

HealthInsFraud_v2.2

August 13, 2021

1 Background to the Case Study:

- 1.0.1 Healthcare programs, in the United States (U.S.), have experienced tremendous growth in patient populations and commensurate costs and Fraud is a major contributor to these inflating healthcare expenses.
- 1.0.2 Many healthcare providers settle huge amounts for patients. But some insured individuals or the provider of health services attempt to make fake claims by giving false claim details such as showing fake bills, submitting same bills repeatedly etc.

2 Business Objective

- 2.0.1 The primary objective is a binary classification task of each of the provider as either a Fraud or a Non-Fraud. Since the data is related to fraud, the dataset is imbalanced as the number of providers, who commit a fraud is very small proportion when compared to the overall dataset.

3 Evaluation Metric

- 3.0.1 I will be considering the No Fraud cases a Positive class and the Fraud cases as a Negative class. Considering the problem statement and the class imbalance in the dataset, I intend to explore the following parameters:
- 3.0.2 1. Precision and Recall: Since we are dealing with a fraud identification problem with a class imbalance, metrics such as Accuracy do not precisely evaluate the models. Hence, I will be going for an evaluation metric such as Precision and Recall.
- 3.0.3 2. AUC score: As it gives an overall view of the classification performance of each of the models
- 3.0.4 3. F1 score or Macro F1 score (due to class imbalances): In this case both precision and recall are particularly important as we need to cover as many fraud cases as possible.

4 Description of the Dataset

- 4.0.1 The dataset consists of the fraud label data for a total of 5410 providers and the labels for these providers are provided in a separate file titled "Train.csv".
- 4.0.2 The dataset that has been provided to us consists of 3 different csv files that has the Inpatient, Outpatient and the Beneficiary data. Each of the healthcare providers are identified by a unique ID and this ID is a part of the outpatient and the Inpatient datasets which also carry the beneficiary ID.

[]:

4.1 Importing the Requisite Libraries

```
[ ]: import os
import csv
import json
import pickle
import shutil
import random
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from datetime import datetime, timedelta
from sklearn.impute import SimpleImputer
from matplotlib.gridspec import GridSpec
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.calibration import CalibratedClassifierCV
from sklearn.feature_extraction.text import CountVectorizer

[ ]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

4.2 Analysis of the Train Datasets

4.3 Loading the Train Datasets

```
[ ]: train_ben= pd.read_csv('/content/drive/MyDrive/Colab Notebooks/
    ↳Train_Beneficiarydata.csv')
train_inpat= pd.read_csv('/content/drive/MyDrive/Colab Notebooks/
    ↳Train_Inpatientdata.csv')
train_outpat= pd.read_csv('/content/drive/MyDrive/Colab Notebooks/
    ↳Train_Outpatientdata.csv')
train_y= pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Train.csv')
```

```
[ ]: train_ben.head()
```

```
[ ]:      BeneID      DOB  ... OPAnnualReimbursementAmt  OPAnnualDeductibleAmt
0  BENE11001  1943-01-01  ...                60                70
1  BENE11002  1936-09-01  ...                30                50
2  BENE11003  1936-08-01  ...                90                40
3  BENE11004  1922-07-01  ...               1810               760
4  BENE11005  1935-09-01  ...               1790              1200
```

[5 rows x 25 columns]

```
[ ]: train_ben.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138556 entries, 0 to 138555
Data columns (total 25 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   BeneID                                     138556 non-null object
1   DOB                                        138556 non-null object
2   DOD                                        1421 non-null  object
3   Gender                                    138556 non-null int64
4   Race                                      138556 non-null int64
5   RenalDiseaseIndicator                    138556 non-null object
6   State                                    138556 non-null int64
7   County                                    138556 non-null int64
8   NoOfMonths_PartACov                      138556 non-null int64
9   NoOfMonths_PartBCov                      138556 non-null int64
10  ChronicCond_Alzheimer                    138556 non-null int64
11  ChronicCond_Heartfailure                 138556 non-null int64
12  ChronicCond_KidneyDisease                138556 non-null int64
13  ChronicCond_Cancer                      138556 non-null int64
14  ChronicCond_ObstrPulmonary               138556 non-null int64
15  ChronicCond_Depression                   138556 non-null int64
16  ChronicCond_Diabetes                     138556 non-null int64
17  ChronicCond_IschemicHeart                138556 non-null int64
18  ChronicCond_Osteoporosis                 138556 non-null int64
19  ChronicCond_rheumatoidarthritis          138556 non-null int64
20  ChronicCond_stroke                       138556 non-null int64
21  IPAnnualReimbursementAmt                 138556 non-null int64
22  IPAnnualDeductibleAmt                    138556 non-null int64
23  OPAnnualReimbursementAmt                 138556 non-null int64
24  OPAnnualDeductibleAmt                    138556 non-null int64
dtypes: int64(21), object(4)
memory usage: 26.4+ MB
```

```
[ ]: train_inpat.head(3)
```

```
[ ]:      BeneID  ClaimID  ... ClmProcedureCode_5 ClmProcedureCode_6
0  BENE11001  CLM46614  ...                NaN                NaN
1  BENE11001  CLM66048  ...                NaN                NaN
2  BENE11001  CLM68358  ...                NaN                NaN
```

```
[3 rows x 30 columns]
```

```
[ ]: train_inpat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40474 entries, 0 to 40473
Data columns (total 30 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   BeneID                                40474 non-null  object
1   ClaimID                               40474 non-null  object
2   ClaimStartDt                          40474 non-null  object
3   ClaimEndDt                            40474 non-null  object
4   Provider                              40474 non-null  object
5   InscClaimAmtReimbursed                40474 non-null  int64
6   AttendingPhysician                   40362 non-null  object
7   OperatingPhysician                   23830 non-null  object
8   OtherPhysician                       4690 non-null   object
9   AdmissionDt                          40474 non-null  object
10  ClmAdmitDiagnosisCode                 40474 non-null  object
11  DeductibleAmtPaid                    39575 non-null  float64
12  DischargeDt                          40474 non-null  object
13  DiagnosisGroupCode                   40474 non-null  object
14  ClmDiagnosisCode_1                   40474 non-null  object
15  ClmDiagnosisCode_2                   40248 non-null  object
16  ClmDiagnosisCode_3                   39798 non-null  object
17  ClmDiagnosisCode_4                   38940 non-null  object
18  ClmDiagnosisCode_5                   37580 non-null  object
19  ClmDiagnosisCode_6                   35636 non-null  object
20  ClmDiagnosisCode_7                   33216 non-null  object
21  ClmDiagnosisCode_8                   30532 non-null  object
22  ClmDiagnosisCode_9                   26977 non-null  object
23  ClmDiagnosisCode_10                  3927 non-null   object
24  ClmProcedureCode_1                   23148 non-null  float64
25  ClmProcedureCode_2                   5454 non-null   float64
26  ClmProcedureCode_3                   965 non-null    float64
27  ClmProcedureCode_4                   116 non-null    float64
28  ClmProcedureCode_5                   9 non-null      float64
29  ClmProcedureCode_6                   0 non-null      float64
dtypes: float64(7), int64(1), object(22)
memory usage: 9.3+ MB
```

```
[ ]: train_outpat.head(3)
```

```
[ ]:      BeneID      ClaimID  ... DeductibleAmtPaid ClmAdmitDiagnosisCode
0  BENE11002  CLM624349  ...              0              56409
1  BENE11003  CLM189947  ...              0              79380
2  BENE11003  CLM438021  ...              0              NaN
```

```
[3 rows x 27 columns]
```

```
[ ]: train_outpat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517737 entries, 0 to 517736
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   BeneID                                517737 non-null object
1   ClaimID                               517737 non-null object
2   ClaimStartDt                          517737 non-null object
3   ClaimEndDt                            517737 non-null object
4   Provider                              517737 non-null object
5   InscClaimAmtReimbursed                517737 non-null int64
6   AttendingPhysician                   516341 non-null object
7   OperatingPhysician                   90617 non-null  object
8   OtherPhysician                        195046 non-null object
9   ClmDiagnosisCode_1                   507284 non-null object
10  ClmDiagnosisCode_2                   322357 non-null object
11  ClmDiagnosisCode_3                   203257 non-null object
12  ClmDiagnosisCode_4                   125596 non-null object
13  ClmDiagnosisCode_5                   74344 non-null  object
14  ClmDiagnosisCode_6                   48756 non-null  object
15  ClmDiagnosisCode_7                   32961 non-null  object
16  ClmDiagnosisCode_8                   22912 non-null  object
17  ClmDiagnosisCode_9                   14838 non-null  object
18  ClmDiagnosisCode_10                  1083 non-null   object
19  ClmProcedureCode_1                   162 non-null    float64
20  ClmProcedureCode_2                   36 non-null     float64
21  ClmProcedureCode_3                   4 non-null      float64
22  ClmProcedureCode_4                   2 non-null      float64
23  ClmProcedureCode_5                   0 non-null      float64
24  ClmProcedureCode_6                   0 non-null      float64
25  DeductibleAmtPaid                    517737 non-null int64
26  ClmAdmitDiagnosisCode                105425 non-null object
dtypes: float64(6), int64(2), object(19)
memory usage: 106.7+ MB
```

```
[ ]: train_y.head()
```

```
[ ]:      Provider PotentialFraud
0  PRV51001                  No
```

1	PRV51003	Yes
2	PRV51004	No
3	PRV51005	Yes
4	PRV51007	No

```
[ ]: print("Shape of Train Beneficiary file Data:", train_ben.shape)
print("Shape of Train In-patient file Data:", train_inpat.shape)
print("Shape of Train Out-patient file Data:", train_outpat.shape)
print("Shape of Train file Data:", train_y.shape)
```

```
Shape of Train Beneficiary file Data: (138556, 25)
Shape of Train In-patient file Data: (40474, 30)
Shape of Train Out-patient file Data: (517737, 27)
Shape of Train file Data: (5410, 2)
```

```
[ ]: out_prov= np.unique(train_outpat['Provider'])
print("The number of Unique Providers in the Train_Outpat file:", len(out_prov))

in_prov= np.unique(train_inpat['Provider'])
print("The number of Unique Providers in the Train_Inpat file", len(in_prov))

com_prov= set(out_prov).intersection(set(in_prov))
print("The number of Providers common to both the Inpat and Outpat files:
→", len(com_prov))

uni_ele= len(out_prov)+len(in_prov)-len(com_prov)
print("Total Number of Unique Providers in Outpatient and Inpatient datasets_
→Together:", uni_ele)
```

```
The number of Unique Providers in the Train_Outpat file: 5012
The number of Unique Providers in the Train_Inpat file 2092
The number of Providers common to both the Inpat and Outpat files: 1694
Total Number of Unique Providers in Outpatient and Inpatient datasets Together:
5410
```

4.4 Observations on the Train Datasets

1. We observed that the labels of Potential Fraud as "Yes" or "No" have been provided to the each of the Providers in the dataset.
2. The number of Unique Providers in the Dataset is 5410 as can be seen in the "Train" file.
3. Hence, checking for the unique Providers in the Inpatient and the Outpatient files.
4. From the above, we observed that that the total number of Providers are spread across the Inpatient and Outpatient Files.

4.4.1 Looking at the different columns present in each of the datasets

```
[ ]: print("The columns in the Outpatient Dataset are:",train_outpat.columns)
      print("="*100)
      print("The columns in the Inpatient Dataset are:",train_inpat.columns)
      print("="*100)
      print("The columns in the Beneficiary Dataset are:",train_ben.columns)
```

```
The columns in the Outpatient Dataset are: Index(['BeneID', 'ClaimID',
'ClaimStartDt', 'ClaimEndDt', 'Provider',
'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
'OtherPhysician', 'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2',
'ClmDiagnosisCode_3', 'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5',
'ClmDiagnosisCode_6', 'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8',
'ClmDiagnosisCode_9', 'ClmDiagnosisCode_10', 'ClmProcedureCode_1',
'ClmProcedureCode_2', 'ClmProcedureCode_3', 'ClmProcedureCode_4',
'ClmProcedureCode_5', 'ClmProcedureCode_6', 'DeductibleAmtPaid',
'ClmAdmitDiagnosisCode'],
dtype='object')
```

```
=====
=====
The columns in the Inpatient Dataset are: Index(['BeneID', 'ClaimID',
'ClaimStartDt', 'ClaimEndDt', 'Provider',
'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
'DeductibleAmtPaid', 'DischargeDt', 'DiagnosisGroupCode',
'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3',
'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6',
'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8', 'ClmDiagnosisCode_9',
'ClmDiagnosisCode_10', 'ClmProcedureCode_1', 'ClmProcedureCode_2',
'ClmProcedureCode_3', 'ClmProcedureCode_4', 'ClmProcedureCode_5',
'ClmProcedureCode_6'],
dtype='object')
```

```
=====
=====
The columns in the Beneficiary Dataset are: Index(['BeneID', 'DOB', 'DOD',
'Gender', 'Race', 'RenalDiseaseIndicator',
'State', 'County', 'NoOfMonths_PartACov', 'NoOfMonths_PartBCov',
'ChronicCond_Alzheimer', 'ChronicCond_Heartfailure',
'ChronicCond_KidneyDisease', 'ChronicCond_Cancer',
'ChronicCond_ObstrPulmonary', 'ChronicCond_Depression',
'ChronicCond_Diabetes', 'ChronicCond_IschemicHeart',
'ChronicCond_Osteoporosis', 'ChronicCond_rheumatoidarthritis',
'ChronicCond_stroke', 'IPAnnualReimbursementAmt',
'IPAnnualDeductibleAmt', 'OPAnnualReimbursementAmt',
'OPAnnualDeductibleAmt'],
dtype='object')
```


4.4.2 Checking for common columns between the Outpatient and the Inpatient datasets separately

```
[ ]: #Checking each of the columns in the Outpatient dataset if they are present in
      ↳the Inpatient Dataset
c_o=[]
for o in train_outpat.columns:
    if o in train_inpat.columns:
        c_o.append(o)

#Checking each of the columns in the Inpatient dataset if they are present in
↳the Outpatient dataset
c_i=[]
for i in train_inpat.columns:
    if i in train_outpat.columns:
        c_i.append(i)
print("Cols of Outpatient dataset also present in Inpatient dataset",len(c_o))
print("Cols of Inpatient dataset also present in Outpatient dataset",len(c_i))

#Checking for common column names in the outpatient and the inpatient datasets
c_s= set(c_o).intersection(set(c_i))
c_s= list(c_s)
print("Common columns between the outpatient and the inpatient_
↳datasets",len(c_s))
```

Cols of Outpatient dataset also present in Inpatient dataset 27

Cols of Inpatient dataset also present in Outpatient dataset 27

Common columns between the outpatient and the inpatient datasets 27

```
[ ]: print(c_s)
```

```
['BeneID', 'ClmProcedureCode_1', 'ClmDiagnosisCode_3', 'OperatingPhysician',
'ClaimStartDt', 'ClmProcedureCode_4', 'DeductibleAmtPaid', 'ClmDiagnosisCode_1',
'ClmDiagnosisCode_8', 'OtherPhysician', 'ClmProcedureCode_3',
'ClmDiagnosisCode_7', 'ClmDiagnosisCode_9', 'ClmProcedureCode_5',
'AttendingPhysician', 'ClmDiagnosisCode_5', 'ClmProcedureCode_6',
'ClmProcedureCode_2', 'ClmDiagnosisCode_10', 'ClaimEndDt',
'ClmAdmitDiagnosisCode', 'ClmDiagnosisCode_6', 'Provider', 'ClaimID',
'InscClaimAmtReimbursed', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_4']
```

4.4.3 Merging the Outpatient and the Inpatient datasets based on the common columns between both the datasets. We will be doing an outer merge as we need to take the union of all the elements in both the datasets

```
[ ]: train_fin_df= pd.
      ↳merge(train_inpat,train_outpat,left_on=c_s,right_on=c_s,how='outer')
train_fin_df.shape
```

```
[ ]: (558211, 30)
```

```
[ ]: train_fin_df.head(2)
```

```
[ ]:      BeneID  ClaimID  ... ClmProcedureCode_5 ClmProcedureCode_6
0  BENE11001  CLM46614  ...                NaN                NaN
1  BENE11001  CLM66048  ...                NaN                NaN
```

```
[2 rows x 30 columns]
```

4.4.4 Merging the resultant dataset with Beneficiary data on the BeneID column in both the datasets

```
[ ]: train_fin= pd.merge(train_fin_df,train_ben, left_on='BeneID',right_on=
    ↳'BeneID',how='outer')
train_fin.shape
```

```
[ ]: (558211, 54)
```

```
[ ]: print("The columns in the final merged dataset are:",train_fin.columns)
```

```
The columns in the final merged dataset are: Index(['BeneID', 'ClaimID',
'ClaimStartDt', 'ClaimEndDt', 'Provider',
'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
'DeductibleAmtPaid', 'DischargeDt', 'DiagnosisGroupCode',
'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3',
'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6',
'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8', 'ClmDiagnosisCode_9',
'ClmDiagnosisCode_10', 'ClmProcedureCode_1', 'ClmProcedureCode_2',
'ClmProcedureCode_3', 'ClmProcedureCode_4', 'ClmProcedureCode_5',
'ClmProcedureCode_6', 'DOB', 'DOD', 'Gender', 'Race',
'RenalDiseaseIndicator', 'State', 'County', 'NoOfMonths_PartACov',
'NoOfMonths_PartBCov', 'ChronicCond_Alzheimer',
'ChronicCond_Heartfailure', 'ChronicCond_KidneyDisease',
'ChronicCond_Cancer', 'ChronicCond_ObstrPulmonary',
'ChronicCond_Depression', 'ChronicCond_Diabetes',
'ChronicCond_IschemicHeart', 'ChronicCond_Osteoporasis',
'ChronicCond_rheumatoidarthritis', 'ChronicCond_stroke',
'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt'],
dtype='object')
```

4.4.5 Merging the Y variable with the final dataset

```
[ ]: train_fin= pd.merge(train_fin,train_y,left_on=
    ↳'Provider',right_on='Provider',how='outer')
train_fin.shape
```

```
[ ]: (558211, 55)
```

```
[ ]: train_fin.columns
```

```
[ ]: Index(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
          'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
          'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
          'DeductibleAmtPaid', 'DischargeDt', 'DiagnosisGroupCode',
          'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3',
          'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6',
          'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8', 'ClmDiagnosisCode_9',
          'ClmDiagnosisCode_10', 'ClmProcedureCode_1', 'ClmProcedureCode_2',
          'ClmProcedureCode_3', 'ClmProcedureCode_4', 'ClmProcedureCode_5',
          'ClmProcedureCode_6', 'DOB', 'DOD', 'Gender', 'Race',
          'RenalDiseaseIndicator', 'State', 'County', 'NoOfMonths_PartACov',
          'NoOfMonths_PartBCov', 'ChronicCond_Alzheimer',
          'ChronicCond_Heartfailure', 'ChronicCond_KidneyDisease',
          'ChronicCond_Cancer', 'ChronicCond_ObstrPulmonary',
          'ChronicCond_Depression', 'ChronicCond_Diabetes',
          'ChronicCond_IschemicHeart', 'ChronicCond_Osteoporosis',
          'ChronicCond_rheumatoidarthritis', 'ChronicCond_stroke',
          'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
          'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'PotentialFraud'],
          dtype='object')
```

```
[ ]: train_fin.head()
```

```
[ ]:      BeneID    ClaimID  ... OPAnnualDeductibleAmt PotentialFraud
0  BENE11001    CLM46614  ...                70             Yes
1  BENE16973    CLM565430  ...               200             Yes
2  BENE17521    CLM34721  ...                20             Yes
3  BENE21718    CLM72336  ...               540             Yes
4  BENE22934    CLM73394  ...               160             Yes
```

```
[5 rows x 55 columns]
```

4.4.6 Checking for the datatypes of all the columns in the final dataset

```
[ ]: train_fin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 558211 entries, 0 to 558210
Data columns (total 55 columns):
#   Column                Non-Null Count  Dtype
---  -
0   BeneID                558211 non-null object
1   ClaimID               558211 non-null object
2   ClaimStartDt          558211 non-null object
3   ClaimEndDt            558211 non-null object
```

4	Provider	558211 non-null	object
5	InscClaimAmtReimbursed	558211 non-null	int64
6	AttendingPhysician	556703 non-null	object
7	OperatingPhysician	114447 non-null	object
8	OtherPhysician	199736 non-null	object
9	AdmissionDt	40474 non-null	object
10	ClmAdmitDiagnosisCode	145899 non-null	object
11	DeductibleAmtPaid	557312 non-null	float64
12	DischargeDt	40474 non-null	object
13	DiagnosisGroupCode	40474 non-null	object
14	ClmDiagnosisCode_1	547758 non-null	object
15	ClmDiagnosisCode_2	362605 non-null	object
16	ClmDiagnosisCode_3	243055 non-null	object
17	ClmDiagnosisCode_4	164536 non-null	object
18	ClmDiagnosisCode_5	111924 non-null	object
19	ClmDiagnosisCode_6	84392 non-null	object
20	ClmDiagnosisCode_7	66177 non-null	object
21	ClmDiagnosisCode_8	53444 non-null	object
22	ClmDiagnosisCode_9	41815 non-null	object
23	ClmDiagnosisCode_10	5010 non-null	object
24	ClmProcedureCode_1	23310 non-null	float64
25	ClmProcedureCode_2	5490 non-null	float64
26	ClmProcedureCode_3	969 non-null	float64
27	ClmProcedureCode_4	118 non-null	float64
28	ClmProcedureCode_5	9 non-null	float64
29	ClmProcedureCode_6	0 non-null	float64
30	DOB	558211 non-null	object
31	DOD	4131 non-null	object
32	Gender	558211 non-null	int64
33	Race	558211 non-null	int64
34	RenalDiseaseIndicator	558211 non-null	object
35	State	558211 non-null	int64
36	County	558211 non-null	int64
37	NoOfMonths_PartACov	558211 non-null	int64
38	NoOfMonths_PartBCov	558211 non-null	int64
39	ChronicCond_Alzheimer	558211 non-null	int64
40	ChronicCond_Heartfailure	558211 non-null	int64
41	ChronicCond_KidneyDisease	558211 non-null	int64
42	ChronicCond_Cancer	558211 non-null	int64
43	ChronicCond_ObstrPulmonary	558211 non-null	int64
44	ChronicCond_Depression	558211 non-null	int64
45	ChronicCond_Diabetes	558211 non-null	int64
46	ChronicCond_IschemicHeart	558211 non-null	int64
47	ChronicCond_Osteoporosis	558211 non-null	int64
48	ChronicCond_rheumatoidarthritis	558211 non-null	int64
49	ChronicCond_stroke	558211 non-null	int64
50	IPAnnualReimbursementAmt	558211 non-null	int64
51	IPAnnualDeductibleAmt	558211 non-null	int64

```

52  OPAnnualReimbursementAmt      558211 non-null  int64
53  OPAnnualDeductibleAmt        558211 non-null  int64
54  PotentialFraud                558211 non-null  object
dtypes: float64(7), int64(22), object(26)
memory usage: 238.5+ MB

```

```

[ ]: #Storing the final dataframe as a pickled file
with open('/content/drive/MyDrive/Colab Notebooks/train_fin.pkl','wb') as tr_df:
    pickle.dump(train_fin,tr_df)

[ ]: #Loading the pickled file
with open('/content/drive/MyDrive/Colab Notebooks/train_fin.pkl','rb') as tr_df:
    train_fin= pickle.load(tr_df)

[ ]: train_fin.shape

[ ]: (558211, 55)

```

4.5 Splitting the Data into Train and Cross Validate Datasets

```

[ ]: y= train_fin['PotentialFraud']
train_fin.drop(['PotentialFraud'],axis=1, inplace= True)

[ ]: train_fin,cv_fin,train_y,cv_y= train_test_split(train_fin,y,test_size=0.
→2,stratify=y,random_state=42)
print(train_fin.shape)
print(train_y.shape)
print(cv_fin.shape)
print(cv_y.shape)

(446568, 54)
(446568,)
(111643, 54)
(111643,)

[ ]: train_fin.reset_index(drop=True,inplace=True)
train_y.reset_index(drop=True,inplace=True)

cv_fin.reset_index(drop=True,inplace=True)
cv_y.reset_index(drop=True,inplace=True)

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_x.pkl','wb') as tr_df:
    pickle.dump(train_fin,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_x.pkl','wb') as cv_df:
    pickle.dump(cv_fin,cv_df)
with open('/content/drive/MyDrive/Colab Notebooks/train_y.pkl','wb') as tr_y:
    pickle.dump(train_y,tr_y)
with open('/content/drive/MyDrive/Colab Notebooks/cv_y.pkl','wb') as c_y:
    pickle.dump(cv_y,c_y)

```

```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_x.pkl','rb') as tr_df:
      train_fin= pickle.load(tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_x.pkl','rb') as cv_df:
      cv_fin= pickle.load(cv_df)
with open('/content/drive/MyDrive/Colab Notebooks/train_y.pkl','rb') as tr_y:
      train_y= pickle.load(tr_y)
with open('/content/drive/MyDrive/Colab Notebooks/cv_y.pkl','rb') as c_y:
      cv_y= pickle.load(c_y)
```

4.6 Looking at the Class Distribution in the Train Dataset

```
[ ]: #Calculating the number of row items where the Provider is NOT a Potentila
      ↪fraud in percentage terms
tr_no_per= np.round((train_y.value_counts()[0])/(train_y.
      ↪value_counts()[0]+train_y.value_counts()[1]),3)*100

#Calculating the number of row items where the Provider is a Potentila fraud in
      ↪percentage terms
tr_yes_per= np.round((train_y.value_counts()[1])/(train_y.
      ↪value_counts()[0]+train_y.value_counts()[1]),3)*100

#Calculating the number of row items where the Provider is NOT a Potentila
      ↪fraud in percentage terms
cv_no_per= np.round((cv_y.value_counts()[0])/(cv_y.value_counts()[0]+cv_y.
      ↪value_counts()[1]),3)*100

#Calculating the number of row items where the Provider is a Potentila fraud in
      ↪percentage terms
cv_yes_per= np.round((cv_y.value_counts()[1])/(cv_y.value_counts()[0]+cv_y.
      ↪value_counts()[1]),3)*100

#Plotting the Potential and Non Potential Fraud scenarios
fig= plt.figure(figsize=(10,5))
gs= GridSpec(1,2, figure=fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

sns.barplot(ax= ax1,x=['Non-Fraud',"Fraud"],y=
      ↪[tr_no_per,tr_yes_per],palette='crest')
sns.barplot(ax= ax2,x=['Non-Fraud',"Fraud"],y=
      ↪[cv_no_per,cv_yes_per],palette='crest')

ax1.title.set_text("Distribution of the Train_y Data Labels")
```

```
ax2.title.set_text("Distribution of the CV_y Data Labels")
ax1.set_ylabel("Percentage of Providers")
ax2.set_ylabel("Percentage of Providers")

plt.show()
```



4.7 Observations

1. We see that there is a 60:40 split between the number of observations belonging to the Non-Fraud class and the Fraud class.
2. Using the Stratify option in the Train-Test split has ensured that class distribution of the observations belonging to the Non-Fraud and Fraud cases has remained the same in both Train and CV datasets

```
[ ]: print("Percentage of Non-Fraud class in Train dataset:",tr_no_per,'%')
      print("Percentage of Fraud class in Train dataset:",tr_yes_per,'%')
      print("Percentage of Non-Fraud class in Cross Validate dataset:",cv_no_per,'%')
      print("Percentage of Fraud class in Cross Validate dataset:",cv_yes_per,'%')
```

```
Percentage of Non-Fraud class in Train dataset: 61.9 %
Percentage of Fraud class in Train dataset: 38.1 %
Percentage of Non-Fraud class in Cross Validate dataset: 61.9 %
Percentage of Fraud class in Cross Validate dataset: 38.1 %
```

4.8 Checking for the percentage of nan values in each of the columns in the Train Data

```
[ ]: na_perc= np.round(((train_fin.isna().sum())/train_fin.shape[0])*100,2)
na_perc_df= na_perc.to_frame()
na_perc_df.reset_index(inplace= True)
na_perc_df.columns= ["col_name", "na_percentage"]
print(na_perc_df)
```

	col_name	na_percentage
0	BeneID	0.00
1	ClaimID	0.00
2	ClaimStartDt	0.00
3	ClaimEndDt	0.00
4	Provider	0.00
5	InscClaimAmtReimbursed	0.00
6	AttendingPhysician	0.27
7	OperatingPhysician	79.50
8	OtherPhysician	64.24
9	AdmissionDt	92.72
10	ClmAdmitDiagnosisCode	73.88
11	DeductibleAmtPaid	0.16
12	DischargeDt	92.72
13	DiagnosisGroupCode	92.72
14	ClmDiagnosisCode_1	1.86
15	ClmDiagnosisCode_2	35.04
16	ClmDiagnosisCode_3	56.47
17	ClmDiagnosisCode_4	70.52
18	ClmDiagnosisCode_5	79.93
19	ClmDiagnosisCode_6	84.86
20	ClmDiagnosisCode_7	88.13
21	ClmDiagnosisCode_8	90.42
22	ClmDiagnosisCode_9	92.50
23	ClmDiagnosisCode_10	99.09
24	ClmProcedureCode_1	95.80
25	ClmProcedureCode_2	99.02
26	ClmProcedureCode_3	99.83
27	ClmProcedureCode_4	99.98
28	ClmProcedureCode_5	100.00
29	ClmProcedureCode_6	100.00
30	DOB	0.00
31	DOD	99.26
32	Gender	0.00
33	Race	0.00
34	RenalDiseaseIndicator	0.00
35	State	0.00
36	County	0.00
37	NoOfMonths_PartACov	0.00

38	NoOfMonths_PartBCov	0.00
39	ChronicCond_Alzheimer	0.00
40	ChronicCond_Heartfailure	0.00
41	ChronicCond_KidneyDisease	0.00
42	ChronicCond_Cancer	0.00
43	ChronicCond_ObstrPulmonary	0.00
44	ChronicCond_Depression	0.00
45	ChronicCond_Diabetes	0.00
46	ChronicCond_IschemicHeart	0.00
47	ChronicCond_Osteoporosis	0.00
48	ChronicCond_rheumatoidarthritis	0.00
49	ChronicCond_stroke	0.00
50	IPAnnualReimbursementAmt	0.00
51	IPAnnualDeductibleAmt	0.00
52	OPAnnualReimbursementAmt	0.00
53	OPAnnualDeductibleAmt	0.00

```
[ ]: #Isolating the column numbers where the NA percentage is Zero
na_col=[]
na_perc= np.round(((train_fin.isna().sum())/train_fin.shape[0])*100,2)
na_perc_df= na_perc.to_frame()
na_perc_df.reset_index(inplace= True)
na_perc_df.columns= ["col_name", "na_percentage"]
for i in range(na_perc_df.shape[0]):
    if na_perc_df.iloc[i,1] == 0:
        na_col.append(i)

print(na_col)
```

```
[0, 1, 2, 3, 4, 5, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53]
```

```
[ ]: #Deleting the columns with 0% NA from the newly created na_perc_df dataframe
na_perc_df.drop(index=na_col,inplace=True)
na_perc_df.reset_index(drop=True,inplace=True)
print(na_perc_df)
```

	col_name	na_percentage
0	AttendingPhysician	0.27
1	OperatingPhysician	79.50
2	OtherPhysician	64.24
3	AdmissionDt	92.72
4	ClmAdmitDiagnosisCode	73.88
5	DeductibleAmtPaid	0.16
6	DischargeDt	92.72
7	DiagnosisGroupCode	92.72
8	ClmDiagnosisCode_1	1.86

9	ClmDiagnosisCode_2	35.04
10	ClmDiagnosisCode_3	56.47
11	ClmDiagnosisCode_4	70.52
12	ClmDiagnosisCode_5	79.93
13	ClmDiagnosisCode_6	84.86
14	ClmDiagnosisCode_7	88.13
15	ClmDiagnosisCode_8	90.42
16	ClmDiagnosisCode_9	92.50
17	ClmDiagnosisCode_10	99.09
18	ClmProcedureCode_1	95.80
19	ClmProcedureCode_2	99.02
20	ClmProcedureCode_3	99.83
21	ClmProcedureCode_4	99.98
22	ClmProcedureCode_5	100.00
23	ClmProcedureCode_6	100.00
24	DOD	99.26

4.9 Checking for the percentage of nan values in each of the columns in the CV Data

```
[ ]: na_perc_cv= np.round(((cv_fin.isna().sum())/cv_fin.shape[0])*100,2)
na_perc_df_cv= na_perc_cv.to_frame()
na_perc_df_cv.reset_index(inplace= True)
na_perc_df_cv.columns= ["col_name", "na_percentage"]
print(na_perc_df_cv)
```

	col_name	na_percentage
0	BeneID	0.00
1	ClaimID	0.00
2	ClaimStartDt	0.00
3	ClaimEndDt	0.00
4	Provider	0.00
5	InscClaimAmtReimbursed	0.00
6	AttendingPhysician	0.27
7	OperatingPhysician	79.47
8	OtherPhysician	64.13
9	AdmissionDt	92.88
10	ClmAdmitDiagnosisCode	73.80
11	DeductibleAmtPaid	0.16
12	DischargeDt	92.88
13	DiagnosisGroupCode	92.88
14	ClmDiagnosisCode_1	1.92
15	ClmDiagnosisCode_2	35.04
16	ClmDiagnosisCode_3	56.43
17	ClmDiagnosisCode_4	70.53
18	ClmDiagnosisCode_5	80.02
19	ClmDiagnosisCode_6	84.95
20	ClmDiagnosisCode_7	88.21

21	ClmDiagnosisCode_8	90.45
22	ClmDiagnosisCode_9	92.56
23	ClmDiagnosisCode_10	99.14
24	ClmProcedureCode_1	95.91
25	ClmProcedureCode_2	99.02
26	ClmProcedureCode_3	99.83
27	ClmProcedureCode_4	99.98
28	ClmProcedureCode_5	100.00
29	ClmProcedureCode_6	100.00
30	DOB	0.00
31	DOD	99.26
32	Gender	0.00
33	Race	0.00
34	RenalDiseaseIndicator	0.00
35	State	0.00
36	County	0.00
37	NoOfMonths_PartACov	0.00
38	NoOfMonths_PartBCov	0.00
39	ChronicCond_Alzheimer	0.00
40	ChronicCond_Heartfailure	0.00
41	ChronicCond_KidneyDisease	0.00
42	ChronicCond_Cancer	0.00
43	ChronicCond_ObstrPulmonary	0.00
44	ChronicCond_Depression	0.00
45	ChronicCond_Diabetes	0.00
46	ChronicCond_IschemicHeart	0.00
47	ChronicCond_Osteoporosis	0.00
48	ChronicCond_rheumatoidarthritis	0.00
49	ChronicCond_stroke	0.00
50	IPAnnualReimbursementAmt	0.00
51	IPAnnualDeductibleAmt	0.00
52	OPAnnualReimbursementAmt	0.00
53	OPAnnualDeductibleAmt	0.00

```
[ ]: na_col_cv=[]
na_perc_cv= np.round(((cv_fin.isna().sum())/cv_fin.shape[0])*100,2)
na_perc_df_cv= na_perc_cv.to_frame()
na_perc_df_cv.reset_index(inplace= True)
na_perc_df_cv.columns= ["col_name","na_percentage"]
for i in range(na_perc_df_cv.shape[0]):
    if na_perc_df_cv.iloc[i,1] == 0:
        na_col_cv.append(i)

print(na_col_cv)
```

```
[0, 1, 2, 3, 4, 5, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53]
```

```
[ ]: na_perc_df_cv.drop(index=na_col_cv,inplace=True)
na_perc_df_cv.reset_index(drop=True,inplace=True)
print(na_perc_df_cv)
```

	col_name	na_percentage
0	AttendingPhysician	0.27
1	OperatingPhysician	79.47
2	OtherPhysician	64.13
3	AdmissionDt	92.88
4	ClmAdmitDiagnosisCode	73.80
5	DeductibleAmtPaid	0.16
6	DischargeDt	92.88
7	DiagnosisGroupCode	92.88
8	ClmDiagnosisCode_1	1.92
9	ClmDiagnosisCode_2	35.04
10	ClmDiagnosisCode_3	56.43
11	ClmDiagnosisCode_4	70.53
12	ClmDiagnosisCode_5	80.02
13	ClmDiagnosisCode_6	84.95
14	ClmDiagnosisCode_7	88.21
15	ClmDiagnosisCode_8	90.45
16	ClmDiagnosisCode_9	92.56
17	ClmDiagnosisCode_10	99.14
18	ClmProcedureCode_1	95.91
19	ClmProcedureCode_2	99.02
20	ClmProcedureCode_3	99.83
21	ClmProcedureCode_4	99.98
22	ClmProcedureCode_5	100.00
23	ClmProcedureCode_6	100.00
24	DOD	99.26

4.10 Analysis of the presence of high percentage of NA values in the Claim Procedure variables using barplots

```
[ ]: clm_proc=
    ['ClmProcedureCode_1','ClmProcedureCode_2','ClmProcedureCode_3','ClmProcedureCode_4','ClmPr
clm_proc_in=[]
clm_proc_out=[]
clm_proc_mer=[]

for i in clm_proc:
    clm_proc_in.append(np.round((train_inpat[i].isna().sum()/
    len(train_inpat[i]))*100,2))
    clm_proc_out.append(np.round((train_outpat[i].isna().sum()/
    len(train_outpat[i]))*100,2))
    clm_proc_mer.append(np.round((train_fin[i].isna().sum()/
    len(train_fin[i]))*100,2))
```

```

fig= plt.figure(figsize=(12,8))
gs= GridSpec(2,2,figure= fig)

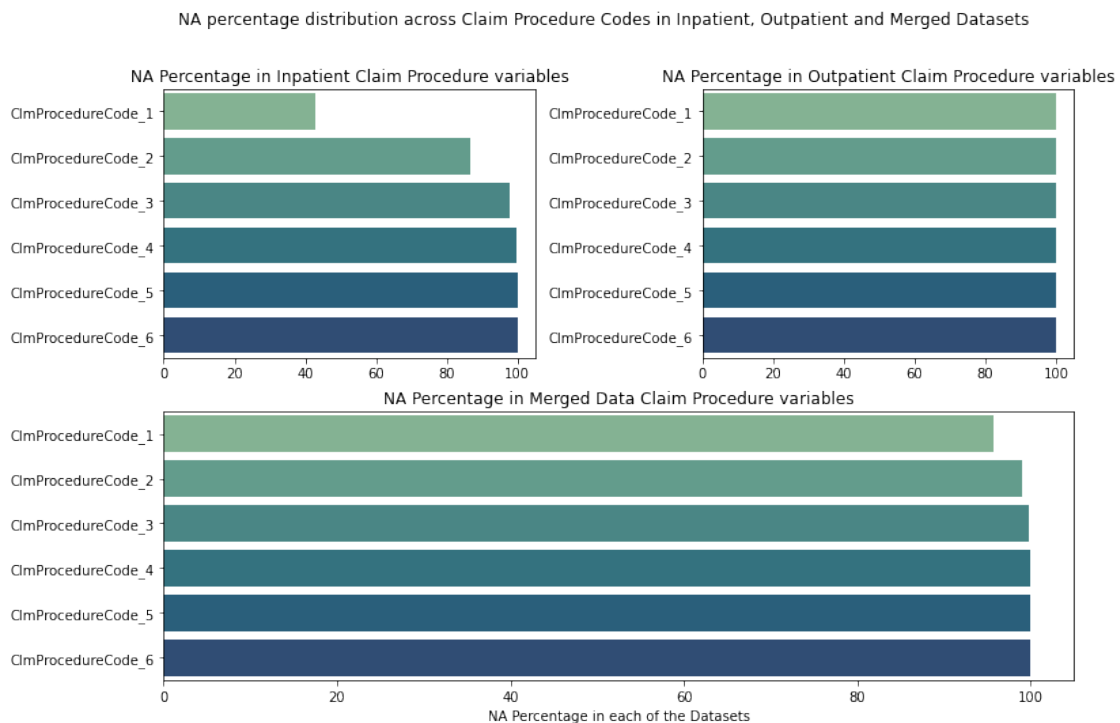
fig.suptitle('NA percentage distribution across Claim Procedure Codes in_
↳Inpatient, Outpatient and Merged Datasets')
ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,:])

sns.barplot(ax=ax1,y= clm_proc,x= clm_proc_in, palette='crest')
sns.barplot(ax=ax2,y= clm_proc,x= clm_proc_out,palette='crest')
sns.barplot(ax=ax3,y= clm_proc,x= clm_proc_mer,palette='crest')

ax1.title.set_text('NA Percentage in Inpatient Claim Procedure variables')
ax2.title.set_text('NA Percentage in Outpatient Claim Procedure variables')
ax3.title.set_text('NA Percentage in Merged Data Claim Procedure variables')

plt.subplots_adjust(wspace=0.45)
plt.xlabel("NA Percentage in each of the Datasets")
plt.show()

```



4.11 Observations

1. We see that there are 100% NA values in the Outpatient dataset in all of the claim Procedure columns. This is because most of the outpatients do not undergo procedures.
2. In case of a need for a complex procedure, the patients are admitted and are treated as inpatients
3. High percentage of the NA values in the merged datasets is not due to missing data but due to the reason that the size of the Outpatient dataset is much higher than the Inpatient dataset
4. As most of the outpatient dataset claim procedure has a high values of NA, they are introducing skewness in the merged dataset.

4.12 Feature Engineering

1. Each of the claim procedure codes indicates a different procedure hence the counting the number of procedures performed effectively captures the information carried by the 6 different claim procedure columns.
2. I have created a new feature capturing the number of procedures performed for each of the patients. Higher the number of procedures it is highly likely that higher is the complexity of the case.

```
[ ]: clm_proc=
      →['ClmProcedureCode_1','ClmProcedureCode_2','ClmProcedureCode_3','ClmProcedureCode_4','ClmPr
```

```
[ ]: #Isolating all the claim procedure columns of the train and crossvalidate
      →datasets into separate dataframes
tr_clm_pr= train_fin[clm_proc]
cv_clm_pr= cv_fin[clm_proc]
print(tr_clm_pr.shape)
print(cv_clm_pr.shape)
```

```
(446568, 6)
(111643, 6)
```

```
[ ]: #Creating a new column called '#_Procedures' to save the counts for each row
      →where the counts of the non-nan values in each of the clm_proc are stored
tr_clm_pr['#_Procedures']= np.zeros(len(train_fin['ClmProcedureCode_1']))

for i in tqdm(range(len(tr_clm_pr['ClmProcedureCode_1']))):
    count= 0
    for j in range(len(clm_proc)):
        if pd.isnull(tr_clm_pr.iloc[i,j])== False:
            count=count+1

    tr_clm_pr['#_Procedures'][i]= count
```

100%| 446568/446568 [02:04<00:00, 3597.97it/s]

```
[ ]: tr_clm_pr['#_Procedures'].describe()
```

```
[ ]: count      446568.000000
      mean       0.053786
      std        0.281055
      min        0.000000
      25%        0.000000
      50%        0.000000
      75%        0.000000
      max        5.000000
      Name: #_Procedures, dtype: float64
```

```
[ ]: cv_clm_pr['#_Procedures'] = np.zeros(len(cv_fin['ClmProcedureCode_1']))

#Looping through each of the claim procedure columns and each of the
→observations
#Counting the number of non-na values in each of the clm_proc columns in the
→each of the obs
#Storing the count values in a separate column titled '#_Procedures'
for i in tqdm(range(len(cv_clm_pr['ClmProcedureCode_1']))):
    count = 0
    for j in range(len(clm_proc)):
        if pd.isnull(cv_clm_pr.iloc[i,j]) == False:
            count = count + 1

    cv_clm_pr['#_Procedures'][i] = count
```

100%|| 111643/111643 [00:31<00:00, 3580.01it/s]

```
[ ]: cv_clm_pr['#_Procedures'].describe()
```

```
[ ]: count      111643.000000
      mean       0.052641
      std        0.278442
      min        0.000000
      25%        0.000000
      50%        0.000000
      75%        0.000000
      max        5.000000
      Name: #_Procedures, dtype: float64
```

```
[ ]: train_fin['#_Procedures'] = tr_clm_pr['#_Procedures']
      cv_fin['#_Procedures'] = cv_clm_pr['#_Procedures']
```

```
[ ]: print(np.unique(train_fin['#_Procedures']))
      print(np.unique(cv_fin['#_Procedures']))
```

```
[0. 1. 2. 3. 4. 5.]
[0. 1. 2. 3. 4. 5.]
```

```

[: uni_proc_tr= np.unique(train_fin['#_Procedures'])
uni_proc_cv= np.unique(cv_fin['#_Procedures'])

tr_proc_counts= np.round((train_fin['#_Procedures'].value_counts()/
→len(train_fin['#_Procedures']))*100,2)
cv_proc_counts= np.round((cv_fin['#_Procedures'].value_counts()/
→len(cv_fin['#_Procedures']))*100,2)

fig= plt.figure(figsize=(10,6))
gs= GridSpec(1,2,figure=fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

sns.barplot(ax= ax1,y= tr_proc_counts, x= uni_proc_tr, palette= 'crest')
sns.barplot(ax= ax2,y= cv_proc_counts, x= uni_proc_cv, palette= 'crest')

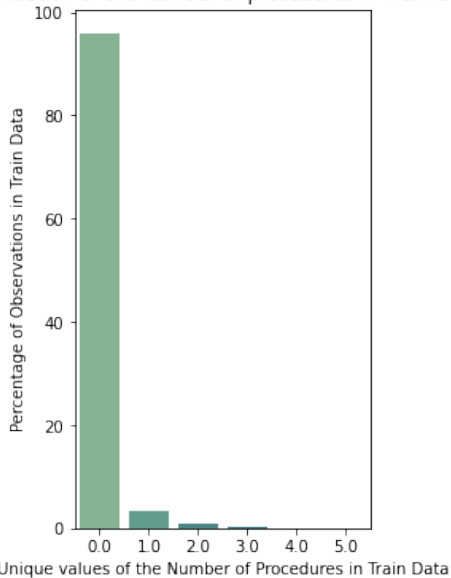
ax1.set_ylabel("Percentage of Observations in Train Data")
ax2.set_ylabel("Percentage of Observations in CV Data")

ax1.set_xlabel('Unique values of the Number of Procedures in Train Data')
ax2.set_xlabel('Unique values of the Number of Procedures in CV Data')

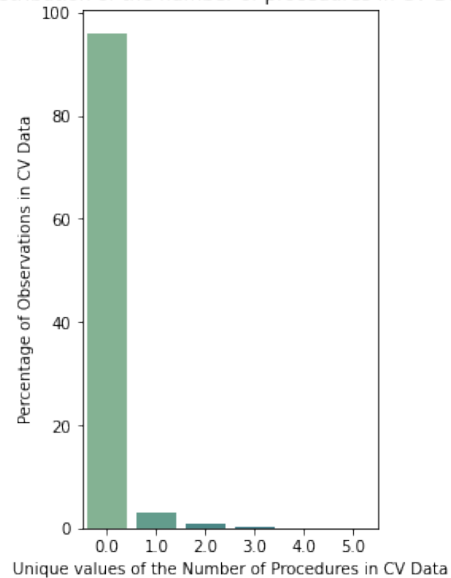
ax1.set_title("Distribution of the number of procedures in Train Data")
ax2.set_title("Distribution of the number of procedures in CV Data")
plt.subplots_adjust(wspace=1)
plt.show()

```

Distribution of the number of procedures in Train Data



Distribution of the number of procedures in CV Data



4.13 Observations

1. As can be seen from the above plot that more than 95% of the procedures have '0' procedures, followed by 1 procedure and so on.
2. This drastic skewness in the data could be due to the fact that the Outpatient dataset is dominant in the overall merged dataset and in majority of the Outpatient cases, the patients do not go through any procedures.
3. In addition to point 2, as most procedures require prepping the patient or stabilizing the patient before the procedure could take 1-2 days hence the patient is most likely to be admitted and treated as an inpatient before carrying out a procedure barring from a very few procedures

```
[ ]: train_fin3['#_Procedures'].value_counts()/len(train_fin3['#_Procedures'])
```

```
[ ]: 0.0    0.958036
      1.0    0.032116
      2.0    0.008113
      3.0    0.001516
      4.0    0.000202
      5.0    0.000018
      Name: #_Procedures, dtype: float64
```

4.13.1 Dropping the 6 Claim procedure code variables

```
[ ]: train_fin.drop(clm_proc, axis=1, inplace= True)
      cv_fin.drop(clm_proc, axis=1, inplace= True)
```

4.13.2 Analysis of the presence of high percentage of NA values in the Claim Diagnosis variables using barplots

```
[ ]: clm_diag=
      → ['ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3', 'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6']
      clm_diag_nai=[]
      clm_diag_nao=[]
      clm_diag_na=[]

      for i in clm_diag:
          clm_diag_nai.append(np.round((train_inpat[i].isna().sum()/
      → len(train_inpat[i]))*100,2))
          clm_diag_nao.append(np.round((train_outpat[i].isna().sum()/
      → len(train_outpat[i]))*100,2))
          clm_diag_na.append(np.round((train_fin[i].isna().sum()/
      → len(train_fin[i]))*100,2))

      fig= plt.figure(figsize=(12,8))
      gs= GridSpec(2,2,figure= fig)
```

```

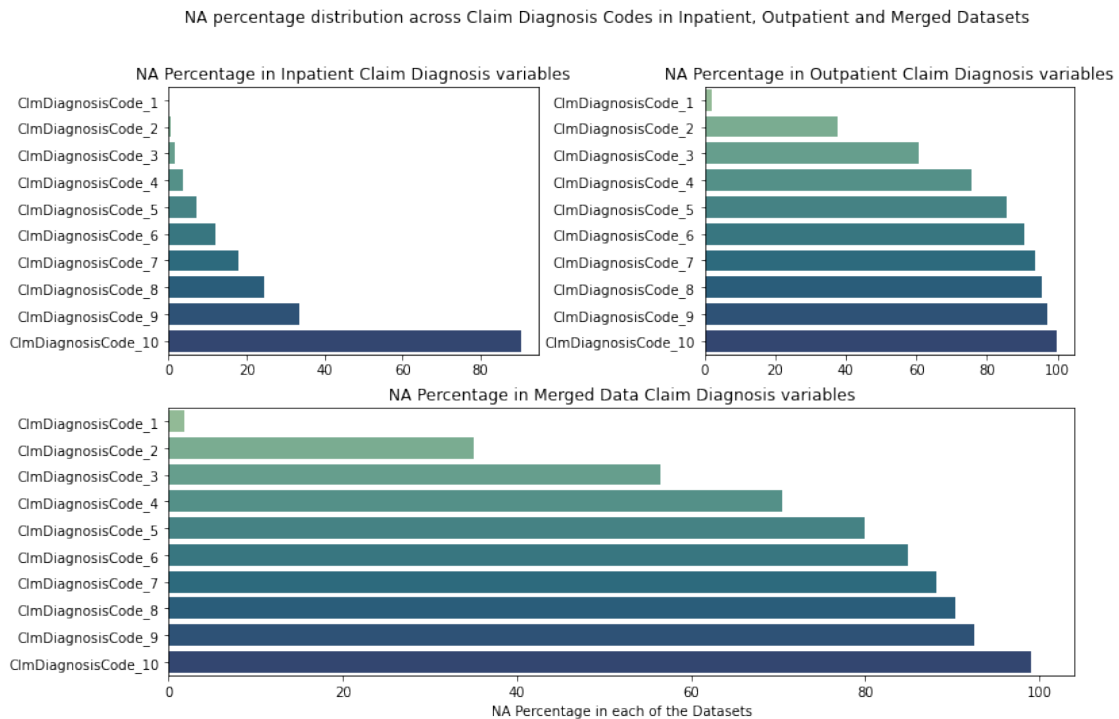
fig.suptitle('NA percentage distribution across Claim Diagnosis Codes in_
→Inpatient, Outpatient and Merged Datasets')
ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,:])

sns.barplot(ax=ax1,y= clm_diag,x= clm_diag_nai,palette='crest')
sns.barplot(ax=ax2,y= clm_diag,x= clm_diag_nao,palette='crest')
sns.barplot(ax=ax3,y= clm_diag,x= clm_diag_na,palette='crest')

ax1.title.set_text('NA Percentage in Inpatient Claim Diagnosis variables')
ax2.title.set_text('NA Percentage in Outpatient Claim Diagnosis variables')
ax3.title.set_text('NA Percentage in Merged Data Claim Diagnosis variables')

plt.subplots_adjust(wspace=0.45)
plt.xlabel("NA Percentage in each of the Datasets")
plt.show()

```



4.14 Observations

1. Very similar observations as the Claims Procedure variable. Even in this case the reasons for the NA values are similar as in the case of ClaimsProcedures variable

4.14.1 Feature Engineering

As employed in the case of Claims Procedure I will be creating a new column called the no.of.diagnosis

Higher the number of Diagnosis the higher is the complexity of the patients diagnosis.

