performance , we can use methods such as tree trimming, combining multiple trees, and adjusting for class distribution.

<u>XGBoost:</u>

I picked XGBoost for my binary classification task because it's good at it and has some cool features. First off, it's super-fast. Unlike other methods, it can do multiple things at once, thanks to its built-in ability to do parallel computing. This makes it speedy, especially when dealing with loads of data. It's also smart about managing its memory, so it doesn't run out when dealing with big datasets.

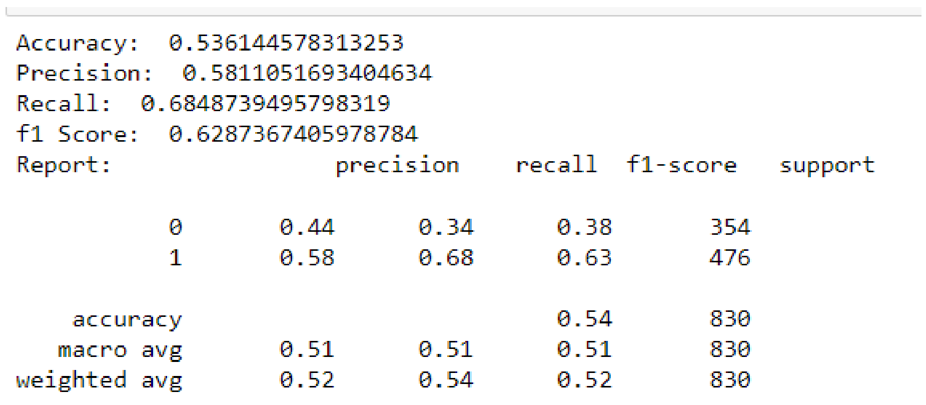
In terms of how well it predicts, XGBoost has some tricks up its sleeve. It's got built-in methods to prevent it from getting too obsessed with the training data, so it doesn't overfit. This helps to keep its predictions accurate on new, unseen data. The algorithm also automatically prunes the trees it builds, meaning it won't make them overly complicated, which is a good thing. This keeps the model in check and robust. One standout feature is that XGBoost can handle missing data without breaking a sweat, which is awesome for real-world datasets where not all information is always available. So, in a nutshell, I went with XGBoost because it's fast, smart about not overthinking the training data, and can handle missing info like a champ – all crucial for my binary classification task.

To fine-tune and train the model, I first implemented the model without any of the hyperparameters or optimization techniques using the following code:



```
from xgboost import XGBClassifier
xgBoost = XGBClassifier()
xgBoost.fit(X_train_pca, y_train)
y_pred_xg = xgBoost.predict(X_test_pca)
```

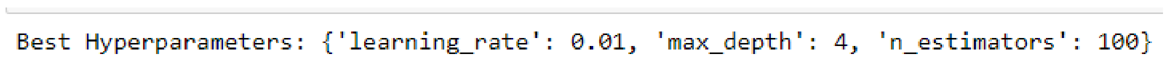
Then for this I found accuracy, precision, recall and f2 score metrics to evaluate my initial model and got the following result:



```
Accuracy:  0.536144578313253
Precision:  0.5811051693404634
Recall:  0.6848739495798319
f1 Score:  0.6287367405978784
Report:                        precision    recall  f1-score   support

           0       0.44      0.34      0.38       354
           1       0.58      0.68      0.63       476

    accuracy                           0.54       830
   macro avg       0.51      0.51      0.51       830
weighted avg       0.52      0.54      0.52       830
```

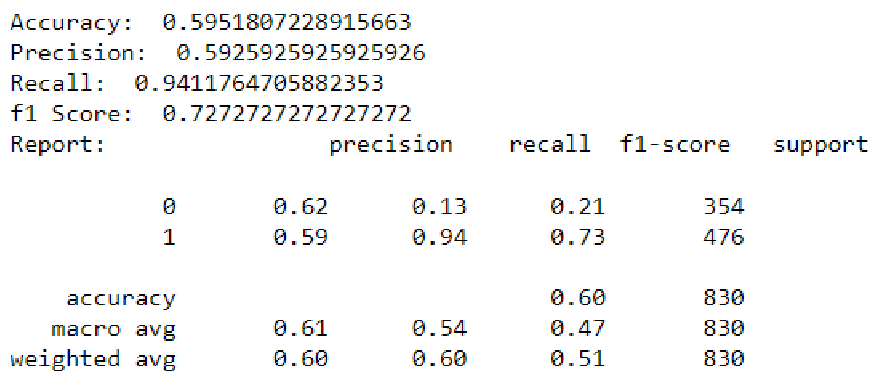
Now, to fine-tune the model and train it more effectively I applied the grid search to the model where the parameters that I am trying to optimize are: learning rate, number of estimators, and maximum depth. Here learning rate indicates the rate at which the model searches for the global minimum, the number of estimators indicates the number of trees to build during the run time of the algorithm and maximum depth indicates the maximum depth of each tree that was constructed in the algorithm.

After applying the grid search I got the following values as the best parameters for my model,

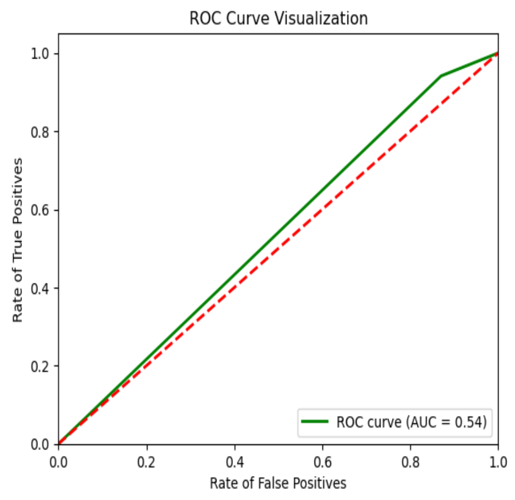


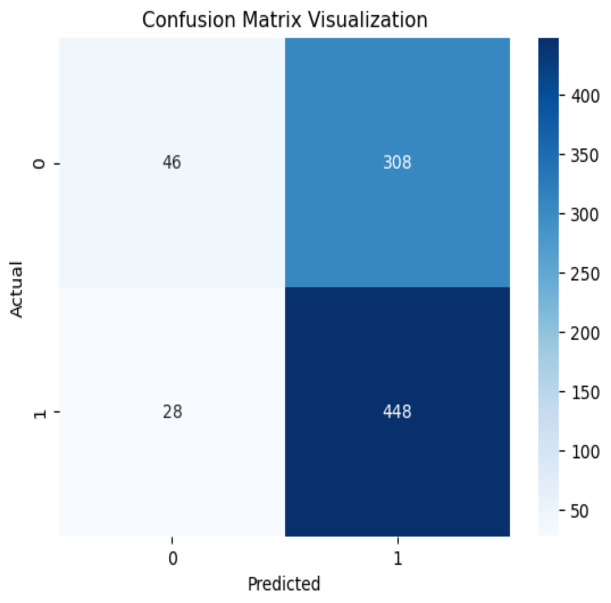
```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 100}
```

