In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
class customLibrary:

    #------------Common code starts from here for Part A and B-----------#
    def convert_DataFrame_Proba_by_threshold(self, df, column_name, threshold):

        df_with_y_hat = df.copy()
        df_with_y_hat.loc[df_with_y_hat[column_name] >= threshold, column_name] = 1
        df_with_y_hat.loc[df_with_y_hat[column_name] < threshold, column_name] = 0

        df_with_y_hat = df_with_y_hat.rename(columns={column_name:'y_hat'})

        return df_with_y_hat

    #A.1 and B.1
    def getConfusionMatrixValues(self, df):

        true_positive = len(df[((df.y == 1) & (df.y_hat == 1))])

        false_positive = len(df[((df.y == 0) & (df.y_hat == 1))])

        false_negative = len(df[((df.y == 1) & (df.y_hat == 0))])

        true_negative = len(df[((df.y == 0) & (df.y_hat == 0))])

        return true_positive, false_positive, false_negative, true_negative


    def compute_Precision(self, true_positive, false_positive):

        if (true_positive + false_positive) > 0:
            return (true_positive / (true_positive + false_positive))
        else:
            return 0

    def compute_Recall(self, true_positive, false_negative):

        if (true_positive + false_negative) > 0:
            return (true_positive / (true_positive + false_negative))
        else:
            return 0

    #A.2 and B.2
    def compute_F1_Score(self, precision, recall):

        if (precision + recall) > 0:
            return (2 * precision * recall) / (precision + recall)
        else:
            return 0

    #A.4 and B.4
    def compute_Accuracy_Score(self, true_positive, true_negative, tot_num_points):
        return (true_positive + true_negative) / tot_num_points


    def compute_TruePositiveRate(self, true_positive, false_negative):
        return self.compute_Recall(true_positive, false_negative)


    def compute_FalsePositiveRate(self, false_positive, true_negative):

        if (false_positive + true_negative) > 0:
            return (false_positive / (false_positive + true_negative))
        else:
            return 0


    #A.3 and B.3
    def compute_AUC_Score(self, df):
        unique_proba = sorted(df.proba.unique(), reverse=True)

        tpr_array = []
        fpr_array = []
        for threshold in unique_proba:

            df_with_y_hat = self.convert_DataFrame_Proba_by_threshold(df, 'proba', threshold)

            t_p, f_p, f_n, t_n = self.getConfusionMatrixValues(df_with_y_hat)

            tpr_array.append(self.compute_TruePositiveRate(t_p, f_n))

            fpr_array.append(self.compute_FalsePositiveRate(f_p, t_n))

        return np.trapz(tpr_array, fpr_array)
    #------------Common code ends here for Part A and B-----------#



    #------------Part C-----------#
    def compute_best_threshold(self, df):

        unique_prob = sorted(df.prob.unique(), reverse = True)

        dict_A = {}
        for threshold in unique_prob:

            df_with_y_hat = self.convert_DataFrame_Proba_by_threshold(df, 'prob', threshold)

            t_p, f_p, f_n, t_n = self.getConfusionMatrixValues(df_with_y_hat)

            A = (500 * f_n) + (100 * f_p)

            dict_A[threshold] = A


        best_threshold = min(dict_A, key=dict_A.get)

        return best_threshold, dict_A[best_threshold]


    #------------Part D-----------#
    #D.1
    def compute_Mean_Square_Error(self, df):

        n = len(df)

        summation_value = 0
        for index, row in df.iterrows():
            summation_value += pow(float(row["y"]) - float(row["pred"]), 2)

        return summation_value / n

    #D.2
    def compute_mean_absolute_percentage_error(self, df):

        summation_error = 0
        summation_actual = df.y.sum()

        for index, row in df.iterrows():
            summation_error += abs(row["y"] - row["pred"])

        return summation_error / summation_actual

    #D.3
    def compute_R_Square(self, df):

        mean_y_pred = df.pred.mean()