```
[ ]: clm_diag=
    → ['ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3', 'ClmDiagnosisCode_4', 'ClmDi

[ ]: tr_clm_dg= train_fin[clm_diag]
    cv_clm_dg= cv_fin[clm_diag]
    print(tr_clm_dg.shape)
    print(cv_clm_dg.shape)
```

```
(446568, 10)
```

```
(111643, 10)
```

```
[ ]: tr_clm_dg['#_DiagnosisCodes']= np.zeros(len(tr_clm_dg['ClmDiagnosisCode_1']))

#Looping through each of the claim diagnosis columns and each of the
    → observations
#Counting the number of non-na values in each of the clm_diag columns in the
    → each of the obs
#Storing the count values in a seperate column titled '#_DiagnosisCodes'
for i in tqdm(range(len(tr_clm_dg['ClmDiagnosisCode_1']))):
    count= 0
    for j in range(len(clm_diag)):
        if pd.isnull(tr_clm_dg.iloc[i,j])== False:
            count=count+1

    tr_clm_dg['#_DiagnosisCodes'][i]= count
```

```
100%|| 446568/446568 [03:09<00:00, 2357.77it/s]
```

```
[ ]: tr_clm_dg['#_DiagnosisCodes'].describe()
```

```
[ ]: count      446568.000000
    mean         3.011736
    std          2.449265
    min          0.000000
    25%          1.000000
    50%          2.000000
    75%          4.000000
    max          10.000000
    Name: #_DiagnosisCodes, dtype: float64
```

```
[ ]: cv_clm_dg['#_DiagnosisCodes']= np.zeros(len(cv_clm_dg['ClmDiagnosisCode_1']))
```

```

#Looping through each of the claim diagnosis columns and each of the
    ↳ observations
#Counting the number of non-na values in each of the clm_diag columns in the
    ↳ each of the obs
#Storing the count values in a separate column titled '#_DiagnosisCodes'
for i in tqdm(range(len(cv_clm_dg['ClmDiagnosisCode_1']))):
    count= 0
    for j in range(len(clm_diag)):
        if pd.isnull(cv_clm_dg.iloc[i,j])== False:
            count=count+1

    cv_clm_dg['_DiagnosisCodes'][i]= count

```

100%|| 111643/111643 [00:47<00:00, 2351.12it/s]

```
[ ]: cv_clm_dg['_DiagnosisCodes'].describe()
```

```
[ ]: count      111643.000000
      mean        3.007542
      std         2.444012
      min         0.000000
      25%         1.000000
      50%         2.000000
      75%         4.000000
      max         10.000000
      Name: #_DiagnosisCodes, dtype: float64

```

```
[ ]: train_fin['_DiagnosisCodes']= tr_clm_dg['_DiagnosisCodes']
      cv_fin['_DiagnosisCodes']= cv_clm_dg['_DiagnosisCodes']

```

```
[ ]: uni_diag_tr= np.unique(train_fin['_DiagnosisCodes'])
      uni_diag_cv= np.unique(cv_fin['_DiagnosisCodes'])

      tr_diag_counts= np.round((train_fin['_DiagnosisCodes'].value_counts()/
          ↳ len(train_fin['_DiagnosisCodes']))*100,2)
      cv_diag_counts= np.round((cv_fin['_DiagnosisCodes'].value_counts()/
          ↳ len(cv_fin['_DiagnosisCodes']))*100,2)

      fig= plt.figure(figsize=(14,6))
      gs= GridSpec(1,2,figure=fig)

      ax1= fig.add_subplot(gs[0,0])
      ax2= fig.add_subplot(gs[0,1])

      sns.barplot(ax= ax1,y= tr_diag_counts, x= uni_diag_tr, palette= 'crest')
      sns.barplot(ax= ax2,y= cv_diag_counts, x= uni_diag_cv, palette= 'crest')

      ax1.set_ylabel("Percentage of Observations in Train Data")

```

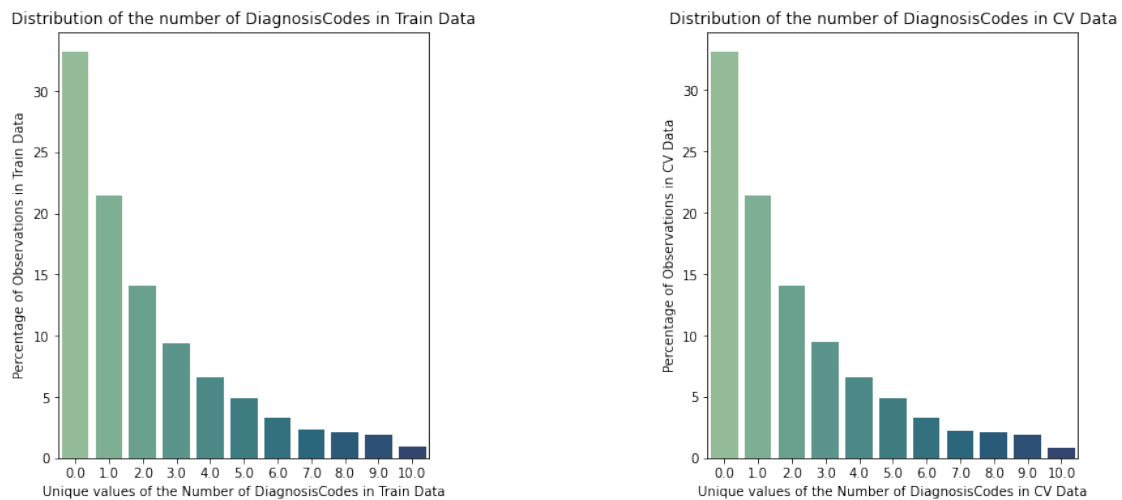
```

ax2.set_ylabel("Percentage of Observations in CV Data")

ax1.set_xlabel('Unique values of the Number of DiagnosisCodes in Train Data')
ax2.set_xlabel('Unique values of the Number of DiagnosisCodes in CV Data')

ax1.set_title("Distribution of the number of DiagnosisCodes in Train Data")
ax2.set_title("Distribution of the number of DiagnosisCodes in CV Data")
plt.subplots_adjust(wspace=0.75)
plt.show()

```



4.15 Observations

1. We see that percentage of the observations belonging to the number of the diagnosis codes keeps reducing.
2. This is in agreement with the general phenomenon that more the number of diagnosis codes more complex the ailment of the patient.
3. Patients with severe ailments are usually inpatients and the number of cases in which the ailment is severe is also low.

4.15.1 Dropping the 10 Claim Diagnosis codes variables

```

[ ]: train_fin.drop(clm_diag,axis=1,inplace= True)
     cv_fin.drop(clm_diag,axis=1,inplace= True)

```

4.15.2 Checking the number of NA values in the Outpatient Dataset just to validate our observations and the feature engineering approach

```
[ ]: print("The NA percentage in the Admission Date variable in Inpatient_␣
      ↳Data", (train_inpat['AdmissionDt'].isna().sum()/
      ↳len(train_inpat['AdmissionDt']))*100)
print("The NA percentage in the Discharge Date variable in Inpatient_␣
      ↳Data", (train_inpat['DischargeDt'].isna().sum()/
      ↳len(train_inpat['DischargeDt']))*100)
```

The NA percentage in the Admission Date variable in Inpatient Data 0.0

The NA percentage in the Discharge Date variable in Inpatient Data 0.0

4.16 Observations

Although we see that Admission Date and the Discharge Date have an NA percentage of 92.5, from the above we see that all the NA values have been added by Outpatient and Beneficiary datasets.

It needs to be noted that AdmissionDate and DischargeDate columns are bound to be missing in the Inpatient Datasets and the Beneficiary Datasets.

4.17 Feature Engineering

I have created a new feature titled "hospital_days" which is taken as a difference between the Discharge Date and the Admission Date features.

I have imputed all the missing values in this feature with Zeros.

I have categorized the "HospitalDays" feature as keeping it a floating point value will introduce too many features and affect the overall distribution of the variable

```
[ ]: train_fin["HospitalDays"] = pd.to_datetime(train_fin['DischargeDt']) - pd.
      ↳to_datetime(train_fin['AdmissionDt'])
train_fin["HospitalDays"] = train_fin["HospitalDays"].dt.days

[ ]: cv_fin["HospitalDays"] = pd.to_datetime(cv_fin['DischargeDt']) - pd.
      ↳to_datetime(cv_fin['AdmissionDt'])
cv_fin["HospitalDays"] = cv_fin["HospitalDays"].dt.days
```

4.17.1 Looking at the distribution of the HospitalDays variable prior to the imputation of the NA values with 0

```
[ ]: train_fin["HospitalDays"].describe()
```

```
[ ]: count    32529.00000
      mean      5.67303
      std      5.65132
```

```
min          0.00000
25%          2.00000
50%          4.00000
75%          7.00000
max          35.00000
Name: HospitalDays, dtype: float64
```

```
[ ]: fig= plt.figure(figsize=(12,6))
gs= GridSpec(1,2,figure=fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

sns.boxplot(ax= ax1,y=train_fin['HospitalDays'],palette='crest')
sns.boxplot(ax= ax2,y=cv_fin['HospitalDays'],palette='crest')

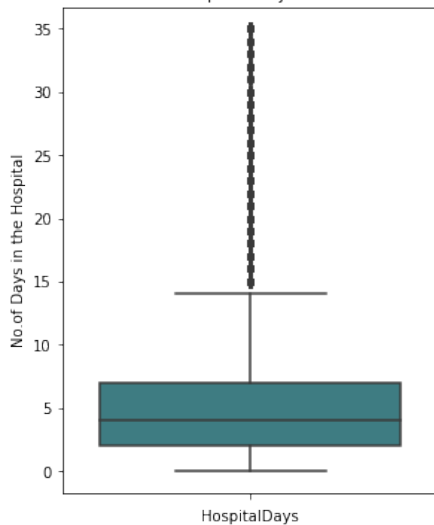
ax1.set_xlabel("HospitalDays")
ax2.set_xlabel("HospitalDays")

ax1.set_ylabel("No.of Days in the Hospital")
ax2.set_ylabel("No.of Days in the Hospital")

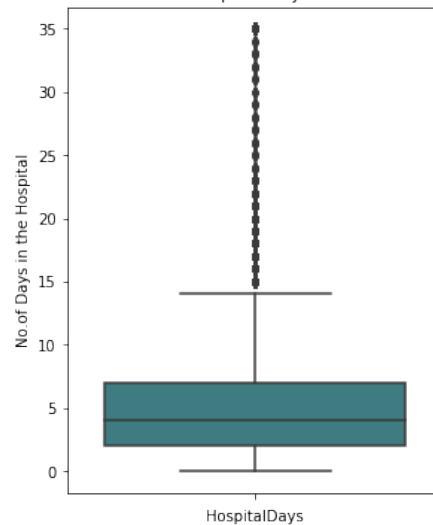
ax1.set_title("Distribution of the Hospital Days variable in Train Data")
ax2.set_title("Distribution of the Hospital Days variable in CV Data")
plt.subplots_adjust(wspace=0.65)

plt.show()
```

Distribution of the Hospital Days variable in Train Data



Distribution of the Hospital Days variable in CV Data



4.18 Observations

1. We see that in both the train and the cv datasets the median or the 50th percentile value is close to 5 while the 25th and the 75th percentile values are close to 3 and 7 respectively

```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n1.pkl','wb') as tr_df:
      pickle.dump(train_fin,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n1.pkl','wb') as cv_df:
      pickle.dump(cv_fin,cv_df)

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n1.pkl','rb') as tr_df:
      train_fin1= pickle.load(tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n1.pkl','rb') as cv_df:
      cv_fin1= pickle.load(cv_df)

[ ]: print(train_fin1['HospitalDays'].isna().sum())
      print(cv_fin1['HospitalDays'].isna().sum())
```

414039

103698

4.19 Imputing the NA values in the HospitalDates variable with 0 values

```
[ ]: train_fin1["HospitalDays"]=train_fin1["HospitalDays"].fillna(0)
      cv_fin1["HospitalDays"]=cv_fin1["HospitalDays"].fillna(0)

[ ]: fig= plt.figure(figsize=(12,6))
      gs= GridSpec(1,2,figure=fig)

      ax1= fig.add_subplot(gs[0,0])
      ax2= fig.add_subplot(gs[0,1])

      sns.boxplot(ax= ax1,y=train_fin1['HospitalDays'],palette='crest')
      sns.boxplot(ax= ax2,y=cv_fin1['HospitalDays'],palette='crest')

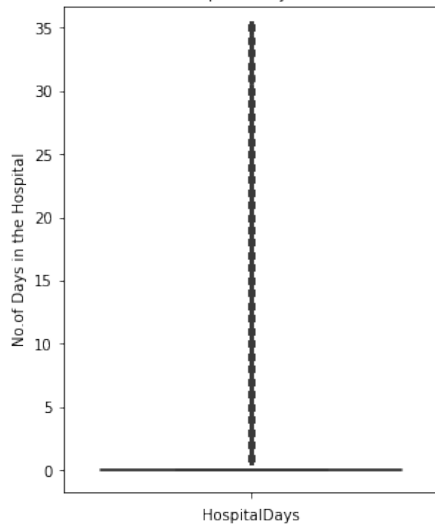
      ax1.set_xlabel("HospitalDays")
      ax2.set_xlabel("HospitalDays")

      ax1.set_ylabel("No.of Days in the Hospital")
      ax2.set_ylabel("No.of Days in the Hospital")

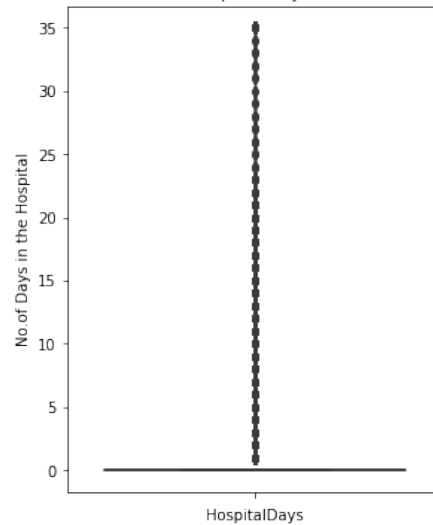
      ax1.set_title("Distribution of the Hospital Days variable in Train Data")
      ax2.set_title("Distribution of the Hospital Days variable in CV Data")
      plt.subplots_adjust(wspace=0.65)

      plt.show()
```


Distribution of the Hospital Days variable in Train Data



Distribution of the Hospital Days variable in CV Data



4.20 Observations

From the above two Box plots it is quite evident that the imputation of the NA values with 0 has drastically impacted the overall distribution of the variable in both the Train and CV datasets

4.20.1 Hence Categorizing the variable as per weeks as it introduces ordinality in the feature as well as separates out the inpatient and the outpatient data

```
[ ]: for i in tqdm(range(len(train_fin1['HospitalDays']))):
    if train_fin1['HospitalDays'][i]==0.0:
        train_fin1["HospitalDays"][i]= 0
    if train_fin1["HospitalDays"][i]>0 and train_fin1["HospitalDays"][i]<=7:
        train_fin1["HospitalDays"][i]= 1
    elif train_fin1["HospitalDays"][i]>7 and train_fin1["HospitalDays"][i]<=14:
        train_fin1["HospitalDays"][i]= 2
    elif train_fin1["HospitalDays"][i]>14 and train_fin1["HospitalDays"][i]<=21:
        train_fin1["HospitalDays"][i]= 3
    elif train_fin1["HospitalDays"][i]>21 and train_fin1["HospitalDays"][i]<=28:
        train_fin1["HospitalDays"][i]= 4
    elif train_fin1["HospitalDays"][i]>28:
        train_fin1["HospitalDays"][i]= 5
```

100%|| 446568/446568 [00:53<00:00, 8359.45it/s]

```
[ ]: for i in tqdm(range(len(cv_fin1['HospitalDays']))):
    if cv_fin1['HospitalDays'][i]==0.0:
        cv_fin1["HospitalDays"][i]= 0
    if cv_fin1["HospitalDays"][i]>0 and cv_fin1["HospitalDays"][i]<=7:
```

```

        cv_fin1["HospitalDays"][i]= 1
    elif cv_fin1["HospitalDays"][i]>7 and cv_fin1["HospitalDays"][i]<=14:
        cv_fin1["HospitalDays"][i]= 2
    elif cv_fin1["HospitalDays"][i]>14 and cv_fin1["HospitalDays"][i]<=21:
        cv_fin1["HospitalDays"][i]= 3
    elif cv_fin1["HospitalDays"][i]>21 and cv_fin1["HospitalDays"][i]<=28:
        cv_fin1["HospitalDays"][i]= 4
    elif cv_fin1["HospitalDays"][i]>28:
        cv_fin1["HospitalDays"][i]= 5

```

100%|| 111643/111643 [00:13<00:00, 8430.96it/s]

```

[ ]: train_fin1= train_fin1.rename(columns={'HospitalDays':'HospitalWeeks'})
     cv_fin1= cv_fin1.rename(columns={'HospitalDays':'HospitalWeeks'})

[ ]: fig=plt.figure(figsize=(14,6))
     gs= GridSpec(1,2,figure=fig)

     ax1= fig.add_subplot(gs[0,0])
     ax2= fig.add_subplot(gs[0,1])

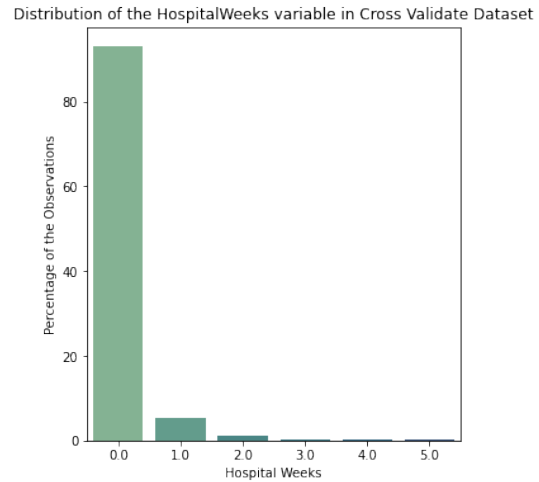
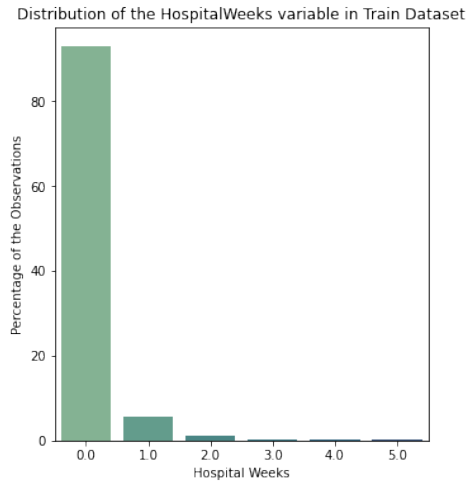
     sns.barplot(ax=ax1, x= np.unique(train_fin1["HospitalWeeks"]),y= np.
         ↳round((train_fin1["HospitalWeeks"].value_counts()/
         ↳len(train_fin1["HospitalWeeks"]))*100,2),palette='crest')
     sns.barplot(ax=ax2, x= np.unique(cv_fin1["HospitalWeeks"]),y= np.
         ↳round((cv_fin1["HospitalWeeks"].value_counts()/
         ↳len(cv_fin1["HospitalWeeks"]))*100,2),palette='crest')

     ax1.set_xlabel("Hospital Weeks")
     ax2.set_xlabel("Hospital Weeks")

     ax1.set_ylabel("Percentage of the Observations")
     ax2.set_ylabel("Percentage of the Observations")

     ax1.set_title("Distribution of the HospitalWeeks variable in Train Dataset")
     ax2.set_title("Distribution of the HospitalWeeks variable in Cross Validate_
         ↳Dataset")
     plt.subplots_adjust(wspace=0.65)
     plt.show()

```



4.21 Observations

1. As mentioned above 0 would be the highest as they are the imputed observations from the Inpatient and the Outpatient Datasets
2. Other than 0, we see that the maximum days spent in the hospital is less than or equal to 1 week and the number of observations keep decreasing with more weeks
3. This seems to be the general trend as there are fewer chronic illness cases in a hospital and most of the Inpatients are predominantly admitted for shorter duration of time

4.22 Dropping the Admission Date and the Discharge Date columns from the dataset

```
[ ]: train_fin1.drop(['AdmissionDt','DischargeDt'], axis=1, inplace=True)
cv_fin1.drop(['AdmissionDt','DischargeDt'], axis=1, inplace=True)

[ ]: na_col_tr=[]
na_perc_tr= np.round(((train_fin1.isna().sum())/train_fin1.shape[0])*100,2)
na_perc_df_tr= na_perc_tr.to_frame()
na_perc_df_tr.reset_index(inplace= True)
na_perc_df_tr.columns= ["col_name","na_percentage"]
for i in range(na_perc_df_tr.shape[0]):
    if na_perc_df_tr.iloc[i,1] == 0:
        na_col_tr.append(i)

print(na_col_tr)
```

```
[0, 1, 2, 3, 4, 5, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38]
```

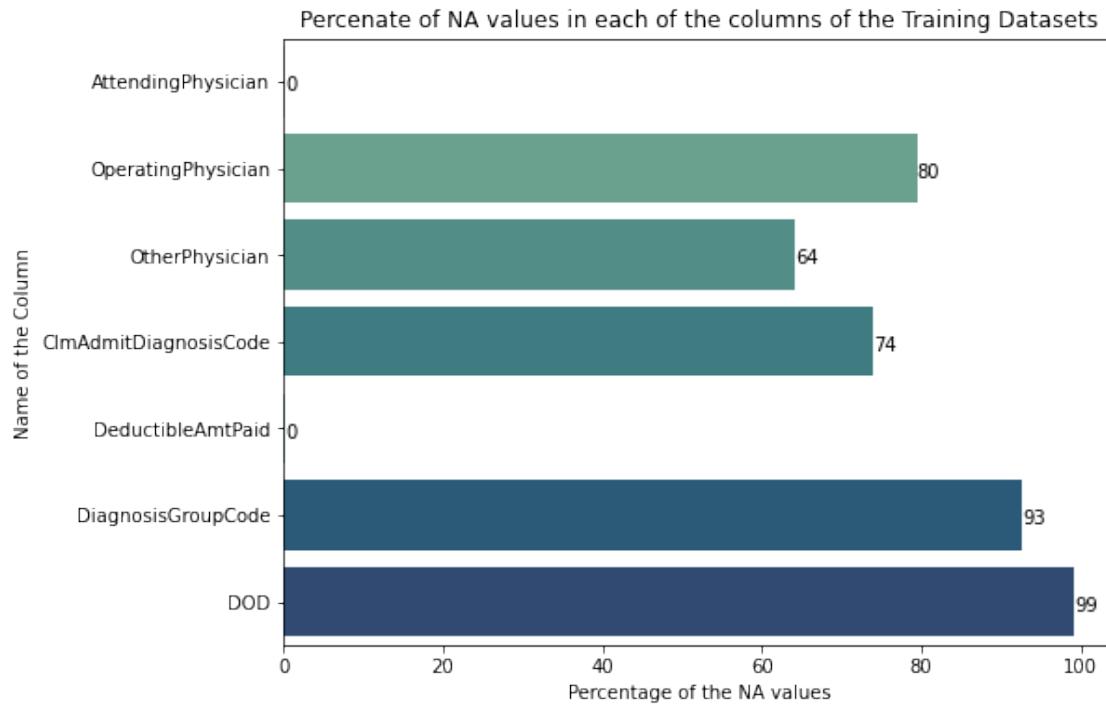
```
[ ]: na_perc_df_tr.drop(index=na_col_tr,inplace=True)
na_perc_df_tr.reset_index(drop=True,inplace=True)
print(na_perc_df_tr)
```

	col_name	na_percentage
0	AttendingPhysician	0.27
1	OperatingPhysician	79.50
2	OtherPhysician	64.24
3	ClmAdmitDiagnosisCode	73.88
4	DeductibleAmtPaid	0.16
5	DiagnosisGroupCode	92.72
6	DOD	99.26

```
[ ]: plt.figure(figsize=(8,6))
ax= sns.barplot(y= na_perc_df_tr['col_name'],x=
    ↳na_perc_df_tr['na_percentage'],palette='crest')

plt.ylabel("Name of the Column")
plt.xlabel("Percentage of the NA values")
plt.title("Percentage of NA values in each of the columns of the Training
    ↳Datasets")

#Source: https://medium.com/@dey.mallika/
    ↳transform-your-graphs-with-seaborn-ea4fa8e606a6
initialx=0
for p in ax.patches:
    ax.text(p.get_width(),initialx+p.get_height()/8,'{:1.0f}'.format(p.
    ↳get_width()))
    initialx+=1
plt.show()
```



4.23 Looking at the distribution of the NA values in the Operating physician and Other physician columns of the Inpatients and the Outpatients datasets

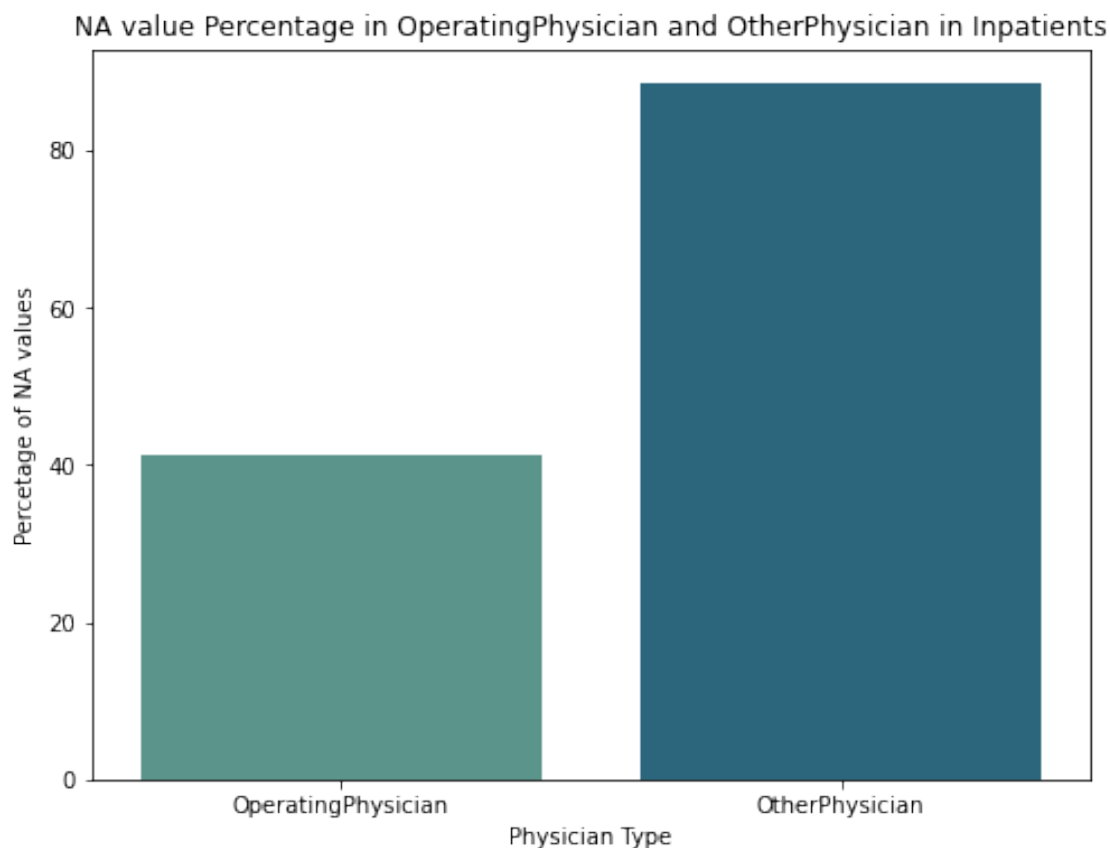
```
[ ]: in_opr_p=np.round((train_inpat['OperatingPhysician'].isna().sum()/
    →len(train_inpat['OperatingPhysician']))*100,2)
in_ot_p= np.round((train_inpat['OtherPhysician'].isna().sum()/
    →len(train_inpat['OtherPhysician']))*100,2)

print("NA percent in OperatingPhysician col in Inpatient Data",in_opr_p)
print("NA percent in OtherPhysician col in Inpatient Data",in_ot_p)
print("*"*100)

plt.figure(figsize=(8,6))
sns.barplot(x=["OperatingPhysician","OtherPhysician"],y=[in_opr_p,in_ot_p],u
    →palette='crest')
plt.xlabel("Physician Type")
plt.ylabel("Percentage of NA values")
plt.title("NA value Percentage in OperatingPhysician and OtherPhysician in_
    →Inpatients")
#plt.grid()
plt.show()
```

NA percent in OperatingPhysician col in Inpatient Data 41.12

NA percent in OtherPhysician col in Inpatient Data 88.41



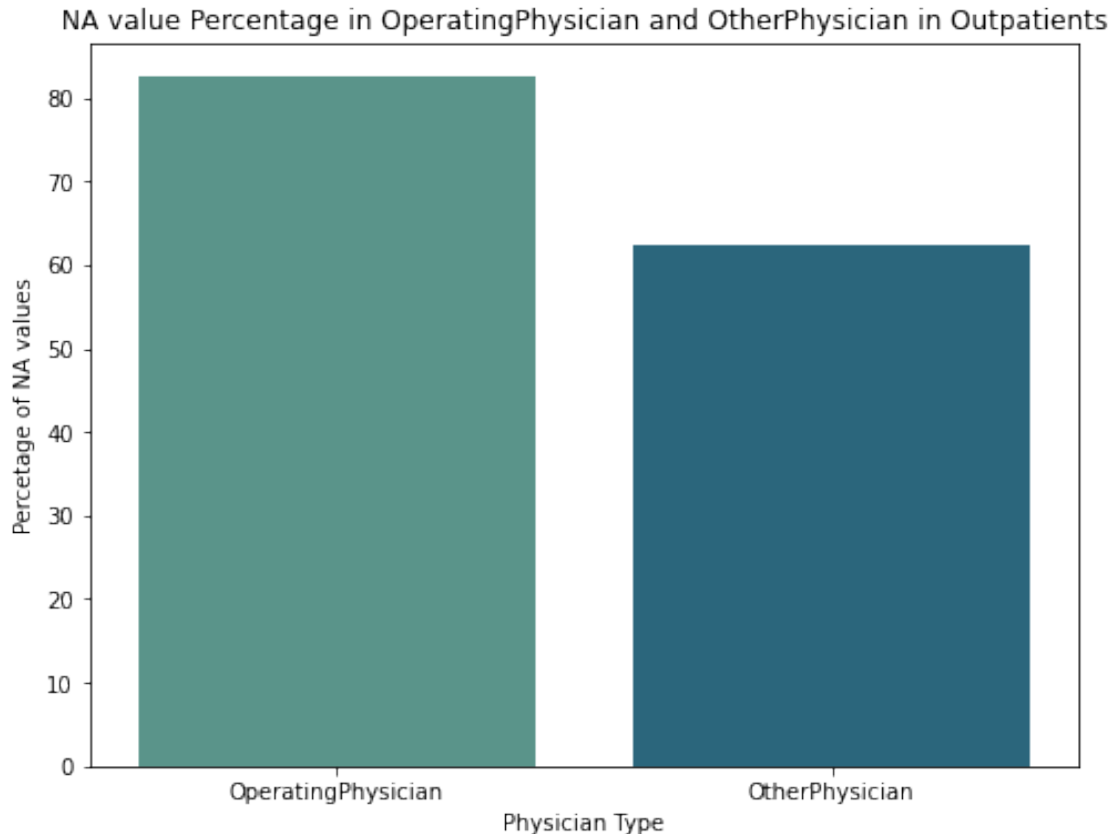
```
[ ]: out_opr_p=np.round((train_outpat['OperatingPhysician'].isna().sum()/
    →len(train_outpat['OperatingPhysician']))*100,2)
out_ot_p= np.round((train_outpat['OtherPhysician'].isna().sum()/
    →len(train_outpat['OtherPhysician']))*100,2)

print("NA percent in OperatingPhysician col in Inpatient Data",out_opr_p)
print("NA percent in OtherPhysician col in Inpatient Data",out_ot_p)

print("*"*100)
plt.figure(figsize=(8,6))
sns.barplot(x=["OperatingPhysician","OtherPhysician"],y=[out_opr_p,out_ot_p],
    →palette='crest')
plt.xlabel("Physician Type")
plt.ylabel("Perctage of NA values")
plt.title("NA value Percentage in OperatingPhysician and OtherPhysician in
    →Outpatients")
#plt.grid()
plt.show()
```

NA percent in OperatingPhysician col in Inpatient Data 82.5

NA percent in OtherPhysician col in Inpatient Data 62.33



4.24 Observations:

1. The above graphs are inline with the reality or practical situation. NA values in Operating Physician and Other Physician datasets doesn't mean that the data is missing.
2. As per my secondary research, Operating Physicians are involved in cases where a surgery or other complications are involved
3. OtherPhysicians are involved in cases where the patient has co-morbidities

4.25 Observations on Inpatient Dataset:

1. We have 41% NA values in Operating Physicians column. This means the 59% (100%-41%) of the Inpatients likely needed a surgery or had other complication whereas 41% didn't have any complications or didn't need surgery

2. We have 88% NA values in Other Physicians column. This means the 12% (100%-88%) of the Inpatients had co-morbidities where as 12% didnt have co-morbidities

4.26 Observations on Outpatient Dataset:

1. We have 82% NA values in Operating Physicians column. Outpatients do not usually go through surgeries or other complicated procedures in a day

2. We have 62% NA values in Other Physicians column. Outpatients do not usually consult other physicians very often.

4.27 Imputing the Attending Physician with Mode or MostFrequent strategy using SimpleImpute

```
[ ]: cat_imp= SimpleImputer(missing_values= np.nan, strategy= 'most_frequent')
train_fin1["AttendingPhysician"]= cat_imp.
    ↳fit_transform(train_fin1['AttendingPhysician'].values.reshape(-1,1))[:,0]

[ ]: train_fin1["AttendingPhysician"].isna().sum()

[ ]: 0
```

4.28 Feature Engineering

A new column has been created to capture the nature of illness of the patient. This column will be categorical and will have the below categories:

1.Simple

2.Operating

3.comorbid

4.Operating&comorbid

```
[ ]: train_fin1["IllnessNature"]= np.zeros(len(train_fin1["AttendingPhysician"]))
cv_fin1["IllnessNature"]= np.zeros(len(cv_fin1["AttendingPhysician"]))

[ ]: for i in tqdm(range(len(train_fin1["AttendingPhysician"]))):
    if pd.isnull(train_fin1["OperatingPhysician"][i])==True and pd.
    ↳isnull(train_fin1["OtherPhysician"][i])==True:
        train_fin1["IllnessNature"][i]= "simple"
    elif pd.isnull(train_fin1["OperatingPhysician"][i])==True and pd.
    ↳isnull(train_fin1["OtherPhysician"][i])==False:
        train_fin1["IllnessNature"][i]= "operating"
    elif pd.isnull(train_fin1["OperatingPhysician"][i])==False and pd.
    ↳isnull(train_fin1["OtherPhysician"][i])==True:
        train_fin1["IllnessNature"][i]= "comorbid"
```



```

        elif pd.isnull(train_fin1["OperatingPhysician"][i])==False and pd.
→isnull(train_fin1["OtherPhysician"][i])==False:
            train_fin1["IllnessNature"][i]= "operating&comorbid"

```

100%|| 446568/446568 [28:45<00:00, 258.81it/s]

```

[ ]: for i in tqdm(range(len(cv_fin1["AttendingPhysician"]))):
        if pd.isnull(cv_fin1["OperatingPhysician"][i])==True and pd.
→isnull(cv_fin1["OtherPhysician"][i])==True:
            cv_fin1["IllnessNature"][i]= "simple"
        elif pd.isnull(cv_fin1["OperatingPhysician"][i])==True and pd.
→isnull(cv_fin1["OtherPhysician"][i])==False:
            cv_fin1["IllnessNature"][i]= "operating"
        elif pd.isnull(cv_fin1["OperatingPhysician"][i])==False and pd.
→isnull(cv_fin1["OtherPhysician"][i])==True:
            cv_fin1["IllnessNature"][i]= "comorbid"
        elif pd.isnull(cv_fin1["OperatingPhysician"][i])==False and pd.
→isnull(cv_fin1["OtherPhysician"][i])==False:
            cv_fin1["IllnessNature"][i]= "operating&comorbid"

```

100%|| 111643/111643 [01:11<00:00, 1567.18it/s]

```

[ ]: print(np.unique(train_fin1["IllnessNature"]))
      print(np.unique(cv_fin1["IllnessNature"]))

```

```

['comorbid' 'operating' 'operating&comorbid' 'simple']
['comorbid' 'operating' 'operating&comorbid' 'simple']

```

```

[ ]: fig=plt.figure(figsize=(14,6))
      gs= GridSpec(1,2,figure=fig)

      ax1= fig.add_subplot(gs[0,0])
      ax2= fig.add_subplot(gs[0,1])

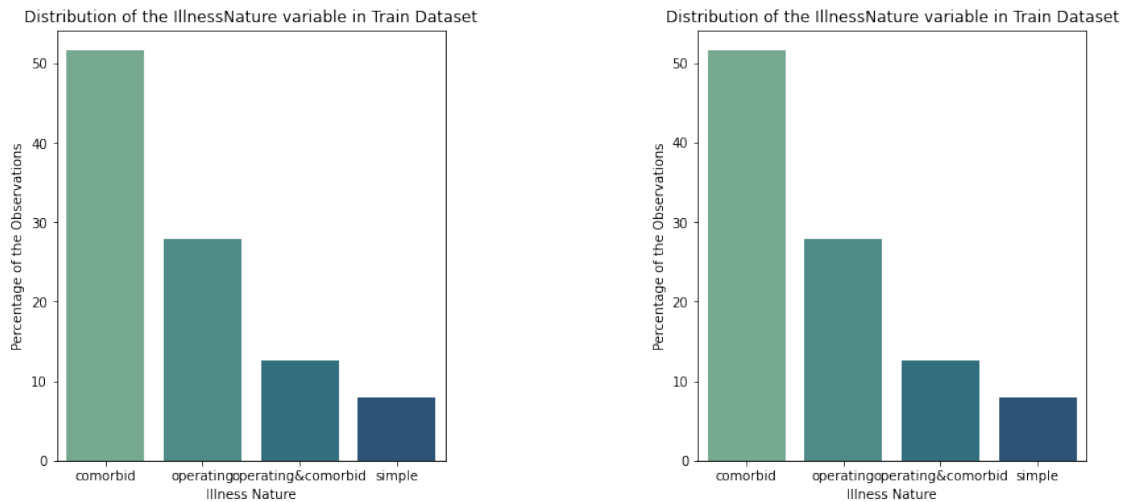
      sns.barplot(ax=ax1, x= np.unique(train_fin1["IllnessNature"]),y= np.
→round((train_fin1["IllnessNature"].value_counts()/
→len(train_fin1["IllnessNature"]))*100,2),palette='crest')
      sns.barplot(ax=ax2, x= np.unique(cv_fin1["IllnessNature"]),y= np.
→round((cv_fin1["IllnessNature"].value_counts()/
→len(cv_fin1["IllnessNature"]))*100,2),palette='crest')

      ax1.set_xlabel("Illness Nature")
      ax2.set_xlabel("Illness Nature")

      ax1.set_ylabel("Percentage of the Observations")
      ax2.set_ylabel("Percentage of the Observations")

```

```
ax1.set_title("Distribution of the IllnessNature variable in Train Dataset")
ax2.set_title("Distribution of the IllnessNature variable in Train Dataset")
plt.subplots_adjust(wspace=0.65)
plt.show()
```



4.29 Observations

1. Comorbid condition or the illness nature has the highest percentage of the observations in the overall dataset followed by the Operating illness nature.
2. Simple illness nature has the lowest percentage of the observations of the total dataset.

```
[ ]: train_fin1.drop(['OperatingPhysician','OtherPhysician'], axis=1,inplace=True)
cv_fin1.drop(['OperatingPhysician','OtherPhysician'], axis=1,inplace=True)
```

```
[ ]: train_fin2.head()
```

```
[ ]: InscClaimAmtReimbursed  DeductibleAmtPaid  ...  CADC_Yes  CADC_No
0                90                0.0  ...  0.465483  0.534517
1               5000             1068.0  ...  0.482759  0.517241
2                400                0.0  ...  0.456727  0.543273
3                 30                0.0  ...  0.456727  0.543273
4               2000             1068.0  ...  0.372093  0.627907
```

[5 rows x 43 columns]

4.30 Feature Engineering

Extracting the number of days from the claim start date and the claim end dates by taking a difference

Converting the number of days during which the claim was in process into number of weeks

```
[ ]: train_fin1["ClaimDays"] = (pd.to_datetime(train_fin1['ClaimEndDt']) - pd.
    ↳to_datetime(train_fin1['ClaimStartDt'])).dt.days
cv_fin1["ClaimDays"] = (pd.to_datetime(cv_fin1['ClaimEndDt']) - pd.
    ↳to_datetime(cv_fin1['ClaimStartDt'])).dt.days

[ ]: train_fin1.drop(['ClaimEndDt', 'ClaimStartDt'], axis=1, inplace=True)
cv_fin1.drop(['ClaimEndDt', 'ClaimStartDt'], axis=1, inplace=True)

[ ]: fig=plt.figure(figsize=(16,10))
gs= GridSpec(2,2,figure=fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,0])
ax4= fig.add_subplot(gs[1,1])

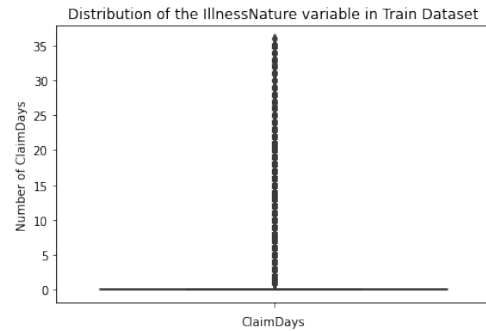
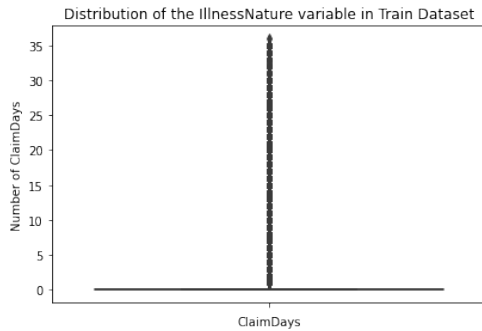
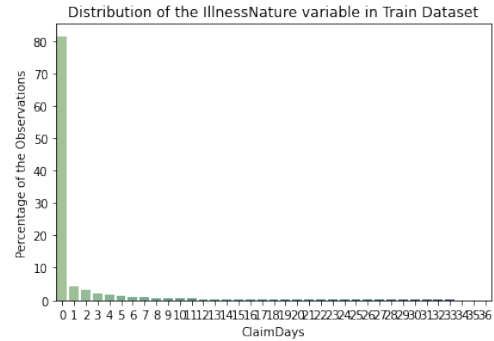
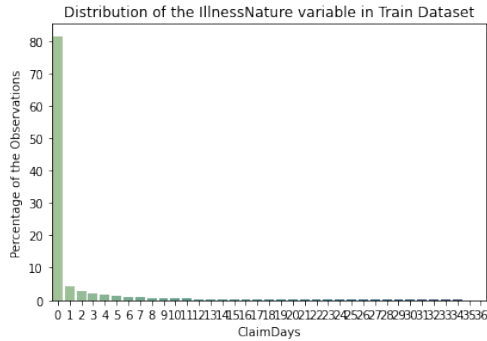
sns.barplot(ax=ax1, x= np.unique(train_fin1["ClaimDays"]), y= np.
    ↳round((train_fin1["ClaimDays"].value_counts()/
    ↳len(train_fin1["ClaimDays"]))*100,2), palette='crest')
sns.barplot(ax=ax2, x= np.unique(cv_fin1["ClaimDays"]), y= np.
    ↳round((cv_fin1["ClaimDays"].value_counts()/
    ↳len(cv_fin1["ClaimDays"]))*100,2), palette='crest')
sns.boxplot(ax=ax3, y= train_fin1["ClaimDays"], palette='crest')
sns.boxplot(ax=ax4, y= cv_fin1["ClaimDays"], palette='crest')

ax1.set_xlabel("ClaimDays")
ax2.set_xlabel("ClaimDays")
ax3.set_xlabel("ClaimDays")
ax4.set_xlabel("ClaimDays")

ax1.set_ylabel("Percentage of the Observations")
ax2.set_ylabel("Percentage of the Observations")
ax3.set_ylabel("Number of ClaimDays")
ax4.set_ylabel("Number of ClaimDays")

ax1.set_title("Distribution of the IllnessNature variable in Train Dataset")
ax2.set_title("Distribution of the IllnessNature variable in Train Dataset")
ax3.set_title("Distribution of the IllnessNature variable in Train Dataset")
ax4.set_title("Distribution of the IllnessNature variable in Train Dataset")

plt.subplots_adjust(wspace=0.45)
plt.subplots_adjust(hspace=0.35)
plt.show()
```



4.31 Observations

1. We see that more than 80% of the claim days are zero and close to 95% of the claim days are less than 7 days.
2. Keeping the variable datatype as floating point value will skew the mean and other distribution related parameters

3. Converting the variable into a Categorical Variable(Ordinal)

```
[ ]: for i in tqdm(range(len(train_fin1['ClaimDays']))):
    if train_fin1['ClaimDays'][i]==0:
        train_fin1['ClaimDays'][i]=0
    elif train_fin1['ClaimDays'][i]>0 and train_fin1['ClaimDays'][i]<=7:
        train_fin1['ClaimDays'][i]= 1
    elif train_fin1['ClaimDays'][i]>7 and train_fin1['ClaimDays'][i]<=14:
        train_fin1['ClaimDays'][i]= 2
    elif train_fin1['ClaimDays'][i]> 14 and train_fin1['ClaimDays'][i]<=21:
        train_fin1['ClaimDays'][i]= 3
    elif train_fin1['ClaimDays'][i]>21 and train_fin1['ClaimDays'][i]<=28:
        train_fin1['ClaimDays'][i]= 4
    elif train_fin1['ClaimDays'][i]>28 and train_fin1['ClaimDays'][i]<=35:
        train_fin1['ClaimDays'][i]=5
    elif train_fin1['ClaimDays'][i]>35:
        train_fin1['ClaimDays'][i]=6
```

100%|| 446568/446568 [00:41<00:00, 10887.56it/s]

```
[ ]: for i in tqdm(range(len(cv_fin1['ClaimDays']))):
    if cv_fin1['ClaimDays'][i]==0:
        cv_fin1['ClaimDays'][i]=0
    elif cv_fin1['ClaimDays'][i]>0 and cv_fin1['ClaimDays'][i]<=7:
        cv_fin1['ClaimDays'][i]= 1
    elif cv_fin1['ClaimDays'][i]>7 and cv_fin1['ClaimDays'][i]<=14:
        cv_fin1['ClaimDays'][i]= 2
    elif cv_fin1['ClaimDays'][i]> 14 and cv_fin1['ClaimDays'][i]<=21:
        cv_fin1['ClaimDays'][i]= 3
    elif cv_fin1['ClaimDays'][i]>21 and cv_fin1['ClaimDays'][i]<=28:
        cv_fin1['ClaimDays'][i]= 4
    elif cv_fin1['ClaimDays'][i]>28 and cv_fin1['ClaimDays'][i]<=35:
        cv_fin1['ClaimDays'][i]=5
    elif cv_fin1['ClaimDays'][i]>35:
        cv_fin1['ClaimDays'][i]=6
```

100%|| 111643/111643 [00:10<00:00, 10793.91it/s]

```
[ ]: train_fin1= train_fin1.rename(columns={'ClaimDays':'ClaimWeeks'})
    cv_fin1= cv_fin1.rename(columns={'ClaimDays':'ClaimWeeks'})

[ ]: fig=plt.figure(figsize=(12,6))
    gs= GridSpec(1,2,figure=fig)

    ax1= fig.add_subplot(gs[0,0])
    ax2= fig.add_subplot(gs[0,1])

    sns.barplot(ax=ax1, x= np.unique(train_fin1["ClaimWeeks"]),y= np.
        ↳round((train_fin1["ClaimWeeks"].value_counts()/
        ↳len(train_fin1["ClaimWeeks"]))*100,2),palette='crest')
    sns.barplot(ax=ax2, x= np.unique(cv_fin1["ClaimWeeks"]),y= np.
        ↳round((cv_fin1["ClaimWeeks"].value_counts()/
        ↳len(cv_fin1["ClaimWeeks"]))*100,2),palette='crest')

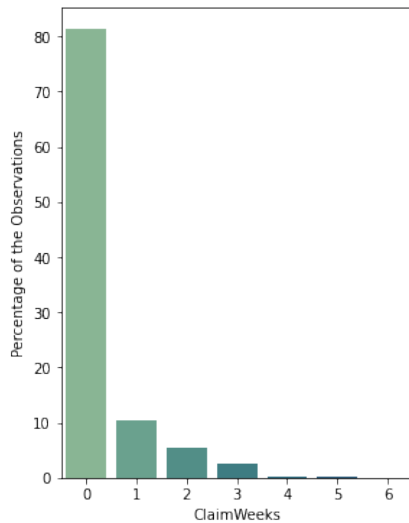
    ax1.set_xlabel("ClaimWeeks")
    ax2.set_xlabel("ClaimWeeks")

    ax1.set_ylabel("Percentage of the Observations")
    ax2.set_ylabel("Percentage of the Observations")

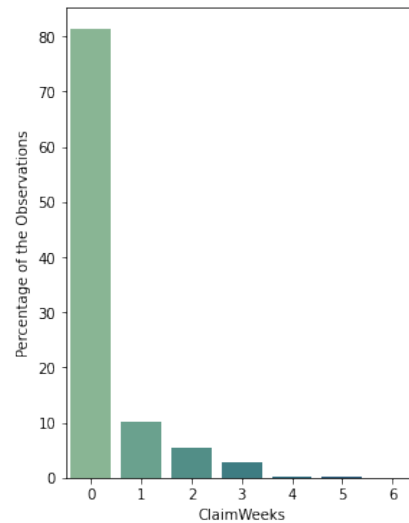
    ax1.set_title("Distribution of the ClaimWeeks variable in Train Dataset")
    ax2.set_title("Distribution of the ClaimWeeks variable in Train Dataset")

    plt.subplots_adjust(wspace=0.75)
    plt.show()
```

Distribution of the ClaimWeeks variable in Train Dataset



Distribution of the ClaimWeeks variable in Train Dataset



4.32 Observations

1. From the above graph we see that about 80% of the claims get settled instantly where the claimweeks are less than 0 weeks
2. We observe that as a total about 15% claims are settled between 0 to 3 weeks.
3. The reason behind the skewness towards 0 could be that the majority of the dataset belongs to the outpatient data and average claims settlement for the inpatient data is close to 1 week.

```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n2.pkl','wb') as tr_df:
      pickle.dump(train_fin1,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n2.pkl','wb') as cv_df:
      pickle.dump(cv_fin1,cv_df)
```

```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n2.pkl','rb') as tr_df:
      train_fin2= pickle.load(tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n2.pkl','rb') as cv_df:
      cv_fin2= pickle.load(cv_df)
```

4.32.1 Looking at the NA value distribtution of the DiagnosisGroupCode variable

```
[ ]: print("Percentage of NA values in Inpateint Data:␣
      →", (train_inpat['DiagnosisGroupCode'].isna().sum()/
      →len(train_inpat['DiagnosisGroupCode']))*100)
```

```
print("Percentage of NA values in Merged Data: ",np.
      ↳round((train_fin2['DiagnosisGroupCode'].isna().sum()/
      ↳len(train_fin2['DiagnosisGroupCode']))*100))
print("Ratio of Outpatient data to Merged Dataset",np.round((train_outpat.
      ↳shape[0]/(train_fin2.shape[0]+cv_fin2.shape[0]))*100))
```

Percentage of NA values in Inpatient Data: 0.0
 Percentage of NA values in Merged Data: 93.0
 Ratio of Outpatient data to Merged Dataset 93.0

4.33 Observations,

1. From the above we see that there are no NA values in the Inpatient data and the variable "DiagnosisGroupCode" is missing in the Outpatient Dataset.

2. https://hmsa.com/portal/provider/zav_pel.fh.DIA.650.htm: clearly states that a Diagnosis-GroupCode is just for the Inpatient Data and not for the outpatient data. Hence NA were introduced during the merger of the Dataset

3 Hence filling the NA as new category with a value of 0

```
[ ]: train_fin2['DiagnosisGroupCode']= train_fin2['DiagnosisGroupCode'].fillna(0)
      cv_fin2['DiagnosisGroupCode']= cv_fin2['DiagnosisGroupCode'].fillna(0)

[ ]: train_fin2['DiagnosisGroupCode']= train_fin2['DiagnosisGroupCode'].astype('str')
      train_fin2['DiagnosisGroupCode']= train_fin2['DiagnosisGroupCode'].astype('str')

      train_fin2['DiagnosisGroupCode'].describe()
      #sns.distplot(train_fin3['DiagnosisGroupCode'].value_counts())
```

```
[ ]: count      446568
      unique       735
      top          0
      freq      414039
      Name: DiagnosisGroupCode, dtype: object
```

```
[ ]: print("Percentage of NA values in Inpatient Data: ")
      ↳", (train_inpat['ClmAdmitDiagnosisCode'].isna().sum()/
      ↳len(train_inpat['ClmAdmitDiagnosisCode']))*100)
print("Percentage of NA values in Outpatient Data: ",np.
      ↳round((train_outpat['ClmAdmitDiagnosisCode'].isna().sum()/
      ↳len(train_outpat['ClmAdmitDiagnosisCode']))*100))
```

Percentage of NA values in Inpatient Data: 0.0
 Percentage of NA values in Outpatient Data: 80.0

4.34 Observations

1. Majority of the categorical columns have a more than 50-100 categories in each of the columns

2. Adopting a one-hot encoding to convert the categorical columns into numerical columns could lead to creation of a lot of columns leading to Curse of Dimensionality
3. Using response coding in order to convert categorical columns to numerical ones.

Response Coding: Calculating the probabilities of each of the categories in a column. Probability is calculated as follows

$P(x=c1/y='yes')$ which is Probability of category in column X, given the Y variable belongs to class 'Yes' and class 'No'

$P(x=c1/y='yes') = (\text{Number of Occurrences of C1 where Y belongs to 'yes' class}) / (\text{Number of Occurrences of where Y='yes' + Number of Occurrences of where Y='No'})$

4.35 Feature Engineering

Response Coding of the DiagnosisGroupCode, State,County, BeneID,ClaimID,Provider,AttendingPhysician

```
[ ]: train_fin2['PotentialFraud'] = train_y
      cv_fin2['PotentialFraud'] = cv_y
```

```
[ ]: def response_coding(tr_data,cv_data,col,y):
      tr_yes_list=[]
      tr_no_list=[]
      cv_yes_list=[]
      cv_no_list=[]

      val_dict= dict(tr_data.groupby([col])[y].value_counts())

      for i in range(len(tr_data[col])):
          t_y= val_dict.get((tr_data[col][i], 'Yes'),0.1)
          t_n= val_dict.get((tr_data[col][i], 'No'),0.1)
          tr_yes_list.append(t_y/(t_y+t_n))
          tr_no_list.append(t_n/(t_y+t_n))

      for j in range(len(cv_data[col])):
          c_y= val_dict.get((cv_data[col][j], 'Yes'),0.1)
          c_n= val_dict.get((cv_data[col][j], 'No'),0.1)
          cv_yes_list.append(c_y/(c_y+c_n))
          cv_no_list.append(c_n/(c_y+c_n))

      return tr_yes_list,tr_no_list,cv_yes_list,cv_no_list,val_dict
```

```
[ ]: train_fin2['State_Yes']= np.zeros(len(train_fin2['State']))
      train_fin2['State_No']= np.zeros(len(train_fin2['State']))

      cv_fin2['State_Yes']= np.zeros(len(cv_fin2['State']))
      cv_fin2['State_No']= np.zeros(len(cv_fin2['State']))

      train_fin2['State_Yes'],train_fin2['State_No'],cv_fin2['State_Yes'],cv_fin2['State_No'],state_
      →response_coding(train_fin2,cv_fin2, 'State', 'PotentialFraud')
```



```
print(state_dict)
```

```
{(1, 'No'): 3609, (1, 'Yes'): 2997, (2, 'No'): 259, (2, 'Yes'): 224, (3, 'No'): 3249, (3, 'Yes'): 2705, (4, 'No'): 2581, (4, 'Yes'): 2209, (5, 'No'): 17879, (5, 'Yes'): 14860, (6, 'No'): 2585, (6, 'Yes'): 2058, (7, 'No'): 1960, (7, 'Yes'): 1651, (8, 'No'): 492, (8, 'Yes'): 466, (9, 'No'): 228, (9, 'Yes'): 187, (10, 'No'): 13365, (10, 'Yes'): 11648, (11, 'No'): 5981, (11, 'Yes'): 4974, (12, 'No'): 594, (12, 'Yes'): 457, (13, 'No'): 975, (13, 'Yes'): 812, (14, 'No'): 8426, (14, 'Yes'): 7133, (15, 'No'): 4869, (15, 'Yes'): 4208, (16, 'No'): 2580, (16, 'Yes'): 2180, (17, 'No'): 1676, (17, 'Yes'): 1489, (18, 'No'): 3605, (18, 'Yes'): 3004, (19, 'No'): 2834, (19, 'Yes'): 2275, (20, 'No'): 1329, (20, 'Yes'): 1133, (21, 'No'): 3907, (21, 'Yes'): 3307, (22, 'No'): 4651, (22, 'Yes'): 4071, (23, 'No'): 7479, (23, 'Yes'): 6199, (24, 'No'): 2722, (24, 'Yes'): 2198, (25, 'No'): 2331, (25, 'Yes'): 1925, (26, 'No'): 4482, (26, 'Yes'): 3829, (27, 'No'): 684, (27, 'Yes'): 600, (28, 'No'): 1453, (28, 'Yes'): 1240, (29, 'No'): 666, (29, 'Yes'): 603, (30, 'No'): 915, (30, 'Yes'): 796, (31, 'No'): 5547, (31, 'Yes'): 4683, (32, 'No'): 1240, (32, 'Yes'): 1004, (33, 'No'): 12242, (33, 'Yes'): 10113, (34, 'No'): 6524, (34, 'Yes'): 5545, (35, 'No'): 322, (35, 'Yes'): 268, (36, 'No'): 7387, (36, 'Yes'): 6241, (37, 'No'): 2441, (37, 'Yes'): 2045, (38, 'No'): 1927, (38, 'Yes'): 1564, (39, 'No'): 8355, (39, 'Yes'): 7179, (41, 'No'): 359, (41, 'Yes'): 336, (42, 'No'): 3605, (42, 'Yes'): 3106, (43, 'No'): 681, (43, 'Yes'): 594, (44, 'No'): 5007, (44, 'Yes'): 4200, (45, 'No'): 11761, (45, 'Yes'): 9921, (46, 'No'): 1213, (46, 'Yes'): 928, (47, 'No'): 642, (47, 'Yes'): 518, (49, 'No'): 5334, (49, 'Yes'): 4349, (50, 'No'): 4093, (50, 'Yes'): 3457, (51, 'No'): 1548, (51, 'Yes'): 1421, (52, 'No'): 3447, (52, 'Yes'): 2818, (53, 'No'): 386, (53, 'Yes'): 329, (54, 'No'): 1485, (54, 'Yes'): 1230}
```

```
[ ]: train_fin2['County_Yes']= np.zeros(len(train_fin2['County']))
train_fin2['County_No']= np.zeros(len(train_fin2['County']))

cv_fin2['County_Yes']= np.zeros(len(cv_fin2['County']))
cv_fin2['County_No']= np.zeros(len(cv_fin2['County']))

train_fin2['County_Yes'],train_fin2['County_No'],cv_fin2['County_Yes'],cv_fin2['County_No'],co
    ↳response_coding(train_fin2,cv_fin2,'County','PotentialFraud')
print(county_dict)
```