Now, keeping these parameters and training the model again, and then testing this model on the test data. I got the following result on the test data,



```
Accuracy:  0.5951807228915663
Precision:  0.5925925925925926
Recall:  0.9411764705882353
f1 Score:  0.7272727272727272
Report:                        precision    recall  f1-score   support

           0       0.62      0.13      0.21       354
           1       0.59      0.94      0.73       476

    accuracy                           0.60       830
   macro avg       0.61      0.54      0.47       830
weighted avg       0.60      0.60      0.51       830
```





As you can see, applying the grid search, finding the best parameters, and training it with them increased my accuracy of the model. Now the model is fully trained. As you can see, the accuracy of the model is 59 percent which means the model predicted 59 per of the values correctly. The precision value is 59.2 per which means the ratio of the positive outcomes i.e., the ratio of true positives to the sum of true positives and false negatives. Recall, which is the count of true cases in the model is about 94 per, and finally the f1 score which is the harmonic mean of precision and recall values is 72 per. As you can see, the model predicted the positive outcomes most of the time correctly, but sometimes it did not predict as accurately as it can. Apart from this, when you see the confusion matrix you can say that, the true predictions are a little more than the false predictions of the model.

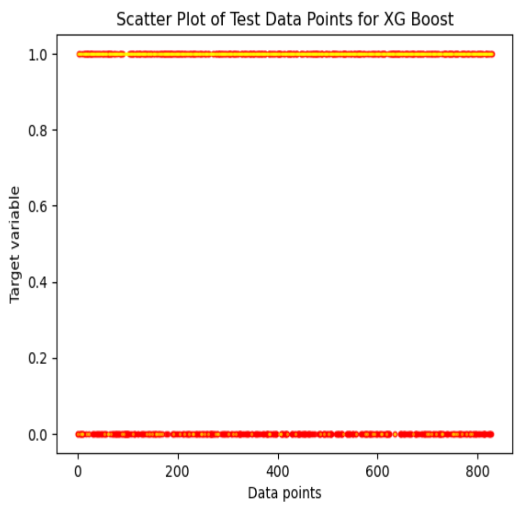
the accuracy is 59%, which means that the model correctly predicts whether a person will take mental health treatment or not about 59% of the time.

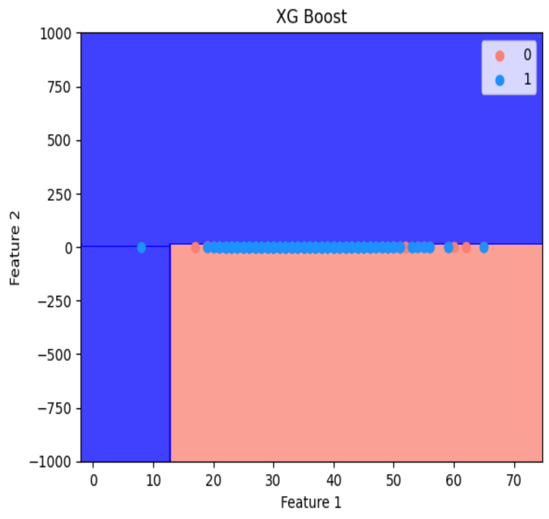
A recall of 94.1% suggests that the model is good at capturing individuals who are actually going to take mental health treatment. It indicates a low rate of false negatives, meaning the model is effective at minimizing instances where it fails to predict individuals who actually take treatment.

A precision of 59.2% means that when the model predicts that a person will take mental health treatment, it is correct about 59.2% of the time. A value of 72.72% suggests a good amount of balance between precision and recall.

To sum up, the model's recall is high, suggesting that it is successful in identifying those who truly seek mental health therapy. Nonetheless, the reduced precision indicates that the accuracy of positive forecasts might be enhanced. An aggregate metric that takes recall and accuracy into account is the F1 score.

Also attaching the decision boundary and scatter plot of the model to check how many points are misrepresented by the model: As you can see from the graphs, the blue point in the orange area are miscalculated by our model and vice versa.





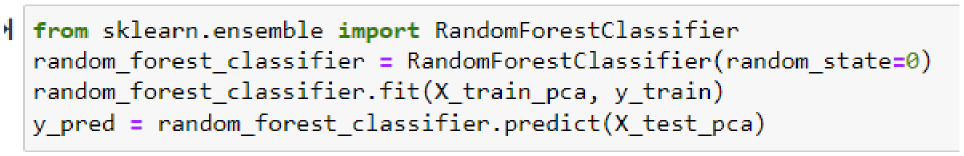
Problems with the model:

XGBoost model might not be doing well due to issues like not having enough varied data, incorrect settings, or struggles in picking the right features for prediction. Problems with imbalanced data, fitting too closely or not enough, and the complexity of predicting mental health treatment based on company priorities could also play a role. Regularly trying different approaches, cleaning your data well, and considering other ways to build the model might help improve its performance.

Random forest:

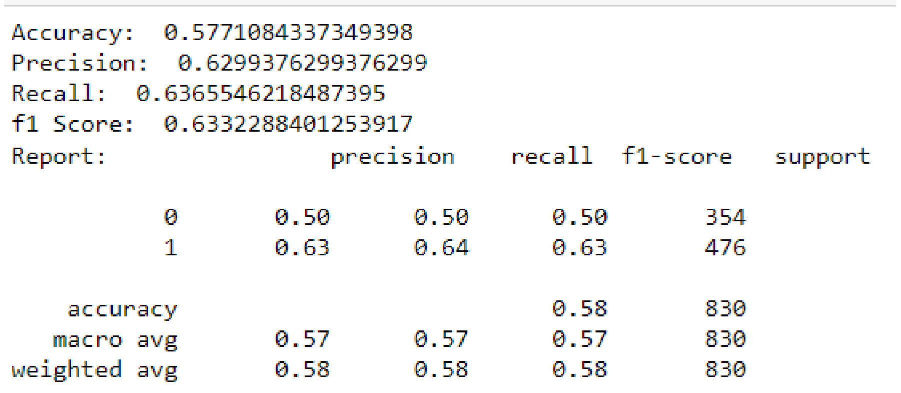
We chose the random forest algorithm as this is also one of the best models to give accurate predictions on the classification dataset. This is because random forest is a type of ensemble learning algorithm, where it combines the data from all the trees and gives the best result from it. This increases the robustness of the algorithm and reduces the overfitting of the algorithm. Apart from this, this model takes the important features and assigns weights to them proportionally. This is very useful while predicting the model because this model does the best feature selection which is useful to increase the performance of the model. Besides all these random forests also have built-in cross-validation techniques that cross-validate the results and weights and prevent the model from overfitting. As my model needs all these advantages to be incorporated to predict the target value, I chose this model.