        SS_Tot = 0
        SS_Res = 0

        for index, row in df.iterrows():
            SS_Tot += pow((row["y"] - mean_y_pred), 2)
            SS_Res += pow((row["y"] - row["pred"]), 2)


        return (1 - (SS_Res/SS_Tot))
```

In [3]:
```
#Initialising the class object
custom_Lib = customLibrary()
```

Part A

In [4]:
```
df = pd.read_csv('5_a.csv')

df_with_y_hat = custom_Lib.convert_DataFrame_Proba_by_threshold(df, 'proba', 0.5)

t_p, f_p, f_n, t_n = custom_Lib.getConfusionMatrixValues(df_with_y_hat)

confusion_matrix = [[t_p, f_p], [f_n, t_n]]

precision = custom_Lib.compute_Precision(t_p, f_p)
recall = custom_Lib.compute_Recall(t_p, f_n)
f1_score = custom_Lib.compute_F1_Score(precision, recall)
acc_score = custom_Lib.compute_Accuracy_Score(t_p, t_n, len(df))
auc_score = custom_Lib.compute_AUC_Score(df)

print('\n-------Confusion Matrix-------')
print(confusion_matrix)

print('\n-------F1 Score-------')
print(f1_score)


print('\n-------Accuracy Score-------')
print(acc_score)

print('\n-------AUC Score-------')
print(auc_score)
```

```
-------Confusion Matrix-------
[[10000, 100], [0, 0]]

-------F1 Score-------
0.995024875621890６

-------Accuracy Score-------
0.9900990099009901

-------AUC Score-------
0.4882990000000004
```

Part B

In [5]:
```
df = pd.read_csv('5_b.csv')

df_with_y_hat = custom_Lib.convert_DataFrame_Proba_by_threshold(df, 'proba', 0.5)

t_p, f_p, f_n, t_n = custom_Lib.getConfusionMatrixValues(df_with_y_hat)

confusion_matrix = [[t_p, f_p], [f_n, t_n]]

precision = custom_Lib.compute_Precision(t_p, f_p)
recall = custom_Lib.compute_Recall(t_p, f_n)
f1_score = custom_Lib.compute_F1_Score(precision, recall)
acc_score = custom_Lib.compute_Accuracy_Score(t_p, t_n, len(df))
auc_score = custom_Lib.compute_AUC_Score(df)

print('\n-------Confusion Matrix-------')
print(confusion_matrix)

print('\n-------F1 Score-------')
print(f1_score)


print('\n-------Accuracy Score-------')
print(acc_score)

print('\n-------AUC Score-------')
print(auc_score)
```

```
-------Confusion Matrix-------
[[59, 239], [45, 9761]]

-------F1 Score-------
0.2791878172588833

-------Accuracy Score-------
0.9718811881188119

-------AUC Score-------
0.9377570000000001
```

Part C

In [6]:
```
df = pd.read_csv('5_c.csv')

best_threshold, lowest_metric = custom_Lib.compute_best_threshold(df)

print('\n-------Lowest Metric-------')
print(lowest_metric)

print('\n-------Best Threshold-------')
print(best_threshold)
```

```
-------Lowest Metric-------
141000

-------Best Threshold-------
0.2300390278970873
```

Part D

In [7]:
```
df = pd.read_csv('5_d.csv')

mse = custom_Lib.compute_Mean_Square_Error(df)

mape = custom_Lib.compute_mean_absolute_percentage_error(df)

r_square = custom_Lib.compute_R_Square(df)

print('\n-------MSE-------')
print(mse)

print('\n-------MAPE-------')
print(mape)

print('\n-------R Square-------')
print(r_square)
```

```
-------MSE-------
177.16569974554707

-------MAPE-------
0.12912029940009687

-------R Square-------
0.9563583447288628
```

In [8]:
```
#Please give the feedback
```