```
{(0, 'No'): 3893, (0, 'Yes'): 3432, (1, 'Yes'): 4, (1, 'No'): 3, (10, 'No'): 4847, (10, 'Yes'): 4059, (11, 'No'): 92, (11, 'Yes'): 79, (14, 'No'): 2, (14, 'Yes'): 1, (20, 'No'): 4444, (20, 'Yes'): 3622, (25, 'No'): 20, (25, 'Yes'): 15, (30, 'No'): 2132, (30, 'Yes'): 1878, (34, 'No'): 5, (34, 'Yes'): 5, (40, 'No'): 2549, (40, 'Yes'): 2291, (50, 'No'): 2724, (50, 'Yes'): 2368, (55, 'No'): 10, (55, 'Yes'): 9, (60, 'No'): 4231, (60, 'Yes'): 3533, (70, 'No'): 2335, (70, 'Yes'): 1999, (80, 'No'): 2055, (80, 'Yes'): 1701, (84, 'Yes'): 3, (84, 'No'): 1, (88, 'No'): 34, (88, 'Yes'): 20, (90, 'No'): 3845, (90, 'Yes'): 3261, (100, 'No'): 2352, (100, 'Yes'): 1970, (110, 'No'): 1281, (110, 'Yes'): 1081, (111,
```

'No'): 47, (111, 'Yes'): 37, (113, 'No'): 6, (113, 'Yes'): 6, (117, 'No'): 6,
 (117, 'Yes'): 3, (120, 'No'): 2558, (120, 'Yes'): 2129, (130, 'No'): 2873, (130,
 'Yes'): 2429, (131, 'Yes'): 7, (131, 'No'): 6, (140, 'No'): 1832, (140, 'Yes'):
 1586, (141, 'No'): 3093, (141, 'Yes'): 2592, (150, 'No'): 3340, (150, 'Yes'):
 2882, (160, 'No'): 3638, (160, 'Yes'): 3116, (161, 'Yes'): 8, (161, 'No'): 4,
 (170, 'No'): 2834, (170, 'Yes'): 2323, (180, 'No'): 1269, (180, 'Yes'): 1086,
 (190, 'No'): 1475, (190, 'Yes'): 1174, (191, 'No'): 37, (191, 'Yes'): 29, (194,
 'No'): 94, (194, 'Yes'): 74, (200, 'No'): 5539, (200, 'Yes'): 4715, (210, 'No'):
 1263, (210, 'Yes'): 1116, (211, 'No'): 6, (211, 'Yes'): 6, (212, 'No'): 11,
 (212, 'Yes'): 6, (213, 'Yes'): 15, (213, 'No'): 14, (220, 'No'): 1812, (220,
 'Yes'): 1494, (221, 'No'): 11, (221, 'Yes'): 6, (222, 'No'): 27, (222, 'Yes'):
 26, (223, 'No'): 7, (223, 'Yes'): 4, (224, 'Yes'): 5, (224, 'No'): 2, (230,
 'No'): 1539, (230, 'Yes'): 1230, (240, 'No'): 2262, (240, 'Yes'): 1892, (241,
 'No'): 91, (241, 'Yes'): 67, (250, 'No'): 2955, (250, 'Yes'): 2422, (251, 'No'):
 39, (251, 'Yes'): 34, (260, 'No'): 2109, (260, 'Yes'): 1755, (270, 'No'): 2221,
 (270, 'Yes'): 1850, (271, 'No'): 11, (271, 'Yes'): 6, (280, 'No'): 2067, (280,
 'Yes'): 1796, (281, 'No'): 25, (281, 'Yes'): 24, (288, 'No'): 72, (288, 'Yes'):
 64, (290, 'No'): 2866, (290, 'Yes'): 2358, (291, 'No'): 80, (291, 'Yes'): 75,
 (292, 'Yes'): 7, (292, 'No'): 6, (300, 'No'): 1734, (300, 'Yes'): 1431, (301,
 'Yes'): 21, (301, 'No'): 13, (310, 'No'): 3126, (310, 'Yes'): 2672, (311, 'No'):
 13, (311, 'Yes'): 12, (312, 'Yes'): 21, (312, 'No'): 19, (320, 'No'): 1689,
 (320, 'Yes'): 1436, (321, 'No'): 18, (321, 'Yes'): 15, (328, 'No'): 9, (328,
 'Yes'): 4, (330, 'No'): 1829, (330, 'Yes'): 1502, (331, 'No'): 1435, (331,
 'Yes'): 1131, (340, 'No'): 1418, (340, 'Yes'): 1212, (341, 'No'): 61, (341,
 'Yes'): 45, (342, 'Yes'): 42, (342, 'No'): 41, (343, 'No'): 23, (343, 'Yes'):
 22, (350, 'No'): 1952, (350, 'Yes'): 1599, (360, 'No'): 1626, (360, 'Yes'):
 1383, (361, 'Yes'): 12, (361, 'No'): 7, (362, 'No'): 9, (362, 'Yes'): 9, (370,
 'No'): 1832, (370, 'Yes'): 1515, (380, 'No'): 1201, (380, 'Yes'): 954, (381,
 'Yes'): 17, (381, 'No'): 13, (390, 'No'): 2682, (390, 'Yes'): 2336, (391,
 'Yes'): 9, (391, 'No'): 6, (392, 'No'): 8, (392, 'Yes'): 6, (400, 'No'): 4193,
 (400, 'Yes'): 3318, (410, 'No'): 1801, (410, 'Yes'): 1529, (411, 'No'): 110,
 (411, 'Yes'): 79, (412, 'No'): 2, (412, 'Yes'): 2, (420, 'No'): 1965, (420,
 'Yes'): 1719, (421, 'No'): 54, (421, 'Yes'): 34, (430, 'No'): 1745, (430,
 'Yes'): 1568, (431, 'No'): 15, (431, 'Yes'): 4, (440, 'No'): 2640, (440, 'Yes'):
 2179, (441, 'No'): 5, (441, 'Yes'): 3, (450, 'No'): 1039, (450, 'Yes'): 940,
 (451, 'No'): 192, (451, 'Yes'): 158, (460, 'No'): 1826, (460, 'Yes'): 1499,
 (461, 'No'): 61, (461, 'Yes'): 45, (462, 'No'): 60, (462, 'Yes'): 36, (470,
 'No'): 4269, (470, 'Yes'): 3530, (471, 'No'): 22, (471, 'Yes'): 22, (480, 'No'):
 2648, (480, 'Yes'): 2295, (490, 'No'): 3339, (490, 'Yes'): 2737, (500, 'No'):
 1691, (500, 'Yes'): 1447, (510, 'No'): 2375, (510, 'Yes'): 2057, (511, 'No'):
 49, (511, 'Yes'): 30, (520, 'No'): 1625, (520, 'Yes'): 1388, (522, 'No'): 16,
 (522, 'Yes'): 6, (530, 'No'): 2091, (530, 'Yes'): 1721, (531, 'Yes'): 16, (531,
 'No'): 14, (540, 'No'): 1441, (540, 'Yes'): 1216, (541, 'No'): 36, (541, 'Yes'):
 27, (542, 'Yes'): 10, (542, 'No'): 5, (550, 'No'): 1873, (550, 'Yes'): 1602,
 (551, 'No'): 83, (551, 'Yes'): 71, (552, 'No'): 23, (552, 'Yes'): 16, (560,
 'No'): 1362, (560, 'Yes'): 1154, (561, 'No'): 59, (561, 'Yes'): 40, (562, 'No'):
 4, (562, 'Yes'): 1, (563, 'No'): 62, (563, 'Yes'): 61, (564, 'No'): 137, (564,
 'Yes'): 114, (570, 'No'): 1682, (570, 'Yes'): 1471, (580, 'No'): 1621, (580,

'Yes'): 1407, (581, 'Yes'): 78, (581, 'No'): 76, (582, 'No'): 9, (582, 'Yes'): 6, (583, 'No'): 4, (583, 'Yes'): 1, (590, 'No'): 3289, (590, 'Yes'): 2626, (591, 'No'): 93, (591, 'Yes'): 85, (592, 'Yes'): 7, (592, 'No'): 5, (600, 'No'): 1264, (600, 'Yes'): 1129, (601, 'Yes'): 25, (601, 'No'): 21, (610, 'No'): 2126, (610, 'Yes'): 1720, (611, 'Yes'): 28, (611, 'No'): 14, (612, 'No'): 4, (612, 'Yes'): 2, (620, 'No'): 2748, (620, 'Yes'): 2446, (621, 'No'): 27, (621, 'Yes'): 11, (622, 'No'): 150, (622, 'Yes'): 118, (630, 'No'): 1298, (630, 'Yes'): 1103, (631, 'No'): 51, (631, 'Yes'): 37, (632, 'Yes'): 6, (632, 'No'): 4, (640, 'No'): 1103, (640, 'Yes'): 928, (641, 'No'): 115, (641, 'Yes'): 94, (650, 'No'): 1428, (650, 'Yes'): 1265, (651, 'No'): 35, (651, 'Yes'): 29, (652, 'No'): 11, (652, 'Yes'): 10, (653, 'No'): 81, (653, 'Yes'): 66, (654, 'No'): 40, (654, 'Yes'): 26, (660, 'No'): 1077, (660, 'Yes'): 894, (661, 'No'): 9, (661, 'Yes'): 9, (662, 'No'): 4, (662, 'Yes'): 2, (670, 'No'): 656, (670, 'Yes'): 545, (671, 'No'): 31, (671, 'Yes'): 12, (672, 'No'): 6, (680, 'No'): 978, (680, 'Yes'): 808, (681, 'No'): 17, (681, 'Yes'): 8, (690, 'No'): 784, (690, 'Yes'): 664, (691, 'Yes'): 10, (691, 'No'): 9, (700, 'No'): 1861, (700, 'Yes'): 1590, (701, 'No'): 34, (701, 'Yes'): 27, (702, 'Yes'): 23, (702, 'No'): 17, (703, 'No'): 17, (703, 'Yes'): 6, (710, 'No'): 807, (710, 'Yes'): 637, (711, 'No'): 102, (711, 'Yes'): 76, (712, 'Yes'): 2, (712, 'No'): 1, (720, 'No'): 850, (720, 'Yes'): 710, (722, 'No'): 19, (722, 'Yes'): 18, (730, 'No'): 743, (730, 'Yes'): 639, (731, 'No'): 63, (731, 'Yes'): 49, (734, 'No'): 53, (734, 'Yes'): 45, (740, 'No'): 645, (740, 'Yes'): 590, (741, 'No'): 10, (741, 'Yes'): 5, (742, 'No'): 1, (743, 'No'): 34, (743, 'Yes'): 31, (744, 'Yes'): 3, (744, 'No'): 1, (750, 'No'): 942, (750, 'Yes'): 785, (751, 'No'): 26, (751, 'Yes'): 14, (752, 'No'): 13, (752, 'Yes'): 12, (753, 'No'): 1, (754, 'No'): 22, (754, 'Yes'): 16, (755, 'No'): 3, (755, 'Yes'): 1, (756, 'No'): 34, (756, 'Yes'): 24, (757, 'No'): 47, (757, 'Yes'): 40, (758, 'Yes'): 23, (758, 'No'): 22, (760, 'No'): 740, (760, 'Yes'): 583, (761, 'No'): 11, (761, 'Yes'): 10, (770, 'No'): 1487, (770, 'Yes'): 1288, (771, 'No'): 48, (771, 'Yes'): 44, (772, 'No'): 35, (772, 'Yes'): 32, (780, 'No'): 1643, (780, 'Yes'): 1412, (782, 'No'): 16, (782, 'Yes'): 14, (783, 'No'): 14, (783, 'Yes'): 11, (784, 'No'): 3, (784, 'Yes'): 1, (785, 'Yes'): 6, (785, 'No'): 5, (790, 'No'): 524, (790, 'Yes'): 426, (791, 'No'): 278, (791, 'Yes'): 215, (792, 'No'): 33, (792, 'Yes'): 28, (793, 'No'): 5, (793, 'Yes'): 4, (794, 'No'): 83, (794, 'Yes'): 70, (795, 'Yes'): 14, (795, 'No'): 10, (796, 'No'): 3, (797, 'Yes'): 9, (797, 'No'): 6, (800, 'No'): 1164, (800, 'Yes'): 987, (801, 'No'): 224, (801, 'Yes'): 204, (802, 'No'): 31, (802, 'Yes'): 24, (803, 'No'): 16, (803, 'Yes'): 14, (810, 'No'): 2095, (810, 'Yes'): 1757, (811, 'No'): 23, (811, 'Yes'): 19, (812, 'Yes'): 40, (812, 'No'): 28, (820, 'No'): 360, (820, 'Yes'): 282, (821, 'No'): 47, (821, 'Yes'): 38, (822, 'No'): 12, (822, 'Yes'): 3, (830, 'No'): 544, (830, 'Yes'): 411, (831, 'No'): 32, (831, 'Yes'): 28, (832, 'No'): 31, (832, 'Yes'): 26, (835, 'No'): 3, (835, 'Yes'): 2, (838, 'No'): 36, (838, 'Yes'): 30, (840, 'No'): 538, (840, 'Yes'): 478, (841, 'No'): 46, (841, 'Yes'): 30, (842, 'No'): 21, (842, 'Yes'): 17, (843, 'Yes'): 92, (843, 'No'): 90, (844, 'No'): 6, (844, 'Yes'): 5, (845, 'No'): 8, (845, 'Yes'): 7, (850, 'No'): 305, (850, 'Yes'): 277, (851, 'No'): 8, (851, 'Yes'): 6, (860, 'No'): 821, (860, 'Yes'): 675, (861, 'No'): 49, (861, 'Yes'): 33, (862, 'No'): 8, (862, 'Yes'): 5, (867, 'No'): 17, (867, 'Yes'): 16, (870, 'No'): 216, (870, 'Yes'): 164, (871, 'No'): 15, (871, 'Yes'): 15, (873, 'No'): 2, (873, 'Yes'): 1, (874, 'No'): 22,

(874, 'Yes'): 14, (875, 'No'): 5, (876, 'No'): 1, (878, 'No'): 3, (878, 'Yes'): 2, (879, 'No'): 40, (879, 'Yes'): 32, (880, 'No'): 427, (880, 'Yes'): 349, (881, 'No'): 46, (881, 'Yes'): 32, (882, 'Yes'): 30, (882, 'No'): 24, (883, 'Yes'): 27, (883, 'No'): 23, (884, 'No'): 19, (884, 'Yes'): 12, (885, 'No'): 61, (885, 'Yes'): 51, (886, 'No'): 5, (886, 'Yes'): 4, (887, 'Yes'): 5, (887, 'No'): 4, (888, 'No'): 16, (888, 'Yes'): 12, (890, 'No'): 567, (890, 'Yes'): 428, (891, 'No'): 41, (891, 'Yes'): 37, (892, 'No'): 196, (892, 'Yes'): 192, (893, 'Yes'): 7, (893, 'No'): 6, (900, 'No'): 356, (900, 'Yes'): 257, (901, 'Yes'): 12, (901, 'No'): 8, (902, 'No'): 19, (902, 'Yes'): 13, (903, 'No'): 2, (903, 'Yes'): 2, (904, 'No'): 5, (904, 'Yes'): 4, (905, 'No'): 10, (905, 'Yes'): 7, (910, 'No'): 1667, (910, 'Yes'): 1359, (911, 'No'): 119, (911, 'Yes'): 100, (912, 'No'): 3, (912, 'Yes'): 3, (913, 'No'): 40, (913, 'Yes'): 25, (920, 'No'): 369, (920, 'Yes'): 316, (921, 'No'): 372, (921, 'Yes'): 261, (930, 'No'): 281, (930, 'Yes'): 251, (931, 'Yes'): 3, (932, 'No'): 7, (932, 'Yes'): 5, (940, 'No'): 1266, (940, 'Yes'): 1109, (941, 'No'): 19, (941, 'Yes'): 8, (942, 'Yes'): 34, (942, 'No'): 23, (943, 'No'): 24, (943, 'Yes'): 20, (944, 'Yes'): 1, (945, 'No'): 36, (945, 'Yes'): 21, (946, 'No'): 42, (946, 'Yes'): 33, (947, 'Yes'): 30, (947, 'No'): 25, (948, 'Yes'): 55, (948, 'No'): 42, (949, 'Yes'): 8, (949, 'No'): 4, (950, 'No'): 315, (950, 'Yes'): 261, (951, 'No'): 46, (951, 'Yes'): 34, (952, 'Yes'): 18, (952, 'No'): 17, (953, 'No'): 90, (953, 'Yes'): 82, (954, 'No'): 23, (954, 'Yes'): 19, (955, 'No'): 7, (955, 'Yes'): 5, (960, 'No'): 336, (960, 'Yes'): 299, (961, 'No'): 36, (961, 'Yes'): 34, (962, 'Yes'): 54, (962, 'No'): 41, (963, 'Yes'): 4, (963, 'No'): 3, (970, 'No'): 451, (970, 'Yes'): 359, (971, 'No'): 69, (971, 'Yes'): 59, (972, 'No'): 31, (972, 'Yes'): 21, (973, 'No'): 66, (973, 'Yes'): 58, (974, 'No'): 82, (974, 'Yes'): 69, (975, 'No'): 13, (975, 'Yes'): 10, (976, 'No'): 21, (976, 'Yes'): 16, (977, 'No'): 32, (977, 'Yes'): 28, (978, 'No'): 32, (978, 'Yes'): 24, (979, 'Yes'): 21, (979, 'No'): 18, (980, 'No'): 112, (980, 'Yes'): 112, (981, 'No'): 186, (981, 'Yes'): 134, (982, 'No'): 139, (982, 'Yes'): 102, (983, 'No'): 45, (983, 'Yes'): 30, (984, 'Yes'): 67, (984, 'No'): 63, (985, 'No'): 26, (985, 'Yes'): 13, (986, 'Yes'): 174, (986, 'No'): 169, (987, 'No'): 21, (987, 'Yes'): 17, (988, 'No'): 144, (988, 'Yes'): 144, (989, 'No'): 419, (989, 'Yes'): 345, (990, 'No'): 146, (990, 'Yes'): 94, (991, 'No'): 303, (991, 'Yes'): 257, (992, 'No'): 84, (992, 'Yes'): 55, (993, 'No'): 19, (993, 'Yes'): 15, (994, 'No'): 39, (994, 'Yes'): 30, (996, 'No'): 26, (996, 'Yes'): 18, (999, 'No'): 321, (999, 'Yes'): 274}

```
[ ]: train_fin2['DGC_Yes']= np.zeros(len(train_fin2['DiagnosisGroupCode']))
train_fin2['DGC_No']= np.zeros(len(train_fin2['DiagnosisGroupCode']))

cv_fin2['DGC_Yes']= np.zeros(len(cv_fin2['DiagnosisGroupCode']))
cv_fin2['DGC_No']= np.zeros(len(cv_fin2['DiagnosisGroupCode']))

train_fin2['DGC_Yes'],train_fin2['DGC_No'],cv_fin2['DGC_Yes'],cv_fin2['DGC_No'],dgc_dict=
    ↳response_coding(train_fin2,cv_fin2,'DiagnosisGroupCode','PotentialFraud')
print(dgc_dict)
```

{('0', 'No'): 179865, ('0', 'Yes'): 151451, ('000', 'No'): 47, ('000', 'Yes'): 39, ('001', 'Yes'): 5, ('001', 'No'): 3, ('002', 'No'): 7, ('002', 'Yes'): 4,

('003', 'No'): 8, ('003', 'Yes'): 2, ('004', 'No'): 5, ('004', 'Yes'): 3,
 ('005', 'No'): 5, ('005', 'Yes'): 4, ('006', 'No'): 5, ('006', 'Yes'): 2,
 ('007', 'Yes'): 3, ('007', 'No'): 2, ('008', 'Yes'): 5, ('008', 'No'): 4,
 ('009', 'No'): 4, ('009', 'Yes'): 3, ('010', 'Yes'): 3, ('010', 'No'): 2,
 ('011', 'No'): 5, ('011', 'Yes'): 5, ('012', 'No'): 7, ('012', 'Yes'): 4,
 ('013', 'No'): 5, ('013', 'Yes'): 3, ('020', 'Yes'): 11, ('020', 'No'): 9,
 ('021', 'Yes'): 11, ('021', 'No'): 10, ('022', 'Yes'): 15, ('022', 'No'): 12,
 ('023', 'No'): 11, ('023', 'Yes'): 3, ('024', 'No'): 10, ('024', 'Yes'): 3,
 ('025', 'Yes'): 16, ('025', 'No'): 13, ('026', 'Yes'): 13, ('026', 'No'): 10,
 ('027', 'No'): 14, ('027', 'Yes'): 10, ('028', 'Yes'): 16, ('028', 'No'): 14,
 ('029', 'Yes'): 16, ('029', 'No'): 14, ('030', 'No'): 15, ('030', 'Yes'): 10,
 ('031', 'Yes'): 15, ('031', 'No'): 14, ('032', 'No'): 19, ('032', 'Yes'): 15,
 ('033', 'No'): 11, ('033', 'Yes'): 9, ('034', 'No'): 19, ('034', 'Yes'): 13,
 ('035', 'No'): 21, ('035', 'Yes'): 12, ('036', 'Yes'): 15, ('036', 'No'): 10,
 ('037', 'No'): 20, ('037', 'Yes'): 11, ('038', 'Yes'): 14, ('038', 'No'): 13,
 ('039', 'Yes'): 18, ('039', 'No'): 17, ('040', 'No'): 15, ('040', 'Yes'): 13,
 ('041', 'Yes'): 12, ('041', 'No'): 10, ('042', 'Yes'): 12, ('042', 'No'): 11,
 ('052', 'No'): 15, ('052', 'Yes'): 10, ('053', 'Yes'): 25, ('053', 'No'): 16,
 ('054', 'Yes'): 21, ('054', 'No'): 15, ('055', 'No'): 21, ('055', 'Yes'): 13,
 ('056', 'No'): 17, ('056', 'Yes'): 8, ('057', 'No'): 17, ('057', 'Yes'): 8,
 ('058', 'Yes'): 12, ('058', 'No'): 11, ('059', 'No'): 9, ('059', 'Yes'): 7,
 ('060', 'No'): 14, ('060', 'Yes'): 14, ('061', 'Yes'): 11, ('061', 'No'): 8,
 ('062', 'No'): 15, ('062', 'Yes'): 5, ('063', 'No'): 15, ('063', 'Yes'): 15,
 ('064', 'Yes'): 14, ('064', 'No'): 9, ('065', 'Yes'): 15, ('065', 'No'): 7,
 ('066', 'No'): 14, ('066', 'Yes'): 11, ('067', 'No'): 17, ('067', 'Yes'): 7,
 ('068', 'No'): 13, ('068', 'Yes'): 12, ('069', 'No'): 15, ('069', 'Yes'): 14,
 ('070', 'No'): 19, ('070', 'Yes'): 16, ('071', 'No'): 14, ('071', 'Yes'): 8,
 ('072', 'No'): 13, ('072', 'Yes'): 9, ('073', 'No'): 12, ('073', 'Yes'): 11,
 ('074', 'No'): 15, ('074', 'Yes'): 9, ('075', 'No'): 17, ('075', 'Yes'): 11,
 ('076', 'Yes'): 14, ('076', 'No'): 13, ('077', 'No'): 15, ('077', 'Yes'): 13,
 ('078', 'Yes'): 16, ('078', 'No'): 10, ('079', 'No'): 12, ('079', 'Yes'): 11,
 ('080', 'Yes'): 16, ('080', 'No'): 11, ('081', 'No'): 15, ('081', 'Yes'): 6,
 ('082', 'No'): 17, ('082', 'Yes'): 11, ('083', 'No'): 10, ('083', 'Yes'): 7,
 ('084', 'Yes'): 13, ('084', 'No'): 11, ('085', 'No'): 16, ('085', 'Yes'): 5,
 ('086', 'No'): 17, ('086', 'Yes'): 6, ('087', 'No'): 18, ('087', 'Yes'): 18,
 ('088', 'No'): 18, ('088', 'Yes'): 11, ('089', 'No'): 13, ('089', 'Yes'): 10,
 ('090', 'No'): 16, ('090', 'Yes'): 9, ('091', 'Yes'): 15, ('091', 'No'): 12,
 ('092', 'Yes'): 14, ('092', 'No'): 8, ('093', 'Yes'): 11, ('093', 'No'): 10,
 ('094', 'No'): 11, ('094', 'Yes'): 7, ('095', 'Yes'): 16, ('095', 'No'): 13,
 ('096', 'No'): 12, ('096', 'Yes'): 5, ('097', 'No'): 18, ('097', 'Yes'): 16,
 ('098', 'No'): 12, ('098', 'Yes'): 11, ('099', 'No'): 12, ('099', 'Yes'): 9,
 ('100', 'Yes'): 14, ('100', 'No'): 13, ('101', 'No'): 11, ('101', 'Yes'): 9,
 ('102', 'No'): 18, ('102', 'Yes'): 14, ('103', 'No'): 9, ('103', 'Yes'): 7,
 ('113', 'No'): 1, ('113', 'Yes'): 1, ('114', 'No'): 1, ('115', 'No'): 2, ('116',
 'No'): 2, ('117', 'Yes'): 1, ('122', 'No'): 3, ('122', 'Yes'): 2, ('124', 'No'):
 2, ('124', 'Yes'): 1, ('125', 'No'): 4, ('129', 'No'): 4, ('129', 'Yes'): 4,
 ('130', 'No'): 8, ('130', 'Yes'): 4, ('131', 'No'): 6, ('131', 'Yes'): 3,
 ('132', 'Yes'): 5, ('132', 'No'): 2, ('133', 'Yes'): 4, ('133', 'No'): 1,

('134', 'No'): 6, ('134', 'Yes'): 4, ('135', 'No'): 6, ('135', 'Yes'): 5,
('136', 'No'): 3, ('136', 'Yes'): 1, ('137', 'Yes'): 5, ('137', 'No'): 4,
('138', 'No'): 7, ('138', 'Yes'): 4, ('139', 'No'): 4, ('146', 'Yes'): 7,
('146', 'No'): 4, ('147', 'No'): 8, ('147', 'Yes'): 2, ('148', 'No'): 3, ('148',
'Yes'): 2, ('149', 'Yes'): 4, ('149', 'No'): 2, ('150', 'No'): 6, ('150',
'Yes'): 5, ('151', 'No'): 4, ('151', 'Yes'): 1, ('152', 'No'): 6, ('152',
'Yes'): 1, ('153', 'Yes'): 7, ('153', 'No'): 3, ('154', 'No'): 2, ('154',
'Yes'): 2, ('155', 'No'): 3, ('155', 'Yes'): 3, ('156', 'No'): 6, ('156',
'Yes'): 1, ('157', 'No'): 7, ('157', 'Yes'): 7, ('158', 'Yes'): 7, ('158',
'No'): 2, ('159', 'No'): 4, ('159', 'Yes'): 3, ('163', 'No'): 43, ('163',
'Yes'): 38, ('164', 'No'): 54, ('164', 'Yes'): 27, ('165', 'No'): 46, ('165',
'Yes'): 37, ('166', 'No'): 59, ('166', 'Yes'): 35, ('167', 'No'): 55, ('167',
'Yes'): 29, ('168', 'No'): 50, ('168', 'Yes'): 47, ('175', 'No'): 62, ('175',
'Yes'): 37, ('176', 'No'): 45, ('176', 'Yes'): 35, ('177', 'Yes'): 54, ('177',
'No'): 44, ('178', 'No'): 42, ('178', 'Yes'): 33, ('179', 'No'): 46, ('179',
'Yes'): 34, ('180', 'No'): 55, ('180', 'Yes'): 41, ('181', 'No'): 49, ('181',
'Yes'): 49, ('182', 'Yes'): 53, ('182', 'No'): 43, ('183', 'No'): 62, ('183',
'Yes'): 47, ('184', 'Yes'): 51, ('184', 'No'): 49, ('185', 'No'): 54, ('185',
'Yes'): 37, ('186', 'Yes'): 51, ('186', 'No'): 49, ('187', 'No'): 49, ('187',
'Yes'): 36, ('188', 'No'): 41, ('188', 'Yes'): 41, ('189', 'No'): 50, ('189',
'Yes'): 33, ('190', 'No'): 46, ('190', 'Yes'): 41, ('191', 'Yes'): 43, ('191',
'No'): 36, ('192', 'No'): 51, ('192', 'Yes'): 41, ('193', 'No'): 56, ('193',
'Yes'): 44, ('194', 'No'): 40, ('194', 'Yes'): 36, ('195', 'Yes'): 44, ('195',
'No'): 34, ('196', 'No'): 51, ('196', 'Yes'): 39, ('197', 'Yes'): 51, ('197',
'No'): 43, ('198', 'No'): 51, ('198', 'Yes'): 42, ('199', 'No'): 46, ('199',
'Yes'): 30, ('200', 'Yes'): 42, ('200', 'No'): 41, ('201', 'No'): 47, ('201',
'Yes'): 40, ('202', 'No'): 49, ('202', 'Yes'): 47, ('203', 'No'): 53, ('203',
'Yes'): 40, ('204', 'Yes'): 55, ('204', 'No'): 50, ('205', 'No'): 51, ('205',
'Yes'): 45, ('206', 'Yes'): 45, ('206', 'No'): 40, ('207', 'No'): 42, ('207',
'Yes'): 41, ('208', 'No'): 56, ('208', 'Yes'): 50, ('215', 'No'): 34, ('215',
'Yes'): 33, ('216', 'No'): 33, ('216', 'Yes'): 33, ('217', 'No'): 48, ('217',
'Yes'): 27, ('218', 'No'): 42, ('218', 'Yes'): 26, ('219', 'Yes'): 35, ('219',
'No'): 29, ('220', 'No'): 32, ('220', 'Yes'): 30, ('221', 'No'): 39, ('221',
'Yes'): 26, ('222', 'No'): 34, ('222', 'Yes'): 24, ('223', 'No'): 42, ('223',
'Yes'): 37, ('224', 'Yes'): 34, ('224', 'No'): 31, ('225', 'Yes'): 33, ('225',
'No'): 31, ('226', 'No'): 33, ('226', 'Yes'): 30, ('227', 'No'): 32, ('227',
'Yes'): 30, ('228', 'No'): 40, ('228', 'Yes'): 26, ('229', 'No'): 46, ('229',
'Yes'): 29, ('230', 'No'): 50, ('230', 'Yes'): 30, ('231', 'Yes'): 37, ('231',
'No'): 34, ('232', 'Yes'): 31, ('232', 'No'): 30, ('233', 'Yes'): 39, ('233',
'No'): 33, ('234', 'No'): 34, ('234', 'Yes'): 31, ('235', 'No'): 44, ('235',
'Yes'): 32, ('236', 'No'): 36, ('236', 'Yes'): 28, ('237', 'No'): 38, ('237',
'Yes'): 27, ('238', 'No'): 36, ('238', 'Yes'): 29, ('239', 'No'): 39, ('239',
'Yes'): 28, ('240', 'No'): 42, ('240', 'Yes'): 38, ('241', 'No'): 48, ('241',
'Yes'): 33, ('242', 'Yes'): 42, ('242', 'No'): 40, ('243', 'No'): 31, ('243',
'Yes'): 25, ('244', 'No'): 43, ('244', 'Yes'): 26, ('245', 'No'): 30, ('245',
'Yes'): 28, ('246', 'Yes'): 39, ('246', 'No'): 33, ('247', 'No'): 39, ('247',
'Yes'): 25, ('248', 'No'): 35, ('248', 'Yes'): 24, ('249', 'No'): 40, ('249',
'Yes'): 37, ('250', 'No'): 37, ('250', 'Yes'): 32, ('251', 'Yes'): 33, ('251',

'No'): 28, ('252', 'No'): 56, ('252', 'Yes'): 29, ('253', 'No'): 38, ('253',
 'Yes'): 28, ('254', 'Yes'): 33, ('254', 'No'): 28, ('255', 'Yes'): 35, ('255',
 'No'): 33, ('256', 'Yes'): 45, ('256', 'No'): 37, ('257', 'No'): 34, ('257',
 'Yes'): 32, ('258', 'No'): 26, ('258', 'Yes'): 24, ('259', 'Yes'): 35, ('259',
 'No'): 31, ('260', 'Yes'): 36, ('260', 'No'): 23, ('261', 'No'): 38, ('261',
 'Yes'): 24, ('262', 'No'): 37, ('262', 'Yes'): 28, ('263', 'No'): 33, ('263',
 'Yes'): 28, ('264', 'No'): 36, ('264', 'Yes'): 36, ('280', 'No'): 44, ('280',
 'Yes'): 30, ('281', 'Yes'): 38, ('281', 'No'): 35, ('282', 'Yes'): 37, ('282',
 'No'): 30, ('283', 'No'): 39, ('283', 'Yes'): 37, ('284', 'No'): 45, ('284',
 'Yes'): 30, ('285', 'No'): 36, ('285', 'Yes'): 28, ('286', 'No'): 36, ('286',
 'Yes'): 33, ('287', 'No'): 36, ('287', 'Yes'): 17, ('288', 'No'): 43, ('288',
 'Yes'): 20, ('289', 'No'): 46, ('289', 'Yes'): 26, ('290', 'Yes'): 36, ('290',
 'No'): 23, ('291', 'Yes'): 37, ('291', 'No'): 23, ('292', 'No'): 37, ('292',
 'Yes'): 35, ('293', 'No'): 37, ('293', 'Yes'): 27, ('294', 'No'): 44, ('294',
 'Yes'): 41, ('295', 'No'): 42, ('295', 'Yes'): 33, ('296', 'No'): 37, ('296',
 'Yes'): 26, ('297', 'Yes'): 25, ('297', 'No'): 22, ('298', 'No'): 42, ('298',
 'Yes'): 20, ('299', 'No'): 34, ('299', 'Yes'): 21, ('300', 'No'): 32, ('300',
 'Yes'): 25, ('301', 'No'): 37, ('301', 'Yes'): 29, ('302', 'No'): 39, ('302',
 'Yes'): 28, ('303', 'Yes'): 41, ('303', 'No'): 38, ('304', 'No'): 32, ('304',
 'Yes'): 27, ('305', 'No'): 34, ('305', 'Yes'): 27, ('306', 'Yes'): 34, ('306',
 'No'): 21, ('307', 'No'): 44, ('307', 'Yes'): 30, ('308', 'No'): 35, ('308',
 'Yes'): 29, ('309', 'Yes'): 41, ('309', 'No'): 32, ('310', 'No'): 34, ('310',
 'Yes'): 24, ('311', 'No'): 39, ('311', 'Yes'): 26, ('312', 'No'): 37, ('312',
 'Yes'): 29, ('313', 'No'): 39, ('313', 'Yes'): 28, ('314', 'No'): 32, ('314',
 'Yes'): 29, ('315', 'No'): 38, ('315', 'Yes'): 30, ('316', 'No'): 47, ('316',
 'Yes'): 36, ('326', 'No'): 29, ('326', 'Yes'): 20, ('327', 'No'): 27, ('327',
 'Yes'): 25, ('328', 'No'): 21, ('328', 'Yes'): 16, ('329', 'Yes'): 20, ('329',
 'No'): 18, ('330', 'No'): 23, ('330', 'Yes'): 20, ('331', 'Yes'): 22, ('331',
 'No'): 16, ('332', 'Yes'): 27, ('332', 'No'): 18, ('333', 'No'): 18, ('333',
 'Yes'): 17, ('334', 'No'): 22, ('334', 'Yes'): 22, ('335', 'No'): 32, ('335',
 'Yes'): 19, ('336', 'No'): 26, ('336', 'Yes'): 17, ('337', 'No'): 33, ('337',
 'Yes'): 18, ('338', 'No'): 24, ('338', 'Yes'): 23, ('339', 'Yes'): 20, ('339',
 'No'): 18, ('340', 'No'): 27, ('340', 'Yes'): 16, ('341', 'No'): 25, ('341',
 'Yes'): 16, ('342', 'No'): 18, ('342', 'Yes'): 14, ('343', 'No'): 26, ('343',
 'Yes'): 19, ('344', 'No'): 27, ('344', 'Yes'): 23, ('345', 'No'): 22, ('345',
 'Yes'): 18, ('346', 'Yes'): 16, ('346', 'No'): 15, ('347', 'No'): 23, ('347',
 'Yes'): 18, ('348', 'No'): 32, ('348', 'Yes'): 18, ('349', 'No'): 25, ('349',
 'Yes'): 16, ('350', 'No'): 24, ('350', 'Yes'): 18, ('351', 'No'): 20, ('351',
 'Yes'): 20, ('352', 'Yes'): 20, ('352', 'No'): 17, ('353', 'No'): 25, ('353',
 'Yes'): 24, ('354', 'Yes'): 21, ('354', 'No'): 19, ('355', 'No'): 23, ('355',
 'Yes'): 13, ('356', 'No'): 25, ('356', 'Yes'): 13, ('357', 'Yes'): 27, ('357',
 'No'): 21, ('358', 'No'): 24, ('358', 'Yes'): 21, ('368', 'Yes'): 28, ('368',
 'No'): 27, ('369', 'No'): 21, ('369', 'Yes'): 19, ('370', 'No'): 24, ('370',
 'Yes'): 22, ('371', 'Yes'): 20, ('371', 'No'): 17, ('372', 'No'): 23, ('372',
 'Yes'): 19, ('373', 'No'): 26, ('373', 'Yes'): 21, ('374', 'No'): 27, ('374',
 'Yes'): 24, ('375', 'No'): 22, ('375', 'Yes'): 22, ('376', 'No'): 23, ('376',
 'Yes'): 23, ('377', 'No'): 23, ('377', 'Yes'): 21, ('378', 'Yes'): 22, ('378',
 'No'): 19, ('379', 'No'): 28, ('379', 'Yes'): 17, ('380', 'Yes'): 22, ('380',

'No'): 16, ('381', 'No'): 23, ('381', 'Yes'): 14, ('382', 'No'): 21, ('382',
 'Yes'): 17, ('383', 'No'): 25, ('383', 'Yes'): 23, ('384', 'Yes'): 22, ('384',
 'No'): 20, ('385', 'No'): 36, ('385', 'Yes'): 18, ('386', 'No'): 34, ('386',
 'Yes'): 24, ('387', 'No'): 16, ('387', 'Yes'): 16, ('388', 'No'): 25, ('388',
 'Yes'): 14, ('389', 'No'): 24, ('389', 'Yes'): 14, ('390', 'No'): 29, ('390',
 'Yes'): 13, ('391', 'No'): 21, ('391', 'Yes'): 16, ('392', 'No'): 33, ('392',
 'Yes'): 19, ('393', 'No'): 29, ('393', 'Yes'): 15, ('394', 'No'): 29, ('394',
 'Yes'): 16, ('395', 'No'): 31, ('395', 'Yes'): 28, ('405', 'No'): 12, ('405',
 'Yes'): 10, ('406', 'No'): 7, ('406', 'Yes'): 7, ('407', 'No'): 14, ('407',
 'Yes'): 10, ('408', 'Yes'): 11, ('408', 'No'): 9, ('409', 'Yes'): 16, ('409',
 'No'): 11, ('410', 'No'): 15, ('410', 'Yes'): 4, ('411', 'No'): 14, ('411',
 'Yes'): 9, ('412', 'No'): 14, ('412', 'Yes'): 8, ('413', 'No'): 10, ('413',
 'Yes'): 8, ('414', 'No'): 16, ('414', 'Yes'): 12, ('415', 'No'): 14, ('415',
 'Yes'): 6, ('416', 'No'): 16, ('416', 'Yes'): 6, ('417', 'No'): 10, ('417',
 'Yes'): 9, ('418', 'No'): 12, ('418', 'Yes'): 6, ('419', 'No'): 12, ('419',
 'Yes'): 10, ('420', 'No'): 13, ('420', 'Yes'): 9, ('421', 'No'): 9, ('421',
 'Yes'): 6, ('422', 'No'): 16, ('422', 'Yes'): 9, ('423', 'No'): 9, ('423',
 'Yes'): 9, ('424', 'No'): 14, ('424', 'Yes'): 12, ('425', 'No'): 14, ('425',
 'Yes'): 9, ('432', 'No'): 16, ('432', 'Yes'): 9, ('433', 'Yes'): 12, ('433',
 'No'): 9, ('434', 'No'): 12, ('434', 'Yes'): 10, ('435', 'Yes'): 12, ('435',
 'No'): 7, ('436', 'Yes'): 12, ('436', 'No'): 6, ('437', 'Yes'): 11, ('437',
 'No'): 6, ('438', 'No'): 12, ('438', 'Yes'): 10, ('439', 'No'): 11, ('439',
 'Yes'): 11, ('440', 'No'): 9, ('440', 'Yes'): 8, ('441', 'No'): 11, ('441',
 'Yes'): 8, ('442', 'Yes'): 8, ('442', 'No'): 5, ('443', 'No'): 9, ('443',
 'Yes'): 7, ('444', 'Yes'): 12, ('444', 'No'): 8, ('445', 'No'): 12, ('445',
 'Yes'): 6, ('446', 'Yes'): 10, ('446', 'No'): 6, ('453', 'No'): 26, ('453',
 'Yes'): 12, ('454', 'Yes'): 18, ('454', 'No'): 16, ('455', 'No'): 20, ('455',
 'Yes'): 12, ('456', 'No'): 19, ('456', 'Yes'): 17, ('457', 'No'): 14, ('457',
 'Yes'): 14, ('458', 'No'): 18, ('458', 'Yes'): 10, ('459', 'No'): 18, ('459',
 'Yes'): 17, ('460', 'Yes'): 13, ('460', 'No'): 12, ('461', 'No'): 20, ('461',
 'Yes'): 14, ('462', 'No'): 13, ('462', 'Yes'): 10, ('463', 'Yes'): 22, ('463',
 'No'): 18, ('464', 'No'): 19, ('464', 'Yes'): 18, ('465', 'No'): 16, ('465',
 'Yes'): 11, ('466', 'No'): 18, ('466', 'Yes'): 11, ('467', 'Yes'): 16, ('467',
 'No'): 15, ('468', 'No'): 22, ('468', 'Yes'): 13, ('469', 'No'): 19, ('469',
 'Yes'): 18, ('470', 'No'): 23, ('470', 'Yes'): 16, ('471', 'No'): 20, ('471',
 'Yes'): 15, ('472', 'No'): 16, ('472', 'Yes'): 12, ('473', 'No'): 20, ('473',
 'Yes'): 9, ('474', 'No'): 20, ('474', 'Yes'): 9, ('475', 'No'): 13, ('475',
 'Yes'): 11, ('476', 'Yes'): 19, ('476', 'No'): 16, ('477', 'No'): 15, ('477',
 'Yes'): 12, ('478', 'Yes'): 18, ('478', 'No'): 17, ('479', 'No'): 19, ('479',
 'Yes'): 12, ('480', 'Yes'): 19, ('480', 'No'): 15, ('481', 'No'): 18, ('481',
 'Yes'): 7, ('482', 'No'): 24, ('482', 'Yes'): 10, ('483', 'No'): 15, ('483',
 'Yes'): 14, ('484', 'No'): 29, ('484', 'Yes'): 20, ('485', 'No'): 15, ('485',
 'Yes'): 15, ('486', 'No'): 16, ('486', 'Yes'): 11, ('487', 'No'): 19, ('487',
 'Yes'): 13, ('488', 'No'): 22, ('488', 'Yes'): 16, ('489', 'No'): 15, ('489',
 'Yes'): 7, ('490', 'No'): 13, ('490', 'Yes'): 9, ('491', 'Yes'): 18, ('491',
 'No'): 13, ('492', 'No'): 16, ('492', 'Yes'): 15, ('493', 'No'): 21, ('493',
 'Yes'): 13, ('494', 'No'): 18, ('494', 'Yes'): 14, ('495', 'No'): 19, ('495',
 'Yes'): 15, ('496', 'No'): 23, ('496', 'Yes'): 13, ('497', 'No'): 19, ('497',

'Yes'): 13, ('498', 'Yes'): 17, ('498', 'No'): 7, ('499', 'No'): 11, ('499',
 'Yes'): 11, ('500', 'Yes'): 17, ('500', 'No'): 16, ('501', 'No'): 23, ('501',
 'Yes'): 13, ('502', 'No'): 19, ('502', 'Yes'): 16, ('503', 'Yes'): 17, ('503',
 'No'): 16, ('504', 'No'): 19, ('504', 'Yes'): 17, ('505', 'Yes'): 8, ('505',
 'No'): 6, ('506', 'Yes'): 23, ('506', 'No'): 20, ('507', 'Yes'): 16, ('507',
 'No'): 14, ('508', 'No'): 21, ('508', 'Yes'): 8, ('509', 'No'): 16, ('509',
 'Yes'): 15, ('510', 'No'): 13, ('510', 'Yes'): 13, ('511', 'No'): 16, ('511',
 'Yes'): 14, ('512', 'No'): 19, ('512', 'Yes'): 14, ('513', 'No'): 15, ('513',
 'Yes'): 11, ('514', 'Yes'): 17, ('514', 'No'): 6, ('515', 'No'): 18, ('515',
 'Yes'): 8, ('516', 'No'): 23, ('516', 'Yes'): 12, ('517', 'Yes'): 16, ('517',
 'No'): 15, ('533', 'No'): 18, ('533', 'Yes'): 15, ('534', 'No'): 19, ('534',
 'Yes'): 17, ('535', 'No'): 18, ('535', 'Yes'): 17, ('536', 'No'): 18, ('536',
 'Yes'): 16, ('537', 'Yes'): 17, ('537', 'No'): 14, ('538', 'No'): 24, ('538',
 'Yes'): 18, ('539', 'No'): 20, ('539', 'Yes'): 20, ('540', 'Yes'): 12, ('540',
 'No'): 11, ('541', 'No'): 17, ('541', 'Yes'): 14, ('542', 'No'): 15, ('542',
 'Yes'): 14, ('543', 'Yes'): 18, ('543', 'No'): 17, ('544', 'No'): 19, ('544',
 'Yes'): 13, ('545', 'Yes'): 19, ('545', 'No'): 13, ('546', 'No'): 24, ('546',
 'Yes'): 14, ('547', 'No'): 20, ('547', 'Yes'): 18, ('548', 'No'): 18, ('548',
 'Yes'): 13, ('549', 'No'): 26, ('549', 'Yes'): 10, ('550', 'No'): 14, ('550',
 'Yes'): 13, ('551', 'Yes'): 18, ('551', 'No'): 15, ('552', 'No'): 23, ('552',
 'Yes'): 14, ('553', 'No'): 16, ('553', 'Yes'): 12, ('554', 'No'): 18, ('554',
 'Yes'): 12, ('555', 'No'): 17, ('555', 'Yes'): 11, ('556', 'Yes'): 16, ('556',
 'No'): 14, ('557', 'No'): 18, ('557', 'Yes'): 10, ('558', 'No'): 19, ('558',
 'Yes'): 11, ('559', 'No'): 15, ('559', 'Yes'): 8, ('560', 'Yes'): 15, ('560',
 'No'): 6, ('561', 'Yes'): 16, ('561', 'No'): 12, ('562', 'No'): 11, ('562',
 'Yes'): 10, ('563', 'No'): 25, ('563', 'Yes'): 16, ('564', 'No'): 17, ('564',
 'Yes'): 16, ('565', 'Yes'): 14, ('565', 'No'): 13, ('566', 'Yes'): 14, ('566',
 'No'): 11, ('573', 'No'): 16, ('573', 'Yes'): 12, ('574', 'Yes'): 12, ('574',
 'No'): 8, ('575', 'Yes'): 14, ('575', 'No'): 12, ('576', 'No'): 16, ('576',
 'Yes'): 4, ('577', 'No'): 16, ('577', 'Yes'): 9, ('578', 'No'): 13, ('578',
 'Yes'): 12, ('579', 'Yes'): 15, ('579', 'No'): 11, ('580', 'No'): 18, ('580',
 'Yes'): 11, ('581', 'Yes'): 9, ('581', 'No'): 7, ('582', 'No'): 7, ('582',
 'Yes'): 4, ('583', 'Yes'): 11, ('583', 'No'): 7, ('584', 'Yes'): 17, ('584',
 'No'): 8, ('585', 'No'): 19, ('585', 'Yes'): 16, ('592', 'No'): 11, ('592',
 'Yes'): 2, ('593', 'No'): 14, ('593', 'Yes'): 8, ('594', 'Yes'): 14, ('594',
 'No'): 10, ('595', 'No'): 9, ('595', 'Yes'): 9, ('596', 'No'): 12, ('596',
 'Yes'): 10, ('597', 'No'): 11, ('597', 'Yes'): 10, ('598', 'No'): 14, ('598',
 'Yes'): 9, ('599', 'No'): 14, ('599', 'Yes'): 10, ('600', 'No'): 9, ('600',
 'Yes'): 6, ('601', 'Yes'): 11, ('601', 'No'): 6, ('602', 'Yes'): 7, ('602',
 'No'): 4, ('603', 'Yes'): 14, ('603', 'No'): 11, ('604', 'No'): 15, ('604',
 'Yes'): 9, ('605', 'Yes'): 8, ('605', 'No'): 7, ('606', 'No'): 9, ('606',
 'Yes'): 7, ('607', 'No'): 14, ('607', 'Yes'): 8, ('614', 'No'): 25, ('614',
 'Yes'): 10, ('615', 'No'): 24, ('615', 'Yes'): 18, ('616', 'No'): 20, ('616',
 'Yes'): 17, ('617', 'Yes'): 23, ('617', 'No'): 20, ('618', 'No'): 22, ('618',
 'Yes'): 13, ('619', 'No'): 24, ('619', 'Yes'): 15, ('620', 'No'): 25, ('620',
 'Yes'): 16, ('621', 'No'): 13, ('621', 'Yes'): 11, ('622', 'Yes'): 19, ('622',
 'No'): 12, ('623', 'No'): 20, ('623', 'Yes'): 11, ('624', 'No'): 20, ('624',
 'Yes'): 13, ('625', 'Yes'): 22, ('625', 'No'): 13, ('626', 'No'): 22, ('626',

'Yes'): 14, ('627', 'No'): 21, ('627', 'Yes'): 21, ('628', 'Yes'): 20, ('628',
 'No'): 15, ('629', 'No'): 20, ('629', 'Yes'): 18, ('630', 'Yes'): 17, ('630',
 'No'): 16, ('637', 'No'): 16, ('637', 'Yes'): 14, ('638', 'Yes'): 23, ('638',
 'No'): 14, ('639', 'Yes'): 18, ('639', 'No'): 16, ('640', 'No'): 22, ('640',
 'Yes'): 14, ('641', 'No'): 28, ('641', 'Yes'): 13, ('642', 'No'): 21, ('642',
 'Yes'): 13, ('643', 'No'): 23, ('643', 'Yes'): 19, ('644', 'Yes'): 18, ('644',
 'No'): 17, ('645', 'Yes'): 18, ('645', 'No'): 17, ('652', 'No'): 30, ('652',
 'Yes'): 18, ('653', 'No'): 24, ('653', 'Yes'): 17, ('654', 'No'): 18, ('654',
 'Yes'): 11, ('655', 'No'): 21, ('655', 'Yes'): 18, ('656', 'No'): 14, ('656',
 'Yes'): 7, ('657', 'No'): 17, ('657', 'Yes'): 16, ('658', 'Yes'): 20, ('658',
 'No'): 14, ('659', 'Yes'): 18, ('659', 'No'): 17, ('660', 'No'): 16, ('660',
 'Yes'): 16, ('661', 'No'): 16, ('661', 'Yes'): 16, ('662', 'No'): 17, ('662',
 'Yes'): 16, ('663', 'No'): 22, ('663', 'Yes'): 18, ('664', 'Yes'): 26, ('664',
 'No'): 20, ('665', 'No'): 21, ('665', 'Yes'): 13, ('666', 'No'): 34, ('666',
 'Yes'): 23, ('667', 'No'): 27, ('667', 'Yes'): 17, ('668', 'Yes'): 19, ('668',
 'No'): 13, ('669', 'Yes'): 25, ('669', 'No'): 14, ('670', 'No'): 26, ('670',
 'Yes'): 11, ('671', 'No'): 15, ('671', 'Yes'): 15, ('672', 'No'): 20, ('672',
 'Yes'): 17, ('673', 'No'): 25, ('673', 'Yes'): 18, ('674', 'No'): 21, ('674',
 'Yes'): 20, ('675', 'Yes'): 18, ('675', 'No'): 15, ('682', 'Yes'): 22, ('682',
 'No'): 20, ('683', 'Yes'): 24, ('683', 'No'): 17, ('684', 'Yes'): 22, ('684',
 'No'): 19, ('685', 'Yes'): 24, ('685', 'No'): 16, ('686', 'No'): 26, ('686',
 'Yes'): 21, ('687', 'No'): 17, ('687', 'Yes'): 15, ('688', 'No'): 17, ('688',
 'Yes'): 13, ('689', 'No'): 18, ('689', 'Yes'): 17, ('690', 'No'): 23, ('690',
 'Yes'): 20, ('691', 'No'): 21, ('691', 'Yes'): 10, ('692', 'No'): 17, ('692',
 'Yes'): 14, ('693', 'No'): 22, ('693', 'Yes'): 14, ('694', 'Yes'): 20, ('694',
 'No'): 17, ('695', 'Yes'): 24, ('695', 'No'): 19, ('696', 'No'): 21, ('696',
 'Yes'): 16, ('697', 'No'): 20, ('697', 'Yes'): 17, ('698', 'Yes'): 19, ('698',
 'No'): 17, ('699', 'No'): 25, ('699', 'Yes'): 9, ('700', 'Yes'): 16, ('700',
 'No'): 15, ('707', 'Yes'): 4, ('707', 'No'): 3, ('708', 'No'): 7, ('708',
 'Yes'): 5, ('709', 'No'): 4, ('709', 'Yes'): 3, ('710', 'Yes'): 6, ('711',
 'No'): 3, ('711', 'Yes'): 3, ('712', 'No'): 7, ('712', 'Yes'): 4, ('713', 'No'): 8,
 ('713', 'Yes'): 5, ('714', 'No'): 5, ('714', 'Yes'): 1, ('715', 'Yes'): 4,
 ('715', 'No'): 1, ('716', 'No'): 5, ('716', 'Yes'): 2, ('717', 'No'): 8, ('717',
 'Yes'): 1, ('718', 'No'): 7, ('718', 'Yes'): 6, ('722', 'No'): 8, ('722',
 'Yes'): 5, ('723', 'Yes'): 7, ('723', 'No'): 6, ('724', 'Yes'): 3, ('724',
 'No'): 2, ('725', 'No'): 3, ('725', 'Yes'): 3, ('726', 'Yes'): 6, ('726', 'No'): 4,
 ('727', 'No'): 7, ('727', 'Yes'): 4, ('728', 'No'): 6, ('728', 'Yes'): 2,
 ('729', 'No'): 4, ('729', 'Yes'): 3, ('730', 'Yes'): 9, ('734', 'Yes'): 4,
 ('734', 'No'): 3, ('735', 'Yes'): 4, ('735', 'No'): 2, ('736', 'Yes'): 6,
 ('736', 'No'): 5, ('737', 'No'): 5, ('737', 'Yes'): 4, ('738', 'No'): 5, ('738',
 'Yes'): 5, ('739', 'No'): 5, ('739', 'Yes'): 3, ('740', 'No'): 5, ('740',
 'Yes'): 5, ('741', 'No'): 5, ('741', 'Yes'): 3, ('742', 'Yes'): 5, ('742',
 'No'): 2, ('743', 'No'): 9, ('743', 'Yes'): 1, ('744', 'No'): 6, ('744', 'Yes'): 3,
 ('745', 'No'): 4, ('745', 'Yes'): 4, ('746', 'No'): 6, ('746', 'Yes'): 3,
 ('747', 'Yes'): 5, ('747', 'No'): 4, ('748', 'No'): 3, ('748', 'Yes'): 3,
 ('749', 'No'): 3, ('749', 'Yes'): 2, ('750', 'No'): 6, ('750', 'Yes'): 2,
 ('754', 'Yes'): 4, ('754', 'No'): 3, ('755', 'Yes'): 7, ('755', 'No'): 5,
 ('756', 'No'): 5, ('756', 'Yes'): 5, ('757', 'No'): 6, ('757', 'Yes'): 5,

('758', 'Yes'): 6, ('758', 'No'): 4, ('759', 'No'): 6, ('759', 'Yes'): 4,
 ('760', 'No'): 6, ('760', 'Yes'): 3, ('761', 'No'): 4, ('761', 'Yes'): 3,
 ('765', 'No'): 1, ('766', 'No'): 1, ('766', 'Yes'): 1, ('768', 'No'): 1, ('769',
 'No'): 2, ('769', 'Yes'): 1, ('770', 'Yes'): 3, ('774', 'No'): 1, ('777', 'No'):
 1, ('777', 'Yes'): 1, ('778', 'Yes'): 3, ('779', 'Yes'): 3, ('779', 'No'): 2,
 ('781', 'No'): 1, ('782', 'Yes'): 3, ('799', 'No'): 17, ('799', 'Yes'): 9,
 ('800', 'No'): 12, ('800', 'Yes'): 10, ('801', 'No'): 14, ('801', 'Yes'): 11,
 ('802', 'Yes'): 11, ('802', 'No'): 8, ('803', 'No'): 12, ('803', 'Yes'): 5,
 ('804', 'No'): 9, ('804', 'Yes'): 9, ('808', 'No'): 11, ('808', 'Yes'): 11,
 ('809', 'No'): 13, ('809', 'Yes'): 10, ('810', 'Yes'): 13, ('810', 'No'): 10,
 ('811', 'Yes'): 18, ('811', 'No'): 12, ('812', 'No'): 14, ('812', 'Yes'): 11,
 ('813', 'No'): 11, ('813', 'Yes'): 7, ('814', 'No'): 12, ('814', 'Yes'): 10,
 ('815', 'No'): 7, ('815', 'Yes'): 6, ('816', 'Yes'): 13, ('816', 'No'): 9,
 ('820', 'No'): 9, ('820', 'Yes'): 4, ('821', 'No'): 5, ('821', 'Yes'): 4,
 ('822', 'Yes'): 3, ('822', 'No'): 1, ('823', 'Yes'): 4, ('823', 'No'): 2,
 ('824', 'Yes'): 4, ('824', 'No'): 1, ('825', 'No'): 5, ('825', 'Yes'): 2,
 ('826', 'No'): 3, ('826', 'Yes'): 3, ('827', 'Yes'): 3, ('827', 'No'): 1,
 ('828', 'Yes'): 5, ('828', 'No'): 1, ('829', 'No'): 5, ('829', 'Yes'): 2,
 ('830', 'No'): 3, ('830', 'Yes'): 2, ('834', 'Yes'): 3, ('834', 'No'): 2,
 ('835', 'Yes'): 3, ('835', 'No'): 2, ('836', 'No'): 7, ('836', 'Yes'): 2,
 ('837', 'No'): 7, ('838', 'Yes'): 1, ('839', 'No'): 4, ('839', 'Yes'): 3,
 ('840', 'Yes'): 8, ('840', 'No'): 1, ('841', 'No'): 5, ('841', 'Yes'): 3,
 ('842', 'No'): 1, ('843', 'No'): 3, ('843', 'Yes'): 1, ('844', 'Yes'): 5,
 ('844', 'No'): 3, ('845', 'No'): 2, ('845', 'Yes'): 1, ('846', 'No'): 2, ('846',
 'Yes'): 1, ('847', 'Yes'): 3, ('847', 'No'): 2, ('848', 'Yes'): 5, ('848',
 'No'): 3, ('849', 'No'): 4, ('849', 'Yes'): 2, ('853', 'No'): 46, ('853',
 'Yes'): 42, ('854', 'No'): 34, ('854', 'Yes'): 34, ('855', 'Yes'): 28, ('855',
 'No'): 22, ('856', 'Yes'): 33, ('856', 'No'): 29, ('857', 'No'): 29, ('857',
 'Yes'): 25, ('858', 'No'): 29, ('858', 'Yes'): 29, ('862', 'No'): 44, ('862',
 'Yes'): 31, ('863', 'No'): 20, ('863', 'Yes'): 14, ('864', 'No'): 45, ('864',
 'Yes'): 39, ('865', 'No'): 32, ('865', 'Yes'): 32, ('866', 'No'): 36, ('866',
 'Yes'): 27, ('867', 'No'): 58, ('867', 'Yes'): 28, ('868', 'Yes'): 31, ('868',
 'No'): 28, ('869', 'Yes'): 28, ('869', 'No'): 26, ('870', 'No'): 43, ('870',
 'Yes'): 34, ('871', 'No'): 40, ('871', 'Yes'): 26, ('872', 'Yes'): 37, ('872',
 'No'): 30, ('876', 'No'): 48, ('876', 'Yes'): 47, ('880', 'No'): 49, ('880',
 'Yes'): 28, ('881', 'No'): 49, ('881', 'Yes'): 41, ('882', 'No'): 64, ('882',
 'Yes'): 56, ('883', 'No'): 67, ('883', 'Yes'): 44, ('884', 'No'): 60, ('884',
 'Yes'): 46, ('885', 'No'): 60, ('885', 'Yes'): 35, ('886', 'No'): 47, ('886',
 'Yes'): 39, ('887', 'No'): 58, ('887', 'Yes'): 41, ('894', 'Yes'): 19, ('894',
 'No'): 16, ('895', 'No'): 23, ('895', 'Yes'): 17, ('896', 'No'): 24, ('896',
 'Yes'): 16, ('897', 'No'): 26, ('897', 'Yes'): 18, ('901', 'No'): 5, ('901',
 'Yes'): 3, ('902', 'Yes'): 8, ('902', 'No'): 6, ('903', 'No'): 8, ('903',
 'Yes'): 6, ('904', 'No'): 11, ('904', 'Yes'): 6, ('905', 'No'): 7, ('905',
 'Yes'): 4, ('906', 'Yes'): 6, ('906', 'No'): 4, ('907', 'Yes'): 7, ('907',
 'No'): 5, ('908', 'Yes'): 9, ('908', 'No'): 7, ('909', 'Yes'): 9, ('909', 'No'):
 3, ('913', 'No'): 7, ('913', 'Yes'): 3, ('914', 'No'): 10, ('914', 'Yes'): 9,
 ('915', 'No'): 11, ('915', 'Yes'): 8, ('916', 'No'): 8, ('916', 'Yes'): 6,
 ('917', 'No'): 10, ('917', 'Yes'): 6, ('918', 'Yes'): 7, ('918', 'No'): 4,

```

('919', 'No'): 6, ('919', 'Yes'): 2, ('920', 'No'): 6, ('920', 'Yes'): 6,
('921', 'Yes'): 6, ('921', 'No'): 5, ('922', 'No'): 10, ('922', 'Yes'): 5,
('923', 'No'): 10, ('923', 'Yes'): 3, ('927', 'Yes'): 1, ('928', 'No'): 1,
('928', 'Yes'): 1, ('929', 'No'): 2, ('933', 'No'): 2, ('935', 'No'): 2, ('935',
'Yes'): 1, ('939', 'Yes'): 52, ('939', 'No'): 44, ('940', 'No'): 57, ('940',
'Yes'): 49, ('941', 'Yes'): 50, ('941', 'No'): 46, ('945', 'No'): 53, ('945',
'Yes'): 48, ('946', 'No'): 55, ('946', 'Yes'): 41, ('947', 'No'): 51, ('947',
'Yes'): 33, ('948', 'No'): 54, ('948', 'Yes'): 38, ('949', 'No'): 59, ('949',
'Yes'): 36, ('950', 'No'): 47, ('950', 'Yes'): 36, ('951', 'No'): 41, ('951',
'Yes'): 35, ('956', 'Yes'): 3, ('956', 'No'): 2, ('957', 'Yes'): 3, ('958',
'Yes'): 4, ('958', 'No'): 2, ('959', 'No'): 2, ('963', 'No'): 2, ('964', 'No'):
4, ('964', 'Yes'): 2, ('965', 'No'): 3, ('965', 'Yes'): 2, ('969', 'No'): 3,
('969', 'Yes'): 2, ('970', 'No'): 3, ('970', 'Yes'): 2, ('974', 'No'): 3,
('974', 'Yes'): 2, ('975', 'No'): 1, ('975', 'Yes'): 1, ('976', 'Yes'): 4,
('976', 'No'): 1, ('977', 'No'): 2, ('977', 'Yes'): 2, ('981', 'Yes'): 13,
('981', 'No'): 7, ('982', 'Yes'): 6, ('982', 'No'): 4, ('983', 'No'): 10,
('983', 'Yes'): 9, ('984', 'No'): 7, ('984', 'Yes'): 7, ('985', 'No'): 7,
('985', 'Yes'): 3, ('986', 'No'): 8, ('986', 'Yes'): 5, ('987', 'No'): 6,
('987', 'Yes'): 4, ('988', 'No'): 8, ('988', 'Yes'): 4, ('989', 'No'): 10,
('989', 'Yes'): 6, ('998', 'No'): 9, ('998', 'Yes'): 8, ('999', 'No'): 8,
('999', 'Yes'): 5, ('OTH', 'No'): 40, ('OTH', 'Yes'): 29}

```