To fine-tune or train the model, I first initialized the algorithm and just ran it without any optimization techniques as shown below:



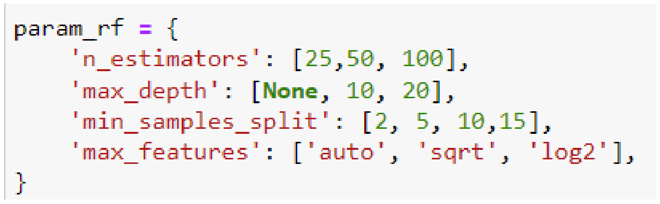
```python
from sklearn.ensemble import RandomForestClassifier
random_forest_classifier = RandomForestClassifier(random_state=0)
random_forest_classifier.fit(X_train_pca, y_train)
y_pred = random_forest_classifier.predict(X_test_pca)
```

The performance metrics that I got from running this algorithm on test data is as follows:



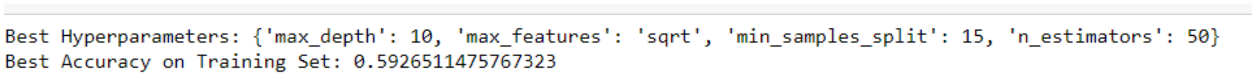
```
Accuracy:  0.5771084337349398
Precision:  0.6299376299376299
Recall:  0.6365546218487395
f1 Score:  0.6332288401253917
Report:                precision    recall  f1-score   support

           0       0.50      0.50      0.50       354
           1       0.63      0.64      0.63       476

    accuracy                           0.58       830
   macro avg       0.57      0.57      0.57       830
weighted avg       0.58      0.58      0.58       830
```

Now, to fine-tune the model and train the model well, I performed a grid search on this algorithm. The parameters that I took to optimize my algorithm are as follows:



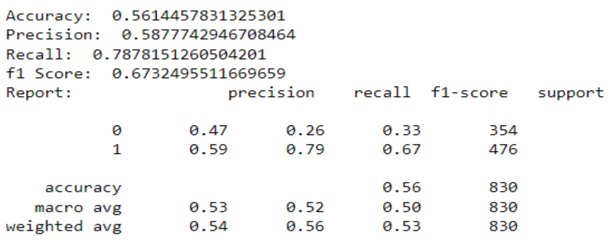
```
param_rf = {
    'n_estimators': [25,50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10,15],
    'max_features': ['auto', 'sqrt', 'log2'],
}
```

Here n_estimators indicated number of trees in the random forest algorithm, max_depth indicates the maximum depth of each of these trees, min_sample_split indicates the minimum number of splits required to split the node, max_features indicates the maximum number of features considered for splitting the node. When I run this param grid on the basic model and train the dataset on it, I got the following result:



```
Best Hyperparameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_split': 15, 'n_estimators': 50}
Best Accuracy on Training Set: 0.5926511475767323
```

As you can see, these are the best parameters for our model, hence now I declared a model with these parameters and trained the dataset on it. Once the training was done, I tested the test data on this trained model and got the following metrics as the results:

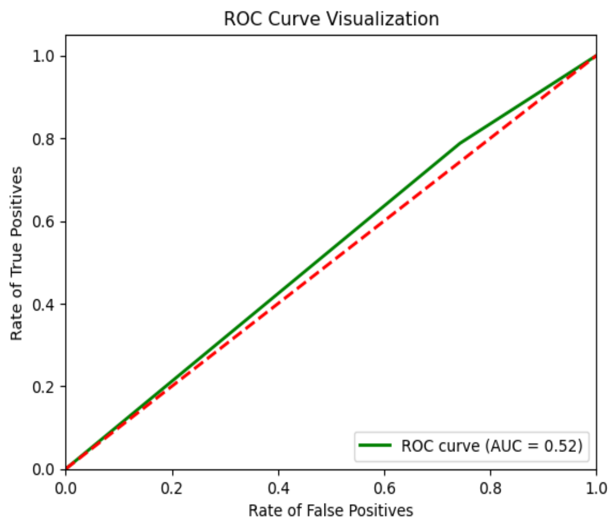


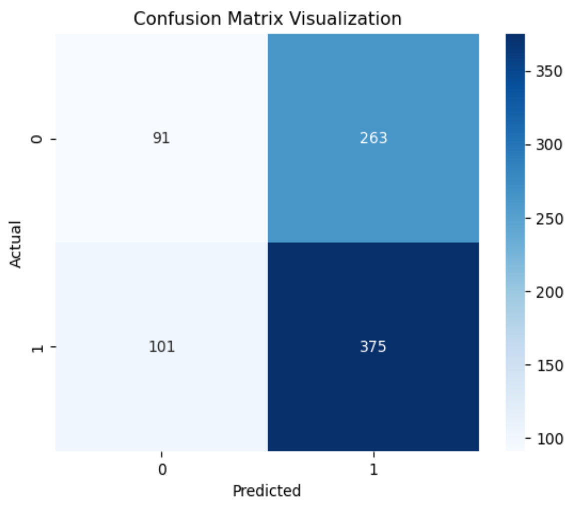
```
Accuracy:  0.5614457831325301
Precision:  0.5877742946708464
Recall:  0.7878151260504201
f1 Score:  0.6732495511669659
Report:                precision    recall  f1-score   support

           0       0.47      0.26      0.33       354
           1       0.59      0.79      0.67       476

    accuracy                           0.56       830
   macro avg       0.53      0.52      0.50       830
weighted avg       0.54      0.56      0.53       830
```

As you can see here, the accuracy of the model is reduced, this is because the model is overfitted to our data, hence it is not able to predict the test data as well as the basic model. As this model did not predict the best results, now the basic model has become my primary model in this algorithm, hence the graphs and analysis for that model are provided.

Analysis:

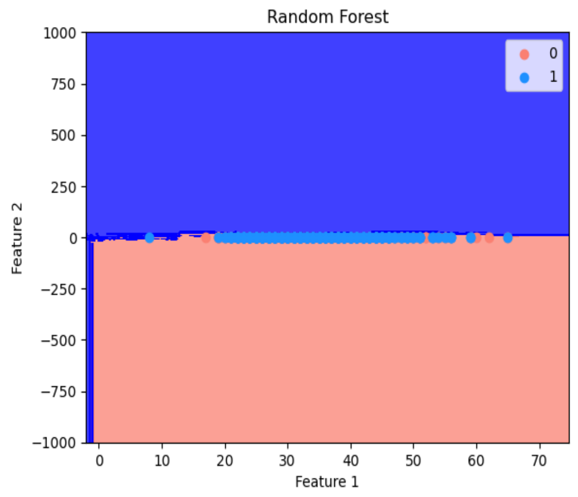
As you can see from the values of the first model, the accuracy that we got is around 57.7 percent, which means the model predicted 58 per of the data correctly. Now, when it comes to a precision value which is 63 per, this indicates that the number of true values i.e., the ratio of true positives to the sum of true positives and false positives is high, which means our model predicted the 63 per of positive cases correctly. The recall value which is the ratio of true positives to the sum of true positives and false negatives indicates how well our model predicted the positive cases is also 63 per. Finally, when it comes to the f1 score the harmonic mean of precision and recall value is 67.3 per. This indicates that there is a good balance between the recall and precision values. Apart from this the values of confusion matrix and roc curve as follows:

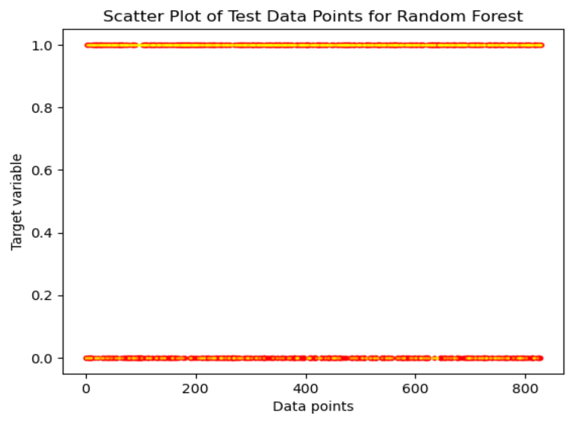




As you can see here, there are 375+91 true to true predictions, 263+101 are predicted incorrectly. Even from this confusion matrix, we can say that the model predicted the data equally, means half of then as false and half of them as true predictions.

Now the decision boundary and scatter plot for the data is as follows:





As you can see from these pictures: the blue points in the orange region are the wrongly predicted labels and vice versa.

According to the problem statement:

An accuracy of 57.7% means that the Random Forest model correctly predicts whether a person will take mental health treatment or not about 57.7% of the time. It gives an overall measure of correctness. With a recall of 63.6%, the model is effective at capturing individuals who are actually going to take mental health treatment. It indicates a relatively low rate of false negatives. A precision of 62.9% means that when the model predicts that a person will take mental health treatment, it is correct about 62.9% of the time. It provides insights into the accuracy of the positive predictions made by the model. A value of 63.3% suggests a reasonable balance between capturing true positives and minimizing false positives. It is a good overall measure, especially when there is an imbalance between classes.

In conclusion, the Random Forest model performs moderately when assessed using the provided criteria. The respectable recall suggests that it is comparatively successful in identifying those who are likely to need mental health care. It can yet be done better, particularly in terms of accuracy and precision in general. To improve the model's performance, more feature engineering or model modifications may be explored.

Why did the model not perform well?

Numerous variables may have contributed to the Random Forest model's subpar performance. The model may be unable to understand complicated interactions if there is insufficient or non-representative data. The model may not be able to comprehend the situation due to inadequate feature representation or a lack of feature engineering. The limits that have been found may also be caused by inadequate hyperparameter settings, an unequal distribution of classes, and problems with data quality. Adding to the difficulties is the potential of over- or underfitting, as well as the inherent difficulty of anticipating mental health therapy based on organizational goals. Addressing these problems and improving model performance requires a thorough examination that includes data exploration, feature analysis, and hyperparameter adjustment.

Voting Classifier (with PCA):

This is also a kind of ensemble learning where the classifier predicts the data from all the input models. The main advantage of this model is that this model combines all the best models for example xgboost, logistic regression, etc, and predicts the outcome. This takes advantage of each of these models combining all the best features and giving the output. As we are combining many models, the diversity of the model is increased, hence reducing the overfitting, and increasing the robustness of the model. As we know XG boost gives us the best feature selection, random forest iterates through all the trees, now this voting classifier considers all these, runs each of the algorithms, grabs the result of each algorithm, and hence performs the voting. The label of the data is interpreted on this voting of this data. Hence, to provide all these advantages to predict the accurate features in my dataset, I chose this model.

As we know there is only one hyperparameter to tune the train data, which is the voting type, I checked the accuracy for both kinds of voting. The inputs for this classifier are the best accuracy models or best-tuned models individually. Now this is how I implemented the hard voting on the dataset:

```
from sklearn.ensemble import VotingClassifier
Voting_classifier = VotingClassifier(estimators=[('lr',best_lr_model),
                                                ('dt', best_decisionTree),
                                                ('xgb',best_xgBoost),
                                                ('rf',random_forest_classifier),
                                                ('svm',best_svm_model)],voting='hard')


Voting_classifier.fit(X_train_pca,y_train)
y_pred_vc = Voting_classifier.predict(X_test_pca)
```

As you can see here, I took all 5 models as input and kept the voting type to hard. Once I declared the model, I trained this model on the training data and then predicted the test data. The performance metrics for this model are as follows:

```
Accuracy:  0.6
Precision:  0.5970350404312669
Recall:  0.930672268907563
f1 Score:  0.7274220032840721
Report:                precision    recall  f1-score   support

           0       0.62      0.16      0.25       354
           1       0.60      0.93      0.73       476

    accuracy                           0.60       830
   macro avg       0.61      0.54      0.49       830
weighted avg       0.61      0.60      0.52       830
```

As you can see here, the accuracy of the model is 60 per, precision is 59.7 per, recall is 93 per and f1 score is 72.7 per. Now performing the same classifier while voting is soft:

```
Voting_classifier_1 = VotingClassifier(estimators=[('lr',best_lr_model),
                                                ('dt', best_decisionTree),
                                                ('xgb',best_xgBoost),
                                                ('rf',random_forest_classifier),
                                                ('svm',best_svm_model)],voting='soft')

Voting_classifier_1.fit(X_train_pca,y_train)
y_pred_vc_1 = Voting_classifier_1.predict_proba(X_test_pca)
y_pred_vc_1= np.argmax(y_pred_vc_1, axis=1)
```

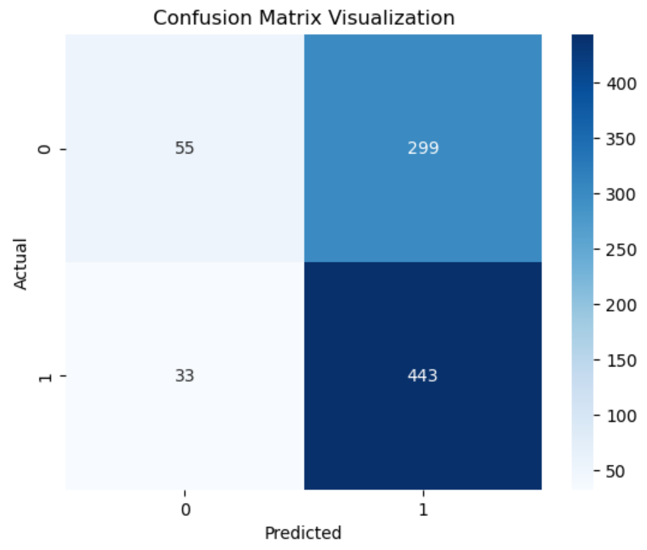
Now, the performance metrics for this method is as follows:

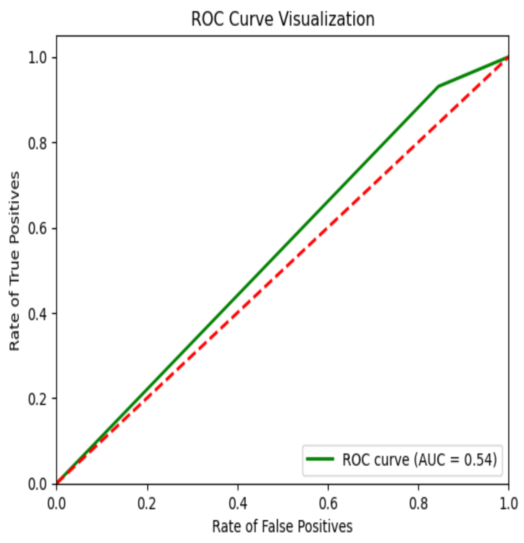
```
Accuracy:  0.5939759036144578
Precision:  0.5943012211668928
Recall:  0.9201680672268907
f1 Score:  0.7221764220939817
Report:                precision    recall  f1-score   support

           0       0.59      0.16      0.25       354
           1       0.59      0.92      0.72       476

    accuracy                           0.59       830
   macro avg       0.59      0.54      0.48       830
weighted avg       0.59      0.59      0.52       830
```

As you can see, the accuracy is 59.3 percent, precision is 59.4 percent, recall is 92.01 percent and finally, f1 score is 72.2 percent. As we can see, the performance metrics of the model are better when the voting hyperparameter is hard. Hence, considering the 1st model as the best model in voting classifier and provided the respective analysis as shown:

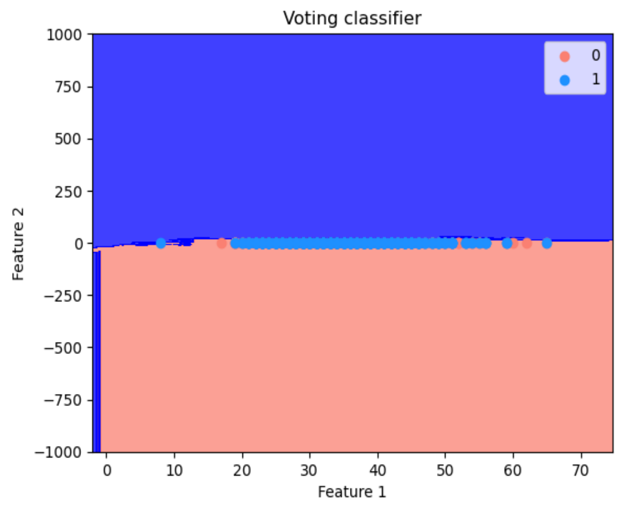
The accuracy obtained from the first model's values is around 60 percent, meaning that the model accurately predicted 60% of the data. Regarding precision, a score of 60 per shows that there all many true values predicted correctly, or that the ratio of true positives to the total of true positives and false positives is large. Now, when coming to the recall value, which means that the ratio of true positives to true positives and false negatives is 93 percent which means that our model predicted the true positives more accurately i.e., true values. Finally, when it comes to the f1 score the harmonic mean of precision and recall value is 72.7 per. This indicates that there is a good balance between the recall and precision values. Apart from this the values of confusion matrix and roc curve as follows:

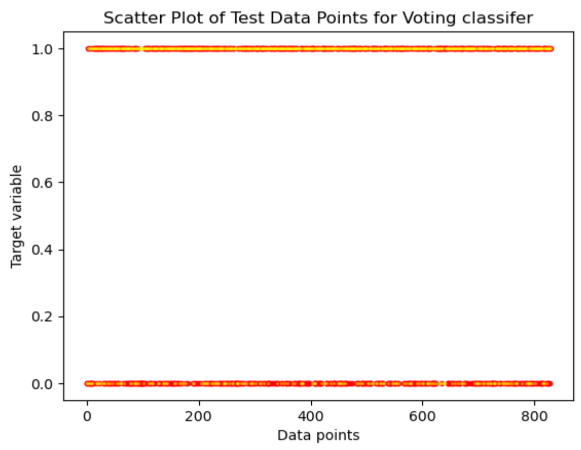
These following confusion matrix and roc curve indicates that how well our model has fitted the dataset. When you see the confusion matrix you can say that the model predicted 55 + 443 entries correctly, 299+33 entries incorrectly. This shows that the data predicted around 60 percent of data correctly. Similarly the ROC curve also shows how much the model fitted the train dataset and how well it is performing.





Apart from this we can also visualize the scatter plot and decision boundary for this algorithm on the test data:





As you can see here, the blue points in orange class indicates that the given model incorrectly predicted those particular samples and vice versa.

According to problem statement:

The accuracy of 60% indicates that the Voting Classifier correctly predicts whether a person will take mental health treatment or not about 60% of the time. It's a measure of overall correctness. With a recall of 93%, the Voting Classifier is highly effective at capturing individuals who are genuinely going to take mental health treatment. This suggests a low rate of false negatives. A precision of 59.7% means that when the Voting Classifier predicts that a person will take mental health treatment, it is correct about 59.7% of the time. It provides insights into the accuracy of positive predictions made by the ensemble. The F1 score, being the harmonic mean of precision and recall, is 72.74%. This suggests a reasonable balance between capturing true positives and minimizing false positives. It's a good overall measure.

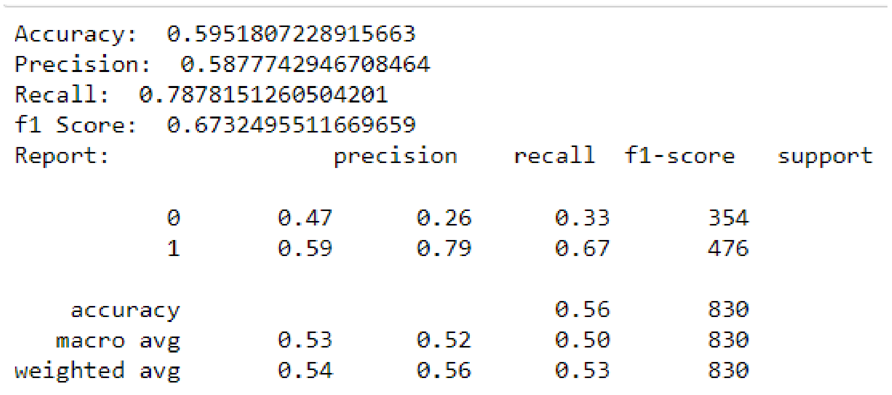
Conclusively, our Voting Classifier exhibits encouraging outcomes, particularly concerning recall, suggesting its efficacy in discerning persons who are inclined to receive mental health therapy. Even if the overall accuracy and precision have increased when compared to individual models, performance may still be enhanced by additional fine-tuning and hyperparameter adjustment. Enhanced prediction skills can result from routinely assessing and refining the ensemble setup.