```

[: train_fin2['BID_Yes']= np.zeros(len(train_fin2['BeneID']))
train_fin2['BID_No']= np.zeros(len(train_fin2['BeneID']))

train_fin2['BID_Yes']= np.zeros(len(train_fin2['BeneID']))
train_fin2['BID_No']= np.zeros(len(train_fin2['BeneID']))

train_fin2['BID_Yes'],train_fin2['BID_No'],cv_fin2['BID_Yes'],cv_fin2['BID_No'],bid_dict=␣
→response_coding(train_fin2,cv_fin2,'BeneID','PotentialFraud')

[: train_fin2['CID_Yes']= np.zeros(len(train_fin2['ClaimID']))
train_fin2['CID_No']= np.zeros(len(train_fin2['ClaimID']))

cv_fin2['CID_Yes']= np.zeros(len(cv_fin2['ClaimID']))
cv_fin2['CID_No']= np.zeros(len(cv_fin2['ClaimID']))

train_fin2['CID_Yes'],train_fin2['CID_No'],cv_fin2['CID_Yes'],cv_fin2['CID_No'],cid_dict=␣
→response_coding(train_fin2,cv_fin2,'ClaimID','PotentialFraud')

[: train_fin2['Pvr_Yes']= np.zeros(len(train_fin2['Provider']))
train_fin2['Pvr_No']= np.zeros(len(train_fin2['Provider']))

cv_fin2['Pvr_Yes']= np.zeros(len(cv_fin2['Provider']))
cv_fin2['Pvr_No']= np.zeros(len(cv_fin2['Provider']))

train_fin2['Pvr_Yes'],train_fin2['Pvr_No'],cv_fin2['Pvr_Yes'],cv_fin2['Pvr_No'],pvr_dict=␣
→response_coding(train_fin2,cv_fin2,'Provider','PotentialFraud')

```

```
[ ]: train_fin2['Ap_Yes']= np.zeros(len(train_fin2['AttendingPhysician']))
train_fin2['Ap_No']= np.zeros(len(train_fin2['AttendingPhysician']))

cv_fin2['Ap_Yes']= np.zeros(len(cv_fin2['AttendingPhysician']))
cv_fin2['Ap_No']= np.zeros(len(cv_fin2['AttendingPhysician']))

train_fin2['Ap_Yes'],train_fin2['Ap_No'],cv_fin2['Ap_Yes'],cv_fin2['Ap_No'],ap_dict=
    ↳response_coding(train_fin2,cv_fin2,'AttendingPhysician','PotentialFraud')

[ ]: train_fin2.
    ↳drop(['State','County','DiagnosisGroupCode','BeneID','ClaimID','Provider','AttendingPhysici
    ↳axis=1, inplace=True)
cv_fin2.
    ↳drop(['State','County','DiagnosisGroupCode','BeneID','ClaimID','Provider','AttendingPhysici
    ↳axis=1, inplace=True)

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n3.pkl','wb') as
    ↳tr_df:
        pickle.dump(train_fin2,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n3.pkl','wb') as cv_df:
    pickle.dump(cv_fin2,cv_df)

[ ]: fig= plt.figure(figsize=(14,10))
gs= GridSpec(2,2,figure= fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,0])
ax4= fig.add_subplot(gs[1,1])

sns.barplot(ax=ax1,x= np.
    ↳unique(train_fin2['NoOfMonths_PartACov']),y=train_fin2['NoOfMonths_PartACov'].
    ↳value_counts()/len(train_fin2['NoOfMonths_PartACov']),palette='crest')
sns.barplot(ax=ax2,x= np.
    ↳unique(cv_fin2['NoOfMonths_PartACov']),y=cv_fin2['NoOfMonths_PartACov'].
    ↳value_counts()/len(cv_fin2['NoOfMonths_PartACov']),palette='crest')
sns.barplot(ax=ax3,x= np.
    ↳unique(train_fin2['NoOfMonths_PartBCov']),y=train_fin2['NoOfMonths_PartBCov'].
    ↳value_counts()/len(train_fin2['NoOfMonths_PartBCov']),palette='crest')
sns.barplot(ax=ax4,x= np.
    ↳unique(cv_fin2['NoOfMonths_PartBCov']),y=cv_fin2['NoOfMonths_PartBCov'].
    ↳value_counts()/len(cv_fin2['NoOfMonths_PartBCov']),palette='crest')

ax1.set_xlabel('Categories in the Variable')
ax2.set_xlabel('Categories in the Variable')
ax3.set_xlabel('Categories in the Variable')
ax4.set_xlabel('Categories in the Variable')
```

```

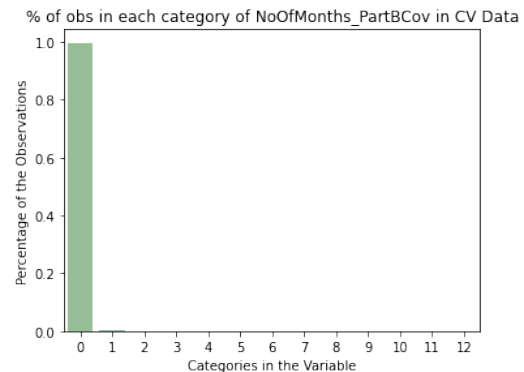
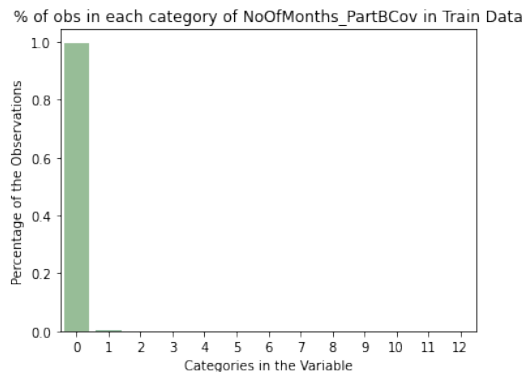
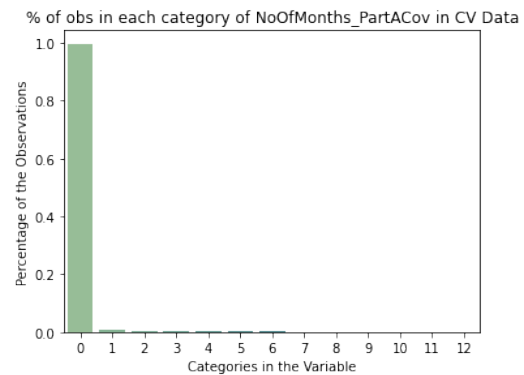
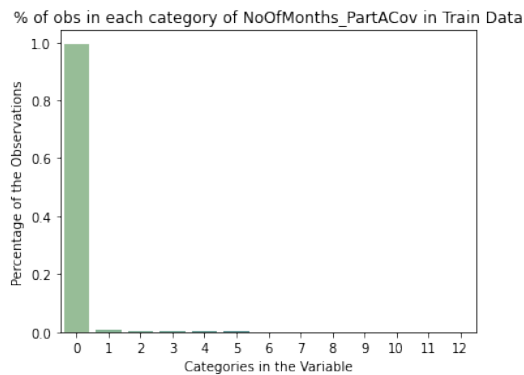
ax1.set_ylabel('Percentage of the Observations')
ax2.set_ylabel('Percentage of the Observations')
ax3.set_ylabel('Percentage of the Observations')
ax4.set_ylabel('Percentage of the Observations')

ax1.set_title('% of obs in each category of NoOfMonths_PartACov in Train Data')
ax2.set_title('% of obs in each category of NoOfMonths_PartACov in CV Data')
ax3.set_title('% of obs in each category of NoOfMonths_PartBCov in Train Data')
ax4.set_title('% of obs in each category of NoOfMonths_PartBCov in CV Data')

plt.subplots_adjust(wspace=0.45)
plt.subplots_adjust(hspace=0.35)

plt.show()

```



4.36 Observations

1. From the above graphs we see that close to 99% of the observations have 0 months in both PartACoverage and PartBCoverage in both the train and cv datasets
2. Since most of the column belongs to same category or has a value of 1 month, the variance in the column is close to 0.

3. Due to 0 variance the contribution of the PartACoverage and PartBCoverage variables to the overall classification of the observations in the PotentialFraud column

Dropping the NoOfMonths PartACov and the NoOfMonths PartBCov variables as they are contributing to the classification task.

```
[ ]: train_fin2.drop(['NoOfMonths_PartACov', 'NoOfMonths_PartBCov'], axis=1, inplace=True)
cv_fin2.drop(['NoOfMonths_PartACov', 'NoOfMonths_PartBCov'], axis=1, inplace=True)

[ ]: train_fin2['DOB'] = pd.to_datetime(train_fin2['DOB'])
train_fin2['DOD'] = pd.to_datetime(train_fin2['DOD'])

cv_fin2['DOB'] = pd.to_datetime(cv_fin2['DOB'])
cv_fin2['DOD'] = pd.to_datetime(cv_fin2['DOD'])
```

4.37 Feature Engineering

Calculating the age of each of the patients as follows:

1. In cases where the Date of Death (DOD) is available, Age = DOD - DOB in years
2. In cases where the Date of Death (DOD) is not available, Age = Max(DOD) - DOB in years

We use Max(DOD) in case 2 to see the year up to which the data has been collected to calculate the patients age at that point of time

```
[ ]: print("The NA percentage in the D.O.B variable: ", train_fin2['DOB'].isna().sum()/len(train_fin2['DOB']))
print("The NA percentage in the D.O.D variable: ", train_fin2['DOD'].isna().sum()/len(train_fin2['DOD']))

print("The NA percentage in the D.O.B variable: ", cv_fin2['DOB'].isna().sum()/len(cv_fin2['DOB']))
print("The NA percentage in the D.O.D variable: ", cv_fin2['DOD'].isna().sum()/len(cv_fin2['DOD']))
```

```
The NA percentage in the D.O.B variable: 0.0
The NA percentage in the D.O.D variable: 0.992601350746135
The NA percentage in the D.O.B variable: 0.0
The NA percentage in the D.O.D variable: 0.9925924598944851
```

```
[ ]: train_fin2['age'] = np.zeros(len(train_fin2['DOB']))

a_max_tr = train_fin2['DOD'].max()

for i in range(len(train_fin2['DOB'])):
    if pd.isnull(train_fin2['DOD'][i]) == True:
        train_fin2['age'][i] = (a_max_tr - train_fin2['DOB'][i]).timedelta(days=365)
    else:
```

```

train_fin2['age'][i]= (train_fin2['DOD'][i]-train_fin2['DOB'][i])/
→timedelta(days=365)

```

```

[: cv_fin2['age']= np.zeros(len(cv_fin2['DOB']))

a_max_cv= cv_fin2['DOD'].max()

for i in range(len(cv_fin2['DOB'])):
    if pd.isnull(cv_fin2['DOD'][i])== True:
        cv_fin2['age'][i]= (a_max_cv-train_fin2['DOB'][i])/timedelta(days=365)
    else:
        cv_fin2['age'][i]= (cv_fin2['DOD'][i]-cv_fin2['DOB'][i])/
→timedelta(days=365)

```

```

[: fig= plt.figure(figsize=(10,6))
gs= GridSpec(1,2,figure=fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

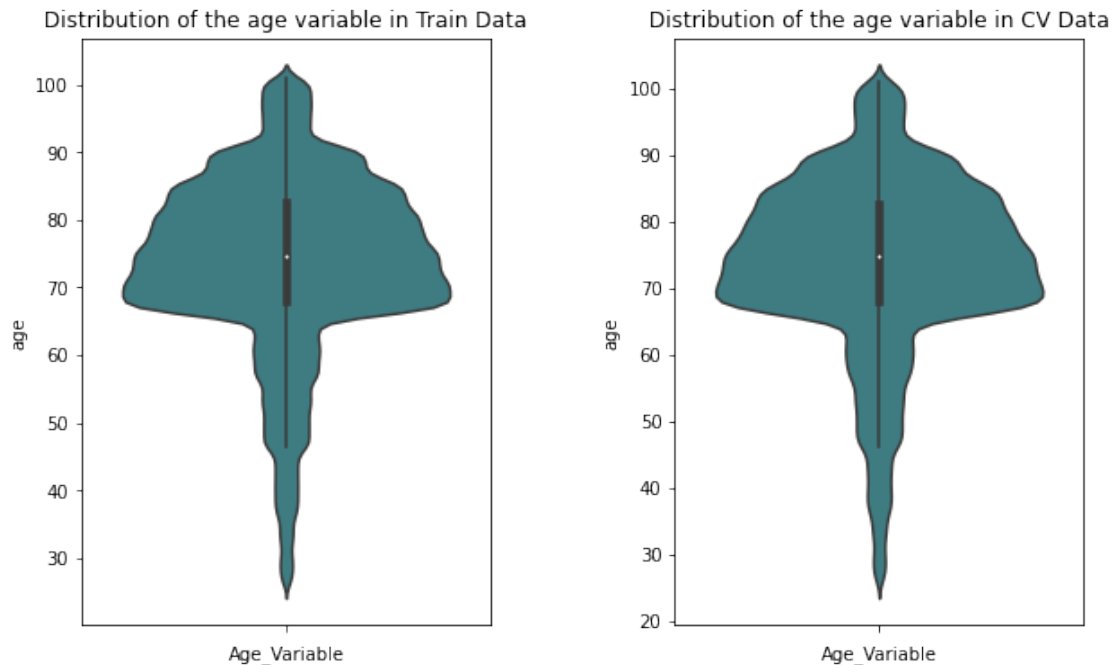
sns.violinplot(y=train_fin3['age'],ax= ax1,palette='crest')
sns.violinplot(y=cv_fin3['age'],ax= ax2,palette='crest')

ax1.set_xlabel('Age_Variable')
ax2.set_xlabel('Age_Variable')

ax1.set_title('Distribution of the age variable in Train Data')
ax2.set_title('Distribution of the age variable in CV Data')

plt.subplots_adjust(wspace=0.45)
plt.show()

```

4.38 Observations

1. From the above graphs we see that a vast majority of the patients in the dataset has a age between 65 and 90.
2. The 50th percentile of the age variable in the train and c datasets seems to be close to 75 years and the 25th and the 75th percentile are at 67 and 82 years resepctively confirming point 1
3. The left/bottom tail of the distribution seems to skewed towards more younger ages raning between early 20s to early 60s with the distribution broadening up with increase in age
4. This is in agreement with the general trend as younger ppl tend to visit the hospital fewer number times than the older population
5. The increase in density could also be due to the fact that older population wiht age between 60 and 80 tend to visit the hospital as a outpatient for their regular check ups or regular visits to theri doctors

Dropping the DOB and DOD columns from the dataset as the information from both the vari-ables has been cptured in the Age variable

```
[ ]: train_fin2.drop(['DOB', 'DOD'], axis=1, inplace=True)
cv_fin2.drop(['DOB', 'DOD'], axis=1, inplace=True)

[ ]: print(np.round((train_inpat['ClmAdmitDiagnosisCode'].isna().sum()/
    →len(train_inpat['ClmAdmitDiagnosisCode']))*100,2), '%')
print(np.round((train_outpat['ClmAdmitDiagnosisCode'].isna().sum()/
    →len(train_outpat['ClmAdmitDiagnosisCode']))*100,2), '%')
```

```
print(np.round((train_fin5['ClmAdmitDiagnosisCode'].isna().sum()/
→len(train_fin5['ClmAdmitDiagnosisCode']))*100,2), '%')
```

0.0 %
79.64 %
73.86 %

Imputing the ClaimAditDiagnosis variable with 0

```
[ ]: train_fin2['ClmAdmitDiagnosisCode']= train_fin2['ClmAdmitDiagnosisCode'].
→fillna(0)
cv_fin2['ClmAdmitDiagnosisCode']= cv_fin2['ClmAdmitDiagnosisCode'].fillna(0)
```

Response Encoding of the ClaimsAdmitDiagnosis variable due to the presence of large number of categories in the variable

```
[ ]: train_fin2['CADC_Yes']= np.zeros(len(train_fin2['ClmAdmitDiagnosisCode']))
train_fin2['CADC_No']= np.zeros(len(train_fin2['ClmAdmitDiagnosisCode']))
cv_fin2['CADC_Yes']= np.zeros(len(cv_fin2['ClmAdmitDiagnosisCode']))
cv_fin2['CADC_No']= np.zeros(len(cv_fin2['ClmAdmitDiagnosisCode']))

train_fin2['CADC_Yes'],train_fin2['CADC_No'],cv_fin2['CADC_Yes'],cv_fin2['CADC_No'],cadc_dict=
→response_coding(train_fin2,cv_fin2,'ClmAdmitDiagnosisCode','PotentialFraud')
```

Dropping the ClmAdmitDiagnosisCode from the dataset

```
[ ]: train_fin2.drop(['ClmAdmitDiagnosisCode'], axis=1, inplace=True)
cv_fin2.drop(['ClmAdmitDiagnosisCode'], axis=1, inplace=True)
```

Replacing the Y and 0 in the RenalDiseaseIndicator varibale with 1 and 0

```
[ ]: train_fin2['RenalDiseaseIndicator']= train_fin2['RenalDiseaseIndicator'].
→map({'Y':1,'0':0})
cv_fin2['RenalDiseaseIndicator']= cv_fin2['RenalDiseaseIndicator'].map({'Y':
→1,'0':0})
```

Replacing the 1 and 2 in the below varibales with 0 and 1 respectively

```
[ ]: nam_cols=
→['ChronicCond_Alzheimer','ChronicCond_Heartfailure','ChronicCond_KidneyDisease','ChronicCon

[ ]: for i in nam_cols:
    train_fin2[i]= train_fin2[i].map({1:0,2:1})
    cv_fin2[i]= cv_fin2[i].map({1:0,2:1})
```

Looking at the distribution of the 0 and 1 categories in each of the below columns

```

[: vars=
    →['RenalDiseaseIndicator','ChronicCond_Alzheimer','ChronicCond_Heartfailure','ChronicCond_Ki
var_uni=[]
var_val=[]

for v in vars:
    var_uni.append(np.unique(train_fin2[v]))
    var_val.append(train_fin2[v].value_counts()/len(train_fin2[v]))

fig= plt.figure(figsize=(15,15))
gs= GridSpec(3,4,figure= fig)

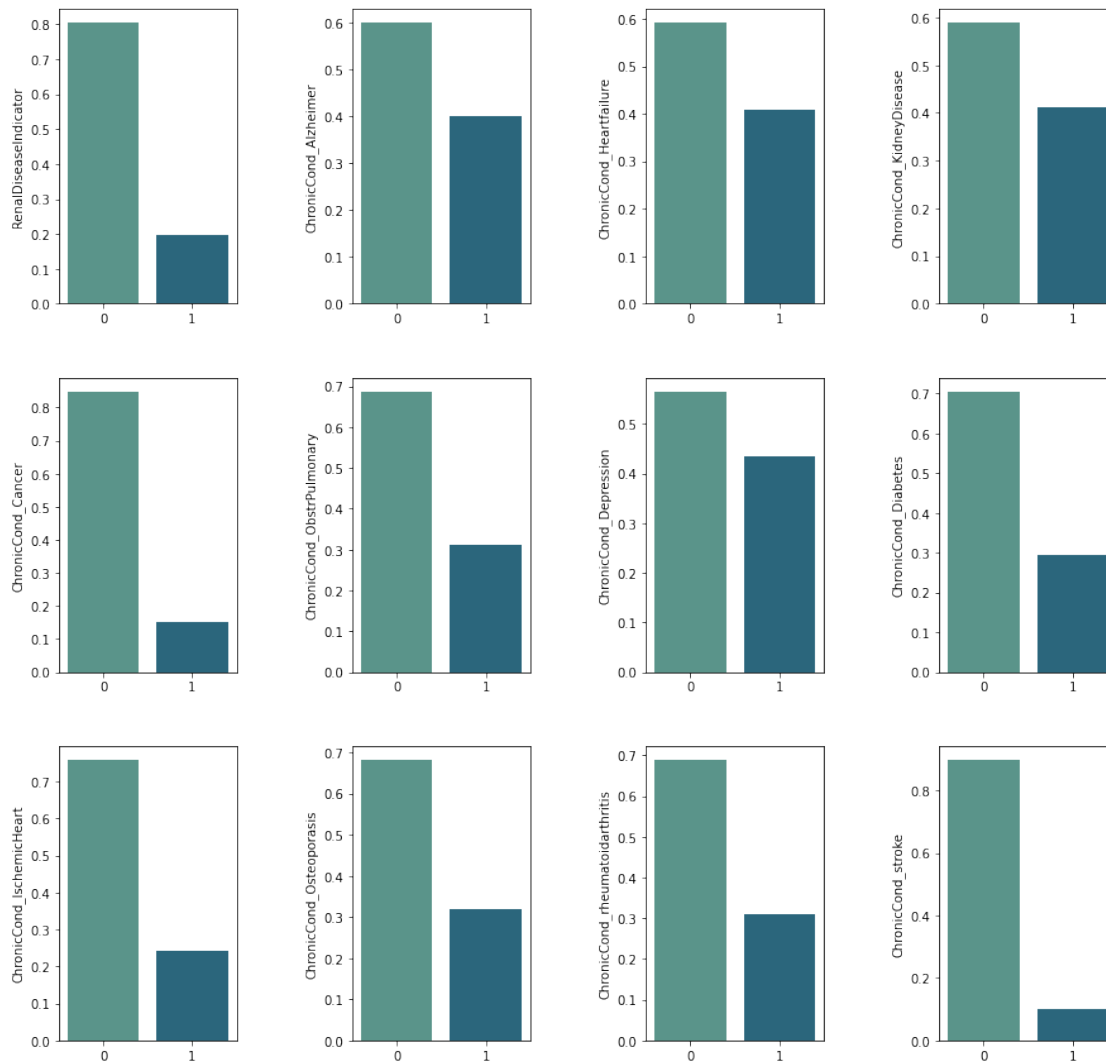
ax=[]

for i in range(3):
    for j in range(4):
        ax.append(fig.add_subplot(gs[i,j]))

for k in range(len(ax)):
    sns.barplot(ax=ax[k],x= var_uni[k],y= var_val[k],palette='crest')
    #ax[k].title.set_text('Percentage distribution of the categories in the
    →Variable')

plt.subplots_adjust(wspace=0.65)
plt.subplots_adjust(hspace=0.25)
#plt.xlabel("NA Percentage in each of the Datasets")
plt.show()

```



```
[ ]: print(np.unique(train_fin5['Race']))
print(train_fin5['Race'].value_counts())
```

```
[1 2 3 5]
1    471036
2     55640
3     19715
5     11820
Name: Race, dtype: int64
```

```
[ ]: train_fin2['Gender']= train_fin2['Gender'].map({1: 'G1',2: 'G2'})
train_fin2['Race']= train_fin2['Race'].map({1: 'R1',2: 'R2',3: 'R3',5: 'R4'})

cv_fin2['Gender']= cv_fin2['Gender'].map({1: 'G1',2: 'G2'})
cv_fin2['Race']= cv_fin2['Race'].map({1: 'R1',2: 'R2',3: 'R3',5: 'R4'})
```

4.39 Imputing the DeductibleAmtPaid with Median strategy using SimpleImpute

```
[ ]: num_imp= SimpleImputer(missing_values= np.nan, strategy= 'median')
train_fin2["DeductibleAmtPaid"]= num_imp.
    ↳fit_transform(train_fin2['DeductibleAmtPaid'].values.reshape(-1,1))[:,0]
cv_fin2["DeductibleAmtPaid"]= num_imp.transform(cv_fin2['DeductibleAmtPaid'].
    ↳values.reshape(-1,1))[:,0]

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n3.pkl','wb') as tr_df:
    ↳tr_df:
        pickle.dump(train_fin2,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n3.pkl','wb') as cv_df:
    pickle.dump(cv_fin2,cv_df)

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n3.pkl','rb') as tr_df:
    ↳tr_df:
        train_fin3= pickle.load(tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n3.pkl','rb') as cv_df:
    cv_fin3= pickle.load(cv_df)
```

5 Multivariate Analysis

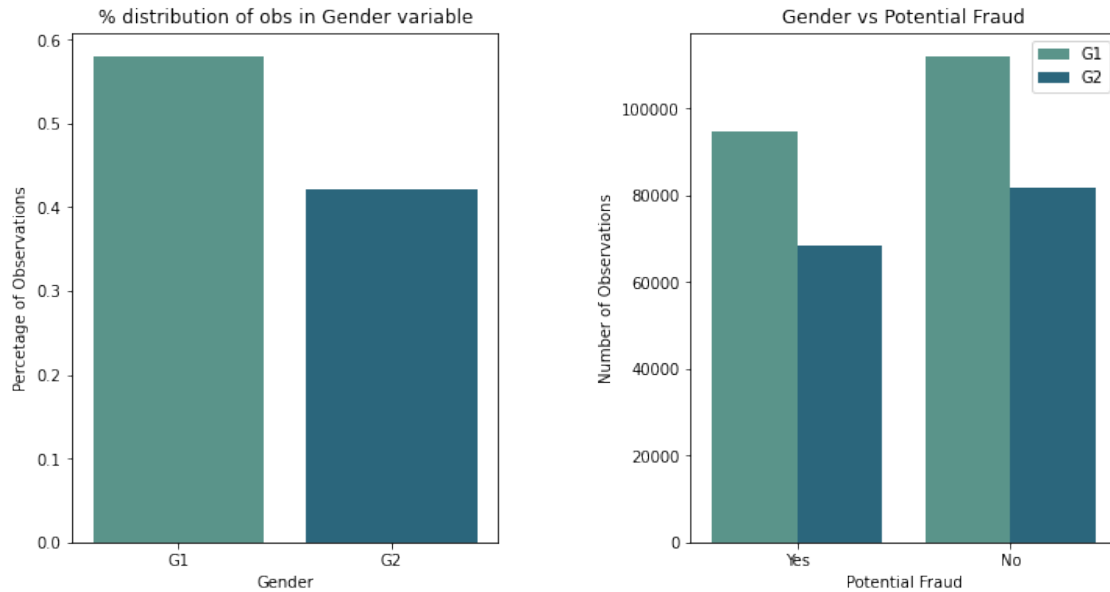
```
[ ]: fig=plt.figure(figsize=(12,6))
gs= GridSpec(1,2,figure= fig)
ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

sns.barplot(ax=ax1,x=np.unique(train_fin3['Gender']),y=train_fin3['Gender'].
    ↳value_counts()/len(train_fin3['Gender']),palette='crest')
sns.countplot(x='PotentialFraud',hue='Gender',data=train_fin3, palette='crest',
    ↳ax=ax2)

ax1.set_xlabel('Gender')
ax1.set_ylabel('Percentage of Observations')
ax1.set_title('% distribution of obs in Gender variable')

ax2.set_xlabel('Potential Fraud')
ax2.set_ylabel('Number of Observations')
ax2.set_title('Gender vs Potential Fraud')

plt.subplots_adjust(wspace=0.45)
plt.legend(labels= np.unique(train_fin3['Gender']))
plt.show()
```



5.1 Observations

1. Gender 1 is dominant in the overall Gender variable
2. Gender 1 is the dominant of both the genders as majority of observations in both Fraud and Non-Fraud cases belong to Gender 1 confirming wiht point 1

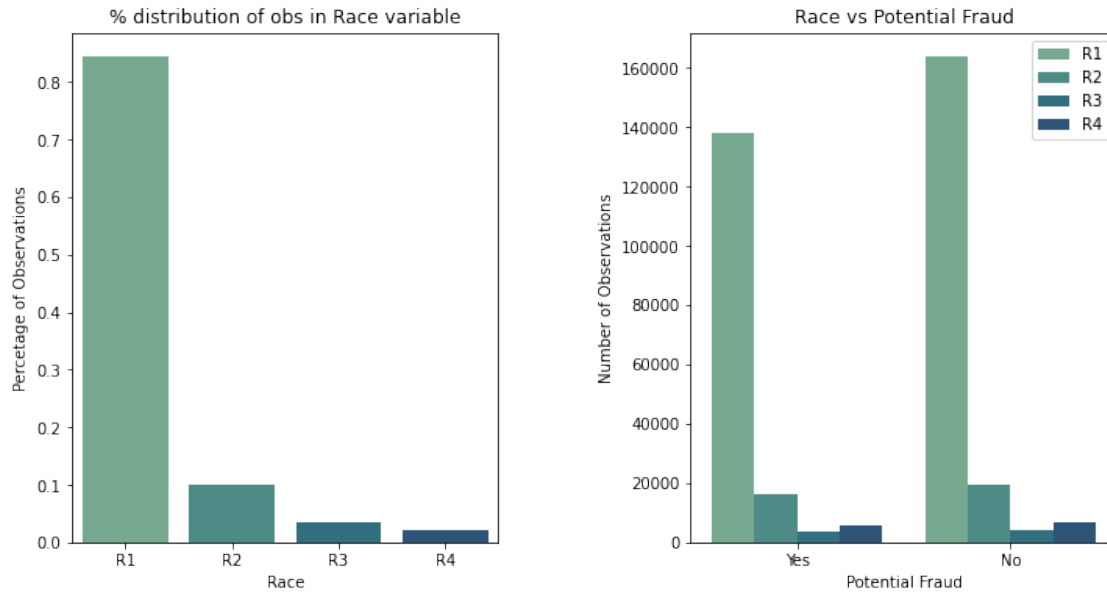
```
[ ]: fig=plt.figure(figsize=(12,6))
gs= GridSpec(1,2,figure= fig)
ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])

sns.barplot(ax=ax1,x=np.unique(train_fin3['Race']),y=train_fin3['Race'].
    ↳value_counts()/len(train_fin3['Race']),palette='crest')
sns.countplot(x='PotentialFraud',hue='Race',data=train_fin3, palette='crest',
    ↳ax=ax2)

ax1.set_xlabel('Race')
ax1.set_ylabel('Percetage of Observations')
ax1.set_title('% distribution of obs in Race variable')

ax2.set_xlabel('Potential Fraud')
ax2.set_ylabel('Number of Observations')
ax2.set_title('Race vs Potential Fraud')

plt.subplots_adjust(wspace=0.45)
plt.legend(labels= np.unique(train_fin3['Race']))
plt.show()
```



5.2 Observations

1. More than 80% of the observations belong to Race 1 as shown in fig.1

2. The distribution of the observations across races in each of the PotentialFraud cases mirror the overall distribution of the Race variable

```
[ ]: num_cols=['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'IPAnnualReimbursementAmt', 'IPAnnualDeco
ax=[]

fig= plt.figure(figsize=(20,20))
gs= GridSpec(3,3,figure= fig)
fig.suptitle('Numerical vs PotentialFraud Variables')

for i in range(3):
    for j in range(3):
        ax.append(fig.add_subplot(gs[i,j]))

for k in range(7):
    sns.violinplot(ax= ax[k],x='PotentialFraud',y=num_cols[k], data=train_fin3,palette='crest')
    ax[k].set_title('{} vs PotentialFraud'.format(num_cols[k]))

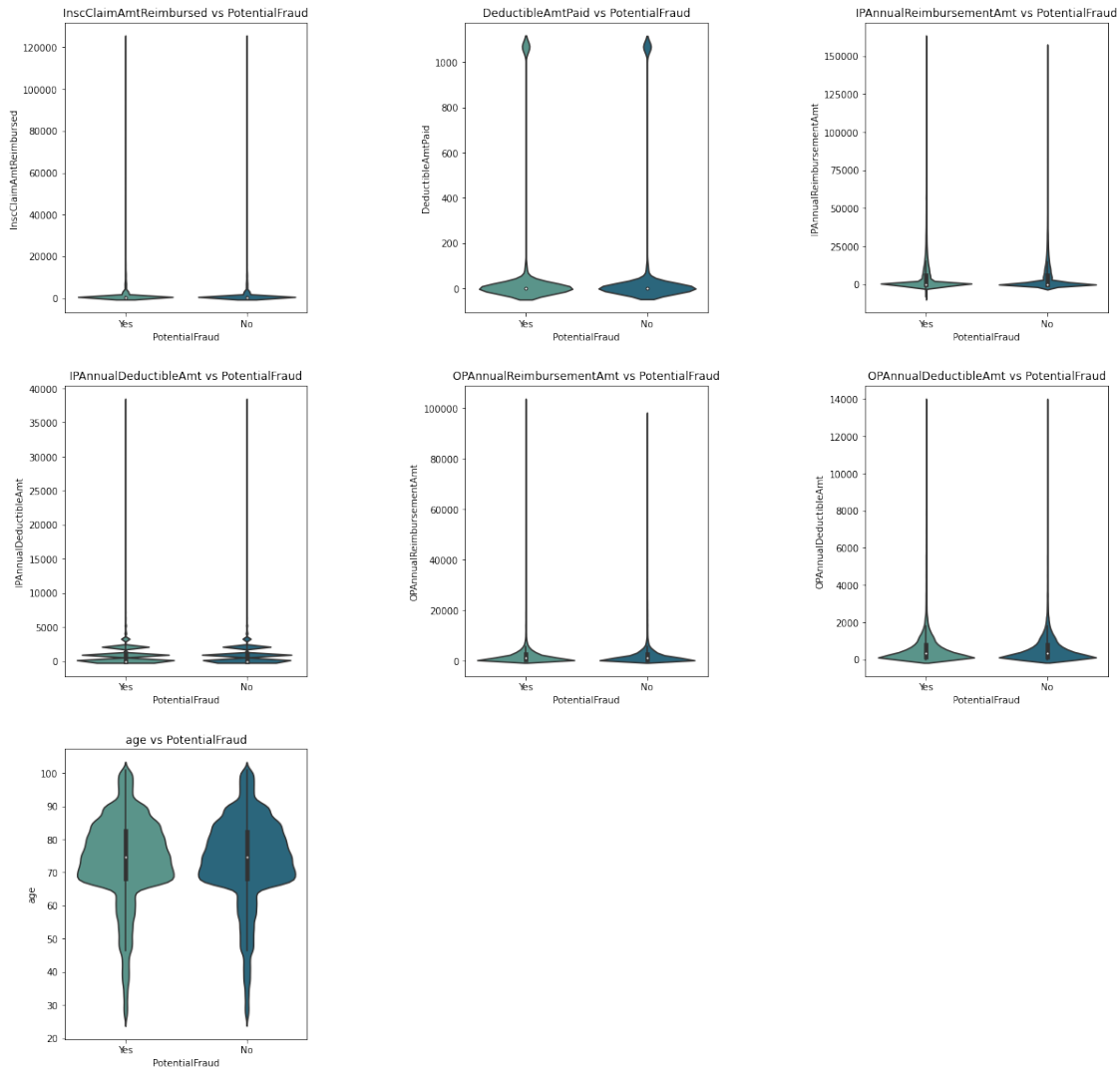
plt.subplots_adjust(wspace=0.65)
plt.subplots_adjust(hspace=0.25)
#plt.xlabel("NA Percentage in each of the Datasets")

ax[7].remove()
```

```
ax[8].remove()
```

```
plt.show()
```

Numerical vs PotentialFraud Variables



5.3 Observations

1. The above graphs compare the distribution of all the numerical variables against the number of classes in the Potential Fraud variable
2. The objective of the above graphs is to see if we could use the box plots to segment the observations within one of the classes of the PotentialFraud variable

3. None of the variables in the above figures is creating a barrier or a level that effectively separates out the 'Yes' and 'No' classes of the PotentialFraud variable
4. It is evident that none of the numerical variables are following a perfect normal distribution and are skewed with longer tails
5. The variables 'InscClaimAmtReimbursed', 'IPAnnualReimbursementAmt', 'OPAnnualReimbursementAmt' are very dense around 0 and taper off towards the tails which means that there are fewer observations with increasing values of the variables
6. IPAnnualDeductibleAmt variable goes through various densities with increasing value of the amount, this could be because the deductible amount is fixed and subscribed by the customer hence the grouping is happening at different levels of values of the Deductible Amount variable.
7. No inference can be drawn in terms of tagging an observation as Fraud and Non-Fraud just by looking at the distribution of the Age variable as the distribution of the Age variable is identical for both the classes of PotentialFraud

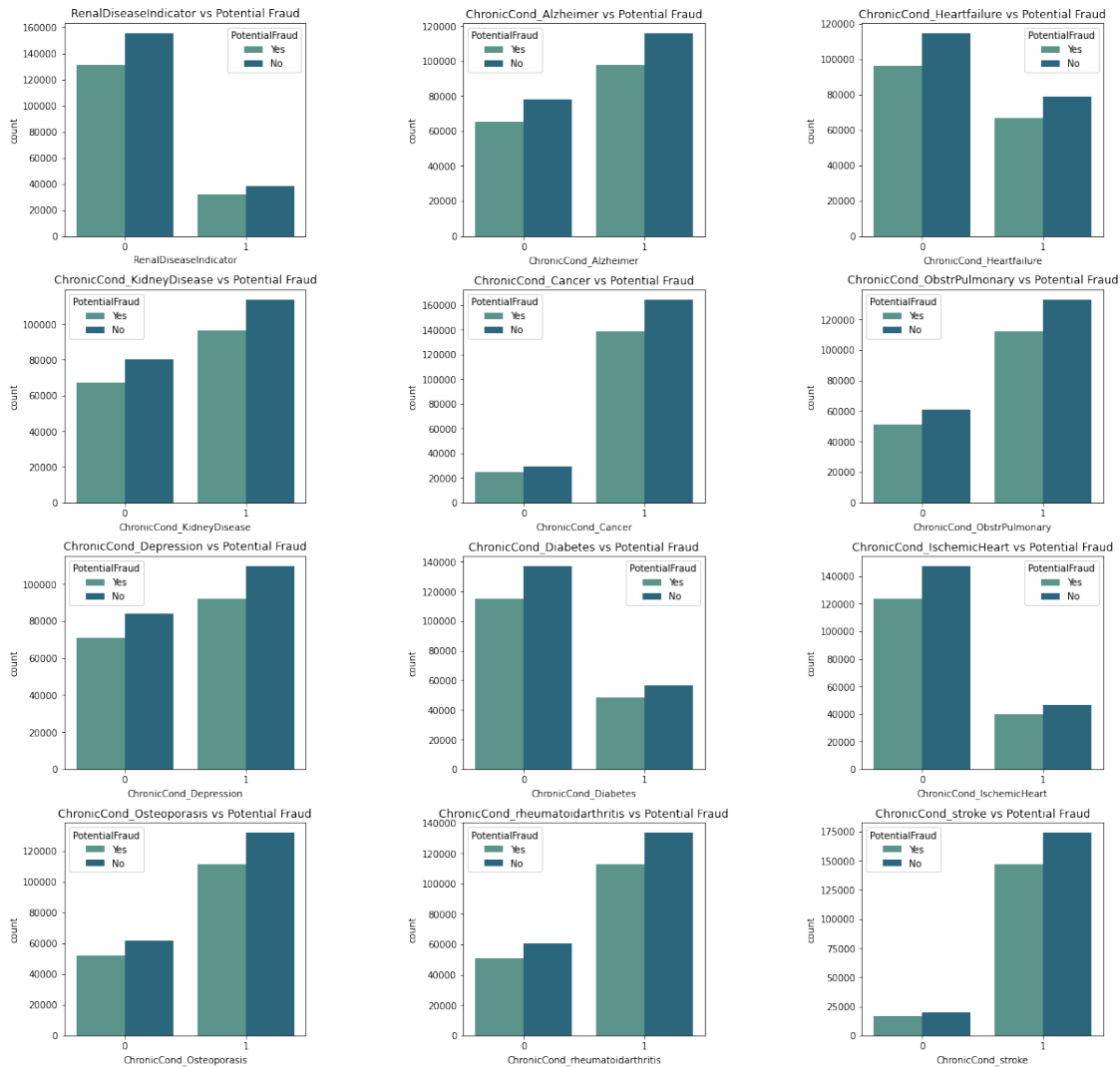
```
[ ]: cat_cols=[
    →['RenalDiseaseIndicator', 'ChronicCond_Alzheimer', 'ChronicCond_Heartfailure', 'ChronicCond_Ki
ax=[]

fig= plt.figure(figsize=(20,20))
gs= GridSpec(4,3,figure= fig)
fig.suptitle('Categorical Variables vs PotentialFraud Variables')

for i in range(4):
    for j in range(3):
        ax.append(fig.add_subplot(gs[i,j]))

for k in range(12):
    sns.countplot(x=cat_cols[k],hue='PotentialFraud',data=train_fin3,
    →palette='crest', ax=ax[k])
    ax[k].set_title('{ } vs Potential Fraud'.format(cat_cols[k]))

plt.subplots_adjust(wspace=0.65)
plt.subplots_adjust(hspace=0.25)
#plt.xlabel("NA Percentage in each of the Datasets")
plt.show()
```



5.4 Observations

1. The above figures compare the distribution of the categorical variables against the number of classes in the Potential Fraud variable
2. The figure gives the counts of each of the classes associated to each of the 'Yes' and 'No' classes of the Potential Fraud variable

```
[ ]: res_cols=['DGC_Yes', 'DGC_No', 'State_Yes', 'State_No', 'County_Yes', 'County_No', 'CID_Yes', 'CID_No', 'ax= []
```

```

fig= plt.figure(figsize=(20,20))
gs= GridSpec(4,4,figure= fig)
fig.suptitle('Response Coded Categorical Variables vs Potential Fraud Variable')

for i in range(4):
    for j in range(4):
        ax.append(fig.add_subplot(gs[i,j]))

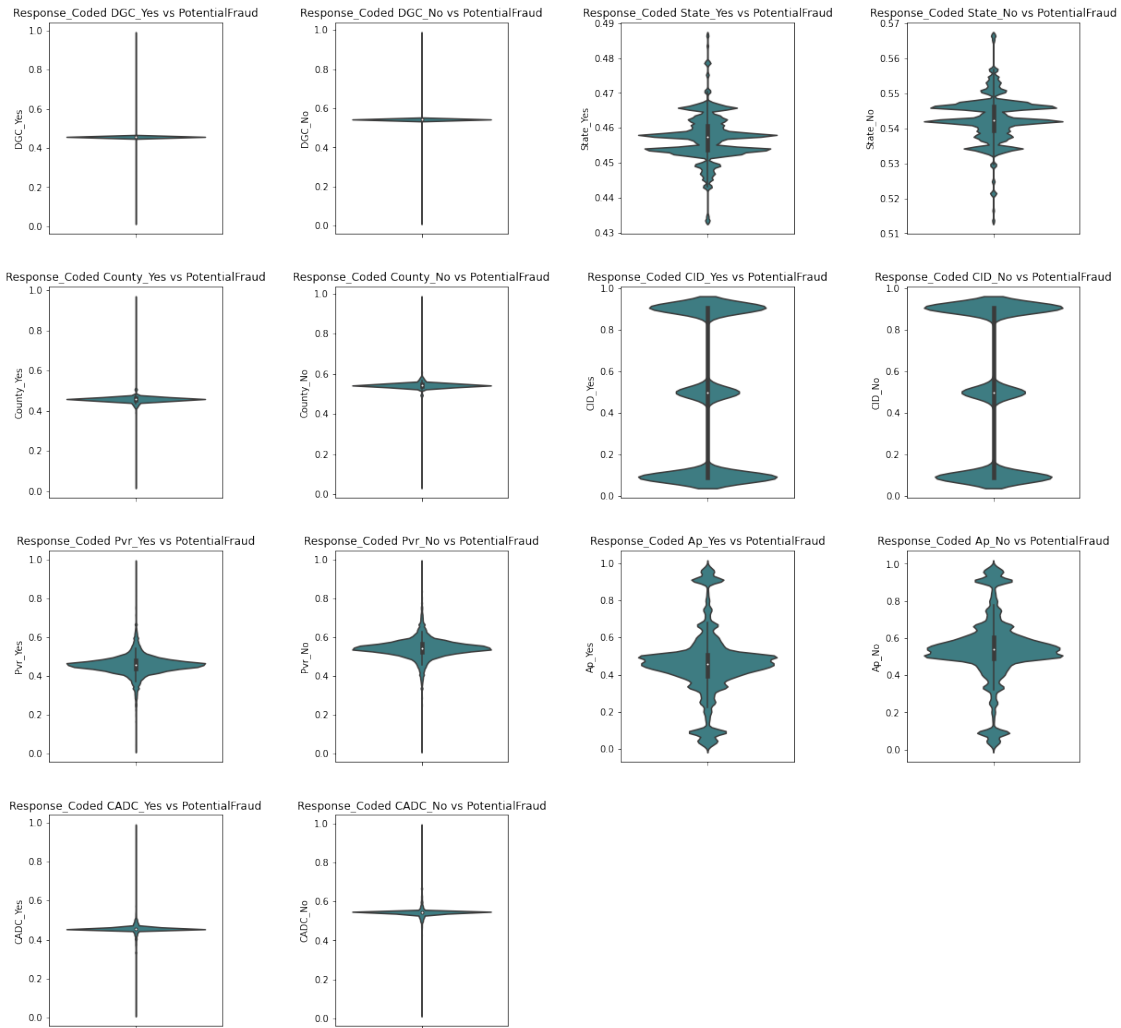
for k in range(14):
    sns.violinplot(ax= ax[k],y=res_cols[k], data= train_fin3,palette='crest')
    ax[k].set_title('Response_Coded {} vs PotentialFraud'.format(res_cols[k]))

plt.subplots_adjust(wspace=0.65)
plt.subplots_adjust(hspace=0.25)

ax[14].remove()
ax[15].remove()

plt.show()

```



5.5 Observations

1. The above graphs compare the distribution of all the variables which have gone through Response Coding against the number of classes in the Potential Fraud variable
2. The variables DiagnosisGroupCodes(DGC),County,Provider(Pvr) and ClaimAdmitDiagnosisCodes(CADC) are very dense around a small number of values. These values do not effectively segregate the PotentialFraud classes.
3. The variables State, AttendingPhysician and ClaimID are densely distributed around some values. For example, for the state variable there a few states where the number of observations are very high which could be due to the size of the states as some states could have more number of people than the others

4. Similarly, ClaimID variable has higher densities around 3 different groups of claimIDs

```
[ ]: num_cols=['InscClaimAmtReimbursed','DeductibleAmtPaid','IPAnnualReimbursementAmt','IPAnnualDec

[ ]: fig= plt.figure(figsize=(14,8))
gs= GridSpec(2,2,figure=fig)

fig.suptitle('Multivariate Analysis of the various numerical analysis')

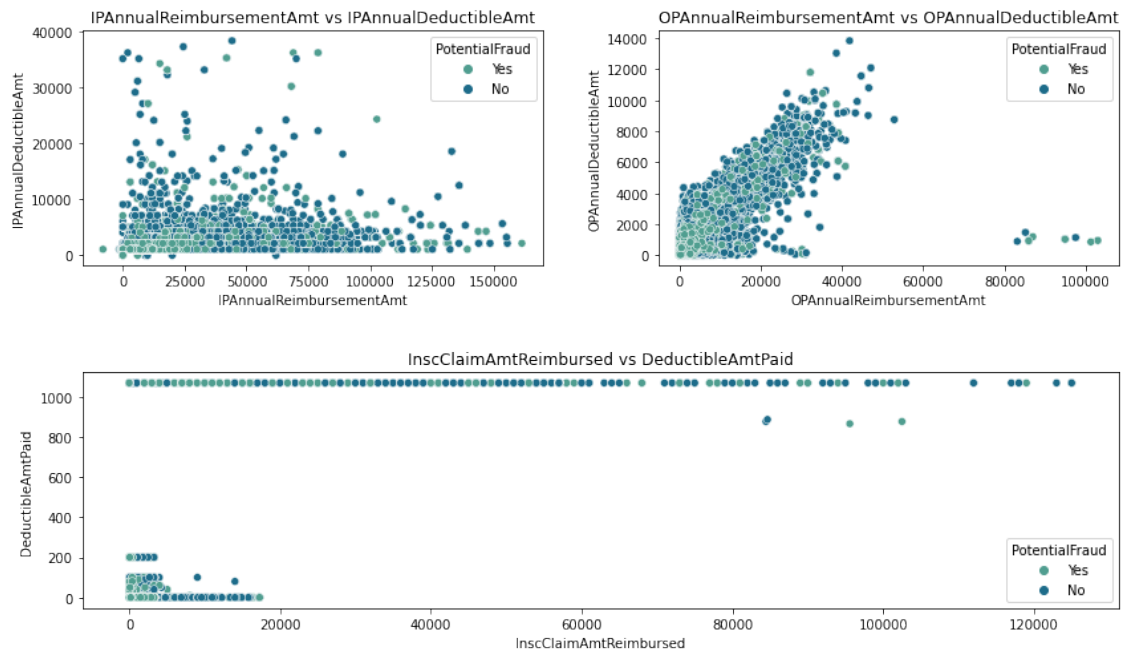
ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,:])

#sns.scatterplot(x='InscClaimAmtReimbursed',y='age',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
sns.scatterplot(x='InscClaimAmtReimbursed',y='DeductibleAmtPaid',
→hue='PotentialFraud',data=train_fin3,ax=ax3,palette='crest')
sns.scatterplot(x='IPAnnualReimbursementAmt',y='IPAnnualDeductibleAmt',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
sns.scatterplot(x='OPAnnualReimbursementAmt',y='OPAnnualDeductibleAmt',
→hue='PotentialFraud',data=train_fin3,ax=ax2,palette='crest')

#ax1.set_title('InscClaimAmtReimbursed vs Age')
ax3.set_title('InscClaimAmtReimbursed vs DeductibleAmtPaid')
ax1.set_title('IPAnnualReimbursementAmt vs IPAnnualDeductibleAmt')
ax2.set_title('OPAnnualReimbursementAmt vs OPAnnualDeductibleAmt')

plt.subplots_adjust(wspace=0.25)
plt.subplots_adjust(hspace=0.45)

plt.show()
```



5.6 Definitions

Medical Reimbursement: Healthcare reimbursement describes the payment that your hospital, doctor, diagnostic facility, or other healthcare providers receive for giving a medical service. Often, health insurer or a government payer covers the cost of all or part of the health care.

Deductible: If a health insurance plan has a deductible of 3000 dollars the insured/individual will have to pay all the medical expenses until 3,000 dollars. Anything above \$3000, the insurance will start paying for the services.

5.7 Observations

1. **IPAnnualReimbursementAmt vs IPAnnualDeductibleAmt:** From the above definition, we see from a high density that the Inpatient deductible amount is fixed in between 0 and 10000 dollars. Due to the same reason we see a lot of grouping of points at the bottom half of the plot.
2. **OPAnnualReimbursementAmt vs OPAnnualDeductibleAmt:** There is a clear increasing trend and a very distinct grouping of the observations tagged as Fraud in the bottom part of the plot. **The ranges for most of the Fraud cases are in the region where Deductible amount is between 0 and 3,000 dollars and the Reimbursement amount between 0 and 10,000 dollars.**
3. **InscClaimAmtReimbursed vs DeductibleAmtPaid:** This plot clearly shows the ranges of Deductible amount paid. We see grouping of observations at levels where deductible amount is 0-150 dollars, 200 dollars and greater than 1000 dollars.

```

[ ]: fig= plt.figure(figsize=(14,8))
gs= GridSpec(2,2,figure=fig)

fig.suptitle('Multivariate Analysis of Age with various Deductible Amount Paid,
→Variables')

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,:])

#sns.scatterplot(x='InscClaimAmtReimbursed',y='age',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
sns.scatterplot(x='age',y='DeductibleAmtPaid',
→hue='PotentialFraud',data=train_fin3,ax=ax3,palette='crest')
sns.scatterplot(x='age',y='IPAnnualDeductibleAmt',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
sns.scatterplot(x='age',y='OPAnnualDeductibleAmt',
→hue='PotentialFraud',data=train_fin3,ax=ax2,palette='crest')

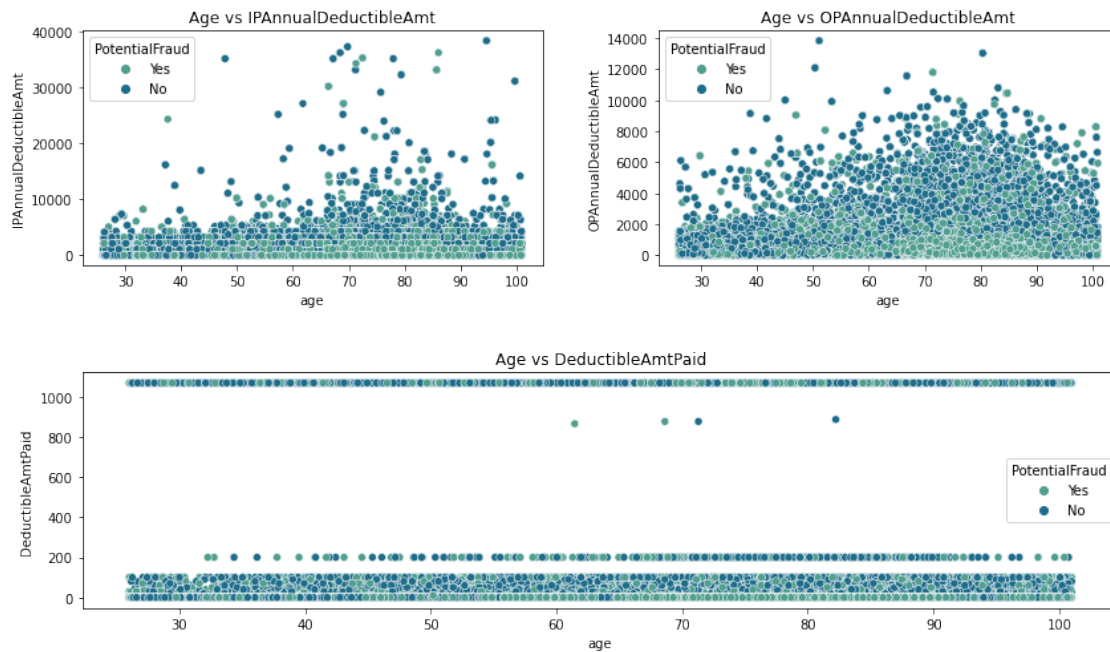
#ax1.set_title('InscClaimAmtReimbursed vs Age')
ax3.set_title('Age vs DeductibleAmtPaid')
ax1.set_title('Age vs IPAnnualDeductibleAmt')
ax2.set_title('Age vs OPAnnualDeductibleAmt')

plt.subplots_adjust(wspace=0.25)
plt.subplots_adjust(hspace=0.45)

plt.show()

```

Multivariate Analysis of Age with various Deductible Amount Paid Variables



5.8 Observations

1. Although small groups of fraud observations are seen in the age range of 45 to 90 years, fraud observations are spread across all the ranges of age as is quite evident from all the 3 plots above.
2. The distribution of the observations along the Y-axis seems to be in accordance with the levels that exist in the Deductible Amount Paid variable.

```
[ ]: fig= plt.figure(figsize=(14,8))
gs= GridSpec(2,2,figure=fig)

fig.suptitle('Multivariate Analysis of Age with various Reimbursement Amount_
→Paid Variables')

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,:])

#sns.scatterplot(x='InscClaimAmtReimbursed',y='age',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
sns.scatterplot(x='age',y='InscClaimAmtReimbursed',
→hue='PotentialFraud',data=train_fin3,ax=ax3,palette='crest')
sns.scatterplot(x='age',y='IPAnnualReimbursementAmt',
→hue='PotentialFraud',data=train_fin3,ax=ax1,palette='crest')
```



```

sns.scatterplot(x='age',y='OPAnnualReimbursementAmt',
               →hue='PotentialFraud',data=train_fin3,ax=ax2,palette='crest')

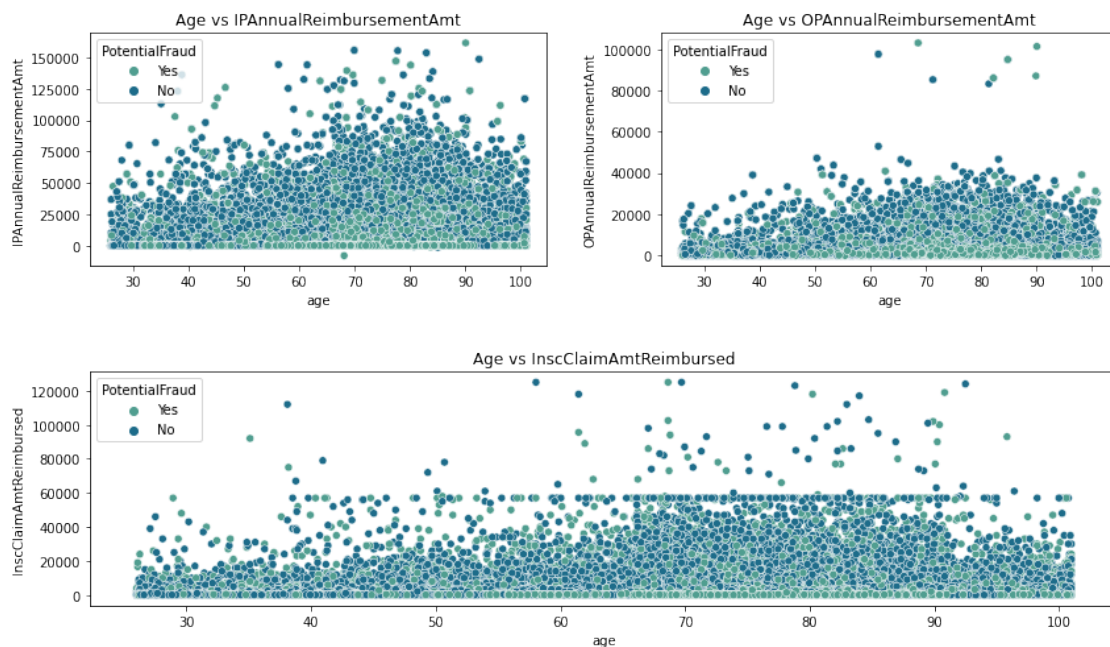
#ax1.set_title('InscClaimAmtReimbursed vs Age')
ax3.set_title('Age vs InscClaimAmtReimbursed')
ax1.set_title('Age vs IPAnnualReimbursementAmt')
ax2.set_title('Age vs OPAnnualReimbursementAmt')

plt.subplots_adjust(wspace=0.25)
plt.subplots_adjust(hspace=0.45)

plt.show()

```

Multivariate Analysis of Age with various Reimbursement Amount Paid Variables



5.9 Observations

1. Very Similar observations as that of the above 'Age with various Deductible Amount Paid Variables' plot.

```
[ ]: len(np.unique(train_fin3['age']))
```

```
[ ]: 1332
```

```
[ ]: train_fin3['age_bins'] = np.zeros(len(train_fin3['age']))
```

```
train_fin3['age_bins'] = pd.cut(x=
    ↳train_fin3['age'],bins=[0,20,30,40,50,60,80,100,110],
    ↳labels=['<20','20s','30s','40s','50s','60+','80+','100+'])
```

```
[ ]: fig= plt.figure(figsize=(14,10))
gs= GridSpec(2,2,figure= fig)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,0])
ax4= fig.add_subplot(gs[1,1])

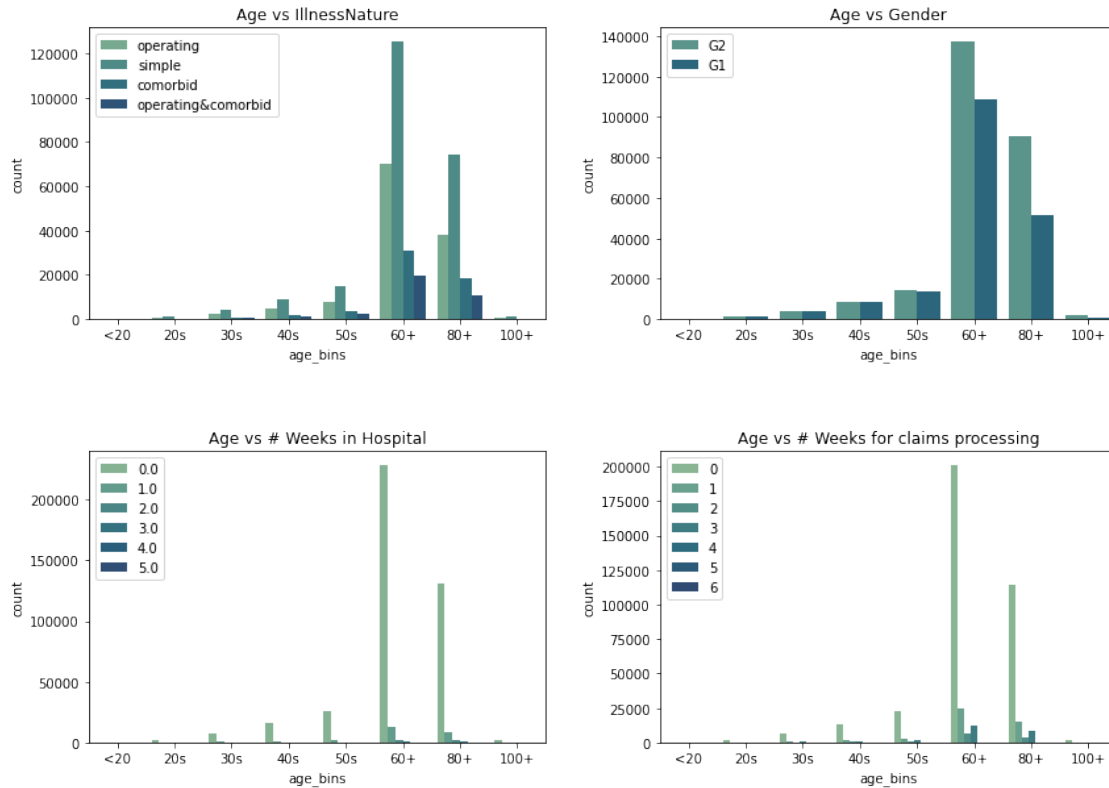
sns.countplot(x='age_bins',hue='IllnessNature',data=train_fin3,
    ↳palette='crest', ax= ax1)
sns.countplot(x='age_bins',hue='Gender',data=train_fin3, palette='crest', ax=
    ↳ax2)
sns.countplot(x='age_bins',hue='HospitalWeeks',data=train_fin3,
    ↳palette='crest', ax= ax3)
sns.countplot(x='age_bins',hue='ClaimWeeks',data=train_fin3, palette='crest',
    ↳ax= ax4)

ax1.legend(loc='upper left')
ax2.legend(loc='upper left')
ax3.legend(loc='upper left')
ax4.legend(loc='upper left')

ax1.set_title('Age vs IllnessNature')
ax2.set_title('Age vs Gender')
ax3.set_title('Age vs # Weeks in Hospital')
ax4.set_title('Age vs # Weeks for claims processing')

plt.subplots_adjust(wspace=0.25)
plt.subplots_adjust(hspace=0.45)

plt.show()
```



```
[ ]: fig= plt.figure(figsize=(14,10))
gs= GridSpec(2,2)

ax1= fig.add_subplot(gs[0,0])
ax2= fig.add_subplot(gs[0,1])
ax3= fig.add_subplot(gs[1,0])
ax4= fig.add_subplot(gs[1,1])

sns.countplot(x='#_Procedures',hue='Gender',data=train_fin3, palette='crest',
    ↳ax= ax1)
sns.countplot(x='#_DiagnosisCodes',hue='Gender',data=train_fin3,
    ↳palette='crest', ax= ax2)
sns.countplot(x='IllnessNature',hue='Gender',data=train_fin3, palette='crest',
    ↳ax= ax3)
sns.countplot(x='Race',hue='Gender',data=train_fin3, palette='crest', ax= ax4)

ax1.legend(loc='upper right')
ax2.legend(loc='upper right')
ax3.legend(loc='upper right')
ax4.legend(loc='upper right')

ax1.set_title('Gender vs #_Procedures')
```

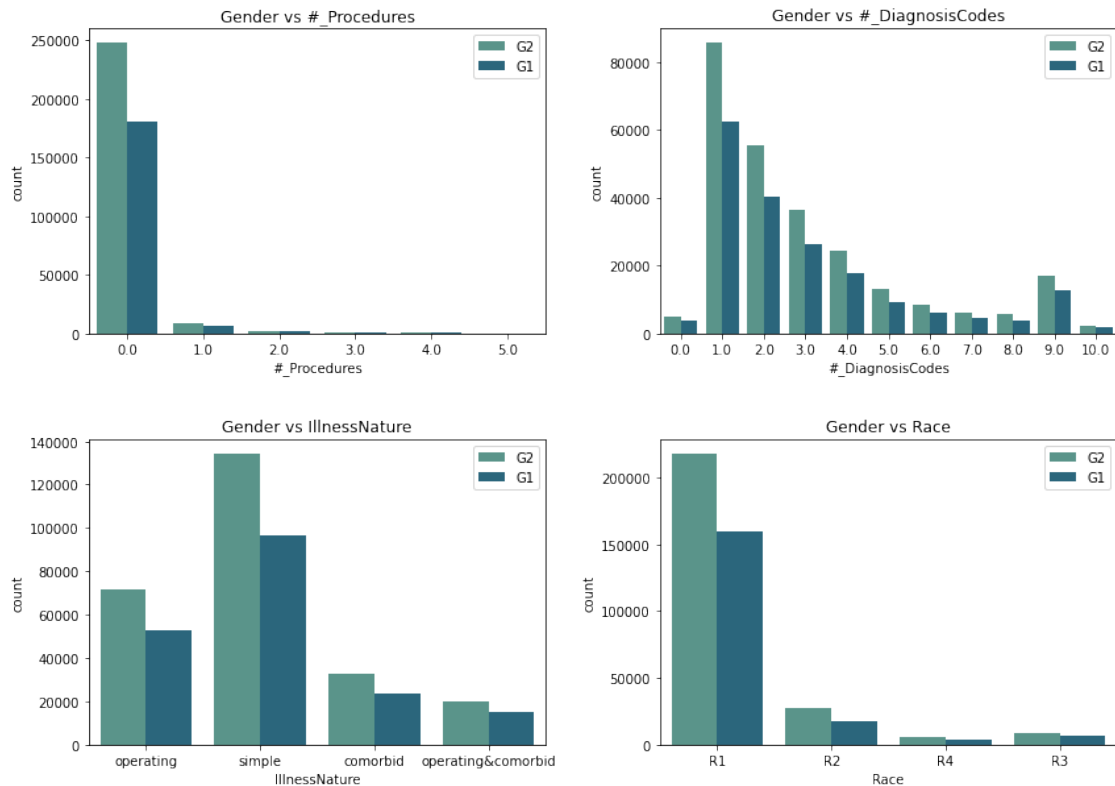
```

ax2.set_title('Gender vs #_DiagnosisCodes')
ax3.set_title('Gender vs IllnessNature')
ax4.set_title('Gender vs Race')

plt.subplots_adjust(wspace=0.25)
plt.subplots_adjust(hspace=0.35)

plt.show()

```



```

[:]: isinstance(train_fin3.iloc[:,2],str)

```

```

[:]: False

```

```

[:]: cat_train= train_fin3.select_dtypes(include=['object']).copy()
cat_train.drop(['PotentialFraud'],axis=1,inplace=True)
cat_train.head()

```

```

[:]: Gender Race IllnessNature
0      G2   R1      operating
1      G1   R1         simple
2      G1   R1         simple
3      G2   R2      operating
4      G2   R1      comorbid

```

5.10 Calculating the Correlation between the Categorical Variables Using CRAMER's V

```
[ ]: contTable_GR= pd.crosstab(cat_train['Gender'],cat_train['Race'])
print("The contTable between Gender and Race Varibales")
print(contTable_GR)
dof_GR= min(contTable_GR.shape[0],contTable_GR.shape[1])-1
print("The number of Degrees of Freedom in Gender vs Race",dof_GR)
print("="*100)

contTable_GI= pd.crosstab(cat_train["Gender"],cat_train["IllnessNature"])
print("The contTable between Gender and IllnessNature Varibales")
print(contTable_GI)
dof_GI= min(contTable.shape[0],contTable.shape[1])-1
print("The number of Degrees of Freedom in Gender vs IllnessNature",dof_GI)
print("="*100)

contTable_RI= pd.crosstab(cat_train["Race"],cat_train["IllnessNature"])
print("The contTable between Race and IllnessNature Varibales")
print(contTable_RI)
dof_RI= min(contTable.shape[0],contTable.shape[1])-1
print("The number of Degrees of Freedom in Race vs IllnessNature",dof_RI)
```

The contTable between Gender and Race Varibales

Race	R1	R2	R3	R4
Gender				
G1	159512	17420	6968	4064
G2	217324	27073	8759	5448

The number of Degrees of Freedom in Gender vs Race 1

=====

The contTable between Gender and IllnessNature Varibales

IllnessNature	comorbid	operating	operating&comorbid	simple
Gender				
G1	23860	52639	14946	96519
G2	32511	71896	20209	133988

The number of Degrees of Freedom in Gender vs IllnessNature 1

=====

The contTable between Race and IllnessNature Varibales

IllnessNature	comorbid	operating	operating&comorbid	simple
Race				
R1	47435	105005	29711	194685
R2	5800	12257	3445	22991
R3	1965	4584	1234	7944
R4	1171	2689	765	4887

The number of Degrees of Freedom in Race vs IllnessNature 1

```
[ ]: pip install researchpy
```

Collecting researchpy

Downloading <https://files.pythonhosted.org/packages/8f/20/e2787cd5eb6d4cfd6bc1f42f5218ca6a2f5552a9fcf021095cd07a4071fd/researchpy-0.3.2-py3-none-any.whl>

Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from researchpy) (1.1.5)

Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from researchpy) (1.19.5)

Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from researchpy) (1.4.1)

Requirement already satisfied: patsy in /usr/local/lib/python3.7/dist-packages (from researchpy) (0.5.1)

Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (from researchpy) (0.10.2)

Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas->researchpy) (2018.9)

Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->researchpy) (2.8.1)

Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy->researchpy) (1.15.0)

Installing collected packages: researchpy

Successfully installed researchpy-0.3.2

```
[ ]: import researchpy
```

```
crosstab_GR,res_GR= researchpy.
```

```
    ↳crosstab(train_fin3['Gender'],train_fin3['Race'], test='chi-square')
```

```
print("The CramersV value between Gender and Race Varibales")
```

```
print(res_GR)
```

```
print("="*100)
```

```
crosstab_GI,res_GI= researchpy.
```

```
    ↳crosstab(train_fin3['Gender'],train_fin3['IllnessNature'], test='chi-square')
```

```
print("The CramersV value between Gender and IllnessNature Varibales")
```

```
print(res_GI)
```

```
print("="*100)
```

```
crosstab_RI,res_RI= researchpy.
```

```
    ↳crosstab(train_fin3['Race'],train_fin3['IllnessNature'], test='chi-square')
```

```
print("The CramersV value between Race and IllnessNature Varibales")
```

```
print(res_RI)
```

The CramersV value between Gender and Race Varibales

	Chi-square test	results
0	Pearson Chi-square (3.0) =	199.6508

```

1          p-value =      0.0000
2          Cramer's V =    0.0211

```

```

=====
=====

```

The CramersV value between Gender and IllnessNature Varibales

```

          Chi-square test results
0 Pearson Chi-square ( 3.0) =    9.9972
1          p-value =      0.0186
2          Cramer's V =    0.0047

```

```

=====
=====

```

The CramersV value between Race and IllnessNature Varibales

```

          Chi-square test results
0 Pearson Chi-square ( 9.0) =   24.0624
1          p-value =      0.0042
2          Cramer's V =    0.0042

```

```

[ ]: #Source: https://www.kaggle.com/chrisbss1/cramer-s-v-correlation-matrix
from scipy.stats import chi2_contingency

def cramers_V(var1,var2) :
    crosstab =np.array(pd.crosstab(var1,var2, rownames=None, colnames=None)) #
    →Cross table building
    stat = chi2_contingency(crosstab)[0] # Keeping of the test statistic of the
    →Chi2 test
    obs = np.sum(crosstab) # Number of observations
    mini = min(crosstab.shape)-1 # Take the minimum value between the columns and
    →the rows of the cross table
    return (stat/(obs*mini))

[ ]: #Source: https://www.kaggle.com/chrisbss1/cramer-s-v-correlation-matrix
rows= []

for var1 in cat_train:
    col = []
    for var2 in cat_train:
        cramers =cramers_V(cat_train[var1], cat_train[var2]) # Cramer's V test
        col.append(round(cramers,2)) # Keeping of the rounded value of the Cramer's
        →V
    rows.append(col)

cramers_results = np.array(rows)
cat_corr = pd.DataFrame(cramers_results, columns = cat_train.columns, index
    →=cat_train.columns)

[ ]: print("The correlation between all the categorical columns")
print(cat_corr)

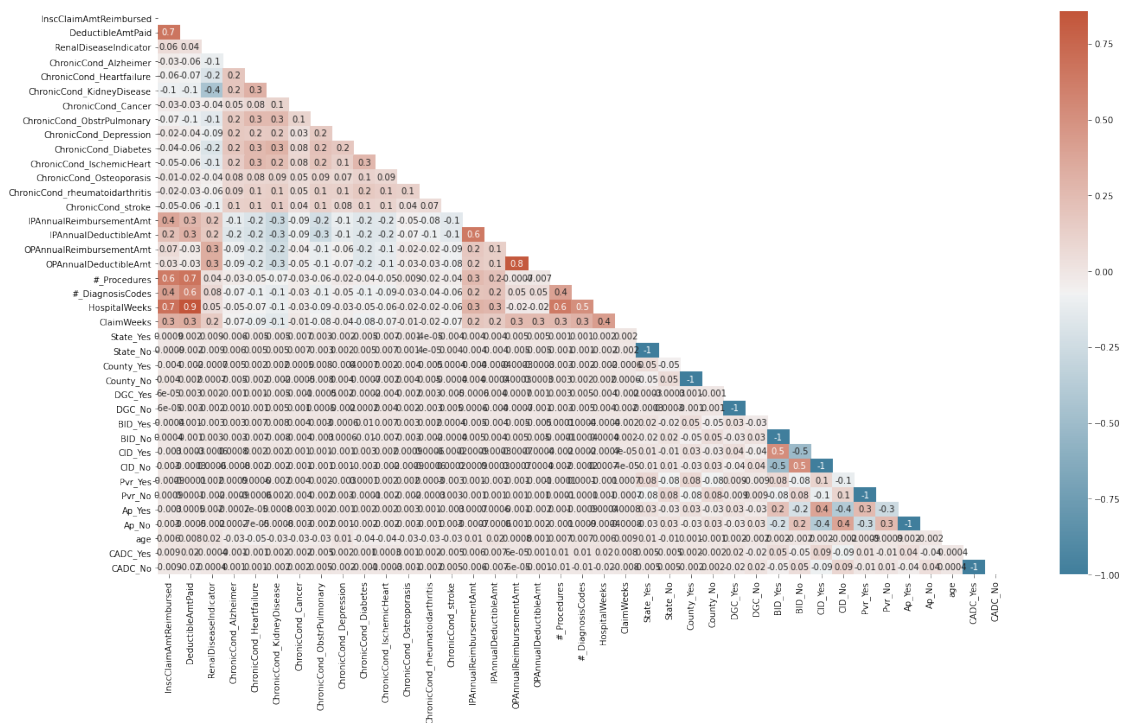
```

The correlation between all the categorical columns

	Gender	Race	IllnessNature
Gender	1.0	0.0	0.0
Race	0.0	1.0	0.0
IllnessNature	0.0	0.0	1.0

5.11 Correlation Analysis amongst the Numerical Variables

```
[ ]: plt.figure(figsize=(22,12))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
matrix = np.triu(train_fin3.corr())
sns.heatmap(train_fin3.corr(),annot=True,cmap=cmap,fmt='.1g', mask= matrix)
plt.show()
```



5.11.1 Dropping the Potential Fraud Column from the Train and the CV Datasets

```
[ ]: train_fin3.drop(['PotentialFraud'],axis=1, inplace=True)
cv_fin3.drop(['PotentialFraud'],axis=1,inplace=True)

[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n4.pkl','wb') as f:
    tr_df = pickle.dump(train_fin3,f)

with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n4.pkl','wb') as f:
    cv_df = pickle.dump(cv_fin3,f)
```



```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_fin_n4.pkl','rb') as f:
      tr_df = pickle.load(f)
      with open('/content/drive/MyDrive/Colab Notebooks/cv_fin_n4.pkl','rb') as f:
        cv_df = pickle.load(f)
```

```
[ ]: train_fin4.head()
```

```
[ ]:      InscClaimAmtReimbursed  DeductibleAmtPaid  ...  CADC_Yes  CADC_No
0                90                0.0  ...  0.465483  0.534517
1               5000              1068.0  ...  0.482759  0.517241
2                400                0.0  ...  0.456727  0.543273
3                 30                0.0  ...  0.456727  0.543273
4               2000              1068.0  ...  0.372093  0.627907
```

[5 rows x 42 columns]

5.11.2 Creating Dummies for the Gender, Race and the IllnessNature Variables in the Train Dataset

```
[ ]: gender_dummies= pd.get_dummies(train_fin4['Gender'])
      train_fin4= train_fin4.join(gender_dummies)
```

```
[ ]: race_dummies= pd.get_dummies(train_fin4['Race'])
      train_fin4= train_fin4.join(race_dummies)
```

```
[ ]: illn_dummies= pd.get_dummies(train_fin4['IllnessNature'])
      train_fin4= train_fin4.join(illn_dummies)
```

5.11.3 Dropping the Gender, Race and the IllnessNature Variables in the Train Dataset

```
[ ]: train_fin4.drop(['Gender','Race','IllnessNature'], axis=1, inplace=True)
```

```
[ ]: train_fin4.head()
```

```
[ ]:      InscClaimAmtReimbursed  DeductibleAmtPaid  ...  operating&comorbid  simple
0                90                0.0  ...                0                0
1               5000              1068.0  ...                0                1
2                400                0.0  ...                0                1
3                 30                0.0  ...                0                0
4               2000              1068.0  ...                0                0
```

[5 rows x 49 columns]

5.11.4 Creating Dummies for the Gender, Race and the IllnessNature Variables in the CV Dataset

```
[ ]: cv_gender_dummies= pd.get_dummies(cv_fin4['Gender'])
cv_fin4= cv_fin4.join(cv_gender_dummies)

[ ]: cv_race_dummies= pd.get_dummies(cv_fin4['Race'])
cv_fin4= cv_fin4.join(cv_race_dummies)

[ ]: cv_illn_dummies= pd.get_dummies(cv_fin4['IllnessNature'])
cv_fin4= cv_fin4.join(cv_illn_dummies)
```

5.11.5 Dropping the Gender, Race and the IllnessNature Variables in the CV Dataset

```
[ ]: cv_fin4.drop(['Gender', 'Race', 'IllnessNature'], axis=1, inplace=True)

[ ]: cv_fin4.head()
```

```
[ ]:      InscClaimAmtReimbursed  DeductibleAmtPaid  ...  operating&comorbid  simple
0                300                0.0  ...                1                0
1                 80               50.0  ...                0                1
2                200                0.0  ...                0                1
3                200                0.0  ...                0                0
4                300                0.0  ...                0                1
```

[5 rows x 49 columns]

```
[ ]:
```

5.11.6 Scaling the Numerical Features using the SKLearn StandardScaler

```
[ ]: scaler= StandardScaler()
num_cols=['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'IPAnnualReimbursementAmt', 'IPAnnualDec

for i in tqdm(num_cols):
    scaler.fit(train_fin4[i].values.reshape(-1,1))
    train_fin4[i]= scaler.transform(train_fin4[i].values.reshape(-1,1))
    cv_fin4[i]= scaler.transform(cv_fin4[i].values.reshape(-1,1))
```

100%| 11/11 [00:00<00:00, 40.17it/s]

```
[ ]: train_fin4.head()

[ ]:      InscClaimAmtReimbursed  DeductibleAmtPaid  RenalDiseaseIndicator  ...  R2  R3
R4
0                -0.237753                -0.286684                0  ...  0  0
0
1                 1.042252                 3.605110                1  ...  0  0
0
```

```

2          -0.156938          -0.286684          0 ... 0 0
0
3          -0.253395          -0.286684          0 ... 1 0
0
4          0.260172           3.605110           0 ... 0 0
0

```

[5 rows x 49 columns]

```
[ ]: cv_fin4.head()
```

```

[ ]:   InscClaimAmtReimbursed  DeductibleAmtPaid  ...  operating&comorbid  simple
0          -0.183007          -0.286684  ...                1          0
1          -0.240360          -0.104484  ...                0          1
2          -0.209077          -0.286684  ...                0          1
3          -0.209077          -0.286684  ...                0          0
4          -0.183007          -0.286684  ...                0          1

```

[5 rows x 49 columns]

```

[ ]: print(train_fin4.shape)
      print(train_y.shape)
      print(cv_fin4.shape)
      print(cv_y.shape)

```

```

(446568, 49)
(446568,)
(111643, 49)
(111643,)

```

5.11.7 Replacing the Yes and No with 1 and 0 in the Train_y and CV_y datasets

```

[ ]: train_y= train_y.map({'Yes':1,'No':0})
      cv_y= cv_y.map({'Yes':1,'No':0})
      print(train_y.head())
      print('='*50)
      print(cv_y.head())

```

```

0    0
1    1
2    1
3    1
4    1
Name: PotentialFraud, dtype: int64
=====
0    1
1    0
2    0

```

```
3    0
4    1
Name: PotentialFraud, dtype: int64
```

```
[ ]: with open('/content/drive/MyDrive/Colab Notebooks/train_hc_fin.pkl','wb') as tr_df:
      pickle.dump(train_fin4,tr_df)
with open('/content/drive/MyDrive/Colab Notebooks/cv_hc_fin.pkl','wb') as cv_df:
      pickle.dump(cv_fin4,cv_df)

[ ]: with open('/home/megha_murthy_n/train_hc_fin.pkl','rb') as tr_df:
      train_fin4= pd.read_pickle(tr_df)
with open('/home/megha_murthy_n/cv_hc_fin.pkl','rb') as cv_df:
      cv_fin4= pd.read_pickle(cv_df)

[ ]: with open('/home/megha_murthy_n/train_y.pkl','rb') as tr_y:
      train_y= pd.read_pickle(tr_y)
with open('/home/megha_murthy_n/cv_y.pkl','rb') as cv_y:
      cv_y= pd.read_pickle(cv_y)

[ ]: print(train_fin4.shape)
      print(train_y.shape)
      print(cv_fin4.shape)
      print(cv_y.shape)
```

```
(446568, 49)
(446568,)
(111643, 49)
(111643,)
```

6 Confusion Matrix

```
[ ]: def plot_confusion_matrix(test_y, predict_y):
      predictions = []

      for i in predict_y:
          if i>=0.5:
              predictions.append(1)
          else:
              predictions.append(0)

      C = confusion_matrix(test_y, predictions)
      print("The Weighted Recall Score: ",recall_score(test_y,
      →predictions,average='weighted'))
      print("The Weighted Precision Score: ",precision_score(test_y,
      →predictions,average='weighted'))
      print("The Weighted F1 Score: ",f1_score(test_y,
      →predictions,average='weighted'))
```

```

print("="*100)

# print("The Micro Recall Score: ", recall_score(test_y,
→ predict_y, average='micro'))
# print("The Micro Precision Score: ", precision_score(test_y,
→ predict_y, average='micro'))
# print("The Micro F1 Score: ", f1_score(test_y, predict_y, average='micro'))

print("="*100)

A = (((C.T)/(C.sum(axis=1))).T)
B = (C/C.sum(axis=0))

labels = [0,1]
cmap=sns.light_palette("green")
# representing A in heatmap format
print("-"*50, "Confusion matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,
→ yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*50, "Precision matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,
→ yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of columns in precision matrix", B.sum(axis=0))

# representing B in heatmap format
print("-"*50, "Recall matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,
→ yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of rows in precision matrix", A.sum(axis=1))

```

```

[ ]: def best_thresholds(test_y, predict_y):
    fpr, tpr, threshold = roc_curve(test_y, predict_y)
    best_t= threshold[np.argmax(tpr*(1-fpr))]

```

```

print('best train threshold :',best_t)

predictions = []
for i in predict_y:
    if i>=best_t:
        predictions.append(1)
    else:
        predictions.append(0)
plot_confusion_matrix(test_y,predictions)

```

7 Building a Random Model

```

[:]: train_data_len = train_fin4.shape[0]
      cv_data_len = cv_fin4.shape[0]
      op_list=[0,1]

      train_predicted_y = np.zeros(train_data_len)

      for i in range(train_data_len):
          train_predicted_y[i] = random.choice(op_list)

      cv_predicted_y = np.zeros(cv_data_len)

      for i in range(cv_data_len):
          cv_predicted_y[i] = random.choice(op_list)

[:]: plot_confusion_matrix(train_y, train_predicted_y)

```

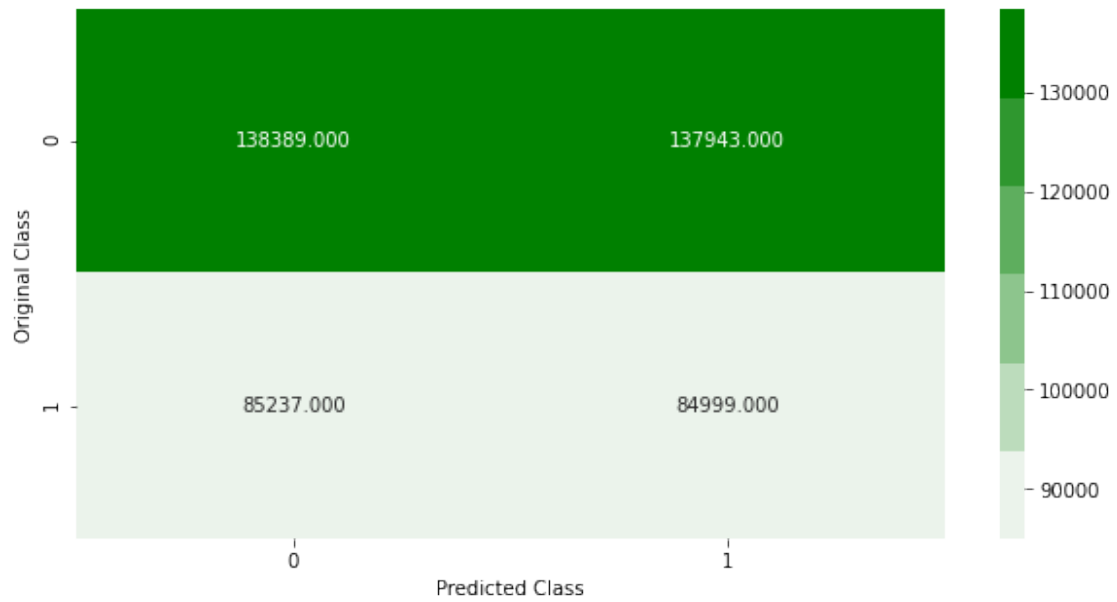
The Weighted Recall Score: 0.5002328872646495
 The Weighted Precision Score: 0.5282732472163528
 The Weighted F1 Score: 0.5073871505917014

=====

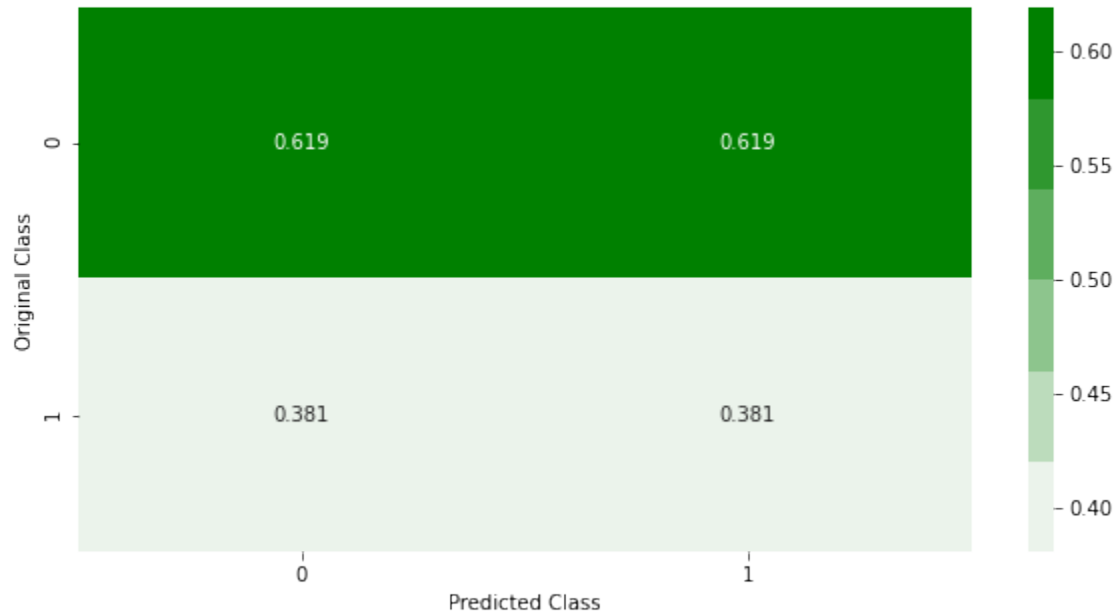
The Micro Recall Score: 0.5002328872646495
 The Micro Precision Score: 0.5002328872646495
 The Micro F1 Score: 0.5002328872646495

=====

----- Confusion matrix

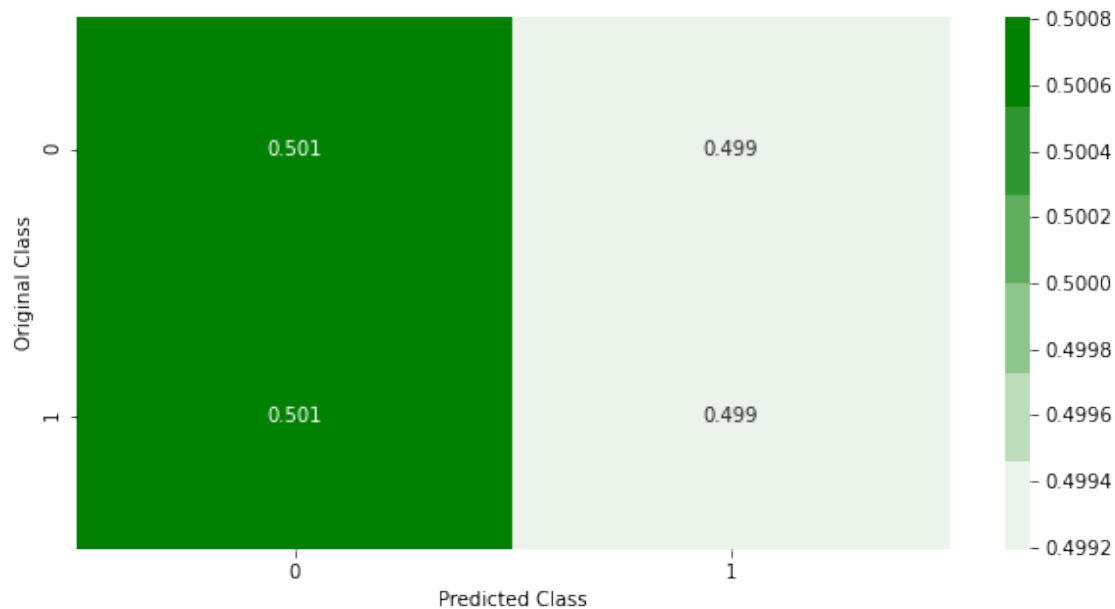


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8 Experiment 1

8.1 Supervised Classification Models WITHOUT correcting for Class Imbalance (without adjusting the class weights parameter)

8.2 Logistic Regression on Datasets without correcting for Class Imbalance

```
[ ]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV

param = { 'C': [0.00001,0.01, 1, 100,1000], 'penalty' : ['l1','l2'] }
lr = LogisticRegression()
lr_tune = RandomizedSearchCV(lr,param,cv=10,n_jobs=-1,verbose=1)
lr_tune.fit(train_fin4,train_y)

print('best parameter :',lr_tune.best_params_)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 46 tasks      | elapsed: 1.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 4.9min finished
```

best parameter : {'penalty': 'l2', 'C': 0.01}


```
[ ]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV

lr_best = LogisticRegression(C=0.01,penalty='l2')
lr_best.fit(train_fin4,train_y)

sig_clf = CalibratedClassifierCV(lr_best, method="sigmoid")
sig_clf.fit(train_fin4,train_y)

train_y_pred = sig_clf.predict_proba(train_fin4)
train_y_pred = train_y_pred[:,1]
cv_y_pred = sig_clf.predict_proba(cv_fin4)
cv_y_pred = cv_y_pred[:,1]
```

8.2.1 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)
```

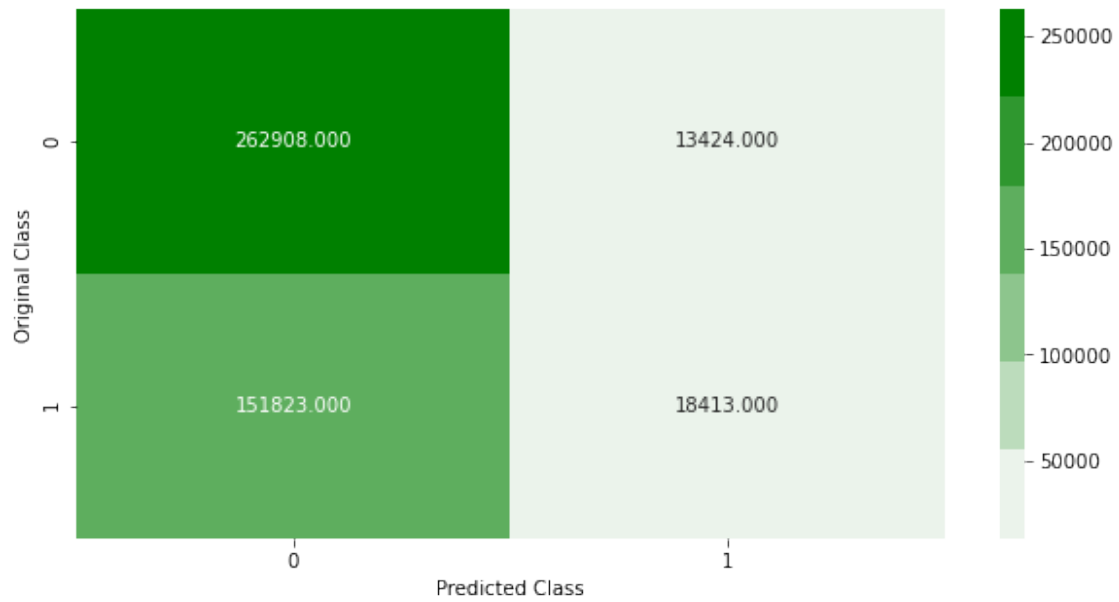
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.6299622901775318

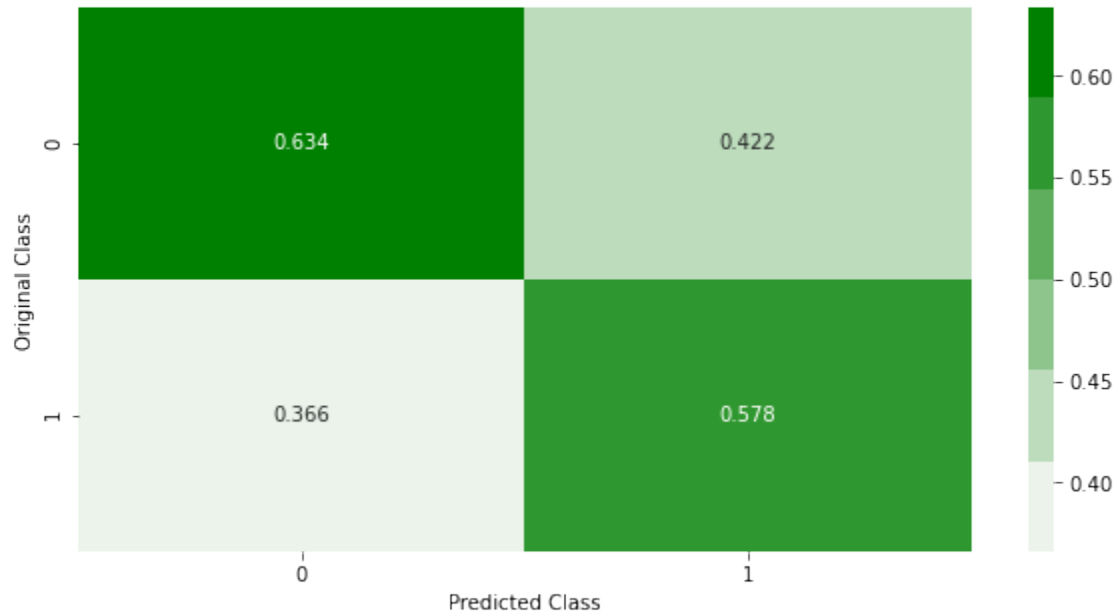
The Weighted Precision Score: 0.6127396073156689

The Weighted F1 Score: 0.5402972828309037

```
=====
=====
=====
=====
----- Confusion matrix
-----
```

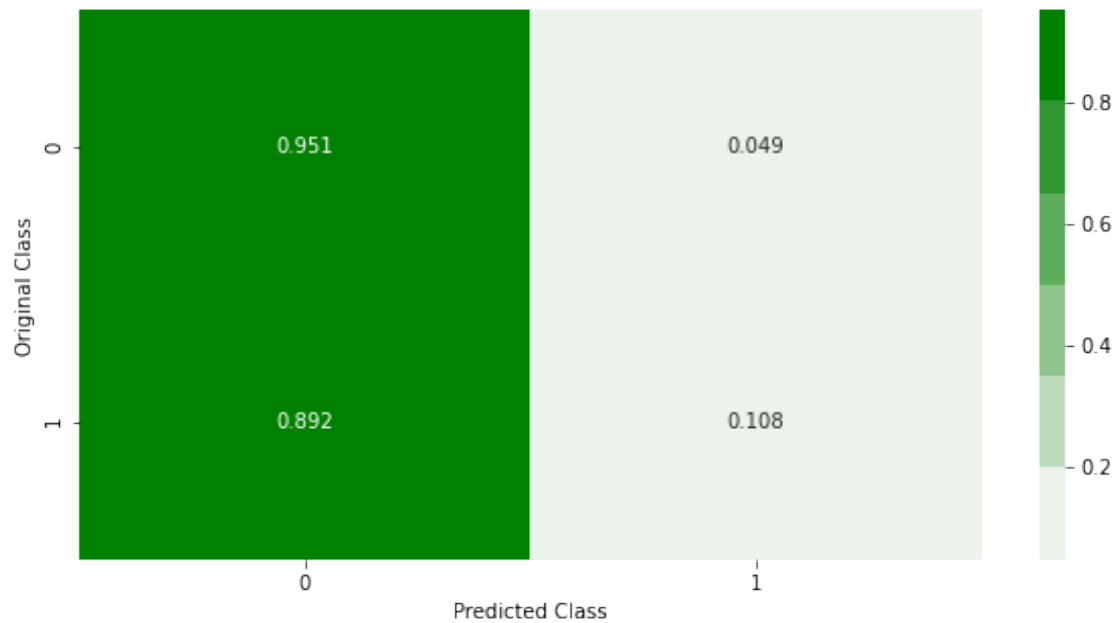


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

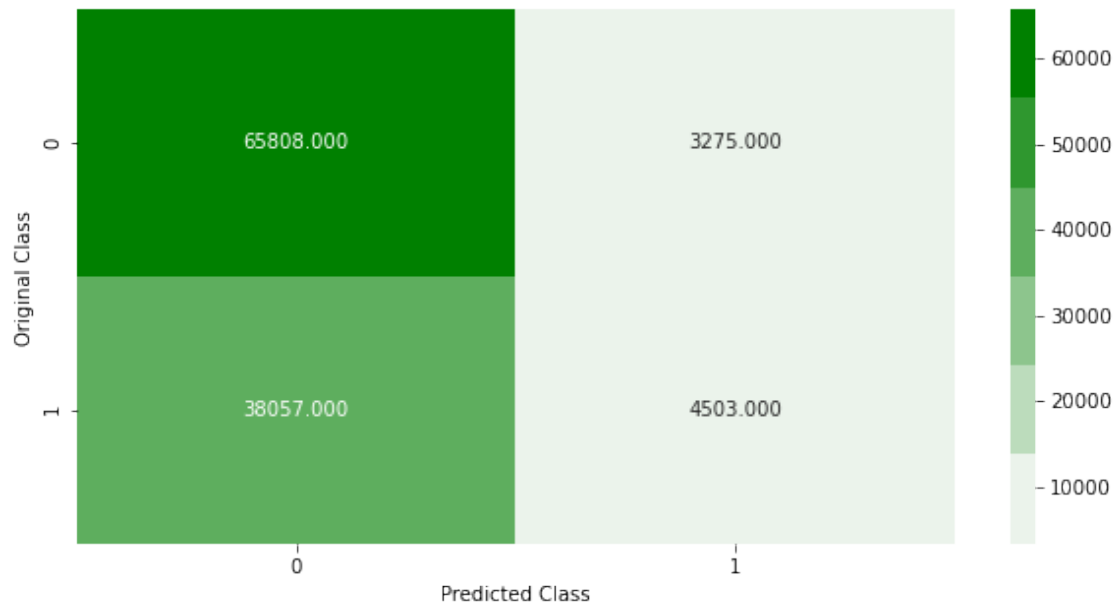
Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.6297842229248587

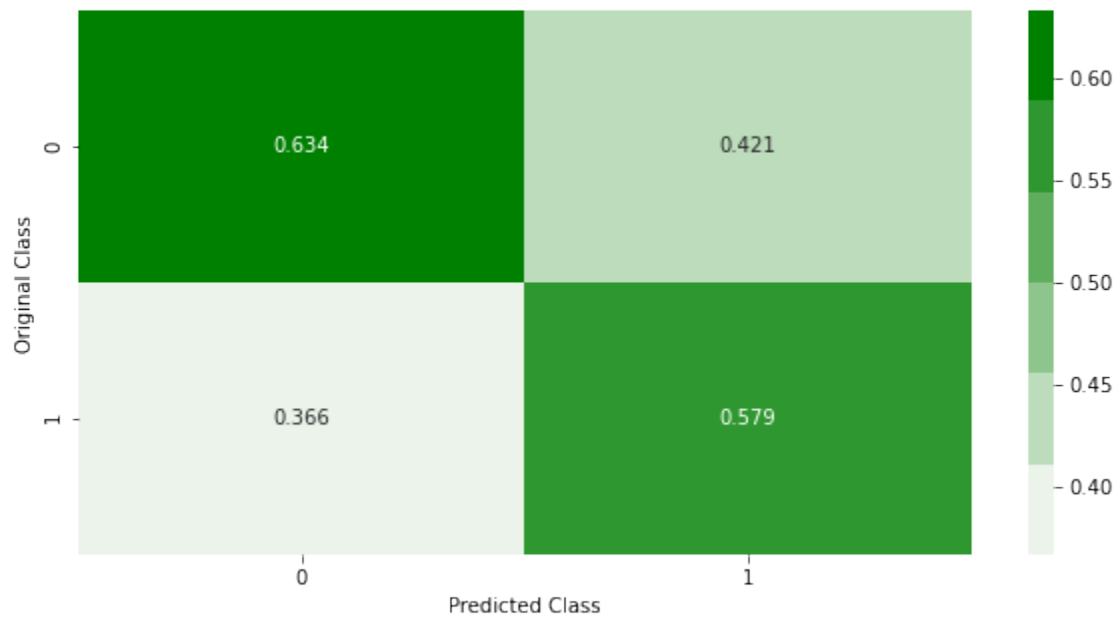
The Weighted Precision Score: 0.6127578633856098

The Weighted F1 Score: 0.5391079064341675

```
=====
=====
=====
----- Confusion matrix
-----
```

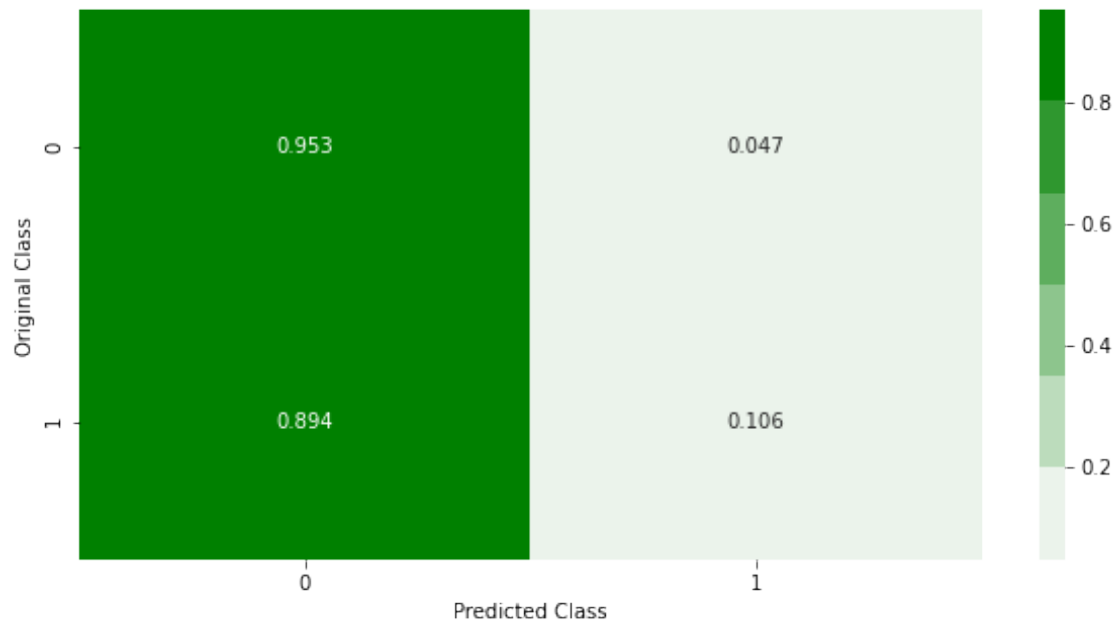


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.2.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

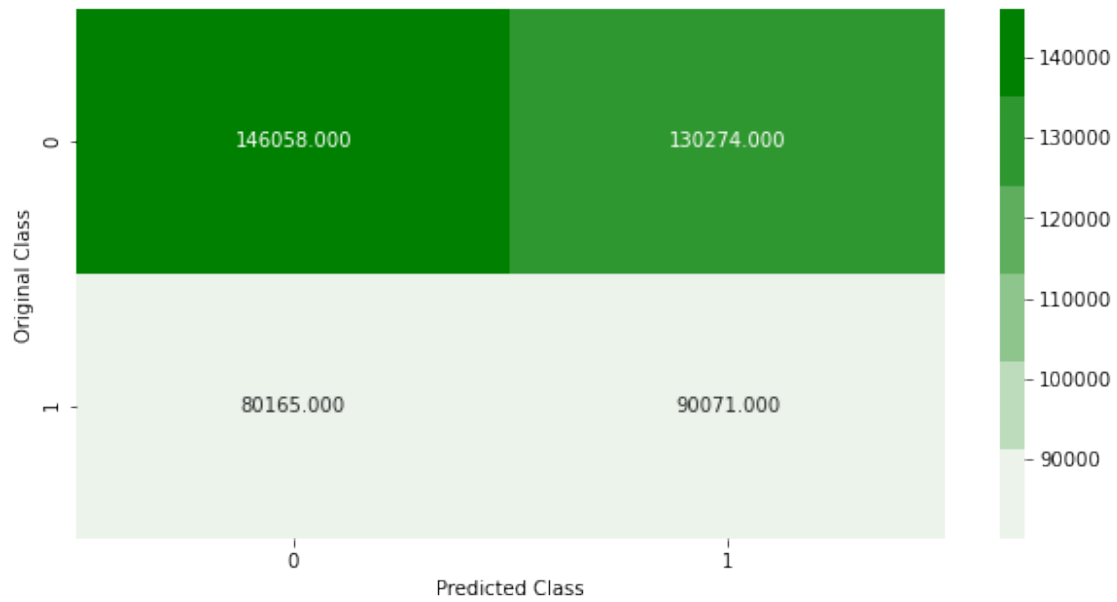
best train threshold : 0.3641454815181372

The Weighted Recall Score: 0.5287638164848355

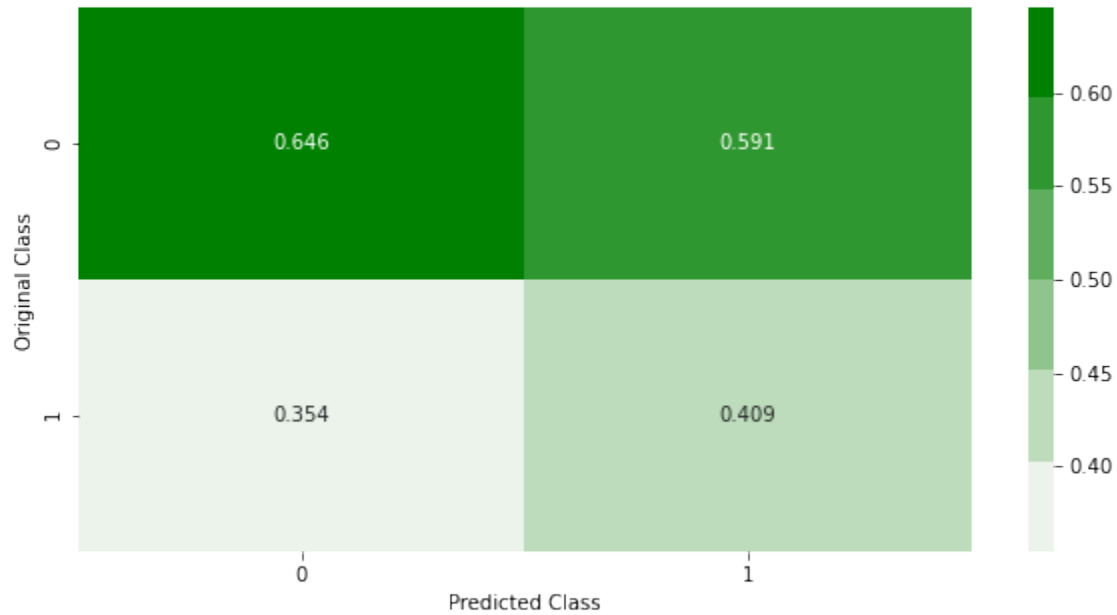
The Weighted Precision Score: 0.5553421906395838

The Weighted F1 Score: 0.5354989599583485

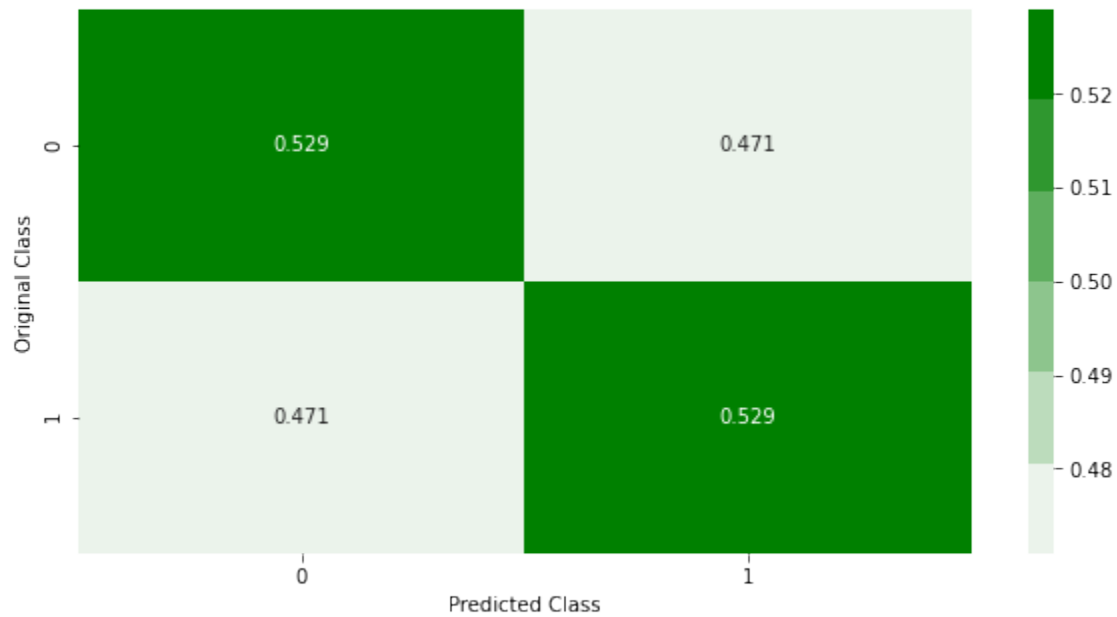
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix

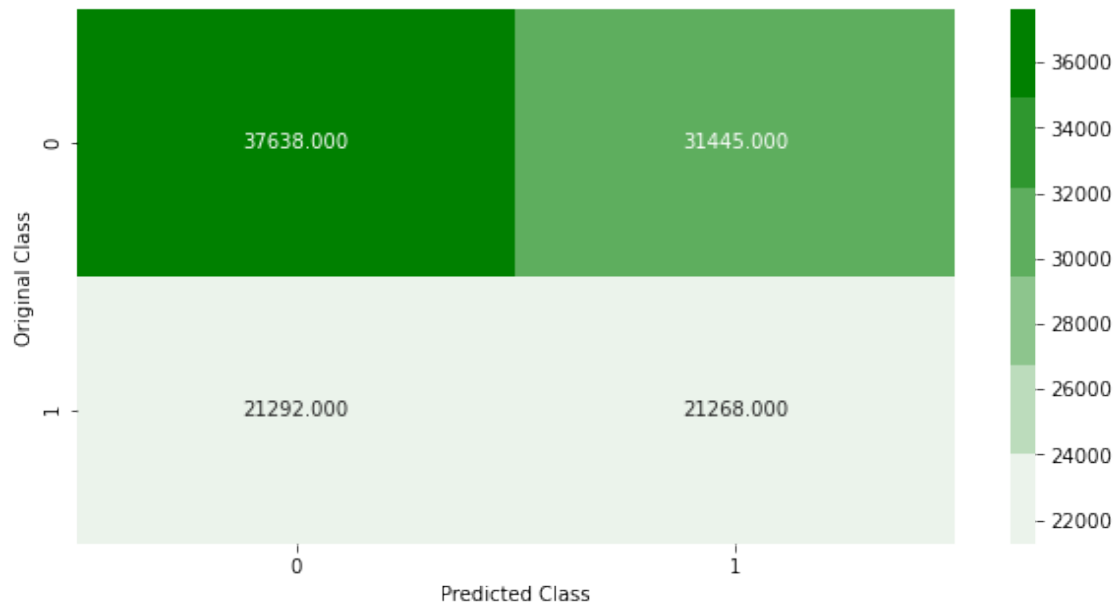


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

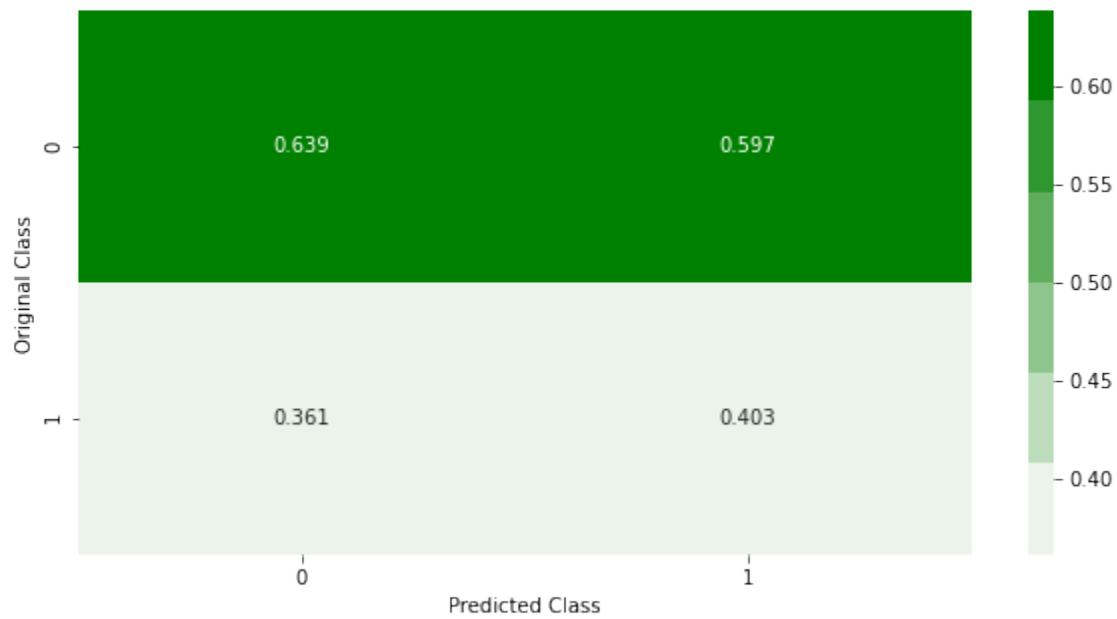


Sum of rows in precision matrix [1. 1.]
 Confusion Matrix for Best Thresold for the CV Data
 best train threshold : 0.426617303742292
 The Weighted Recall Score: 0.5276282435978968
 The Weighted Precision Score: 0.5490197356370331
 The Weighted F1 Score: 0.5340655335640317

```
=====
=====
=====
=====
----- Confusion matrix
-----
```