Problems with this model:

Several factors, including imbalanced classes, suboptimal hyperparameter tuning, possible problems with data quality and feature relevance, and a lack of model diversity, could limit the accuracy of your voting classifier. To increase the ensemble's overall accuracy, it is crucial to ensure a range of base models, adjust their hyperparameters, carefully evaluate and improve the quality of the data, and deal with class imbalances.

Voting classifier (without PCA):

I also implemented the voting classifier on the given dataset without performing any PCA. This means that I want to observe how the original dataset works on this model. When I apply the model, train it and finally test it on the dataset, I got the following results:



```
Accuracy:  0.5951807228915663
Precision:  0.5877742946708464
Recall:  0.7878151260504201
f1 Score:  0.6732495511669659
Report:               precision    recall  f1-score   support

           0       0.47      0.26      0.33       354
           1       0.59      0.79      0.67       476

    accuracy                           0.56       830
   macro avg       0.53      0.52      0.50       830
weighted avg       0.54      0.56      0.53       830
```

This shows that, the accuracy of the voting classifier when applied without pca is about 59.5 percent, precision is 58.7 percent, recall is 78.78 percent and finally the f1 score is 67.32 percent.

The accuracy represents the proportion of correctly classified instances among the total instances. In this case, the Voting Classifier without PCA correctly predicts whether a person will take mental health treatment or not about 59.5% of the time.

Precision is the percentage of predicted positive instances that are truly positive. A precision of 58.7% means that when the Voting Classifier predicts that a person will take mental health treatment, it is correct about 58.7% of the time.

Recall, also known as sensitivity, is the percentage of actual positive instances that the model correctly identifies. With a recall of 78.78%, the Voting Classifier without PCA is effective at capturing individuals who are genuinely going to take mental health treatment.

The F1 score, being the harmonic mean of precision and recall, is 67.32%. This balanced metric indicates the trade-off between precision and recall and suggests a reasonable compromise between capturing true positives and minimizing false positives.

With balanced precision, recall, and F1 score, the Voting Classifier without PCA exhibits a moderate level of performance. The classifier's capacity to accurately categorize instances and successfully identify people in need of mental health treatment is demonstrated by these metrics. To improve the performance of the model, additional analysis and possible modifications to the data or model may be taken into consideration.

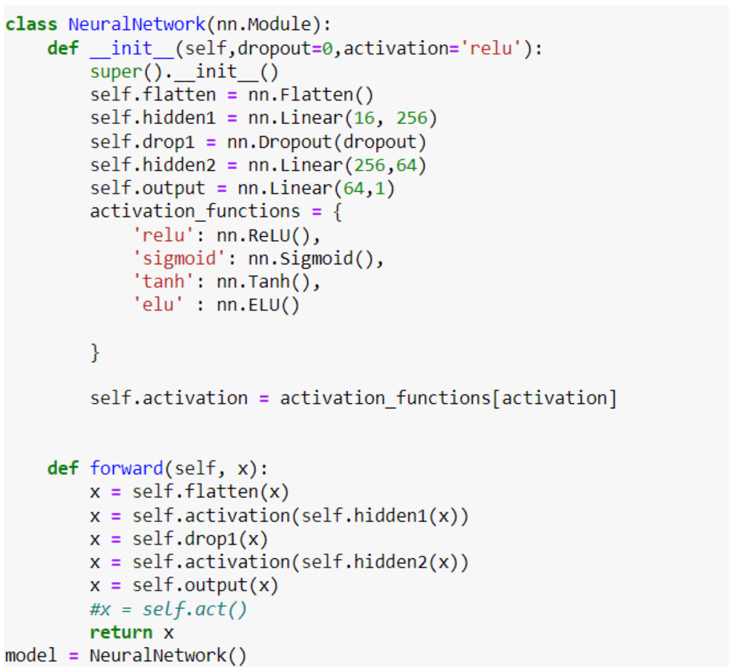
Why did this model did not work on our dataset?

Due to potential issues with tuning or a lack of diversity among its individual models, the Voting Classifier without PCA may not have performed better on the dataset. Furthermore, the lack of PCA may make it harder for it to comprehend the data. You may want to experiment with different model combinations, adjust the models' parameters, and see if adding PCA increases the predictive accuracy overall in order to improve the model's performance.

Neural Network

We chose Neural networks for our binary classification because of it's powerful and robust features. Neural networks has layered structure of neurons which means it can capture complex patterns. They possess the ability to independently extract relevant features. Neural networks are adaptable to different types of data inputs and they demonstrate resilience when dealing with noisy data, ensuring precise outcomes despite data imperfections. As the amount of training data grows, neural networks tend to increase in performance, and they are equipped with a broad spectrum of optimization methods for navigating and refining their complex parameter landscapes. These networks are adept at generalizing their learning to accurately predict on unseen data, and when deployed in ensembles, they show improved classification results and reduced overfitting risks. Additionally, neural networks are designed for online learning, allowing them to adapt continuously as new information streams in, a feature particularly useful in dynamic data environments. The success of neural networks in binary classification relies on having a large and well-represented dataset, a suitably chosen network architecture, and meticulous training to prevent overfitting or underfitting.