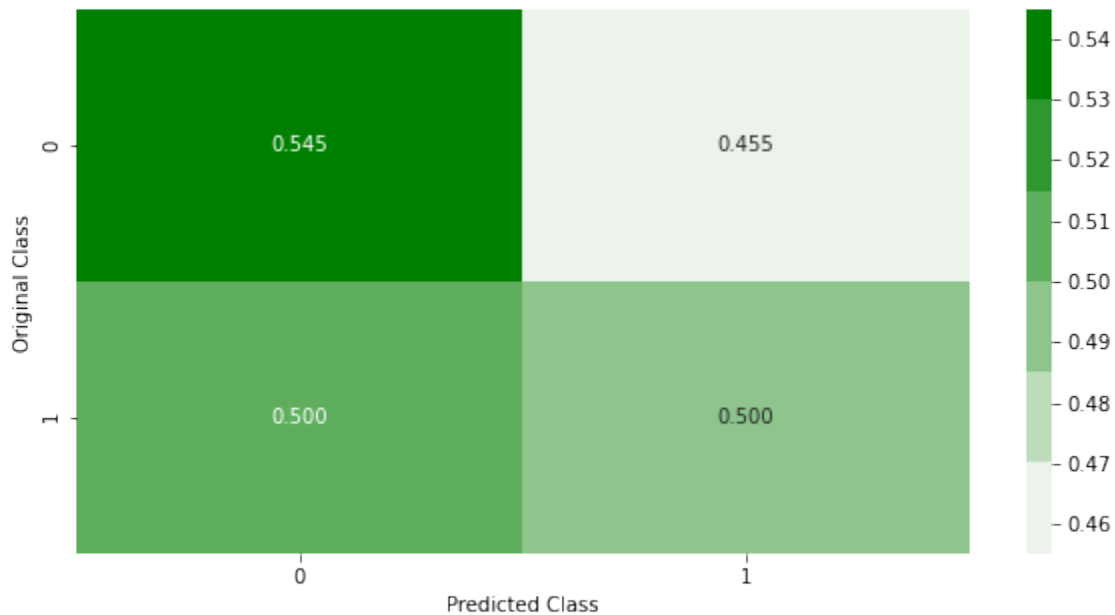


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.3 Random Forest Classifier without correcting for Class Imbalance

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(class_weight='balanced')
param = { 'n_estimators': [10,50,100], 'max_depth' : [
    → [2,6,10,14], 'min_samples_split': [5,50,100,250], 'criterion' : ['gini']}

rf_tune = RandomizedSearchCV(rf,param,cv=10,n_jobs=-1,verbose=1)
rf_tune.fit(train_fin4, train_y)
print('best parameter :',rf_tune.best_params_)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 46 tasks | elapsed: 18.0min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 40.0min finished
```

```
best parameter : {'n_estimators': 50, 'min_samples_split': 5, 'max_depth': 14,
'criterion': 'gini'}
```

8.3.1 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)
```

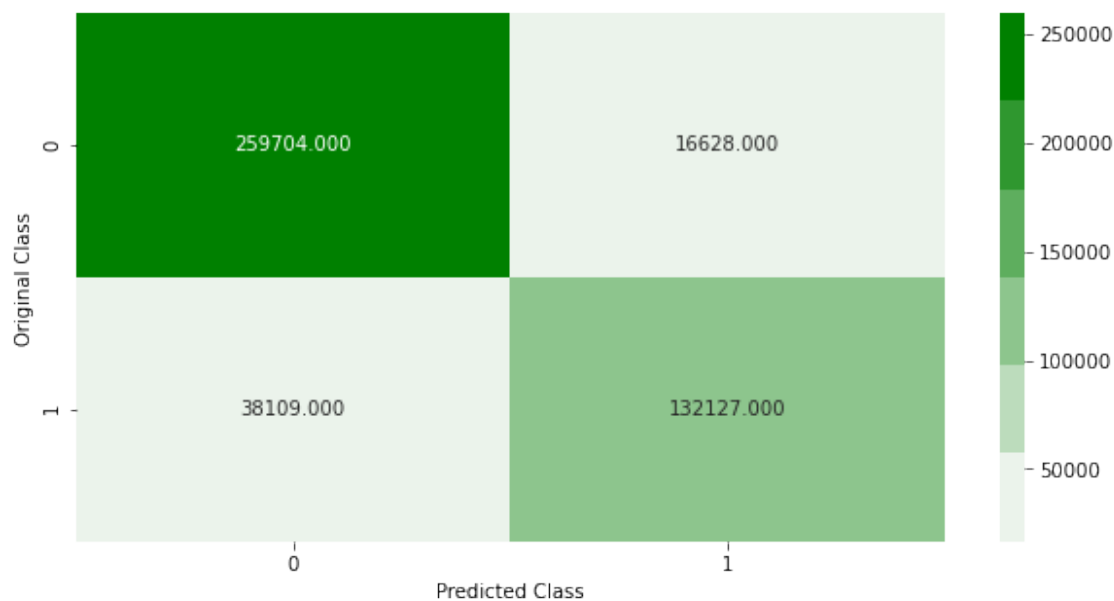
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8774274018738467

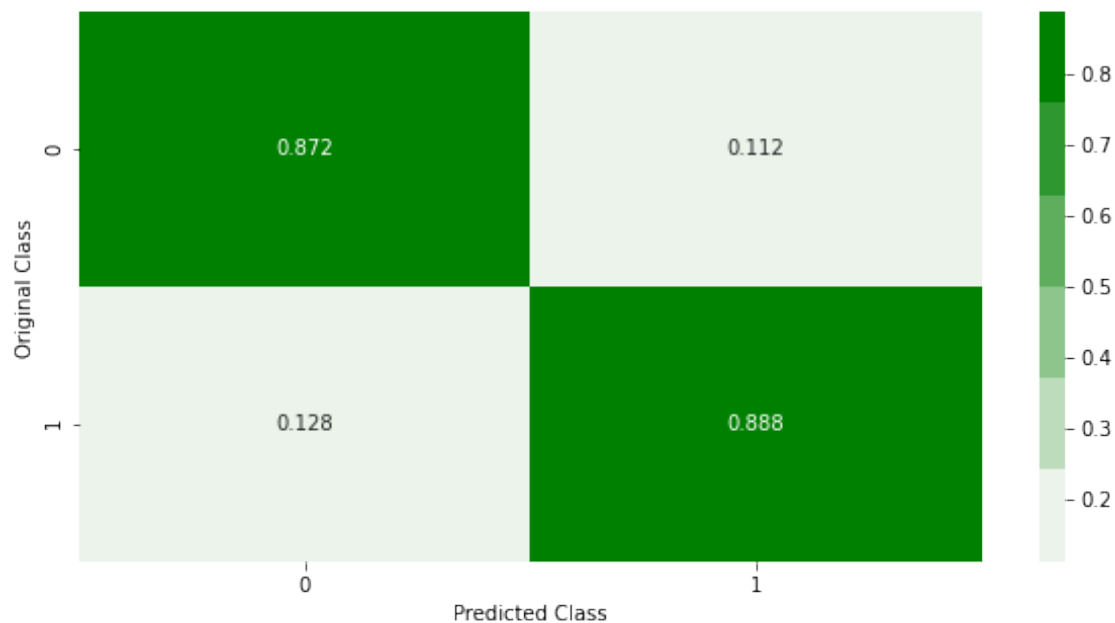
The Weighted Precision Score: 0.8782057823160677

The Weighted F1 Score: 0.8755933135116761

```
=====
=====
=====
----- Confusion matrix
-----
```

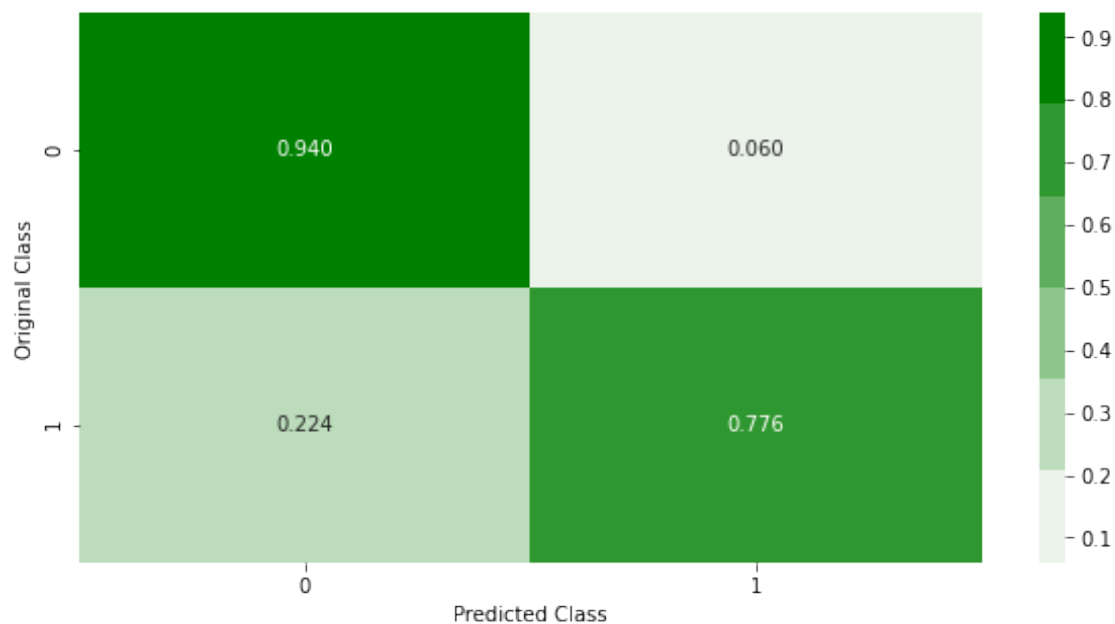


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

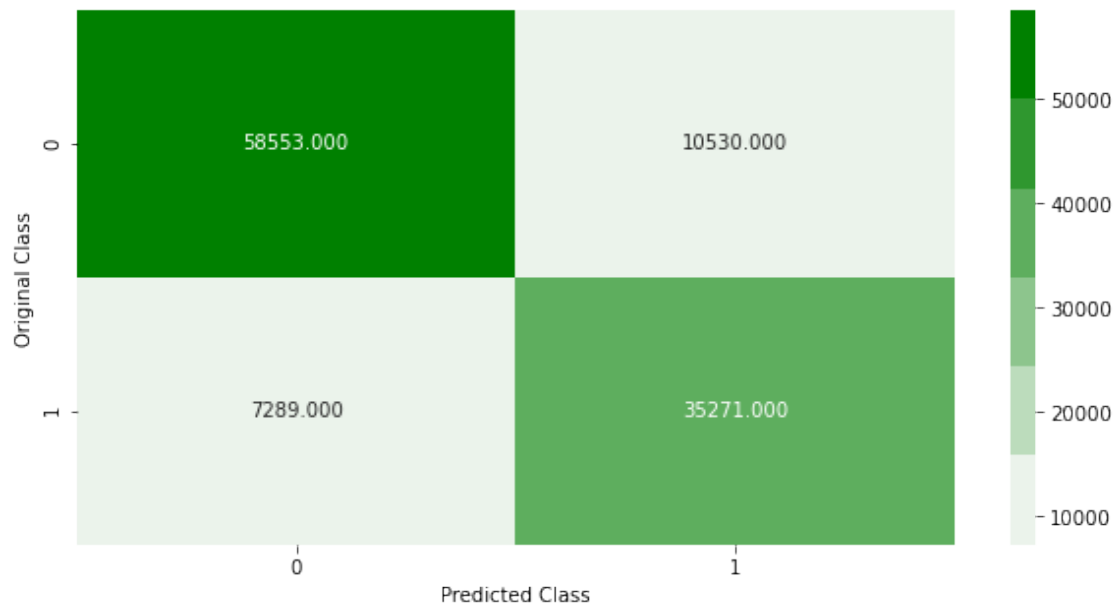
The Weighted Recall Score: 0.8403930385245828
The Weighted Precision Score: 0.8438535122626364
The Weighted F1 Score: 0.8414032169680887

=====

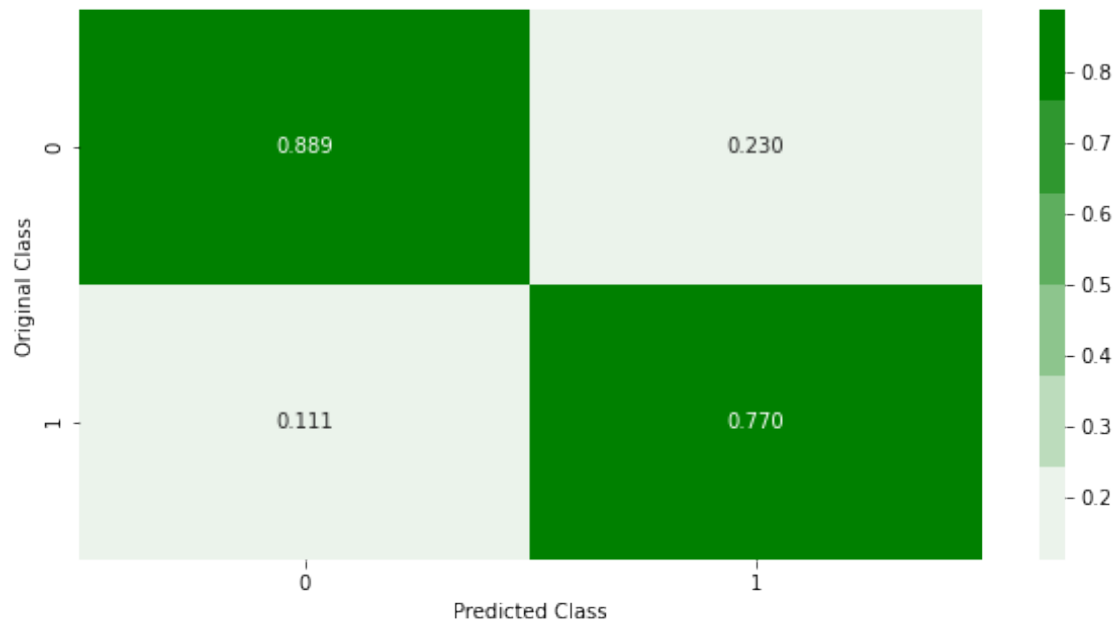
=====

=====

----- Confusion matrix

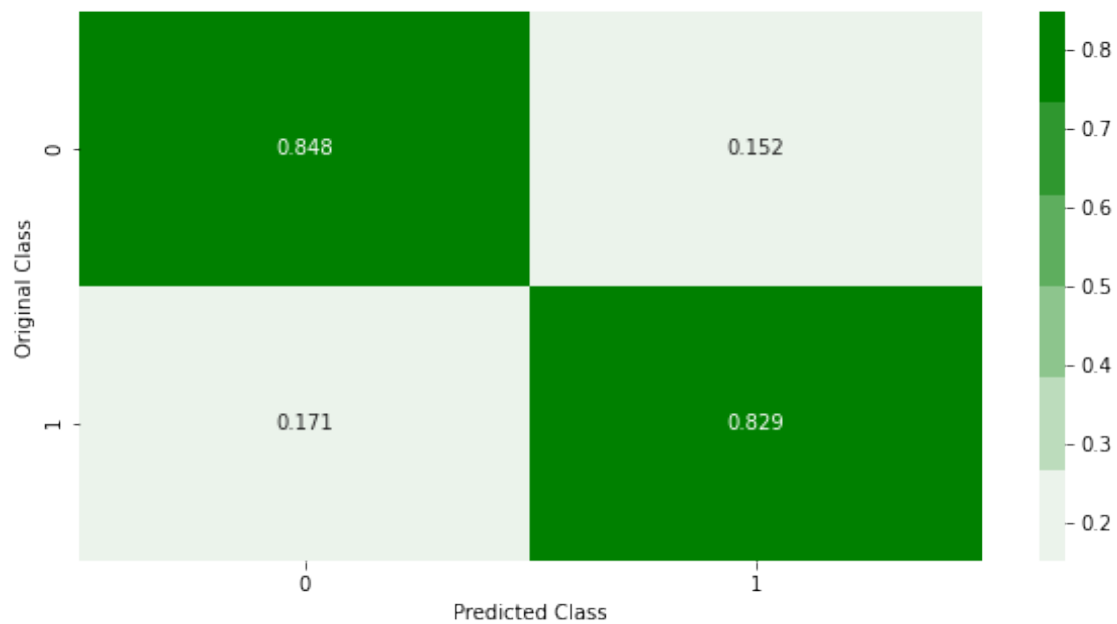


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.3.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

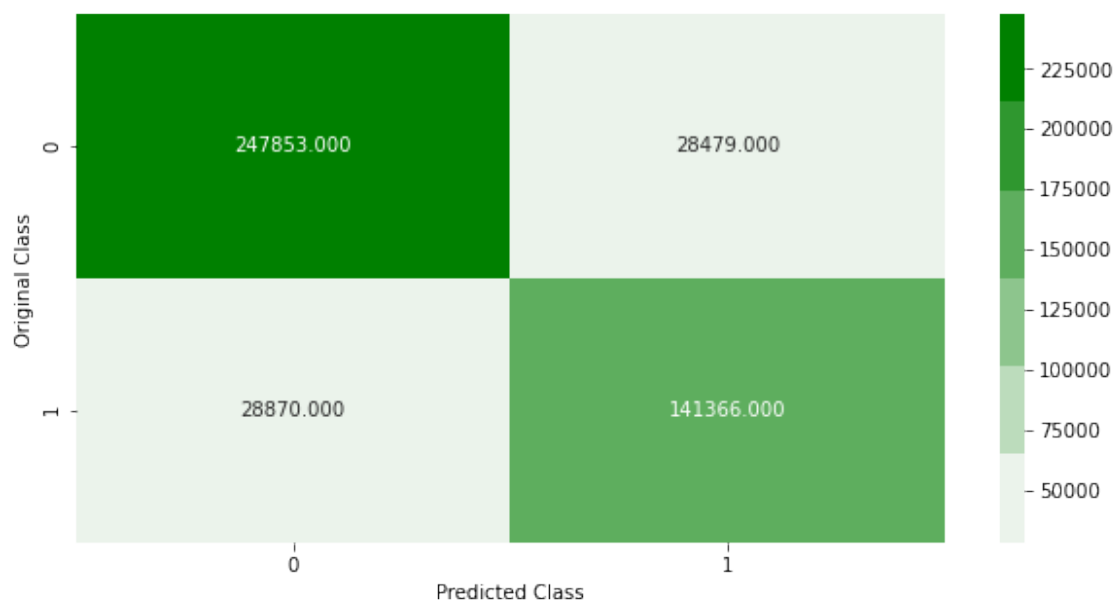
best train threshold : 0.4054152130536083

The Weighted Recall Score: 0.8715783486501496

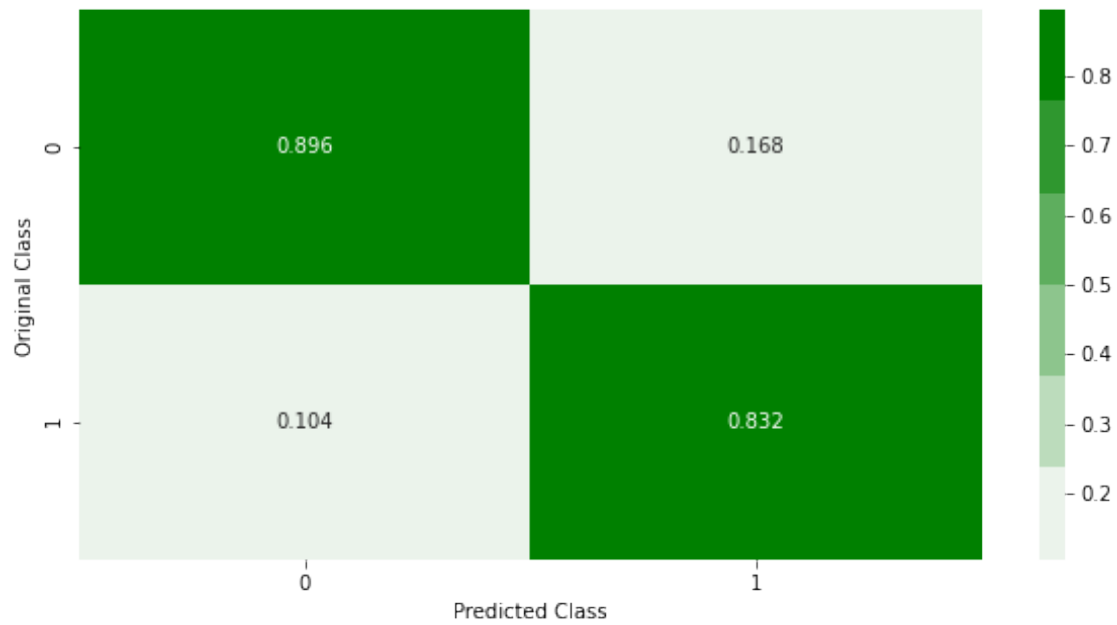
The Weighted Precision Score: 0.8715228830366565

The Weighted F1 Score: 0.8715499196943943

```
=====
=====
=====
----- Confusion matrix
-----
```

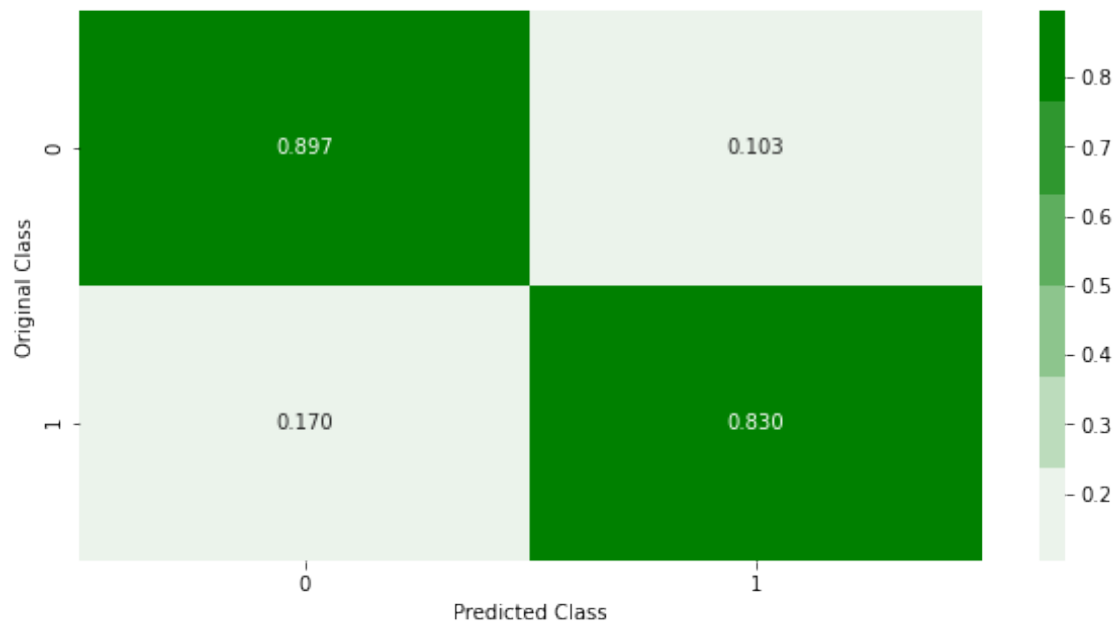


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Thresold for the CV Data

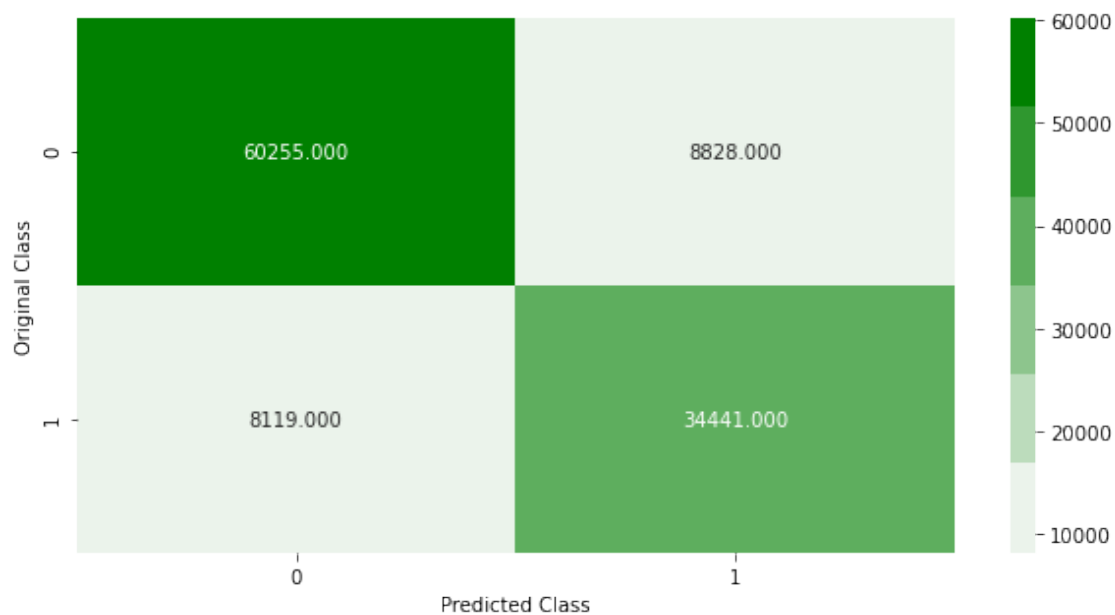
best train threshold : 0.5334865134820612
The Weighted Recall Score: 0.848203649131607
The Weighted Precision Score: 0.8487452410371084
The Weighted F1 Score: 0.8484391334321548

=====

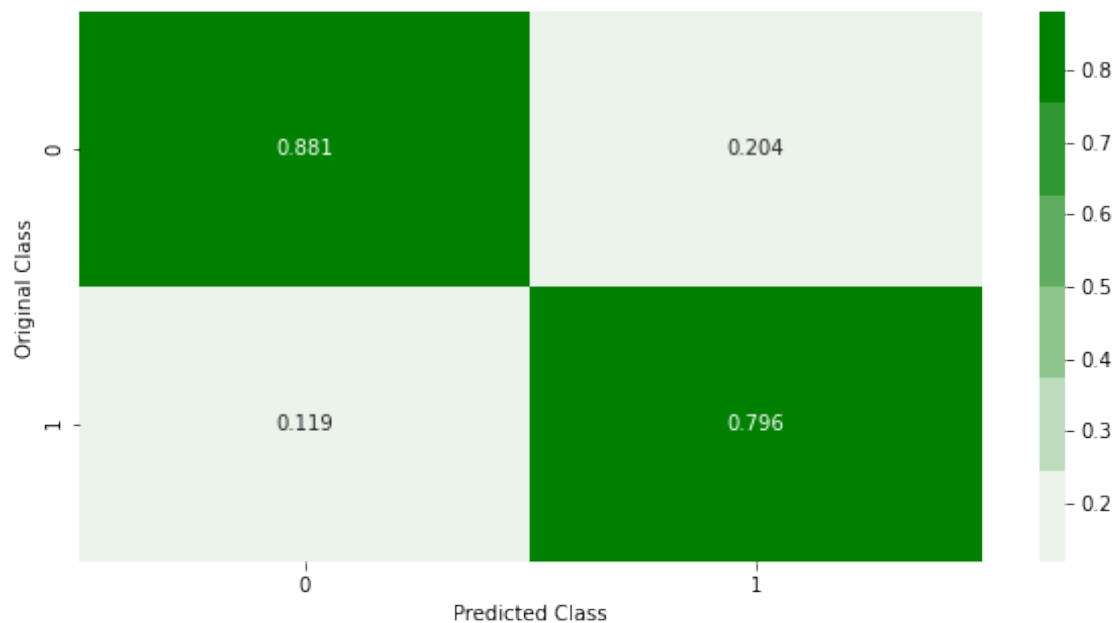
=====

=====

----- Confusion matrix

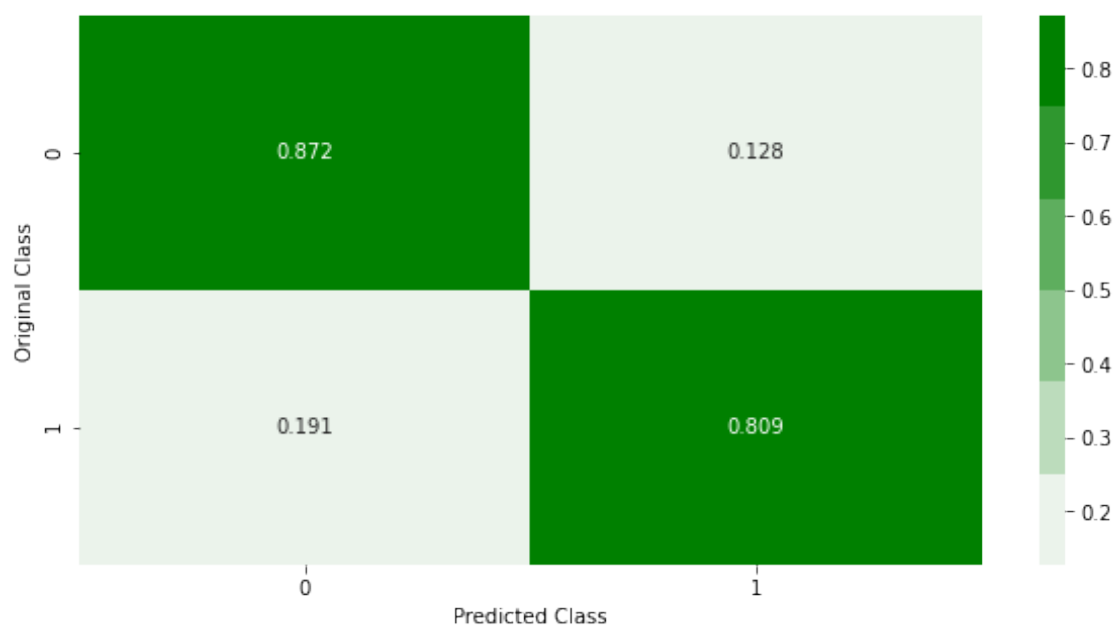


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.4 lightGBM Classifier without correcting for Class Imbalance

```
[ ]: import lightgbm as lgb
      clf= lgb.LGBMClassifier()
      param_grid ={'num_leaves': [1000,1050,1100], 'reg_alpha': [0.01,1,10]}

      lgb_tune = RandomizedSearchCV(clf,param_grid,cv=5,scoring='f1')
      lgb_tune.fit(train_fin4, train_y)

      print('best parameter :',lgb_tune.best_params_)
```

best parameter : {'reg_alpha': 0.01, 'num_leaves': 1100}

```
[ ]: lgb_best= lgb.LGBMClassifier(reg_alpha=0.01,num_leaves=1100)
      lgb_best.fit(train_fin4,train_y)
```

```
[ ]: LGBMClassifier(num_leaves=1100, reg_alpha=0.01)
```

```
[ ]: train_y_pred= lgb_best.predict_proba(train_fin4)
      train_y_pred= train_y_pred[:,1]

      cv_y_pred= lgb_best.predict_proba(cv_fin4)
      cv_y_pred= cv_y_pred[:,1]
```

8.4.1 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)
```

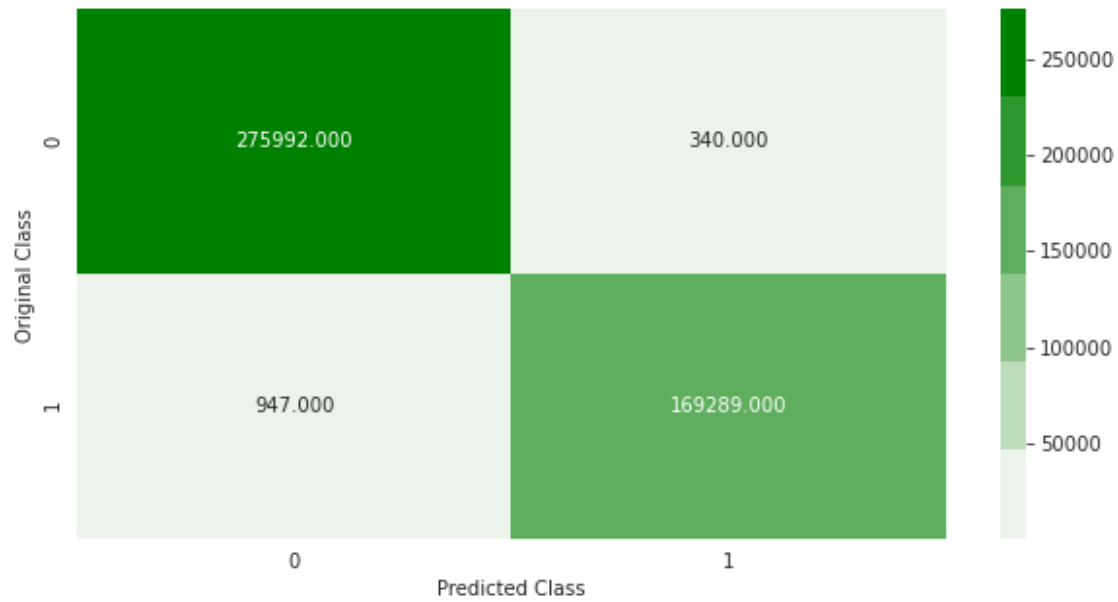
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.9971180200999624

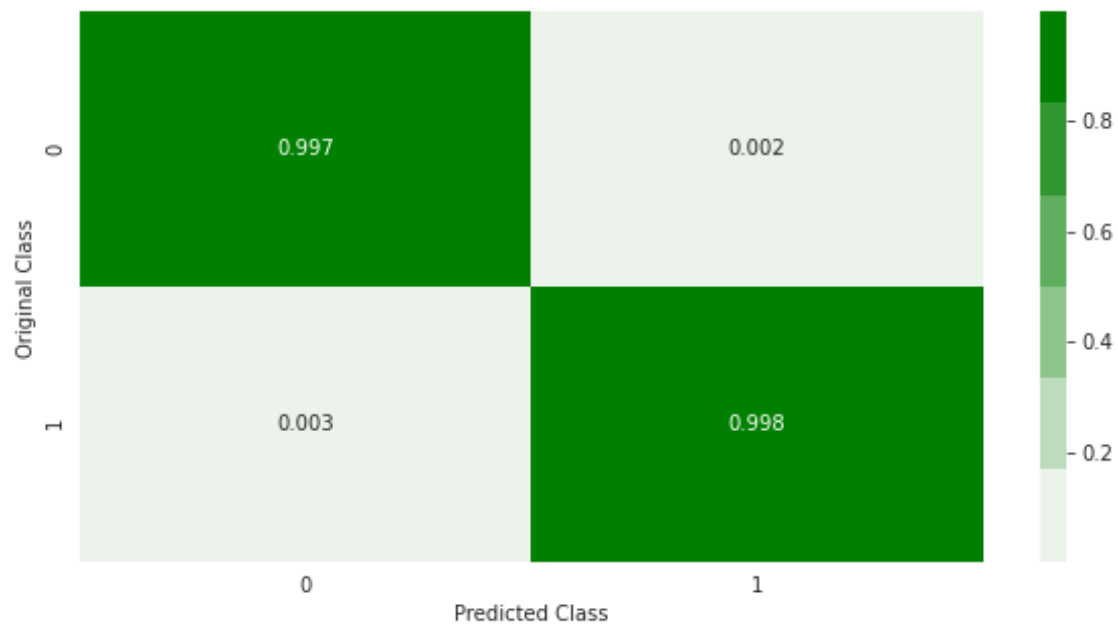
The Weighted Precision Score: 0.9971199436521826

The Weighted F1 Score: 0.9971170274139116

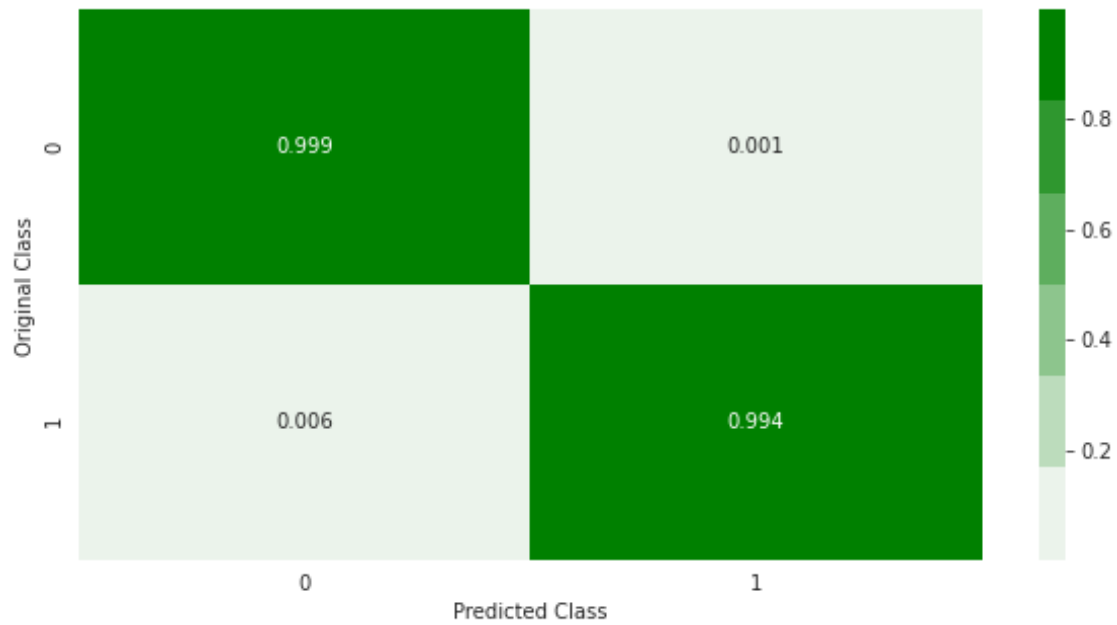
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix

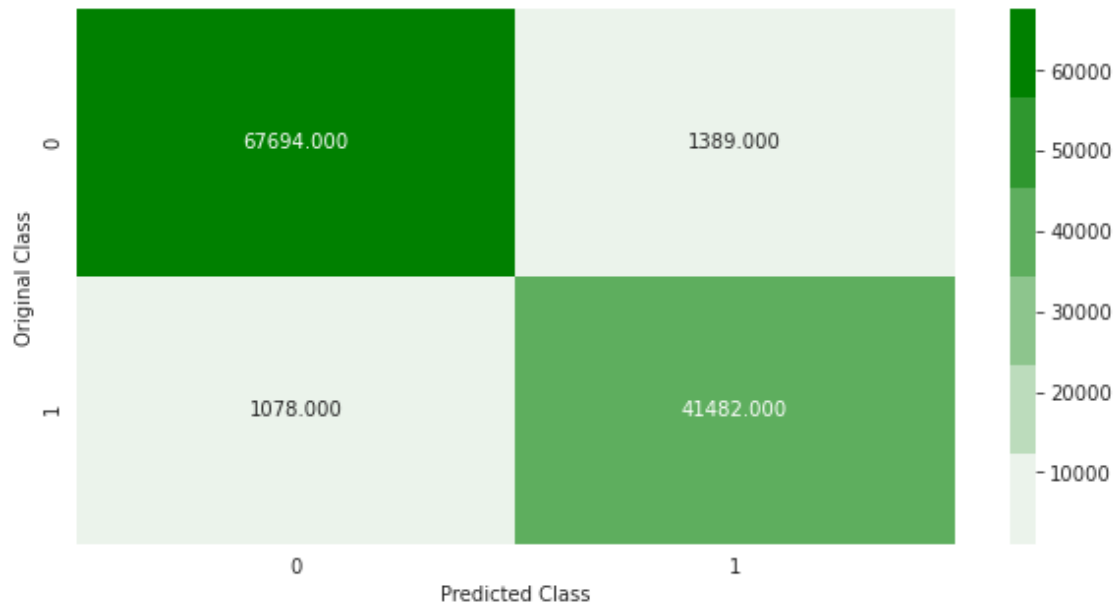


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

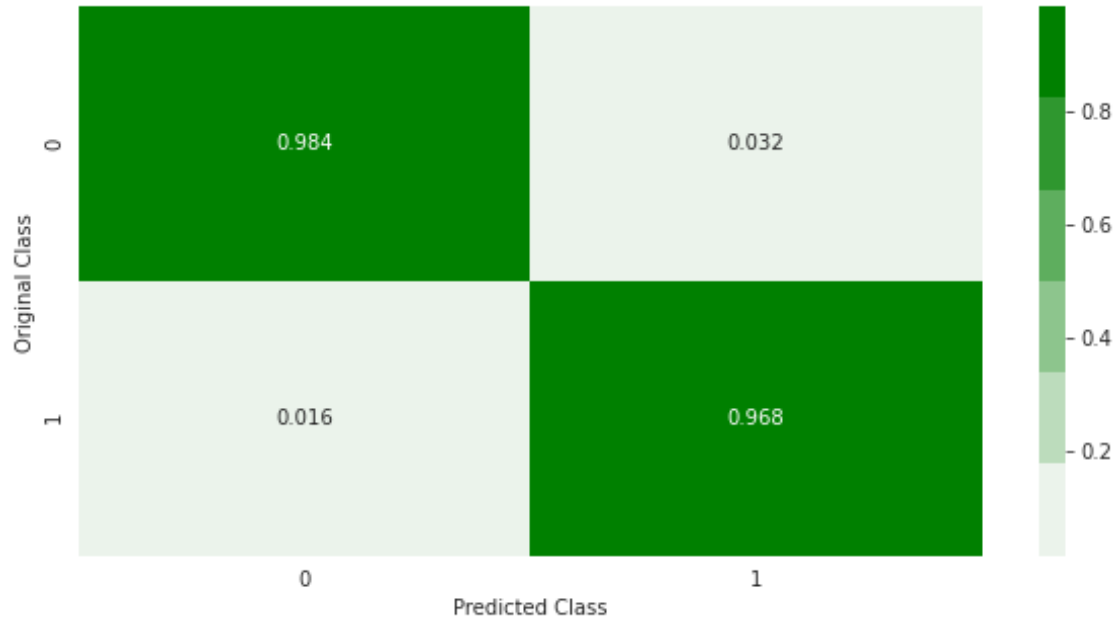


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score:  0.977902779395036
The Weighted Precision Score:  0.9779493683615297
The Weighted F1 Score:  0.9779180747720804
```

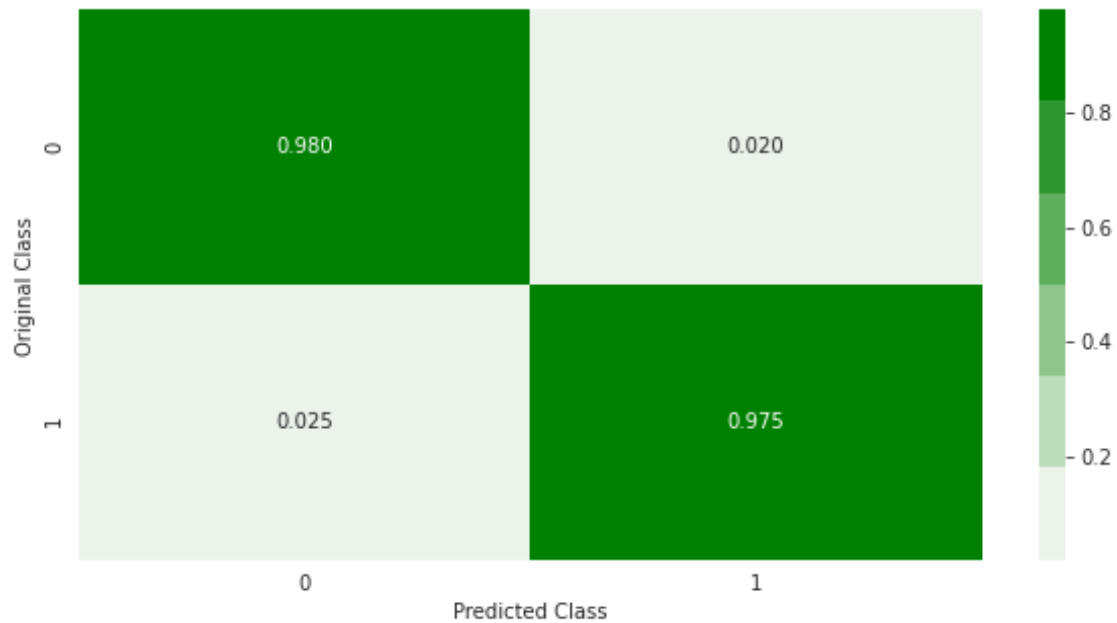
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.4.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

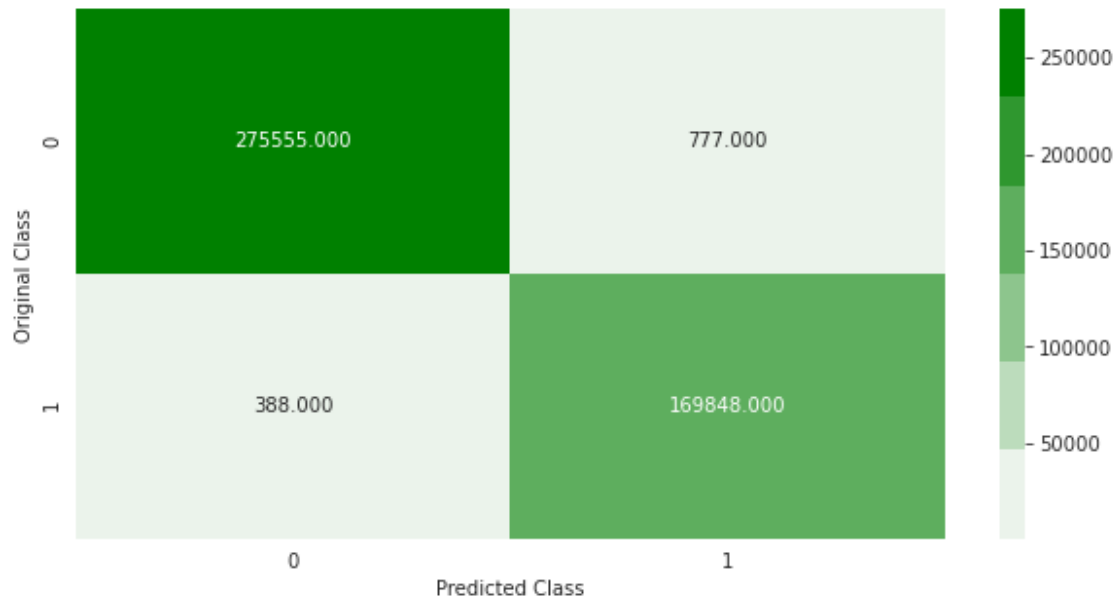
best train threshold : 0.39595528346587905

The Weighted Recall Score: 0.9973912147758012

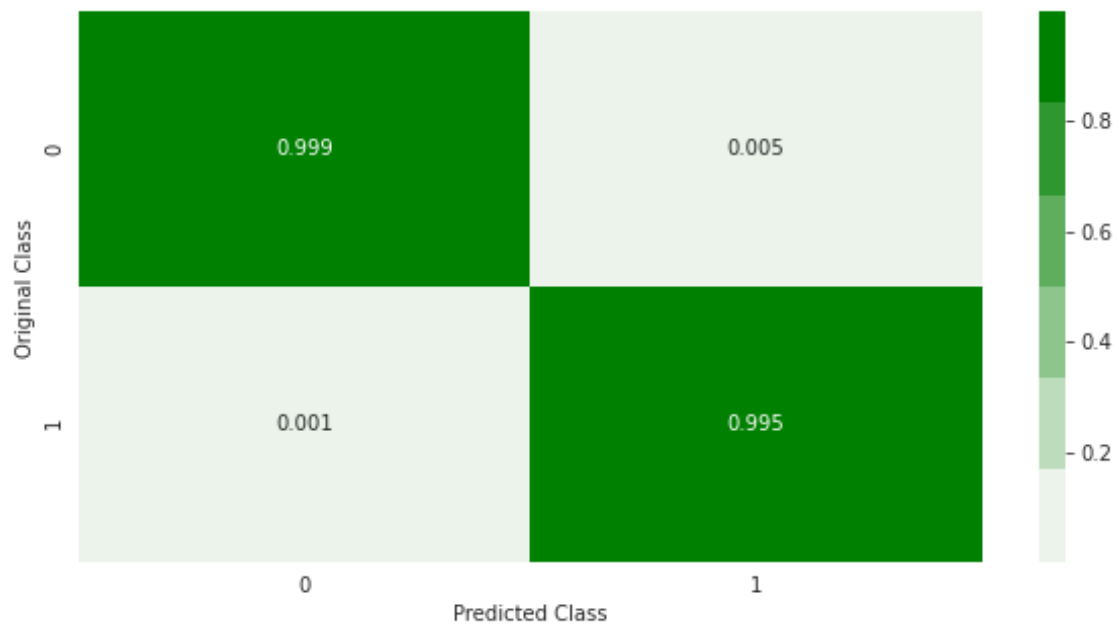
The Weighted Precision Score: 0.9973939567504178

The Weighted F1 Score: 0.9973917846237337

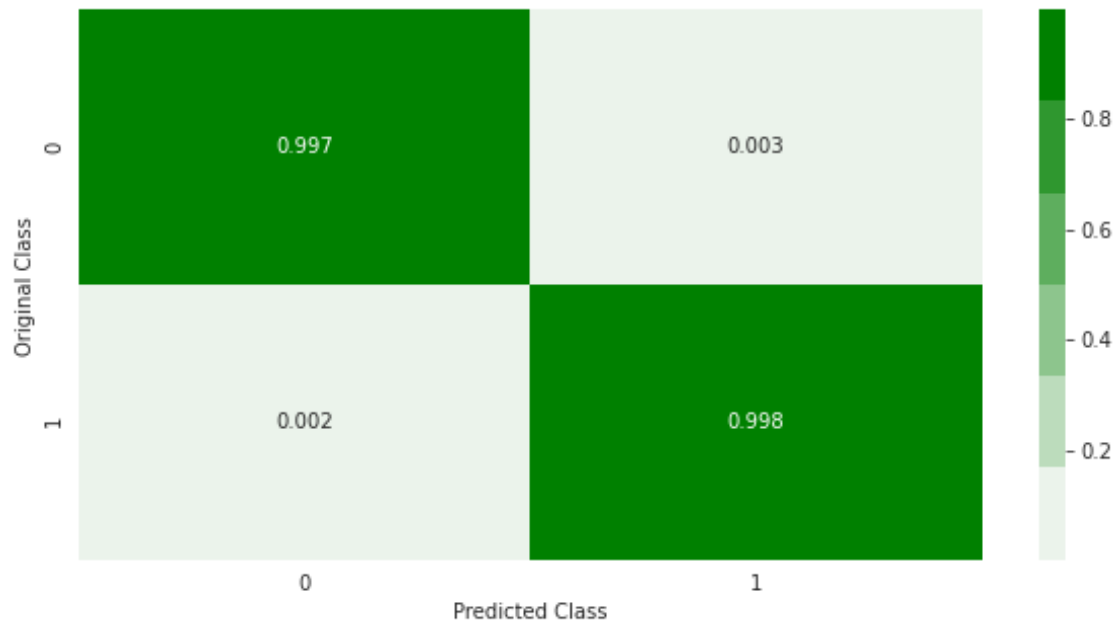
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix

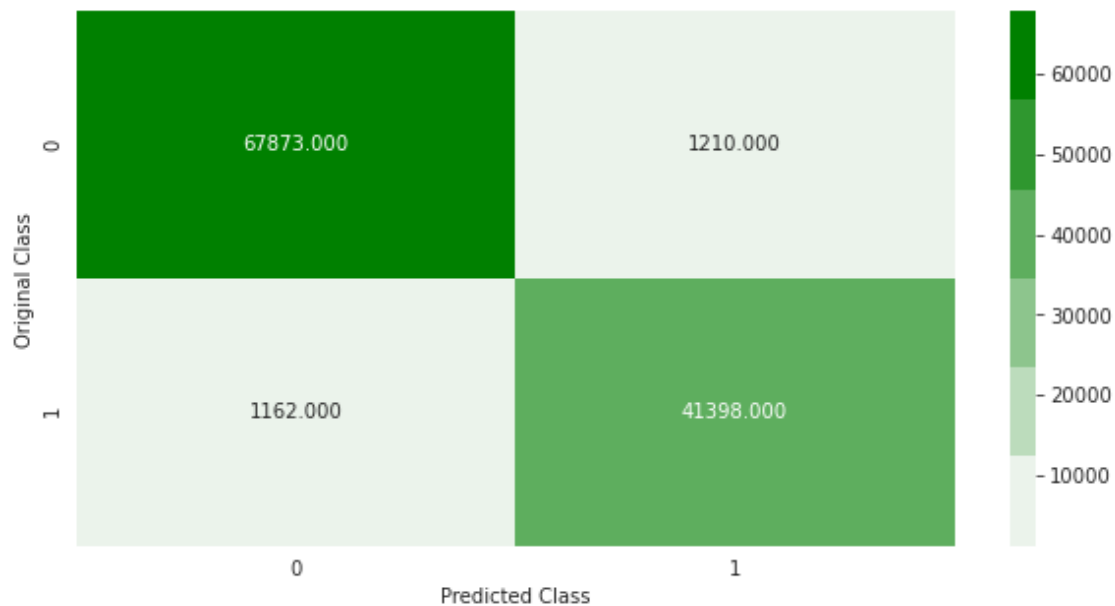


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

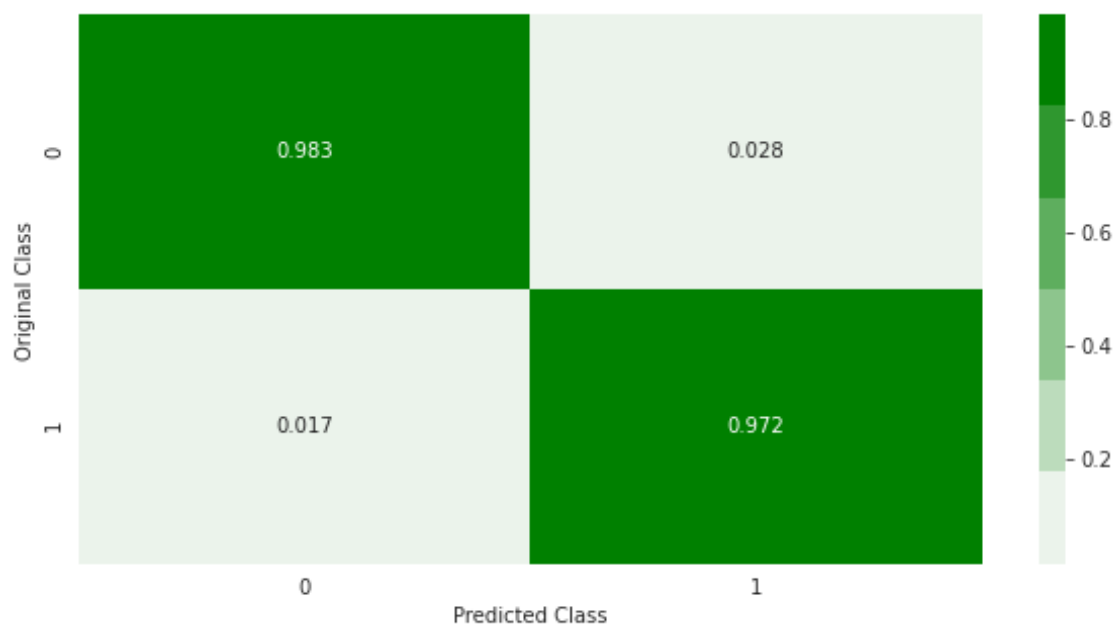


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.5275101624160168
The Weighted Recall Score:  0.9787537060093333
The Weighted Precision Score:  0.9787586788808385
The Weighted F1 Score:  0.9787560012787132
```

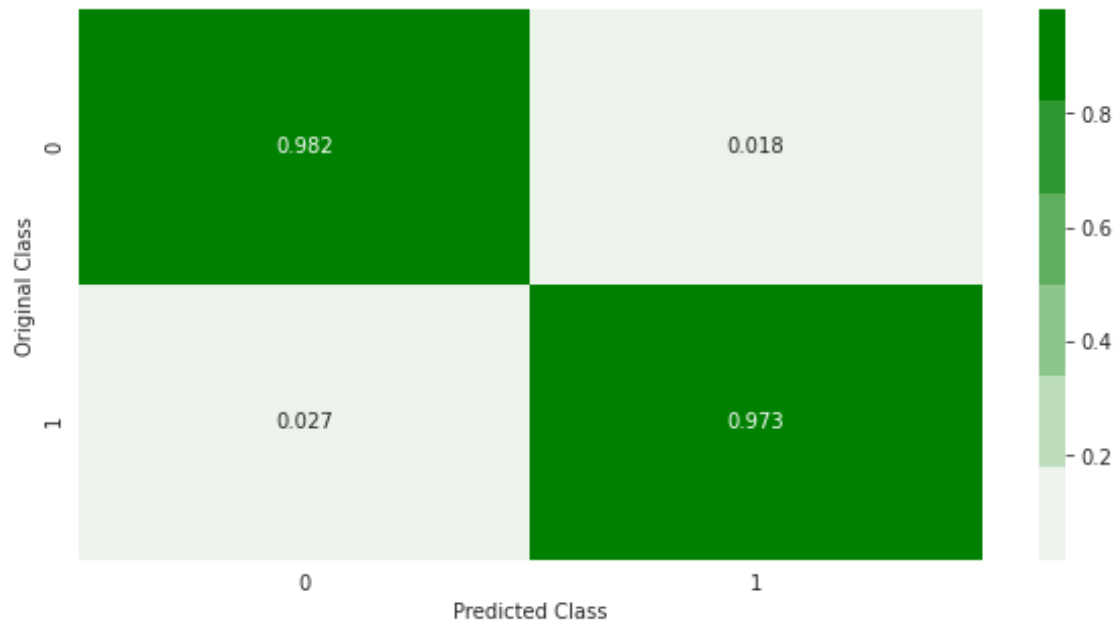
```
=====
=====
=====
----- Confusion matrix
-----
```

----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.5 Stacking Classifier

```
[ ]: from sklearn.ensemble import StackingClassifier
estimators=[('rf',
    ↳ RandomForestClassifier(n_estimators=50,min_samples_split=5,max_depth=14,criterion='gini')),
    ↳ LGBMClassifier(reg_alpha=1,num_leaves=300))]

stk_clf = StackingClassifier(estimators=estimators,
    ↳ final_estimator=LogisticRegression())

stk_clf.fit(train_fin4, train_y)
```

```
[ ]: StackingClassifier(estimators=[('rf',
    RandomForestClassifier(max_depth=14,
                           min_samples_split=5,
                           n_estimators=50)),
    ('lgb',
     LGBMClassifier(num_leaves=300, reg_alpha=1))],
    final_estimator=LogisticRegression())
```

```
[ ]: train_y_pred_stk= stk_clf.predict_proba(train_fin4)
train_y_pred_stk= train_y_pred_stk[:,1]

cv_y_pred_stk= stk_clf.predict_proba(cv_fin4)
cv_y_pred_stk= cv_y_pred_stk[:,1]
```

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred_stk)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred_stk)
```

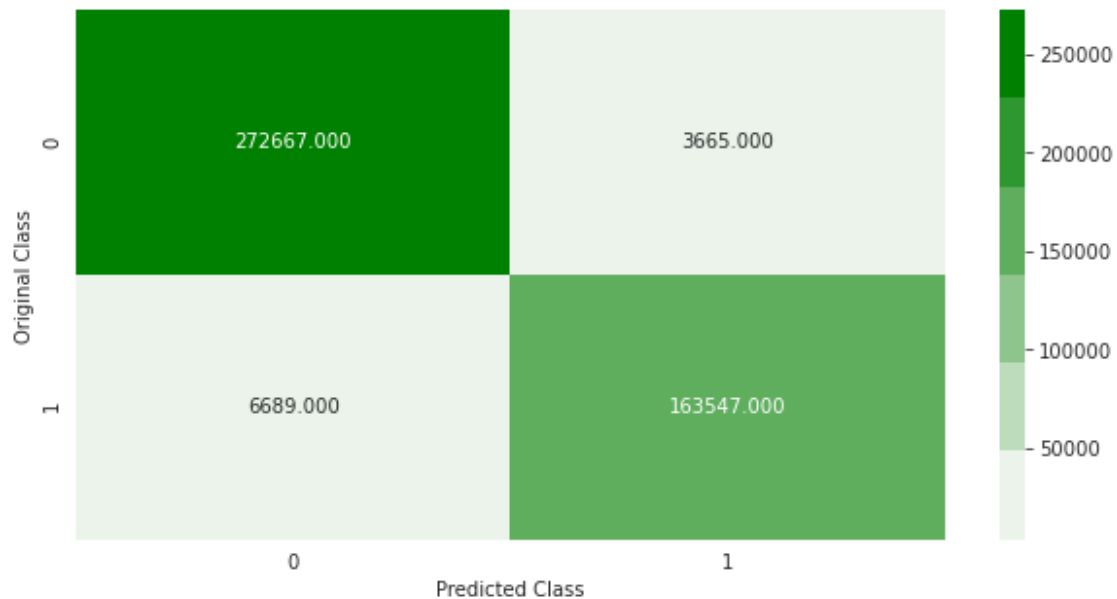
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.9768142813636445

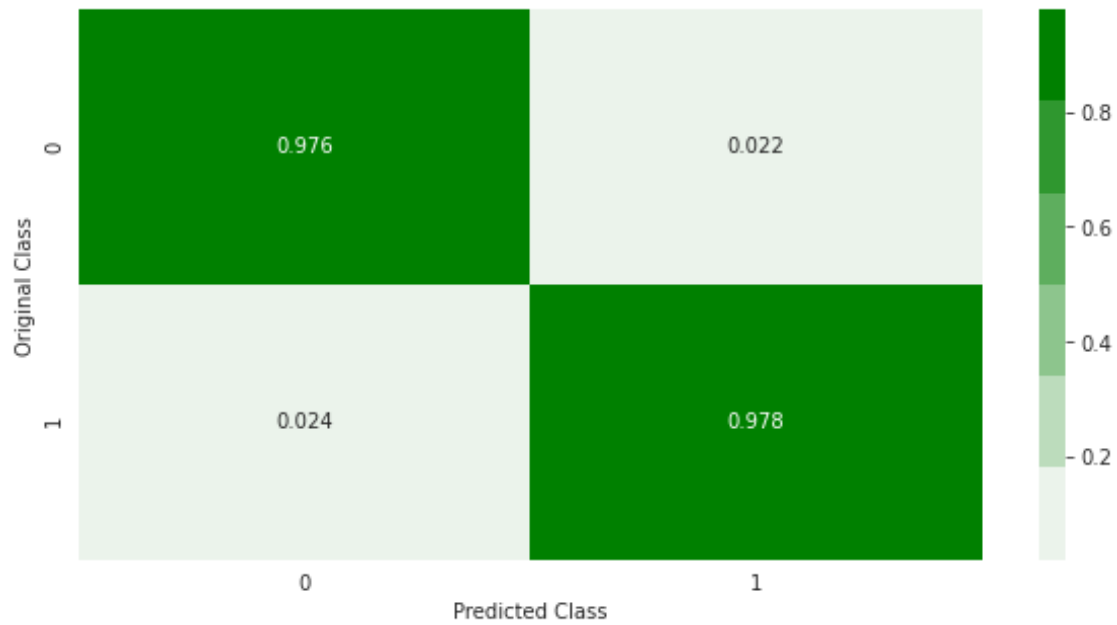
The Weighted Precision Score: 0.9768280012201177

The Weighted F1 Score: 0.9767734805495516

```
=====
=====
=====
----- Confusion matrix
-----
```

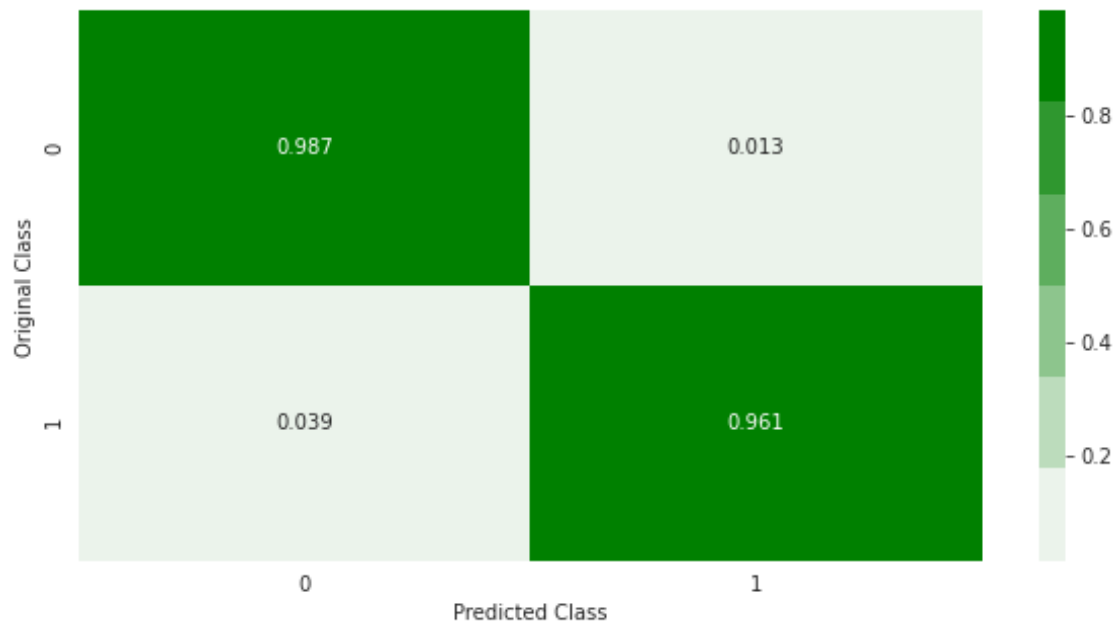


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.9511568123393316
The Weighted Precision Score: 0.953581282743027
The Weighted F1 Score: 0.9514765509463554

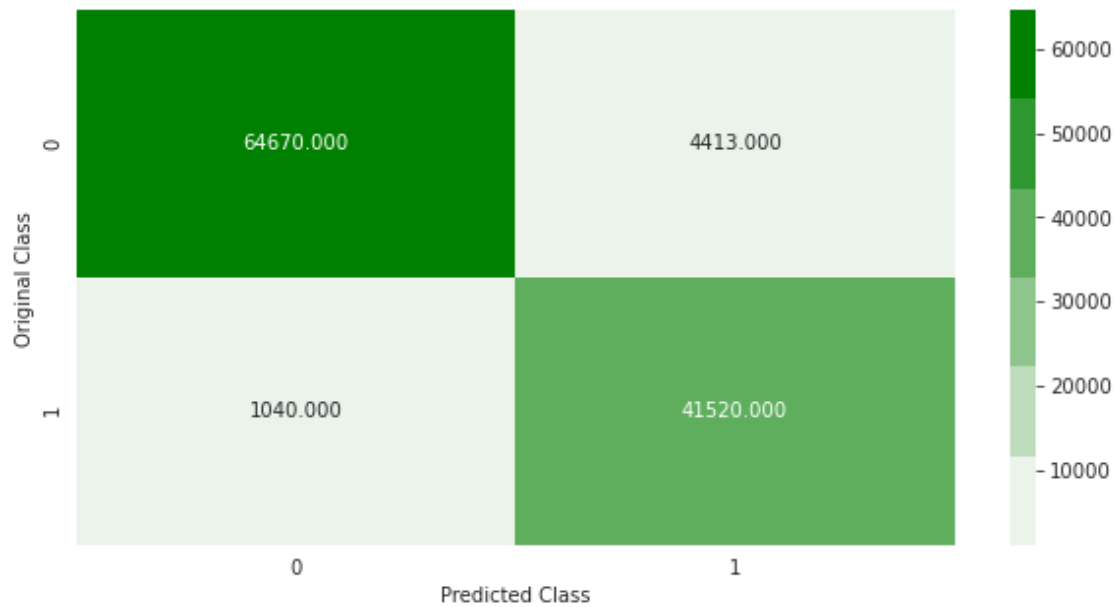
=====

=====

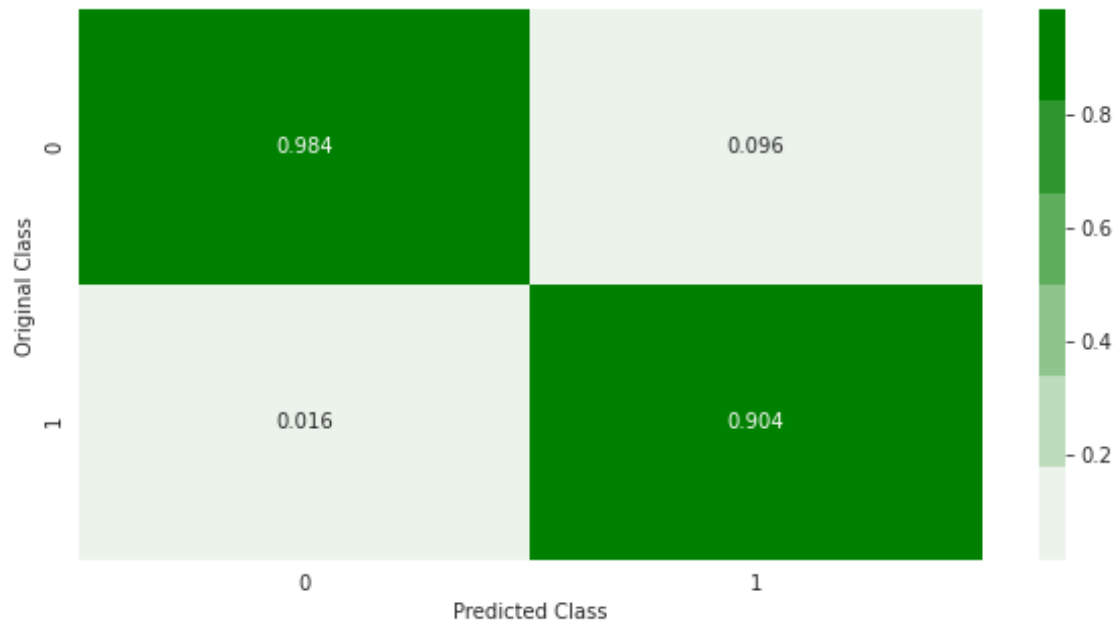
=====

=====

----- Confusion matrix

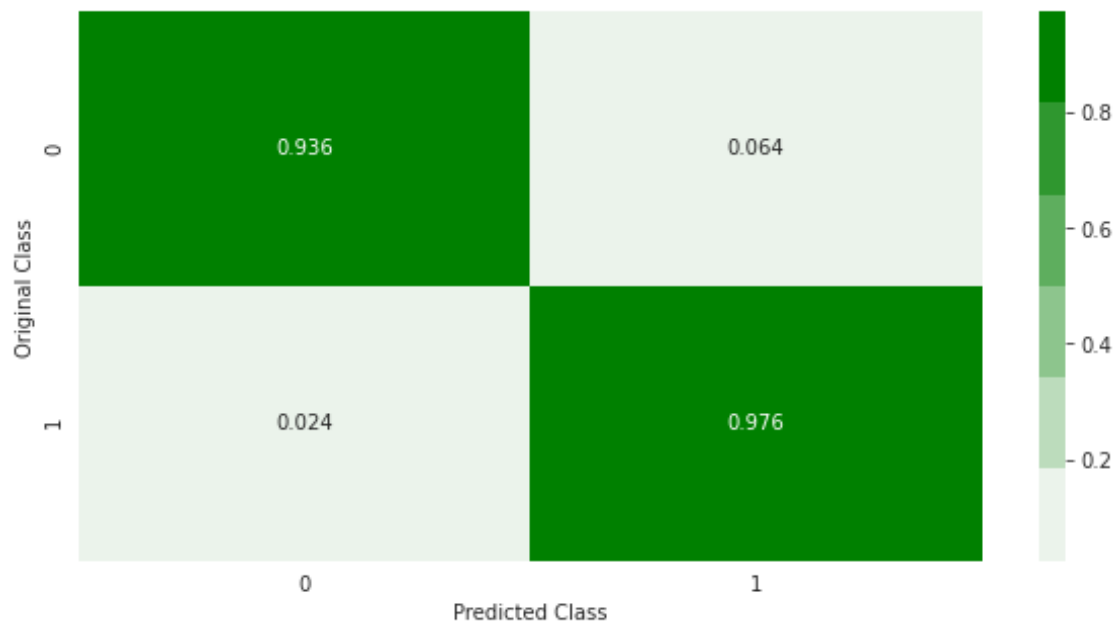


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred_stk)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred_stk)
```

Confusion Matrix for Best Thresold for the Train Data

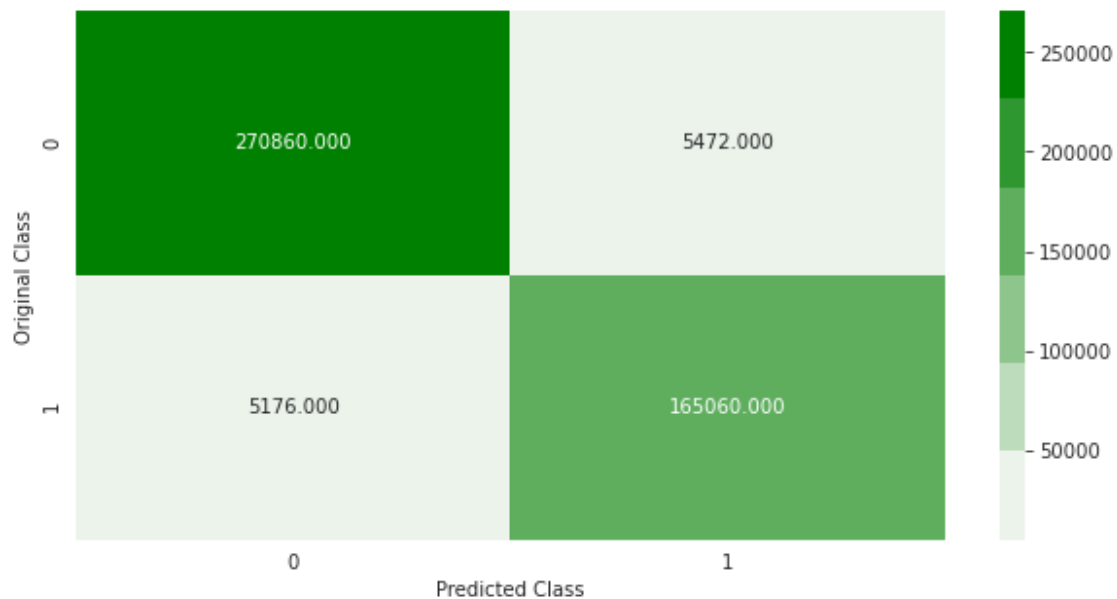
best train threshold : 0.32621624058594195

The Weighted Recall Score: 0.9761559269808854

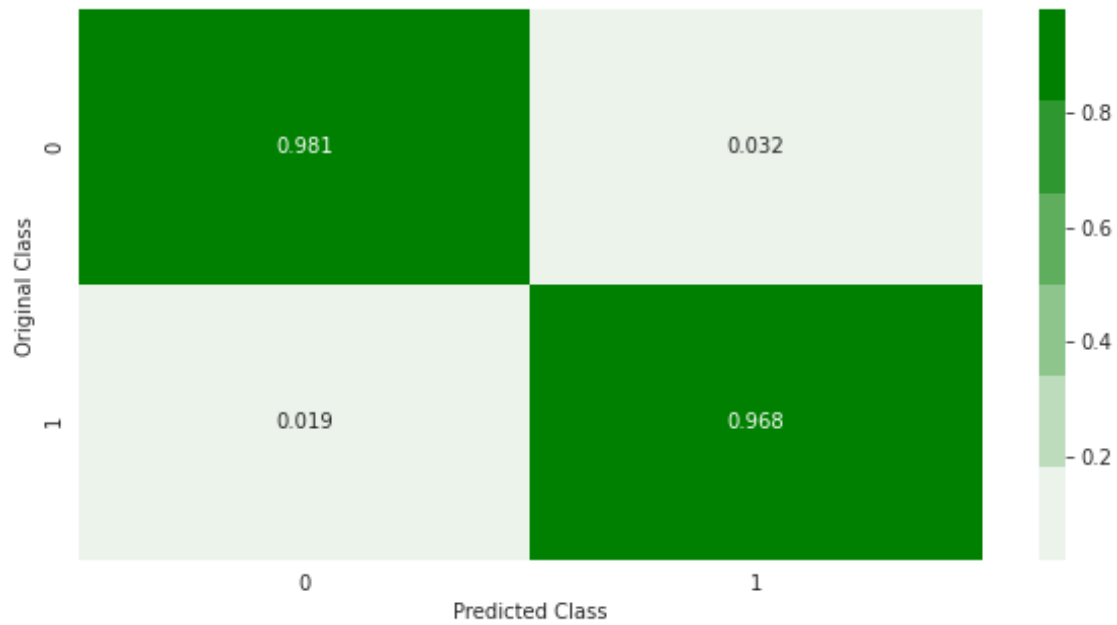
The Weighted Precision Score: 0.9761647669470406

The Weighted F1 Score: 0.9761598940567482

```
=====
=====
=====
----- Confusion matrix
-----
```

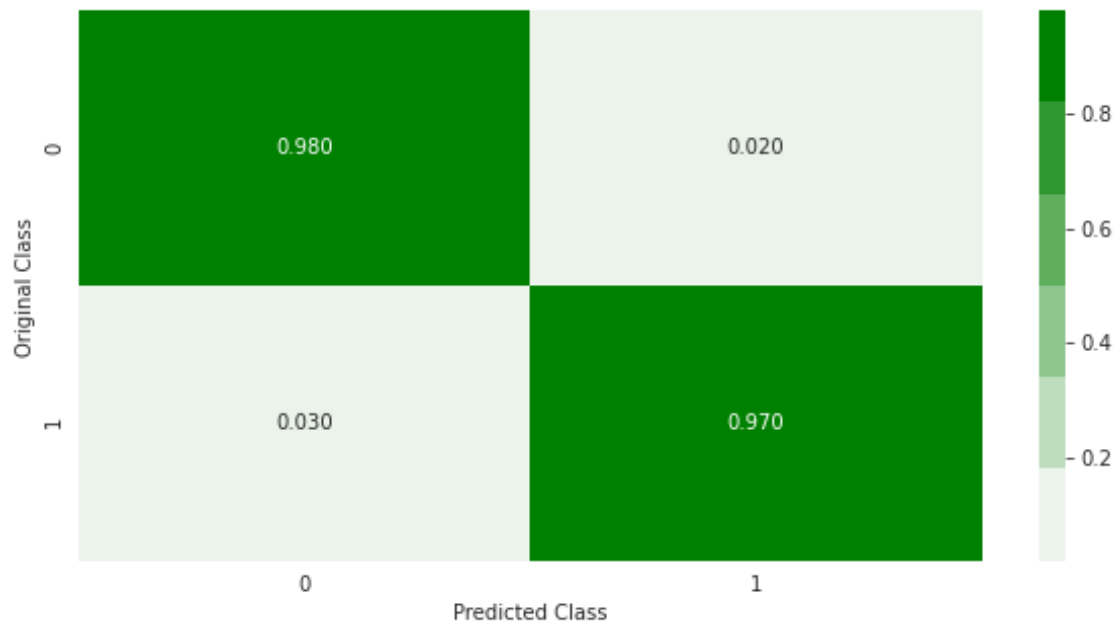


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix

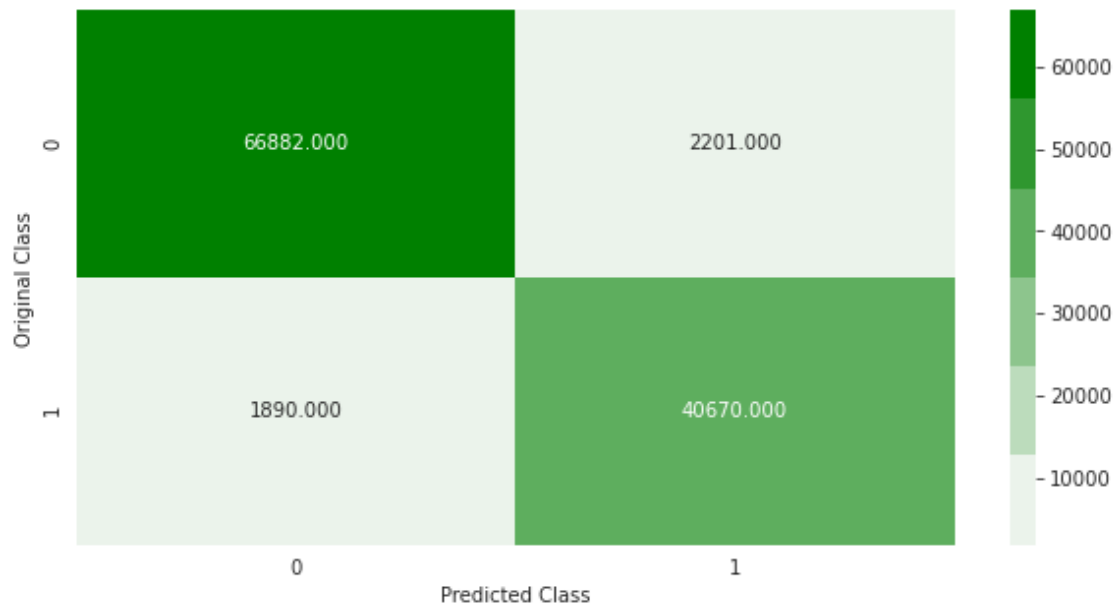


Sum of rows in precision matrix [1. 1.]

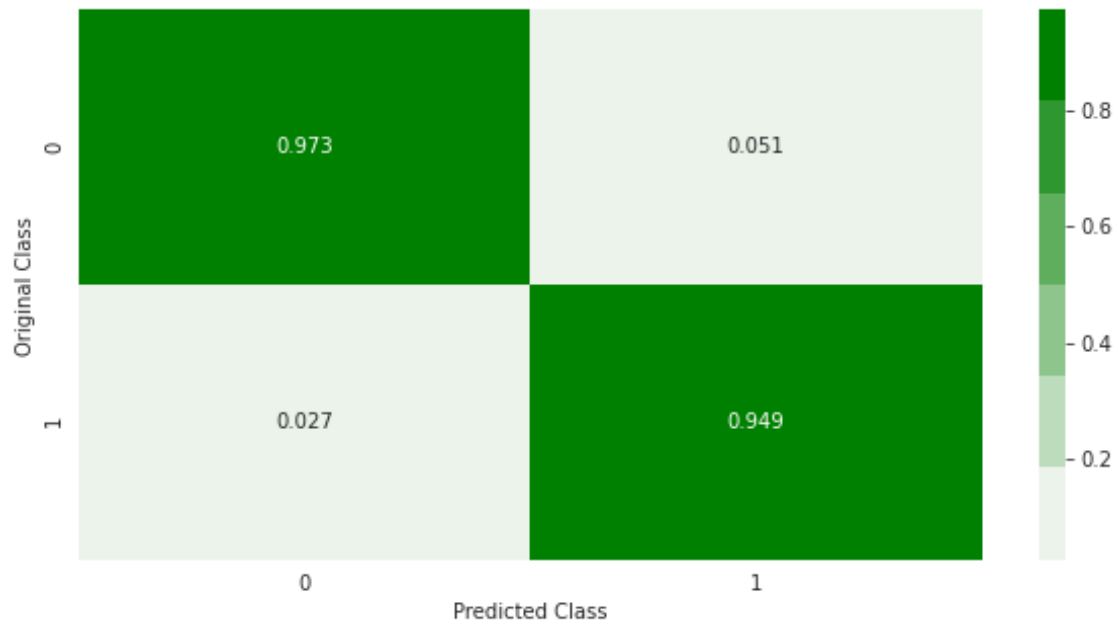
Confusion Matrix for Best Thresold for the CV Data

best train threshold : 0.8104481378443146
The Weighted Recall Score: 0.9633564128516791
The Weighted Precision Score: 0.963422873113766
The Weighted F1 Score: 0.963381777013612

```
=====
=====
=====
----- Confusion matrix
-----
```

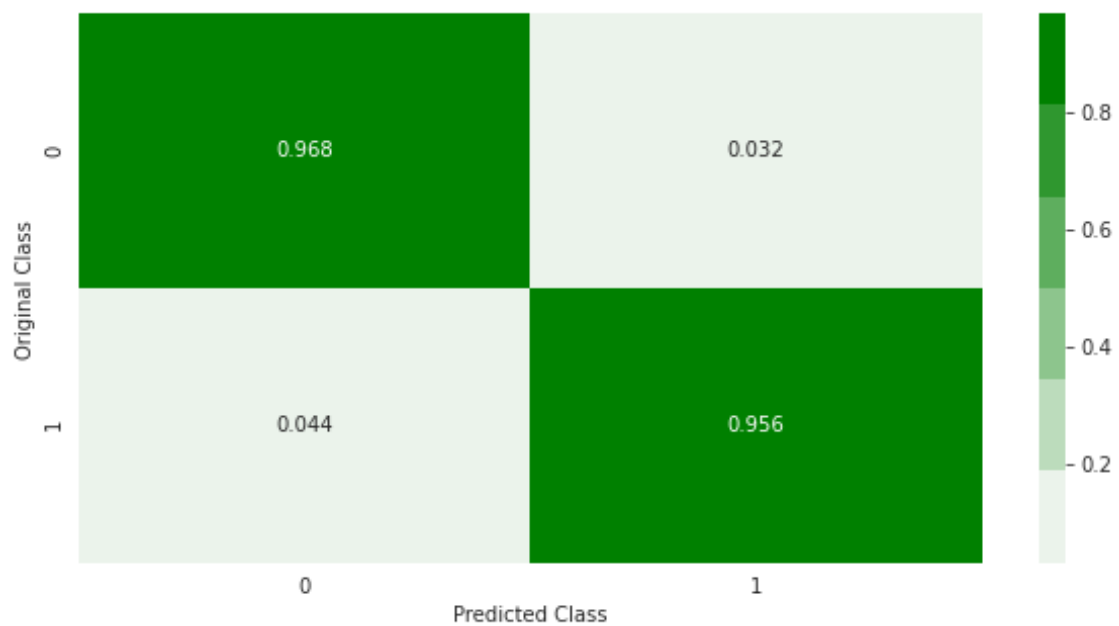


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

8.6 Observations on Experiment 1:

1. From the above we see that the lightGBM model has out performed Random Forests classifier and logistic regression
2. lightGBM outputs on TRAIN DATA- Recall_Score: 0.91079, Precision_Score:0.91058 and a weighted F1 score:0.91065
3. lightGBM outputs on CV DATA- Recall_Score: 0.910, Precision_Score:0.909 and a weighted F1 score:0.909

8.7 Next Steps:

1. Check for multicorrelation amongst the features and see if decreasing the multicorrelation could improve the performance of the models
2. Use VIF scores to see for multicorrelation and evaluate the features with high VIF scores

9 Experiment 2: Using the Class Weights Parameter and obtaining the best class weights

9.1 Logistic Regression with Best Class Weights

```
[ ]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, StratifiedKFold

lr_cw = LogisticRegression(C=0.01,penalty='l2')

weights = np.linspace(0.0,0.99,10)

param_grid = {'class_weight': [{0:x, 1:1.0-x} for x in weights]}

gridsearch = GridSearchCV(estimator= lr_cw,
                           param_grid= param_grid,
                           cv=StratifiedKFold(),
                           n_jobs=-1,
                           scoring='f1',
                           verbose=2).fit(train_fin4, train_y)

sns.set_style('whitegrid')
plt.figure(figsize=(12,8))
weigh_data = pd.DataFrame({'score': gridsearch.cv_results_['mean_test_score'],
                           'weight': (1- weights)})
sns.lineplot(weigh_data['weight'], weigh_data['score'])
plt.xlabel('Weight for class 1')
plt.ylabel('F1 score')
plt.xticks([round(i/10,1) for i in range(0,11,1)])
plt.title('Scoring for different class weights', fontsize=24)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression


```
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

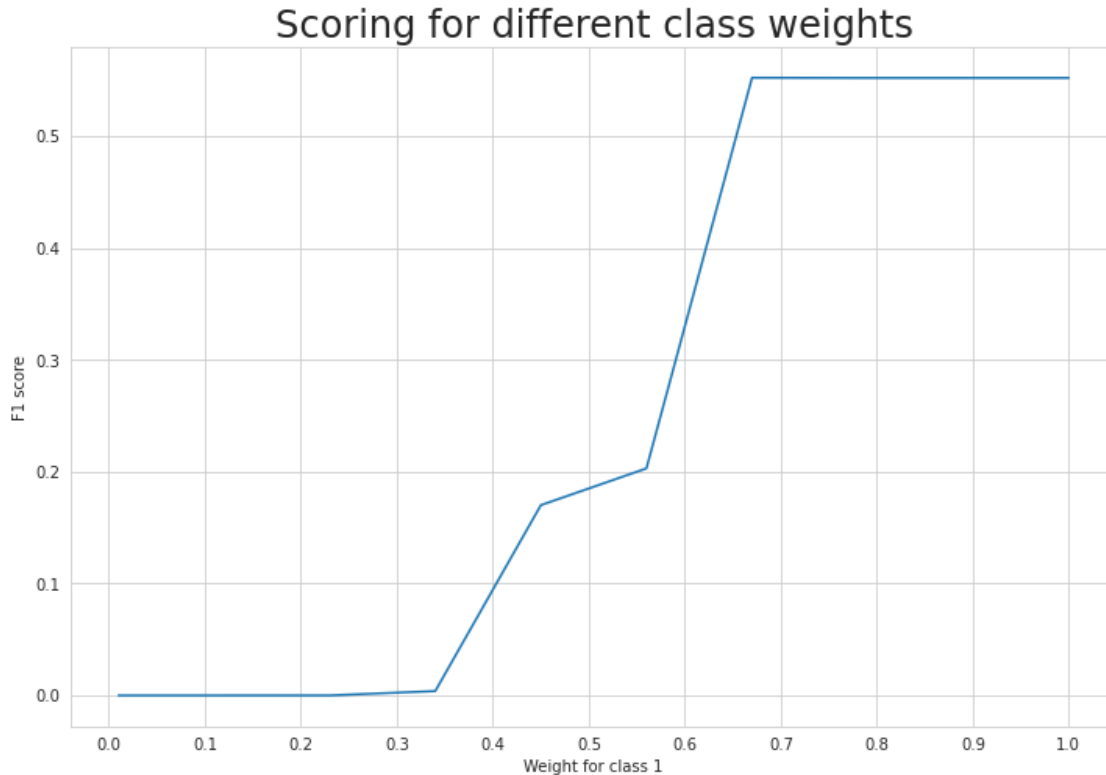
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

[]: Text(0.5, 1.0, 'Scoring for different class weights')



```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

```
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 5.0s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 4.7s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 4.5s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 10.2s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 10.5s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 9.3s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 11.2s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 9.9s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 9.7s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 10.2s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 10.0s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 8.1s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 9.9s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 14.5s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 19.7s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 20.3s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 20.3s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 21.0s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 21.8s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 21.6s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 21.8s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 21.1s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 21.0s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 9.8s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 9.5s
```

/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:

ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:

ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

```
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 5.0s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 4.8s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 9.7s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 10.5s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 10.2s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 9.7s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 10.3s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 10.0s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 9.9s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 10.0s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 10.0s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 9.6s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 11.7s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 19.5s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 19.7s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 21.0s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 21.3s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 20.2s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 21.2s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 21.4s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 19.3s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 22.2s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 9.3s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 10.2s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 9.7s
```

```
[ ]: lr_cw = LogisticRegression(C=0.01, penalty='l2', class_weight={0: 0.33, 1: 0.67})
lr_cw.fit(train_fin4, train_y)
```

```
[ ]: LogisticRegression(C=0.01, class_weight={0: 0.33, 1: 0.67})
```

```
[ ]: train_y_pred= lr_cw.predict_proba(train_fin4)
train_y_pred= train_y_pred[:,1]

cv_y_pred= lr_cw.predict_proba(cv_fin4)
cv_y_pred= cv_y_pred[:,1]
```

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred)
```

```
print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)
```

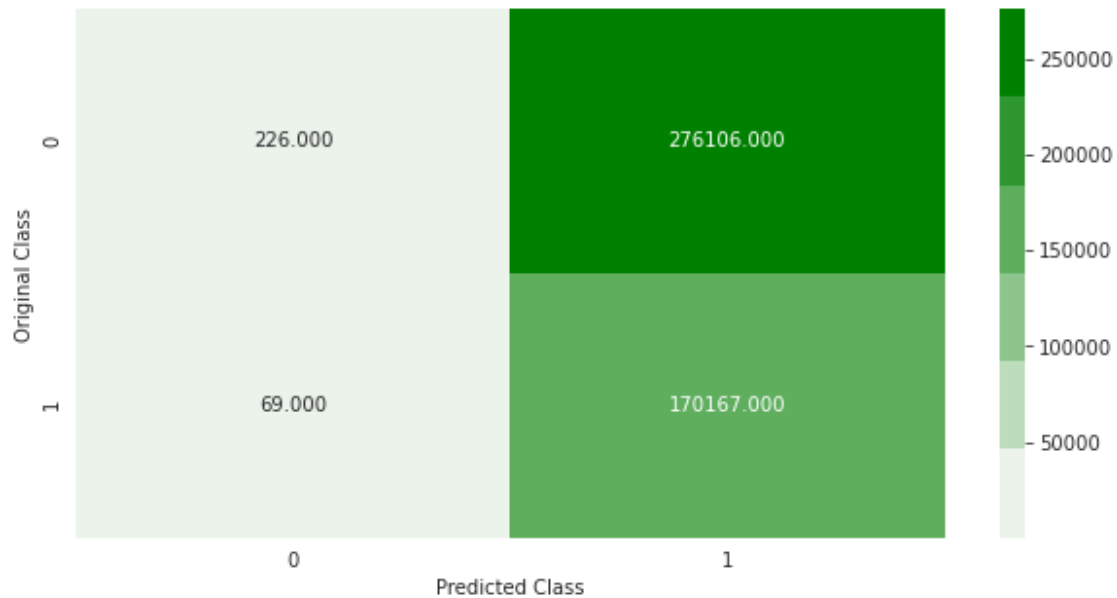
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.3815611508213755

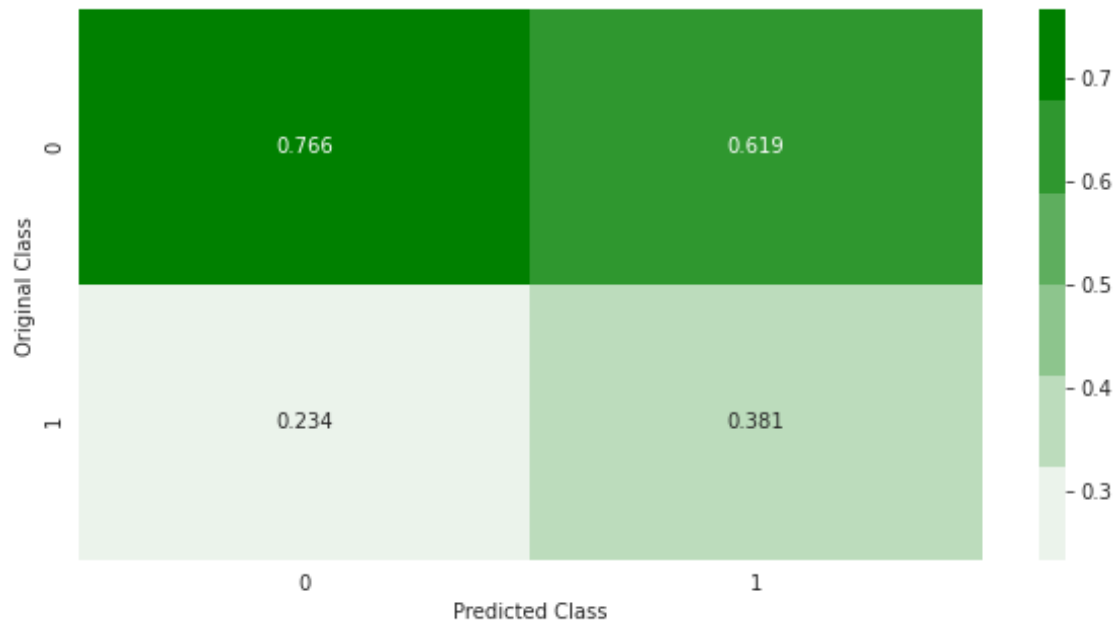
The Weighted Precision Score: 0.6194142545718646

The Weighted F1 Score: 0.2114517778339373

```
=====
=====
=====
=====
----- Confusion matrix
-----
```

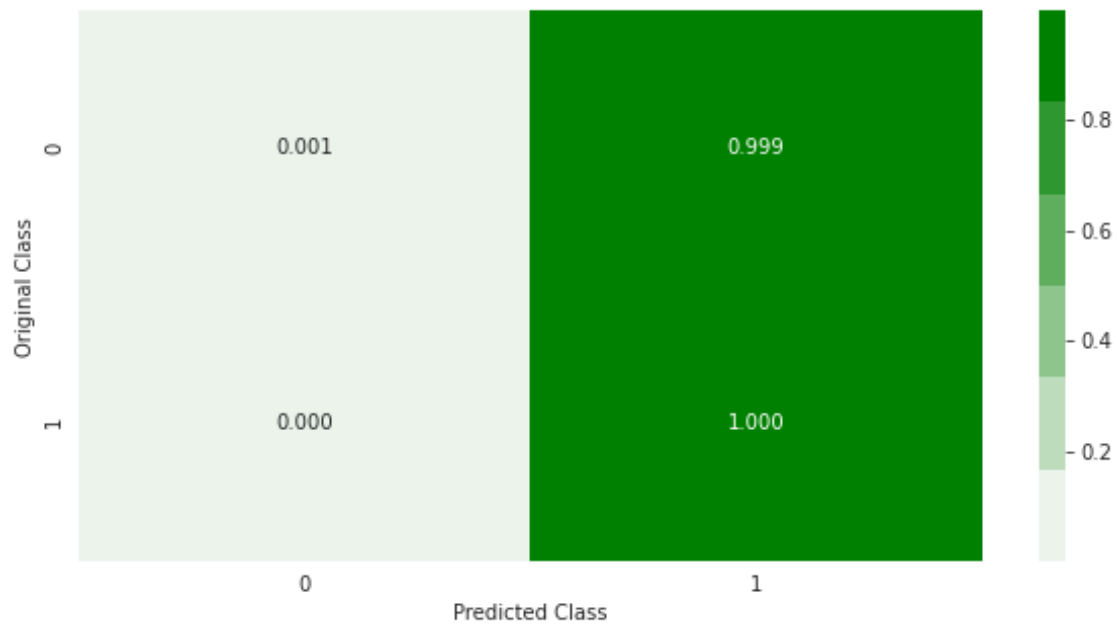


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

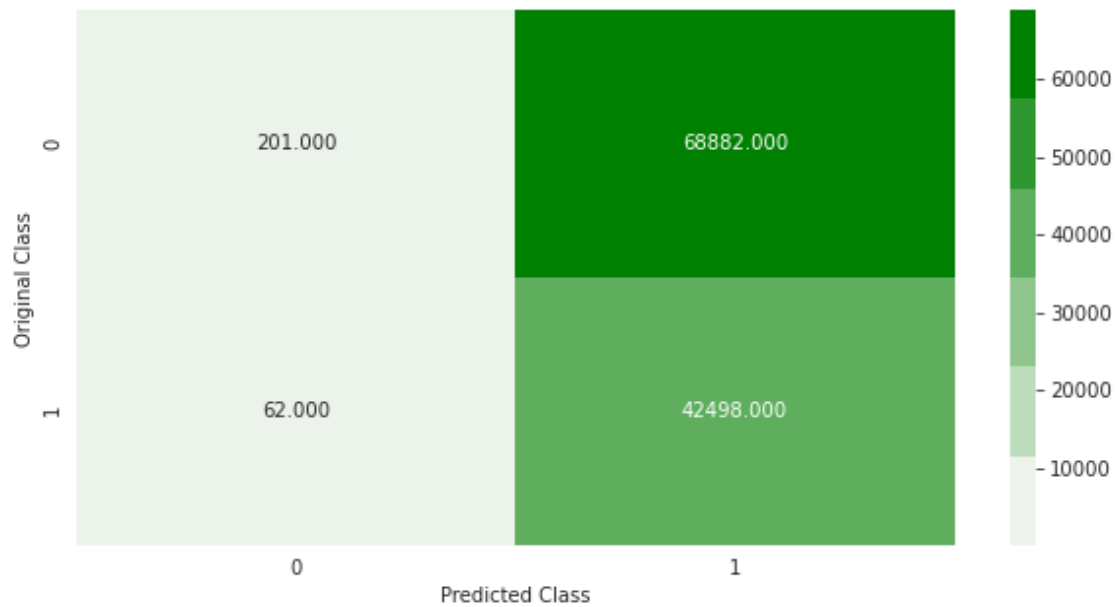
The Weighted Recall Score: 0.3824601631987675
The Weighted Precision Score: 0.618367555306142
The Weighted F1 Score: 0.21407015629523626

=====

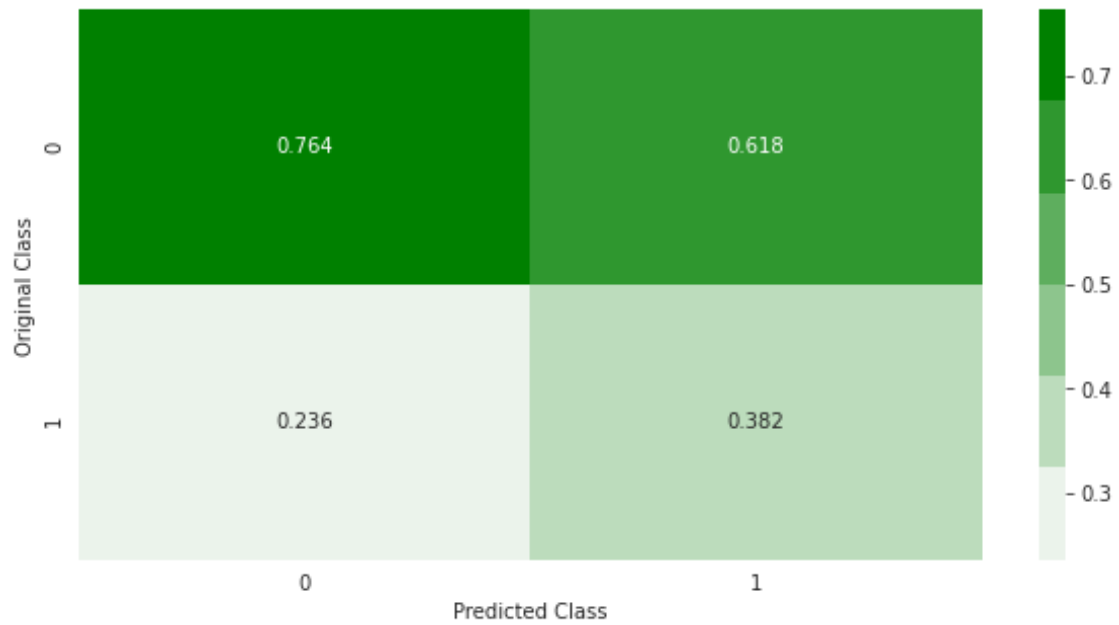
=====

=====

----- Confusion matrix

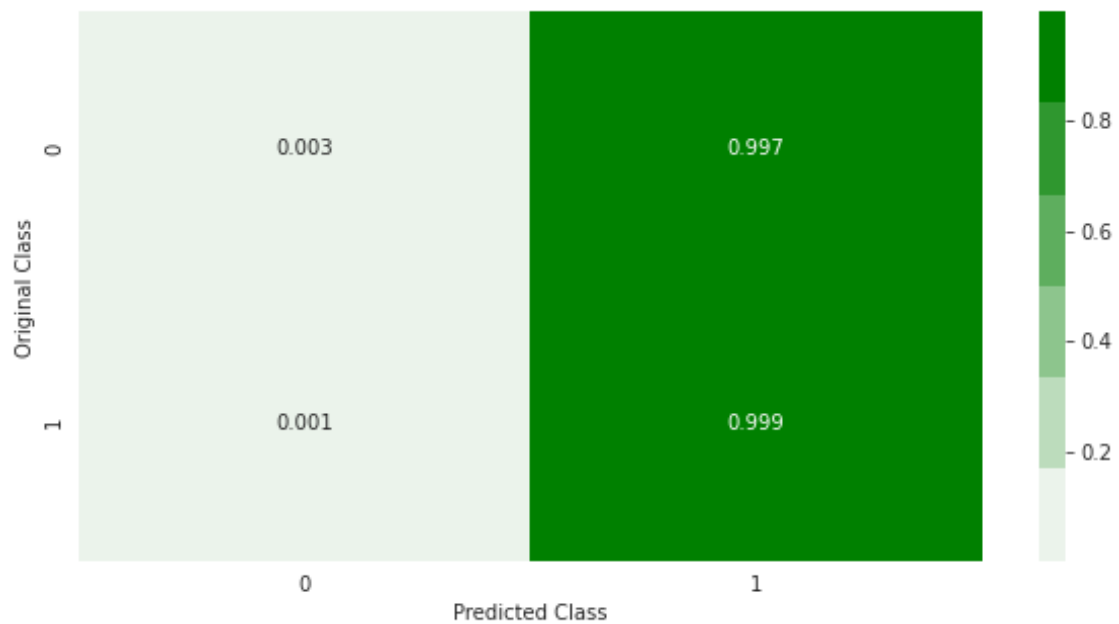


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

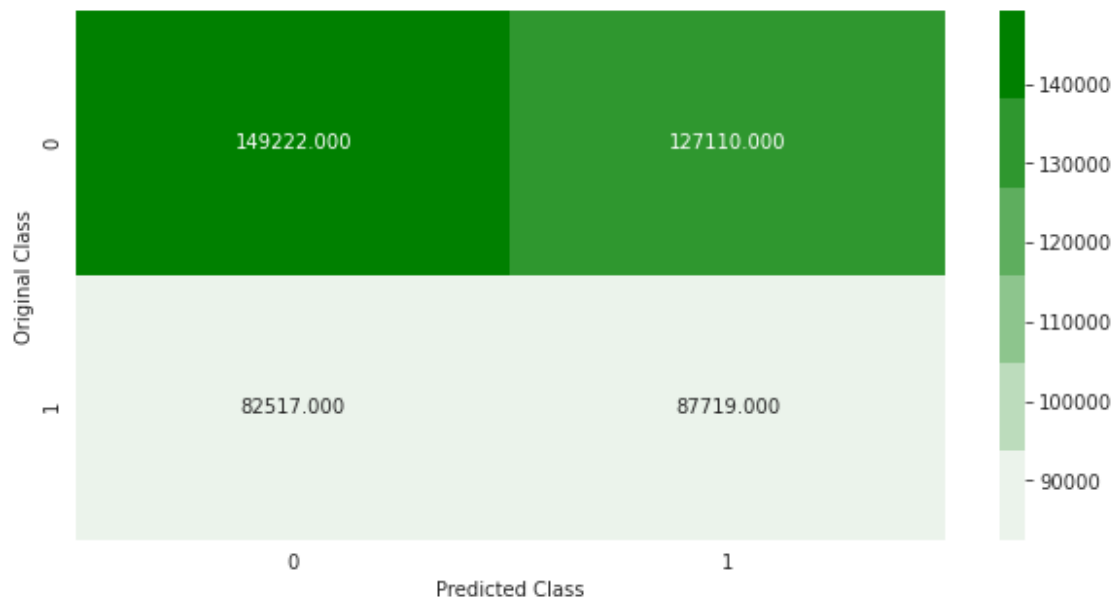
best train threshold : 0.5374186011425456

The Weighted Recall Score: 0.5305821285895989

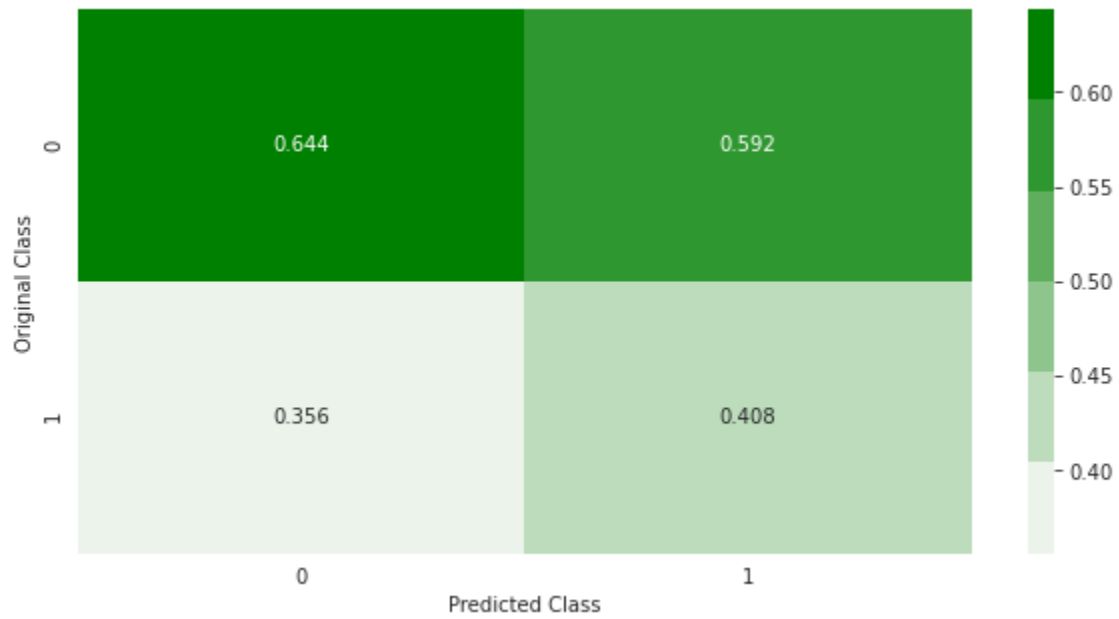
The Weighted Precision Score: 0.5541087269787596

The Weighted F1 Score: 0.5371627083630407

```
=====
=====
=====
----- Confusion matrix
-----
```

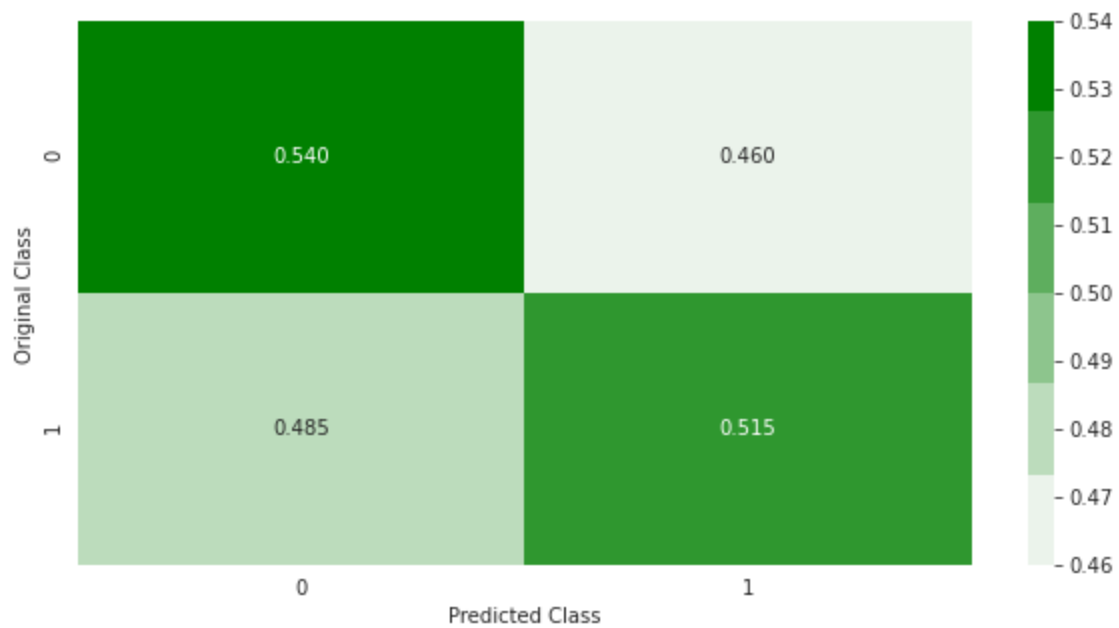


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Thresold for the CV Data

best train threshold : 0.5967761103595537
The Weighted Recall Score: 0.5218419426206748
The Weighted Precision Score: 0.5477761476947325
The Weighted F1 Score: 0.5286592810207519

=====

=====

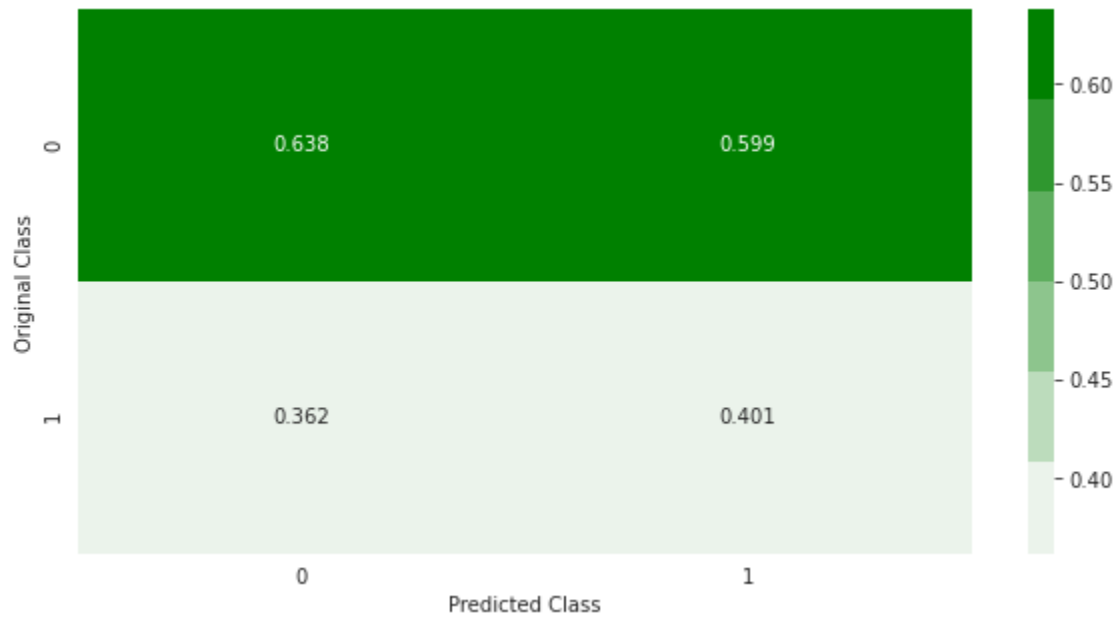
=====

=====

----- Confusion matrix

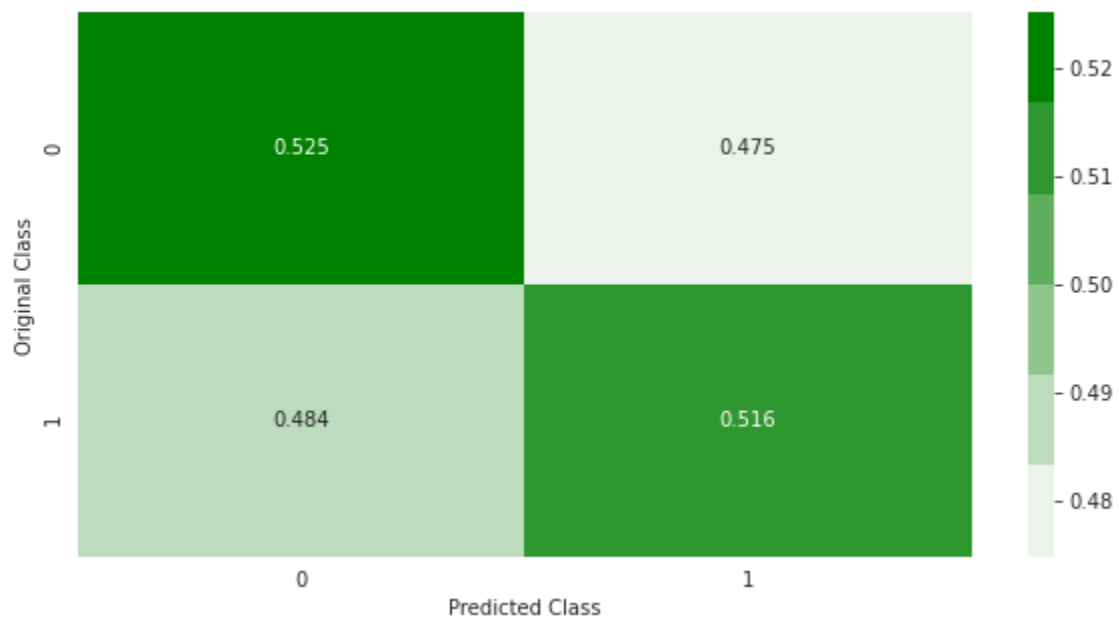


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

9.2 LGBM with Class Weights parameter adjusted