Below is the implementation of the Neural network model:

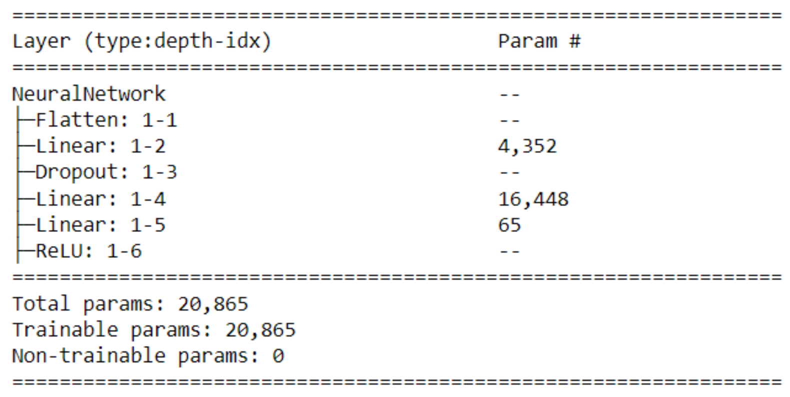


```python
class NeuralNetwork(nn.Module):
    def __init__(self,dropout=0,activation='relu'):
        super().__init__()
        self.flatten = nn.Flatten()
        self.hidden1 = nn.Linear(16, 256)
        self.drop1 = nn.Dropout(dropout)
        self.hidden2 = nn.Linear(256,64)
        self.output = nn.Linear(64,1)
        activation_functions = {
            'relu': nn.ReLU(),
            'sigmoid': nn.Sigmoid(),
            'tanh': nn.Tanh(),
            'elu' : nn.ELU()

        }

        self.activation = activation_functions[activation]


    def forward(self, x):
        x = self.flatten(x)
        x = self.activation(self.hidden1(x))
        x = self.drop1(x)
        x = self.activation(self.hidden2(x))
        x = self.output(x)
        #x = self.act()
        return x
model = NeuralNetwork()
```

Our model includes two liner hidden layers with dropout and activation functions.

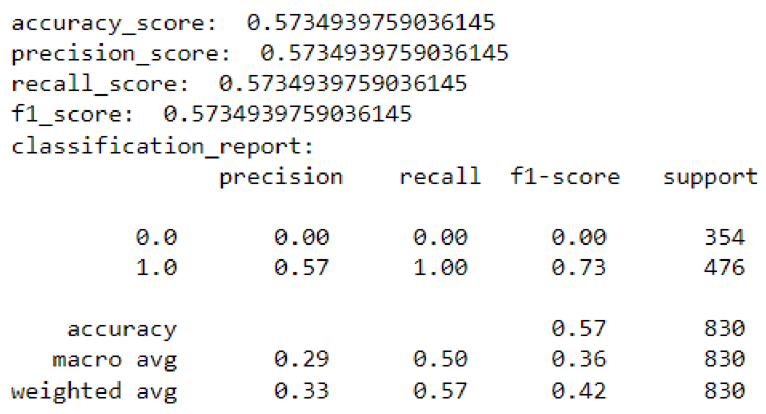
It features an initial flattening layer to transform input data into a 1D tensor, followed by two hidden layers with 256 and 64 neurons respectively. The model employs dropout for regularization. It supports different activation functions—ReLU, Sigmoid, Tanh, and ELU. The final output layer reduces the dimension to a single neuron, which determines the binary output.

Below is the demonstrated summary of our defined Neural Network model:



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
NeuralNetwork                            --
├─Flatten: 1-1                           --
├─Linear: 1-2                            4,352
├─Dropout: 1-3                           --
├─Linear: 1-4                            16,448
├─Linear: 1-5                            65
├─ReLU: 1-6                              --
=================================================================
Total params: 20,865
Trainable params: 20,865
Non-trainable params: 0
=================================================================
```

When we apply this model on the data, we get the following results on the test data. In this model we used, reLU activation function in all the layers except the last one. As this problem is a binary classification problem, we used sigmoid activation function in the last layer. Hence, we get the probabilities and test results for both classes. The loss function we used is binary loss function and the optimization technique used is SGD.



```
accuracy_score:  0.5734939759036145
precision_score:  0.5734939759036145
recall_score:  0.5734939759036145
f1_score:  0.5734939759036145
classification_report:
               precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       354
         1.0       0.57      1.00      0.73       476

    accuracy                           0.57       830
   macro avg       0.29      0.50      0.36       830
weighted avg       0.33      0.57      0.42       830
```

As you can see here, the accuracy, precision and recall values are 57.3 per. You can also observe that the f1 score is also 57.3 percent.

According to the problem statement:

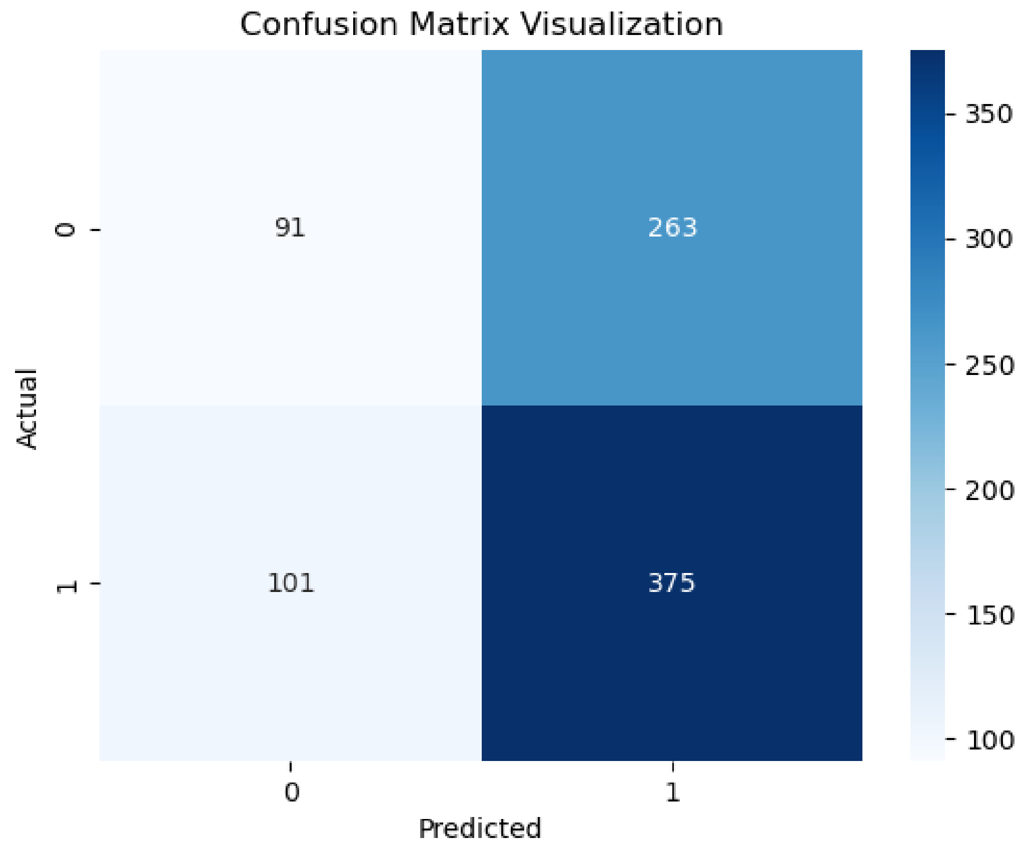
The accuracy represents the proportion of correctly classified instances among the total instances. In this case, the model correctly predicts whether a person will take mental health treatment or not about 57.3% of the time.

Precision is the percentage of predicted positive instances that are truly positive. A precision of 57.3% means that when the neural network predicts that a person will take mental health treatment, it is correct about 57.3% of the time.

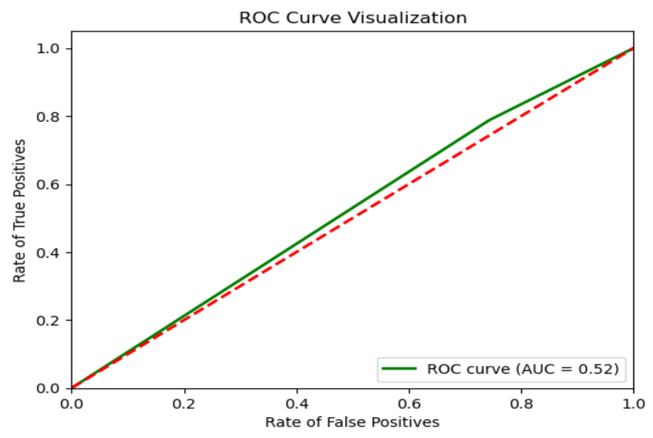
Recall of 57.3%, the model is effective at capturing individuals who are genuinely going to take mental health treatment.

The F1 score, being the harmonic mean of precision and recall, is also 57.3%. This balanced metric indicates the trade-off between precision and recall and suggests a reasonable compromise between capturing true positives and minimizing false positives.

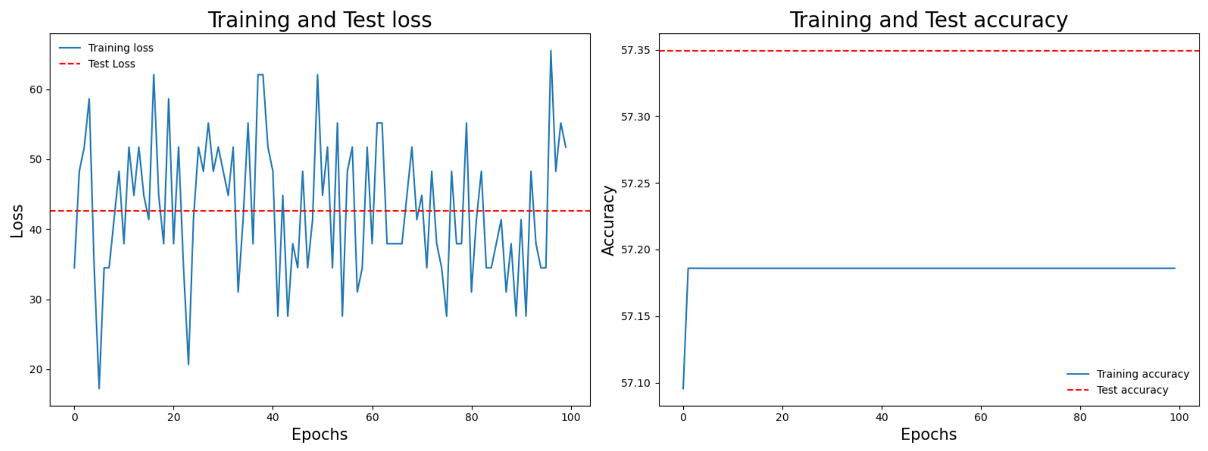
In conclusion, the neural network model exhibits steady performance with balanced F1 score, recall, and precision. To improve overall model performance, more research and possible modifications—like tweaking model architecture or hyperparameter tuning—might be investigated.



As visualized in the above confusion matrix, we can see that the number of times the model correctly predicted the negative classes and positive classes are 91+375 times. The incorrect classifications i.e, the false positives and false negatives are101+263. So we can see that our model is better at predicting if an individual needs treatment compared to predictions where individual doesn't need mental health diagnosis. The counts of false positives and false negatives indicate that we can further improve the model for better performance.



This ROC curve indicates the balance between the rate of true positive rate to the rate of false positive rates. As the line is nearer to the middle and converges at 1, we can say that the model is not too good on the data but can be considered and is expected to produce good results only.



When you see these graphs, here the training loss fluctuates over the 100 epochs. This means that the model is slowly learning on our data and trying to predict the accurate results as much as possible. But when you see the test loss this is almost the average of all the train losses which means that the model learned the dataset equally well and trying to predict the data. Similarly, when it comes to the train accuracy, as you can see, initially the model accuracy is zero and even over the epochs the accuracy didn’t change much. This indicates that the model is trying to learn the data as much as possible but the accuracy isn’t improving much. Hence, the model is learning but it did not show much effect on the overall result. Now, when it comes to the test accuracy it is slightly more the train accuracy and it predicted 57 percent of data correctly.

There are some issues with the neural network that we employed for our dataset. It struggles to significantly increase its accuracy even though it is gradually learning from the data during training (as indicated by fluctuating training loss). Although the model appears to be making an attempt to comprehend the data, its influence on the outcome is not particularly evident. It appears to be struggling with accurately capturing patterns, as evidenced by the gradual increase in training accuracy. At 57%, the test accuracy is marginally higher than the training, but it is still below training levels. We may want to think about adjusting the neural network's parameters or structure to improve performance.

Summary:

Of all these models, the voting classifier with PCA performed relatively well on my dataset. With an accuracy of 60% and a well-balanced precision of 60% and high recall of 93%, the Voting Classifier with PCA is clearly the best-performing model, earning an F1 score of 72.7%. It's a good tool for predicting whether people will choose to receive mental health treatment based on company priorities because of its ability to minimize false negatives and accurately predict positive cases. By including PCA, a method to lower dimensionality is suggested, which could improve computational efficiency. However, particular objectives and interpretability issues should be taken into account when choosing the final model. Overall, the Voting Classifier—especially when combined with PCA—showcased successful predictions by combining information from various models, which made it a strong option for handling the challenges involved in anticipating patients' treatment decisions in the context of mental health and company priorities.

References to the model:

https://scikit-learn.org/stable/

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

Peer Evaluation Form for Final Group Work

CSE 487/587B

- Please write the names of your group members.

Group member 1 : Sri Lakshmi Sahithi

Group member 2 : Mahitha Balumuri

Group member 3 : Sai Grandhi

- Rate each groupmate on a scale of 5 on the following points, with 5 being HIGHEST and 1 being LOWEST.

Evaluation Criteria	**Group member 1**	**Group member 2**	**Group member 3**
How effectively did your group mate work with you?	5	5	5
Contribution in writing the report	5	5	5
Demonstrates a cooperative and supportive attitude.	5	5	5
Contributes significantly to the success of the project .	5	5	5
TOTAL	20 (average 5)	20 (average 5)	20 (average 5)

Also please state the overall contribution of your teammate in percentage below, with total of all the three members accounting for 100% (33.33+33.33+33.33 ~ 100%) :

Group member 1 : 33.33%

Group member 2 : 33.33%

Group member 3 : 33.33%