```
[ ]: import lightgbm as lgb
      from sklearn.model_selection import GridSearchCV, StratifiedKFold

      lgb_cw= lgb.LGBMClassifier(reg_alpha=1,num_leaves=300)

      weights = np.linspace(0.0,0.99,10)

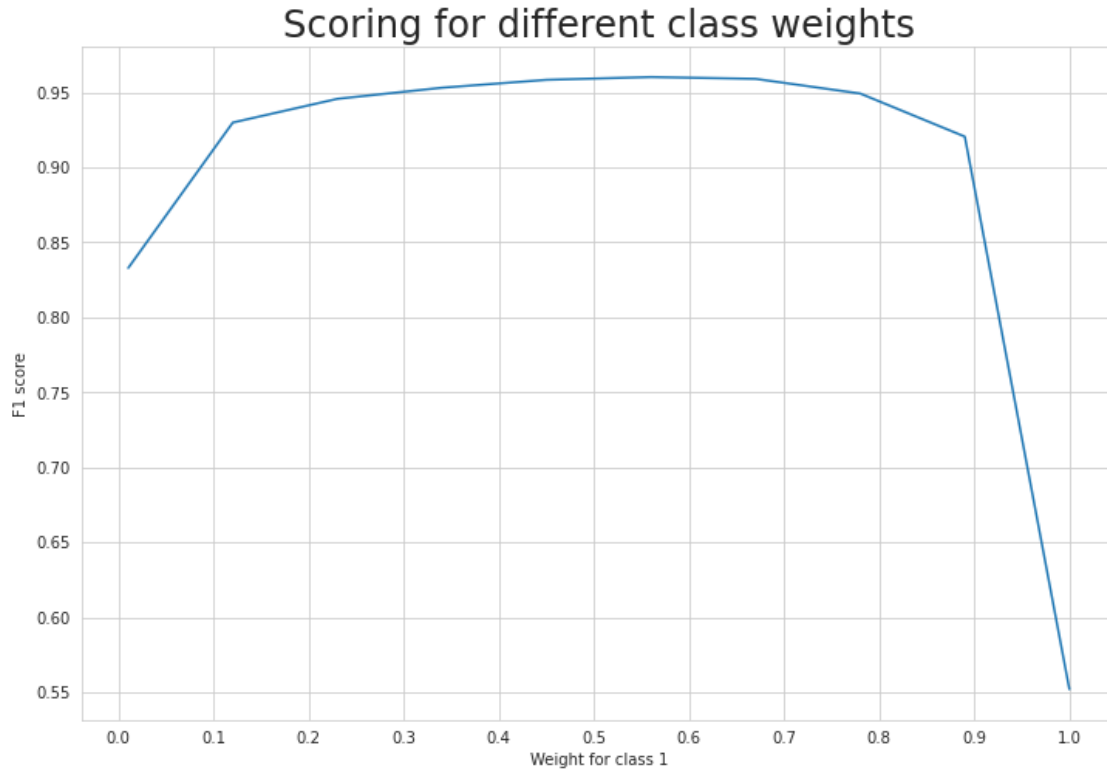
      param_grid = {'class_weight': [{0:x, 1:1.0-x} for x in weights]}

      gridsearch = GridSearchCV(estimator= lgb_cw,
                                param_grid= param_grid,
                                cv=StratifiedKFold(),
                                n_jobs=-1,
                                scoring='f1',
                                verbose=2).fit(train_fin4, train_y)

      sns.set_style('whitegrid')
      plt.figure(figsize=(12,8))
      weigh_data = pd.DataFrame({'score': gridsearch.cv_results_['mean_test_score'],
                                →'weight': (1- weights)})
      sns.lineplot(weigh_data['weight'], weigh_data['score'])
      plt.xlabel('Weight for class 1')
      plt.ylabel('F1 score')
      plt.xticks([round(i/10,1) for i in range(0,11,1)])
      plt.title('Scoring for different class weights', fontsize=24)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
[ ]: Text(0.5, 1.0, 'Scoring for different class weights')
```



```
[ ]: print(gridsearch.best_params_)

{'class_weight': {0: 0.44, 1: 0.56}}
```

```
[ ]: lgb_cw = lgb.LGBMClassifier(reg_alpha=1,num_leaves=300,class_weight={0: 0.44, 1:
    ↳ 0.56})
lgb_cw.fit(train_fin4, train_y)
```

```
[ ]: LGBMClassifier(class_weight={0: 0.44, 1: 0.56}, num_leaves=300, reg_alpha=1)
```

```
[ ]: train_y_pred_cw49= lgb_cw.predict_proba(train_fin4)
train_y_pred_cw49= train_y_pred_cw49[:,1]

cv_y_pred_cw49= lgb_cw.predict_proba(cv_fin4)
cv_y_pred_cw49= cv_y_pred_cw49[:,1]
```

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred_cw49)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred_cw49)
```

Confusion Matrix for the Train Data
The Weighted Recall Score: 0.9747832356998263

The Weighted Precision Score: 0.9748376482722947
The Weighted F1 Score: 0.9747221695098178

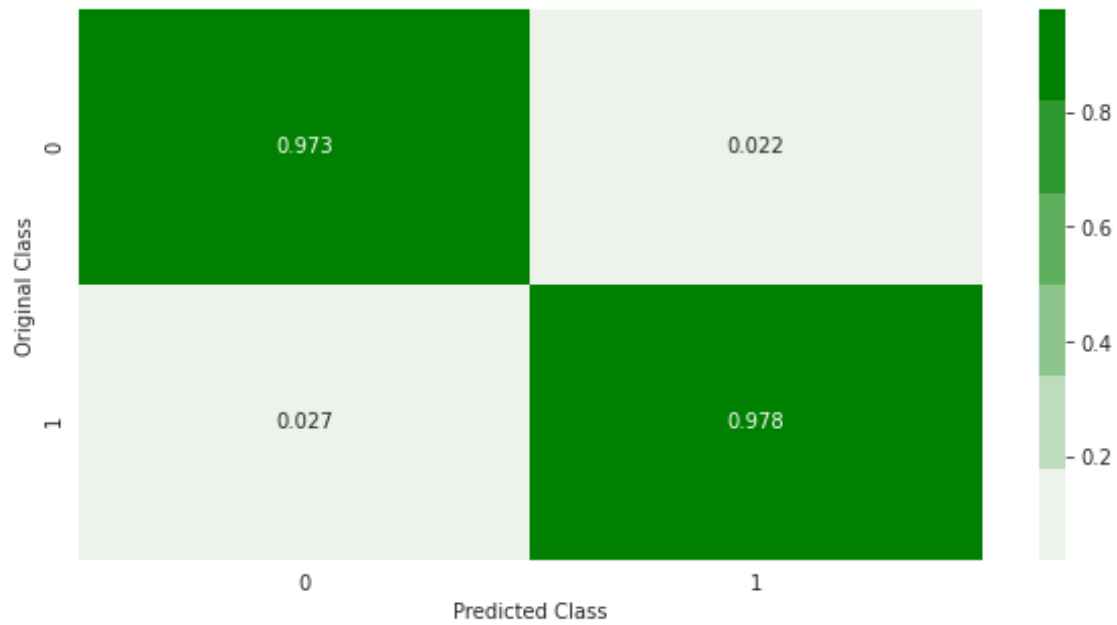
=====

=====

----- Confusion matrix

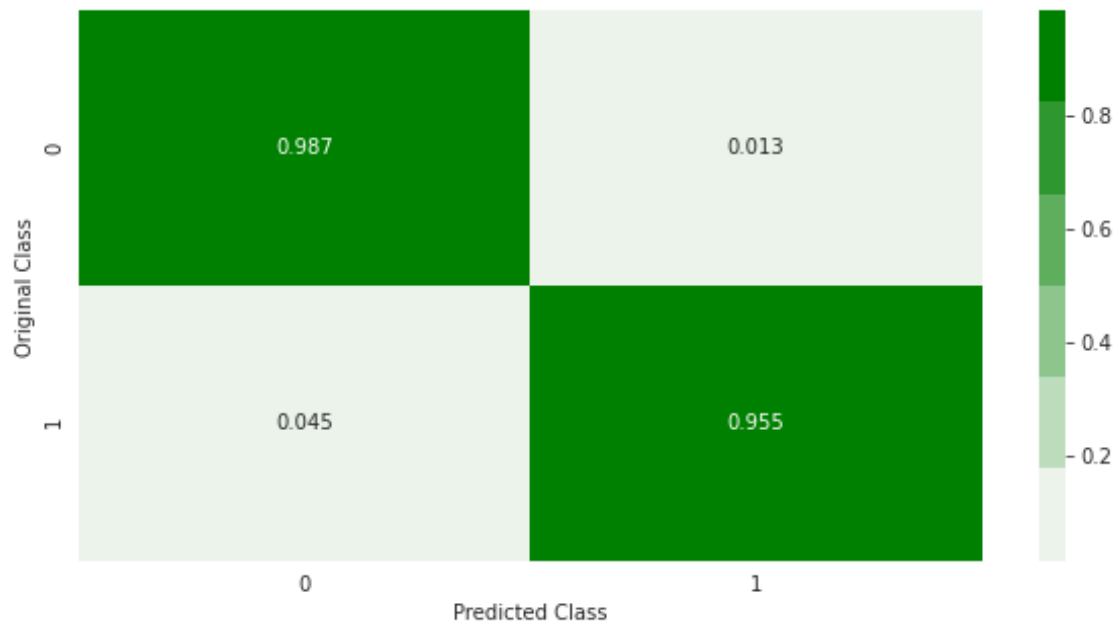


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

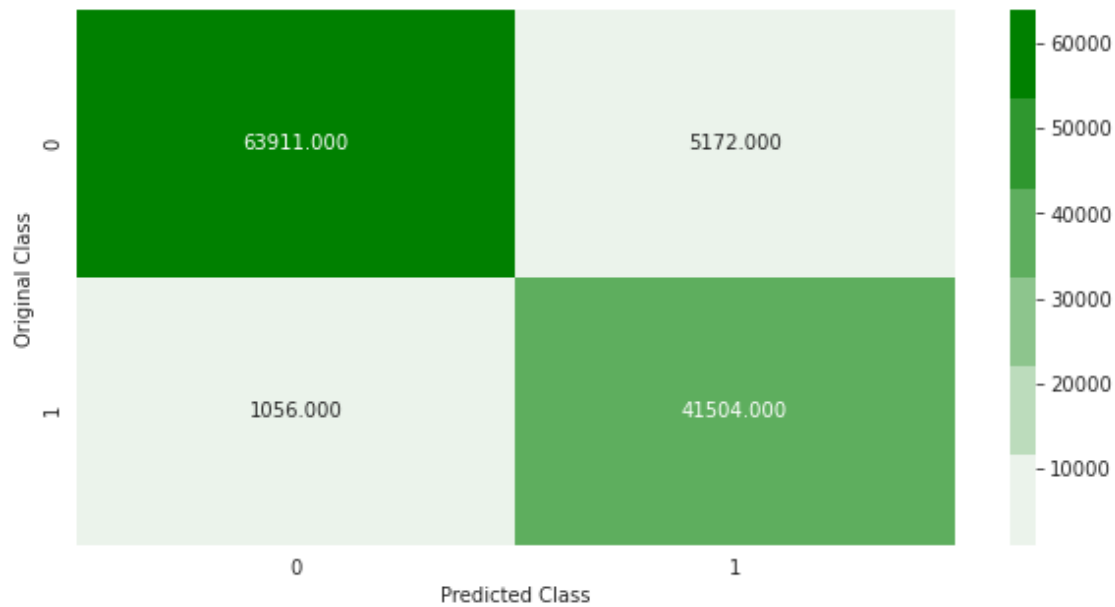
The Weighted Recall Score: 0.9442150425911163
The Weighted Precision Score: 0.947700940038562
The Weighted F1 Score: 0.9446451424231506

=====

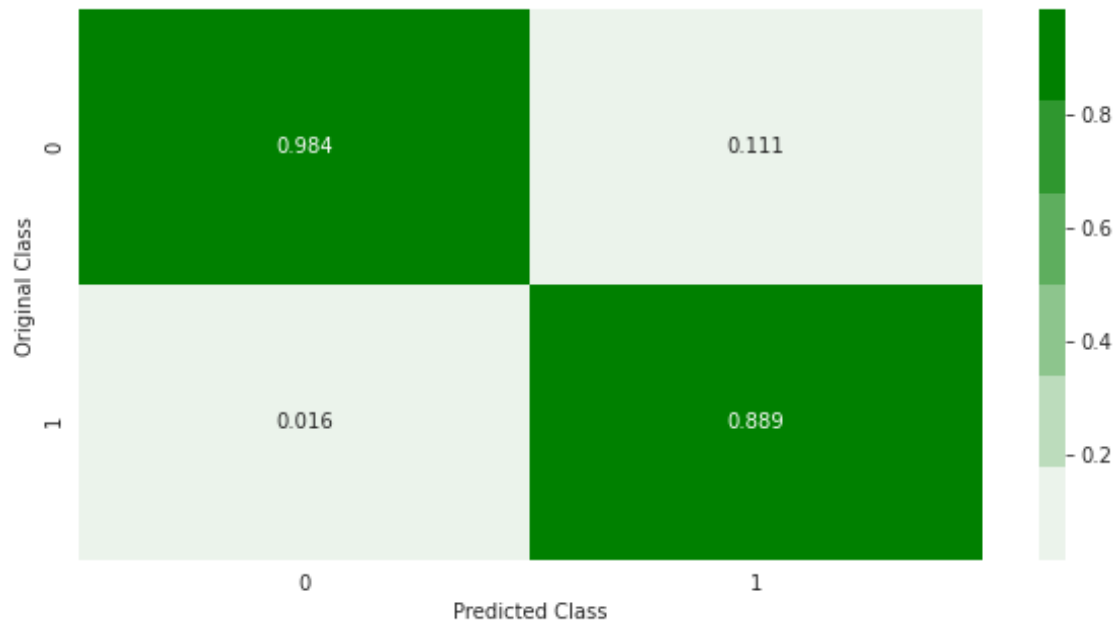
=====

=====

----- Confusion matrix

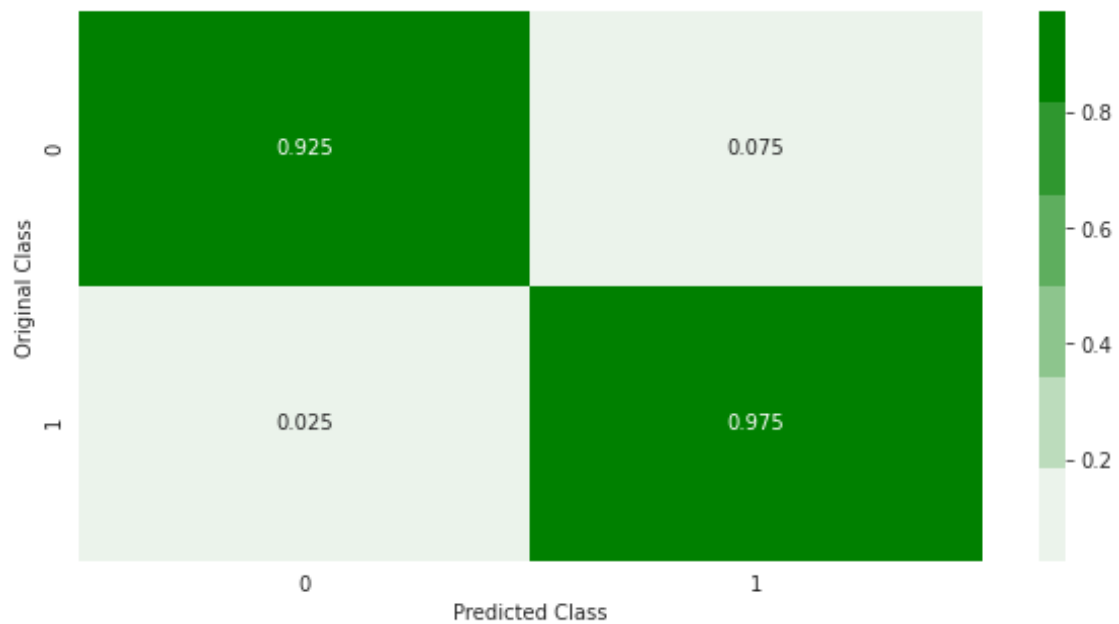


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred_cw49)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred_cw49)
```

Confusion Matrix for Best Thresold for the Train Data

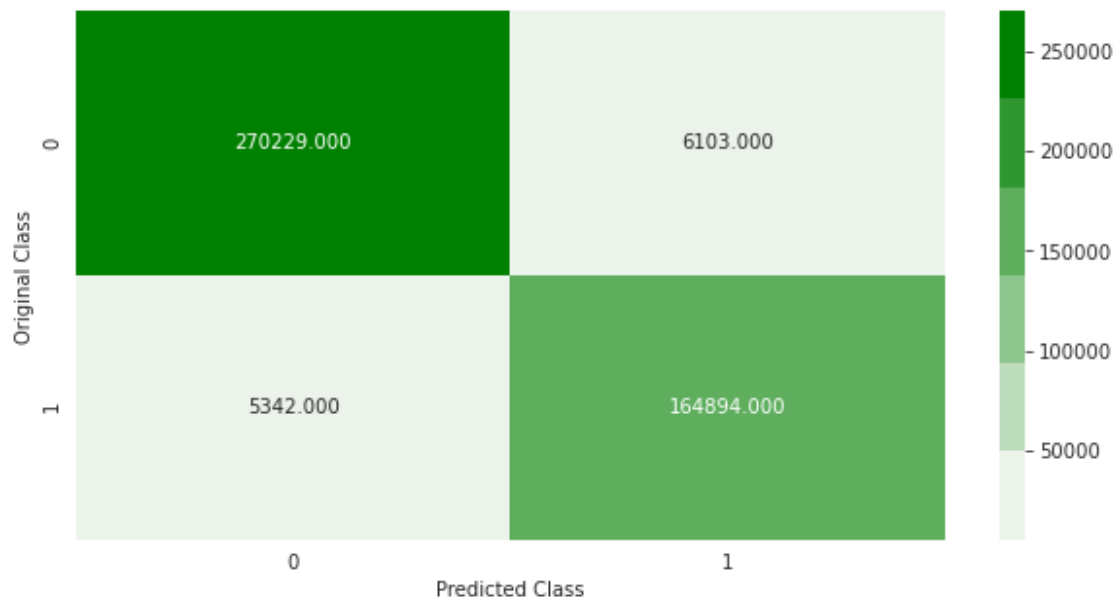
best train threshold : 0.40945335720582726

The Weighted Recall Score: 0.9743712043854463

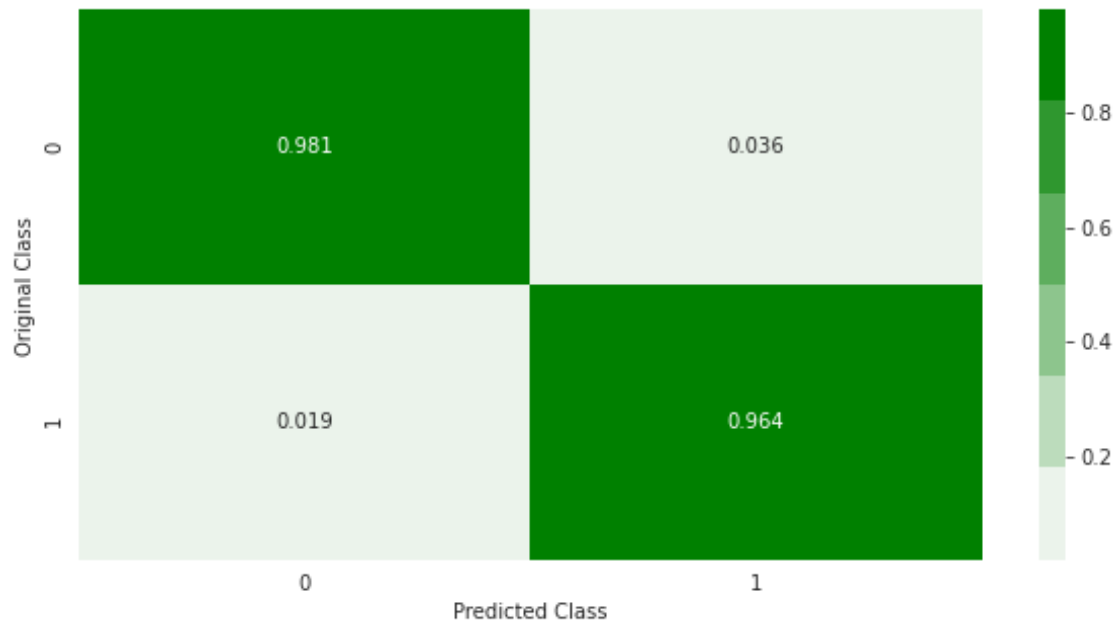
The Weighted Precision Score: 0.974398990682053

The Weighted F1 Score: 0.9743821130547151

```
=====
=====
=====
----- Confusion matrix
-----
```

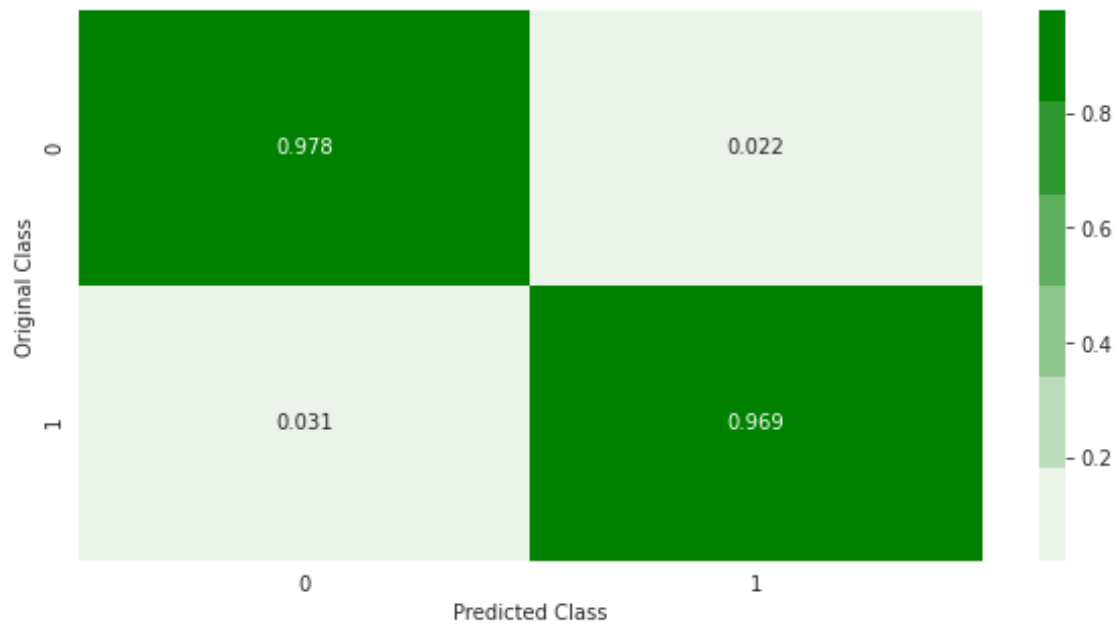


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Threshold for the CV Data

best train threshold : 0.6208150919672095
The Weighted Recall Score: 0.9599974920057683
The Weighted Precision Score: 0.9602063789525368
The Weighted F1 Score: 0.9600603119452032

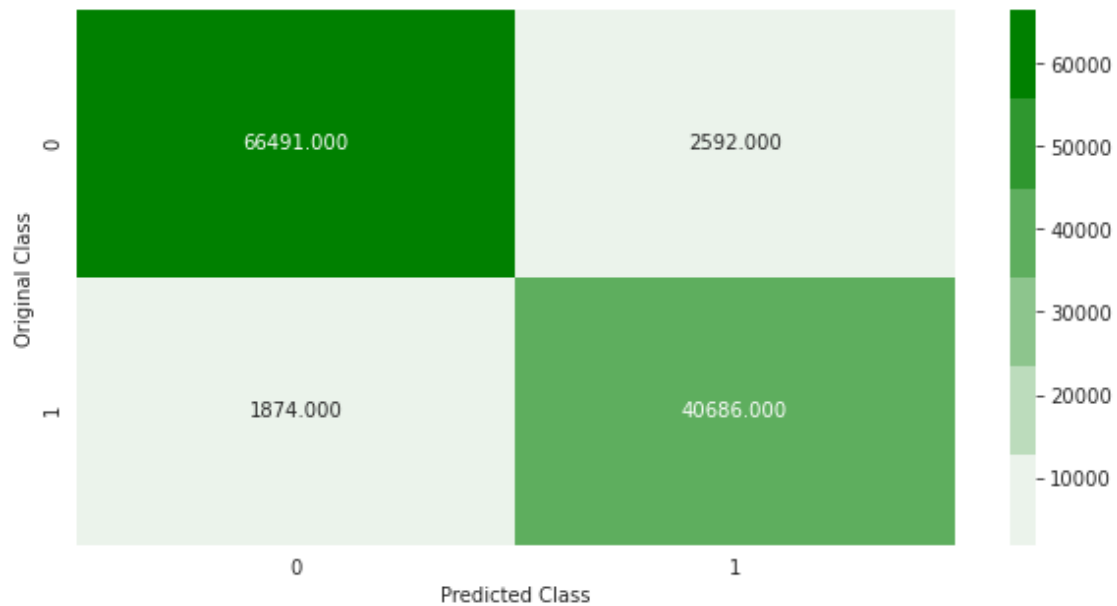
=====

=====

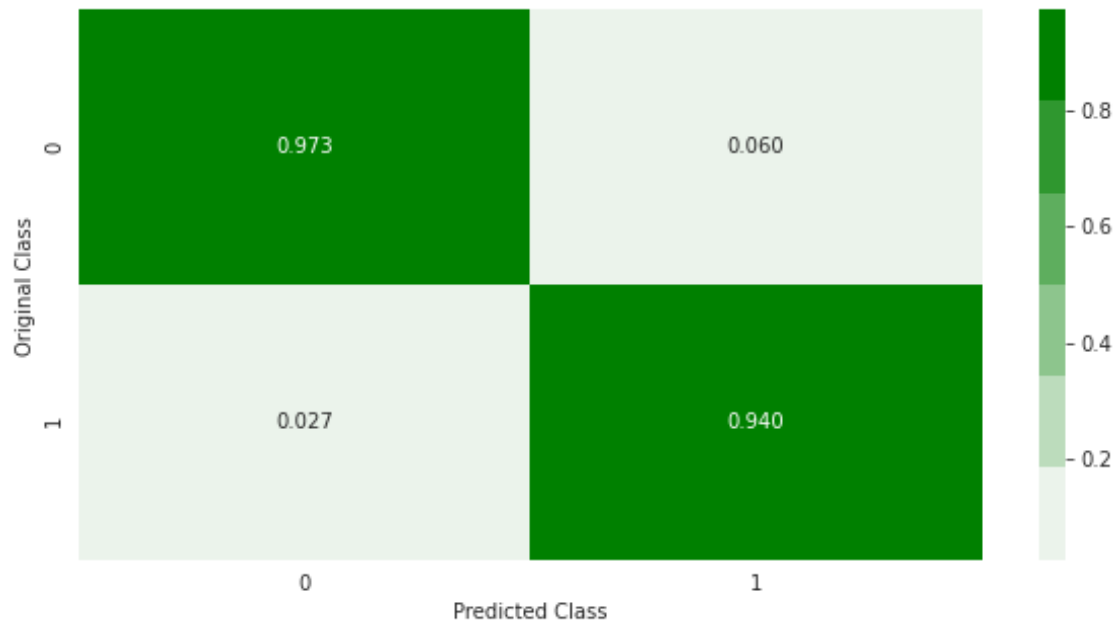
=====

=====

----- Confusion matrix

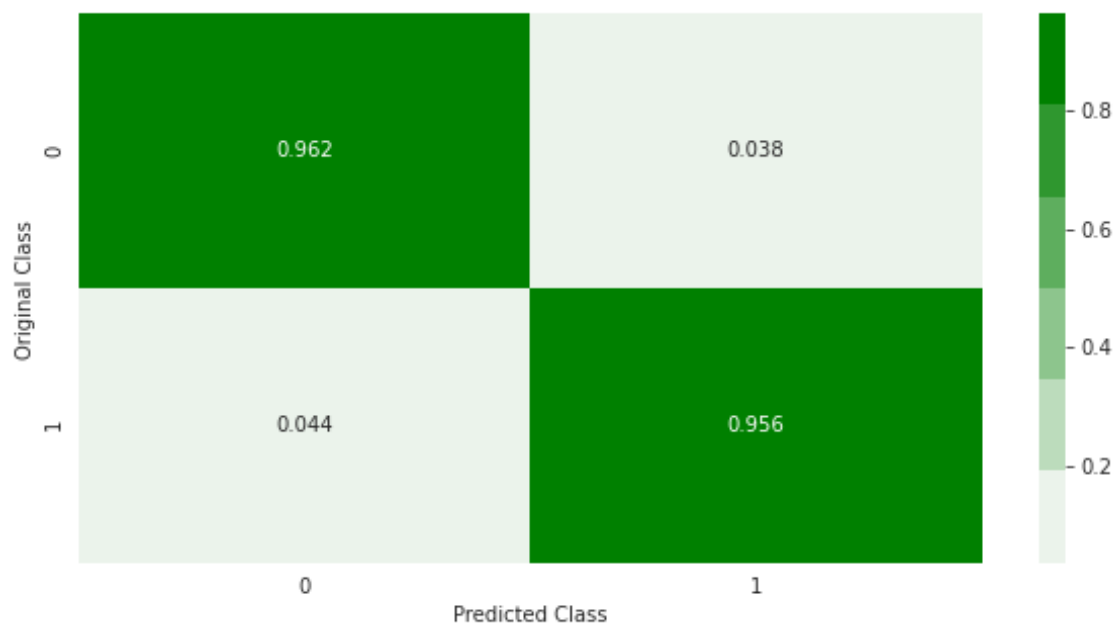


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10 Experiment 3: Calculate VIF values to look for Dummy Variable Trap and remove features with very high VIF values

10.0.1 Standardizing the below columns using the StandardScaler function

```
[ ]: cols=['State_Yes', 'State_No', 'County_Yes',  
          'County_No', 'DGC_Yes', 'DGC_No', 'BID_Yes', 'BID_No', 'CID_Yes',  
          'CID_No', 'Pvr_Yes', 'Pvr_No', 'Ap_Yes', 'Ap_No', 'CADC_Yes',  
          'CADC_No', 'comorbid', 'operating', 'operating&comorbid', 'simple',  
          'G1', 'G2', 'R1', 'R2', 'R3', 'R4']  
  
scaler= StandardScaler()  
  
for i in tqdm(cols):  
    scaler.fit(train_fin4[i].values.reshape(-1,1))  
    train_fin4[i]= scaler.transform(train_fin4[i].values.reshape(-1,1))  
    cv_fin4[i]= scaler.transform(cv_fin4[i].values.reshape(-1,1))
```

```
100%|  
| 26/26  
[00:00<00:00, 135.07it/s]
```

10.0.2 Calculating the VIF of each of the columns to look at the correlation between the variables

```
[ ]: from statsmodels.stats.outliers_influence import variance_inflation_factor  
vif_data= pd.DataFrame()  
vif_data['Feature']= train_fin4.columns  
vif_data["VIF"] = [variance_inflation_factor(train_fin4.values, i)  
                   for i in range(len(train_fin4.columns))]  
print(vif_data.shape)
```

```
(49, 2)
```

```
[ ]: print(vif_data)
```

	Feature	VIF
0	InscClaimAmtReimbursed	2.375139e+00
1	DeductibleAmtPaid	4.275599e+00
2	RenalDiseaseIndicator	1.593612e+00
3	ChronicCond_Alzheimer	2.731314e+00
4	ChronicCond_Heartfailure	2.104513e+00
5	ChronicCond_KidneyDisease	3.510568e+00
6	ChronicCond_Cancer	5.417377e+00
7	ChronicCond_ObstrPulmonary	3.721796e+00
8	ChronicCond_Depression	2.474271e+00
9	ChronicCond_Diabetes	1.756374e+00

10	ChronicCond_IschemicHeart	1.549091e+00
11	ChronicCond_Osteoporosis	3.095205e+00
12	ChronicCond_rheumatoidarthritis	3.311176e+00
13	ChronicCond_stroke	6.867418e+00
14	IPAnnualReimbursementAmt	1.989896e+00
15	IPAnnualDeductibleAmt	1.876353e+00
16	OPAnnualReimbursementAmt	3.441052e+00
17	OPAnnualDeductibleAmt	3.425232e+00
18	#_Procedures	2.244568e+00
19	#_DiagnosisCodes	1.599522e+00
20	HospitalWeeks	4.149848e+00
21	ClaimWeeks	1.383856e+00
22	State_Yes	8.230322e+06
23	State_No	inf
24	County_Yes	8.721207e+06
25	County_No	1.965715e+07
26	DGC_Yes	4.868259e+09
27	DGC_No	5.304493e+08
28	BID_Yes	5.714229e+06
29	BID_No	1.199238e+07
30	CID_Yes	2.940472e+06
31	CID_No	3.439359e+06
32	Pvr_Yes	5.309890e+06
33	Pvr_No	1.107643e+07
34	Ap_Yes	9.482256e+06
35	Ap_No	2.498421e+06
36	age	1.034420e+00
37	CADC_Yes	3.669717e+07
38	CADC_No	7.018030e+06
39	comorbid	4.962378e+06
40	operating	3.248956e+07
41	operating&comorbid	1.171098e+08
42	simple	1.818669e+07
43	G1	8.938538e+06
44	G2	8.938538e+06
45	R1	1.102771e+07
46	R2	1.726438e+06
47	R3	1.178675e+07
48	R4	2.077028e+07

10.1 Dummy Variable Trap

10.1.1 The Very high values of VIF is due to the Dummy Variable Trap hence dropping one of the dummies created by each of the variables

```
[ ]: no_cols=['State_No', 'County_No', 'DGC_No', 'BID_No', 'CID_No', 'Pvr_No', 'Ap_No', 'CADC_No', 'simple']
train_fin4.drop(no_cols,axis=1,inplace=True)
cv_fin4.drop(no_cols,axis=1,inplace=True)
```

10.1.2 Calculating the VIF after dropping the columns with dummy variable trap

```
[ ]: from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data= pd.DataFrame()
vif_data['Feature']= train_fin4.columns
vif_data["VIF"] = [variance_inflation_factor(train_fin4.values, i)
                    for i in range(len(train_fin4.columns))]
print(vif_data)
```

	Feature	VIF
0	InscClaimAmtReimbursed	2.375031
1	DeductibleAmtPaid	4.275598
2	RenalDiseaseIndicator	1.592143
3	ChronicCond_Alzheimer	2.731238
4	ChronicCond_Heartfailure	2.104513
5	ChronicCond_KidneyDisease	3.509186
6	ChronicCond_Cancer	5.416189
7	ChronicCond_ObstrPulmonary	3.720638
8	ChronicCond_Depression	2.473698
9	ChronicCond_Diabetes	1.756353
10	ChronicCond_IschemicHeart	1.549060
11	ChronicCond_Osteoporosis	3.093420
12	ChronicCond_rheumatoidarthritis	3.310835
13	ChronicCond_stroke	6.865476
14	IPAnnualReimbursementAmt	1.989904
15	IPAnnualDeductibleAmt	1.875896
16	OPAnnualReimbursementAmt	3.441047
17	OPAnnualDeductibleAmt	3.425159
18	#_Procedures	2.244576
19	#_DiagnosisCodes	1.599494
20	HospitalWeeks	4.149672
21	ClaimWeeks	1.383851
22	State_Yes	1.016849
23	County_Yes	1.010703
24	DGC_Yes	1.002316
25	BID_Yes	1.373763
26	CID_Yes	1.554839
27	Pvr_Yes	1.083912

28	Ap_Yes	1.254956
29	age	1.034385
30	CADC_Yes	1.009199
31	comorbid	1.340115
32	operating	1.126418
33	operating&comorbid	1.106856
34	G1	1.011287
35	R1	6.358506
36	R2	5.117445
37	R3	2.565035

10.1.3 Dropping all the columns with VIF value more than 4

```
[ ]: vif_cols=[]
for i in range(vif_data.shape[0]):
    if vif_data['VIF'][i]>=4:
        vif_cols.append(vif_data['Feature'][i])
print(vif_cols)
```

```
['DeductibleAmtPaid', 'ChronicCond_Cancer', 'ChronicCond_stroke',
'HospitalWeeks', 'R1', 'R2']
```

```
[ ]: vif_cols= ['DeductibleAmtPaid', 'ChronicCond_Cancer', 'ChronicCond_stroke',
→'HospitalWeeks', 'R1', 'R2']
train_fin4.drop(vif_cols,axis=1,inplace=True)

cv_fin4.drop(vif_cols,axis=1,inplace=True)
```

```
[ ]: with open('/home/megha_murthy_n/train_fin32.pkl','wb') as tr_f32:
    pickle.dump(train_fin4,tr_f32)
with open('/home/megha_murthy_n/cv_fin32.pkl','wb') as cv_f32:
    pickle.dump(cv_fin4,cv_f32)
```

```
[ ]: with open('/home/megha_murthy_n/train_fin32.pkl','rb') as tr_f32:
    train_fin32= pd.read_pickle(tr_f32)
with open('/home/megha_murthy_n/cv_fin32.pkl','rb') as cv_f32:
    cv_fin32= pd.read_pickle(cv_f32)
```

10.2 Supervised Models after dropping variables with Dummy Variable Trap and variables with high VIF values

10.3 Logistic Regression after dropping the features with high VIF Values

10.3.1 Running Logistic Regression on the final dataset post dropping the variables with large VIF and dummy variable trap

```
[ ]: from sklearn.linear_model import LogisticRegression
      from sklearn.model_selection import RandomizedSearchCV

      param = {'C': [0.00001, 0.01, 1, 100, 1000], 'penalty' : ['l1', 'l2']}
      lr = LogisticRegression()
      lr_tune = RandomizedSearchCV(lr, param, cv=5)
      lr_tune.fit(train_fin4, train_y)

      print('best parameter : ', lr_tune.best_params_)
```

```
best parameter : {'penalty': 'l2', 'C': 1}
```

```
[ ]: lr_best = LogisticRegression(C=1, penalty='l2')
      lr_best.fit(train_fin4, train_y)

      sig_clf = CalibratedClassifierCV(lr_best, method="sigmoid")
      sig_clf.fit(train_fin4, train_y)

      train_y_pred = sig_clf.predict_proba(train_fin4)
      train_y_pred = train_y_pred[:, 1]
      cv_y_pred = sig_clf.predict_proba(cv_fin4)
      cv_y_pred = cv_y_pred[:, 1]
```

10.3.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)
```

Confusion Matrix for the Train Data

The Weighted Recall Score: 0.6244424141452142

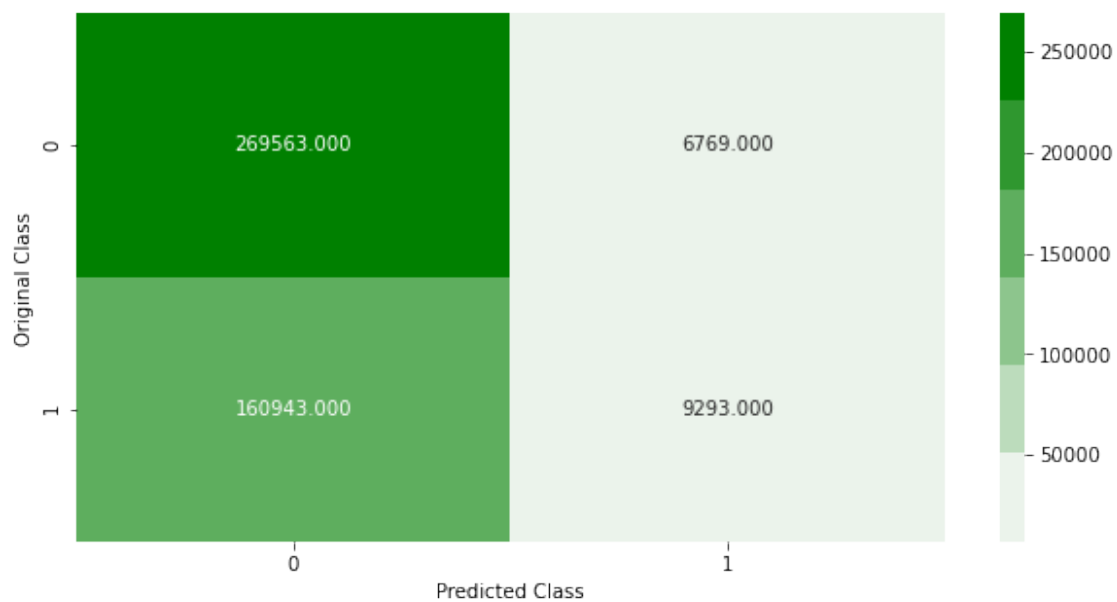
The Weighted Precision Score: 0.6080146513408525

The Weighted F1 Score: 0.5100008700079026

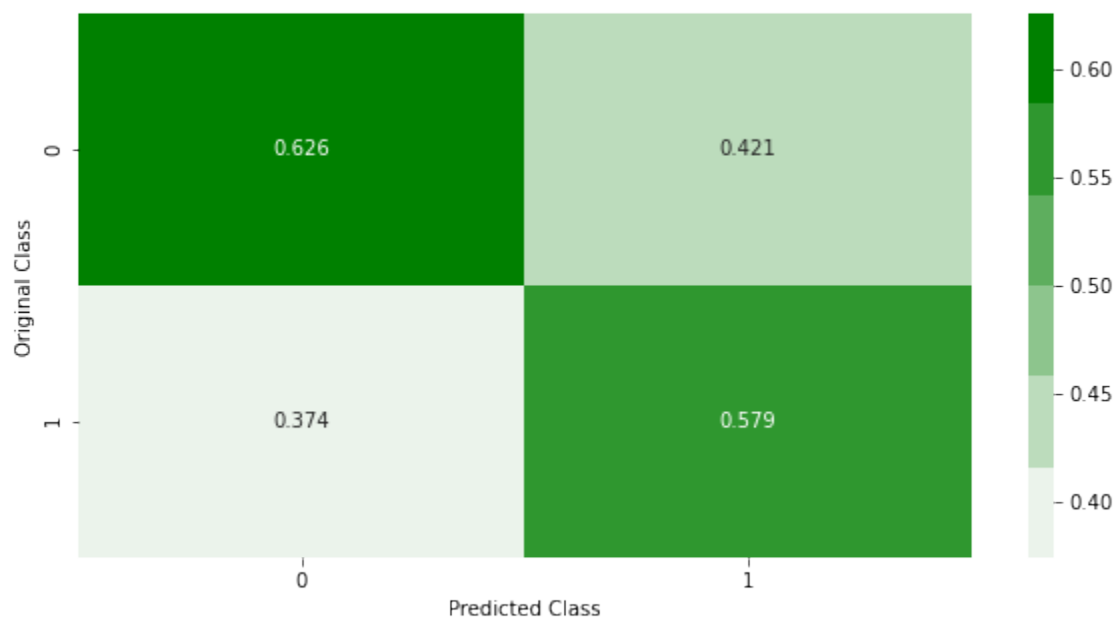
```
=====
=====
=====
```

=====

----- Confusion matrix

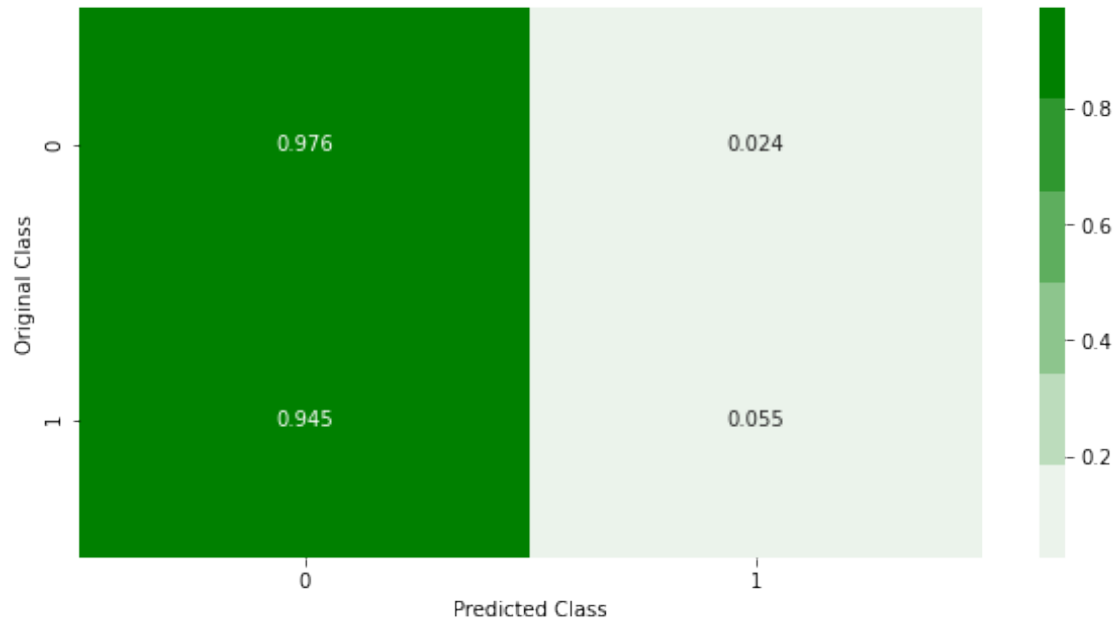


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

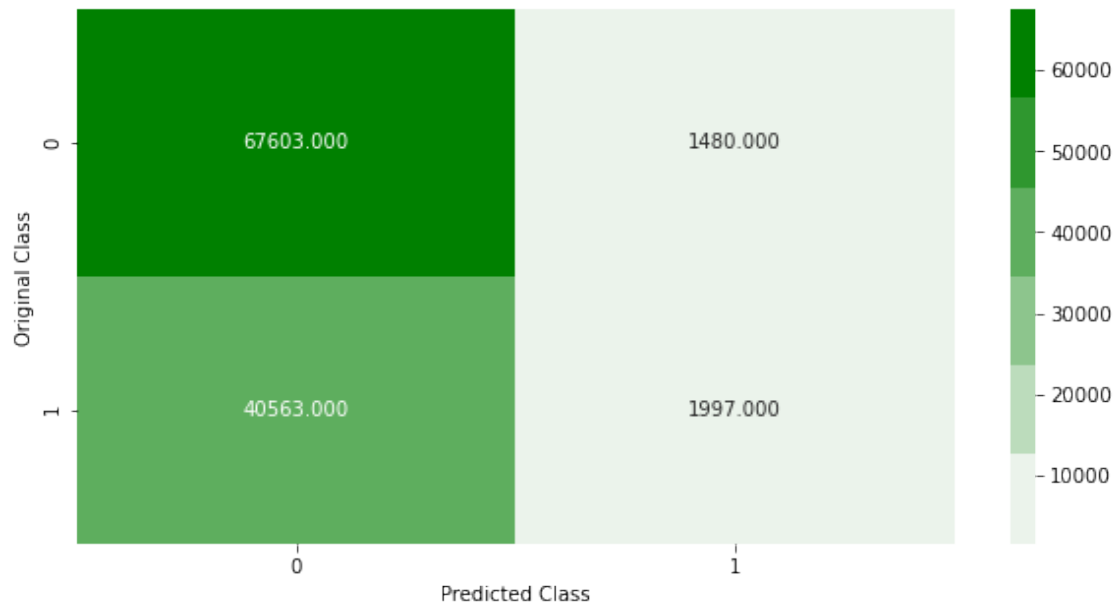
Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.6234157090010122

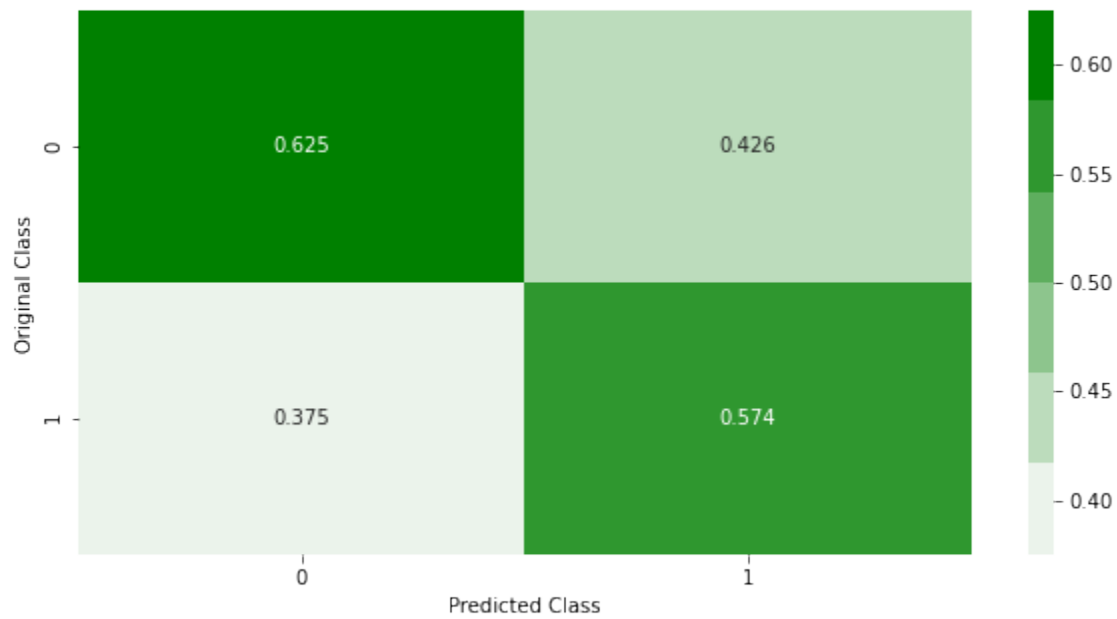
The Weighted Precision Score: 0.605685524382721

The Weighted F1 Score: 0.5050835350657582

=====
=====
=====
=====
----- Confusion matrix

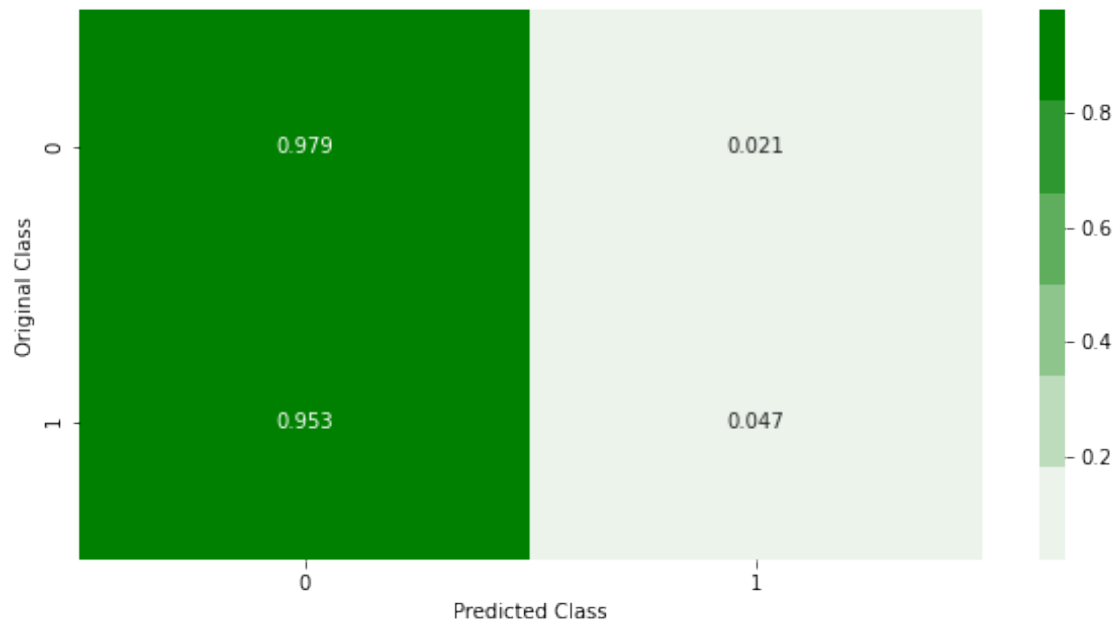


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.3.3 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

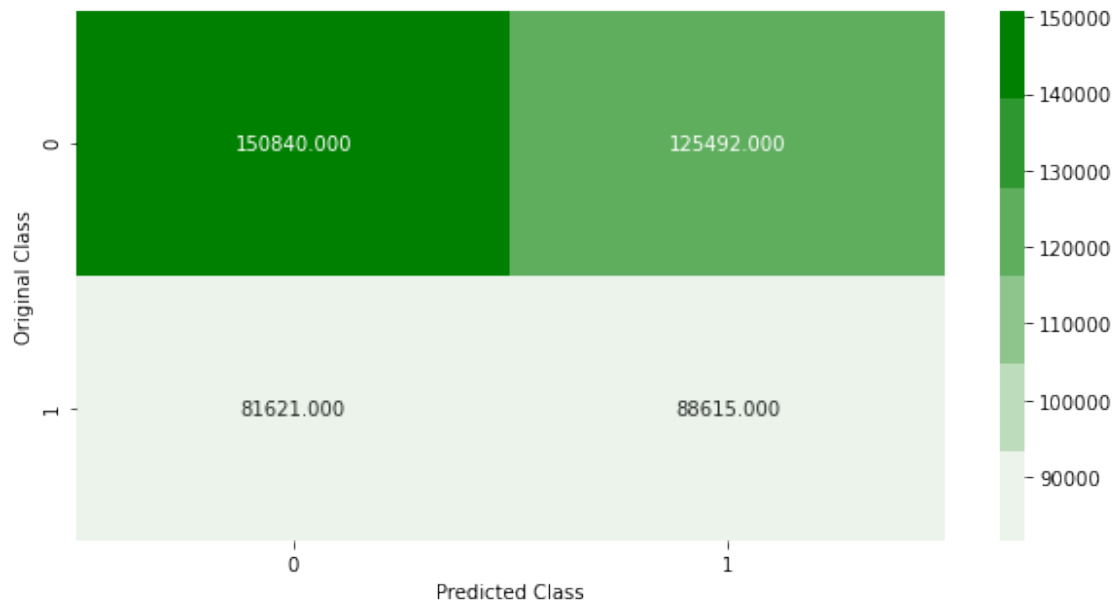
best train threshold : 0.3716093098780414

The Weighted Recall Score: 0.5362117303523763

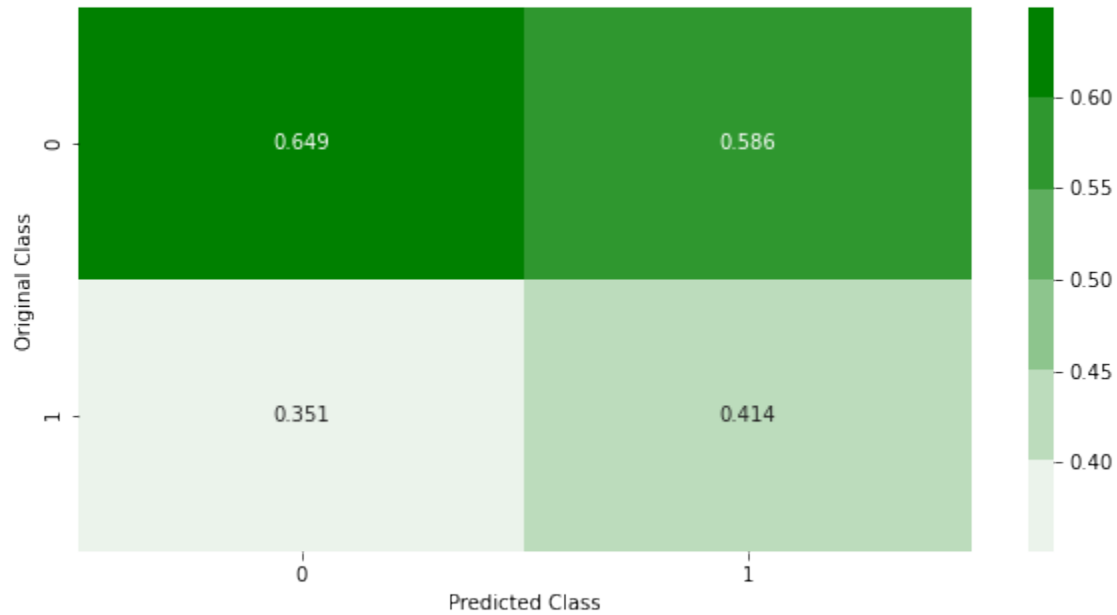
The Weighted Precision Score: 0.5592983317121565

The Weighted F1 Score: 0.542686168642926

```
=====
=====
=====
----- Confusion matrix
-----
```

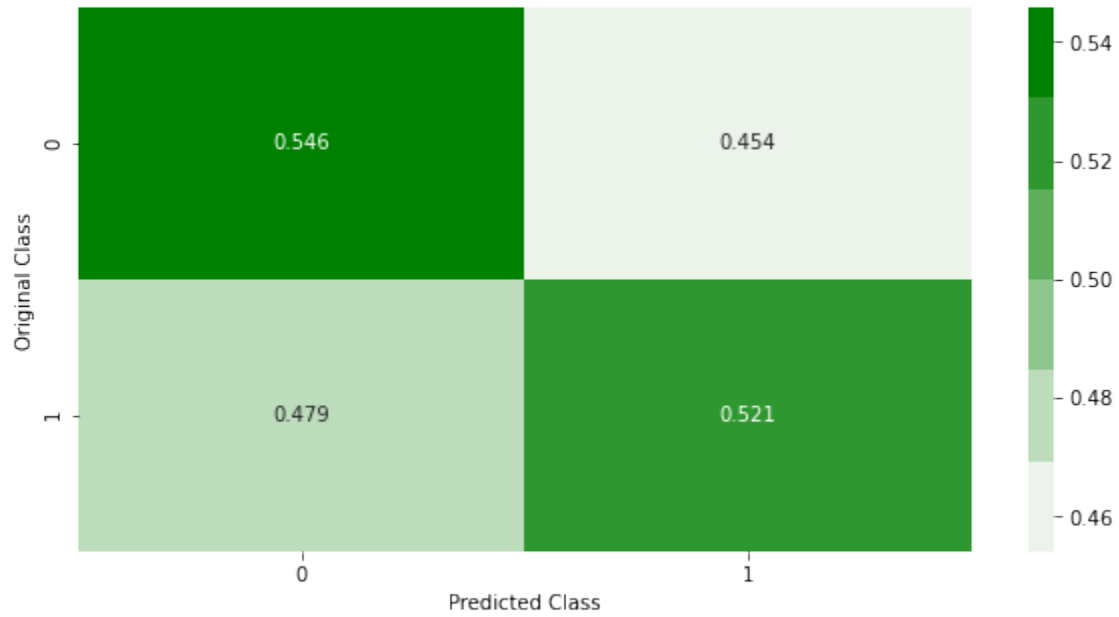



----- Precision matrix



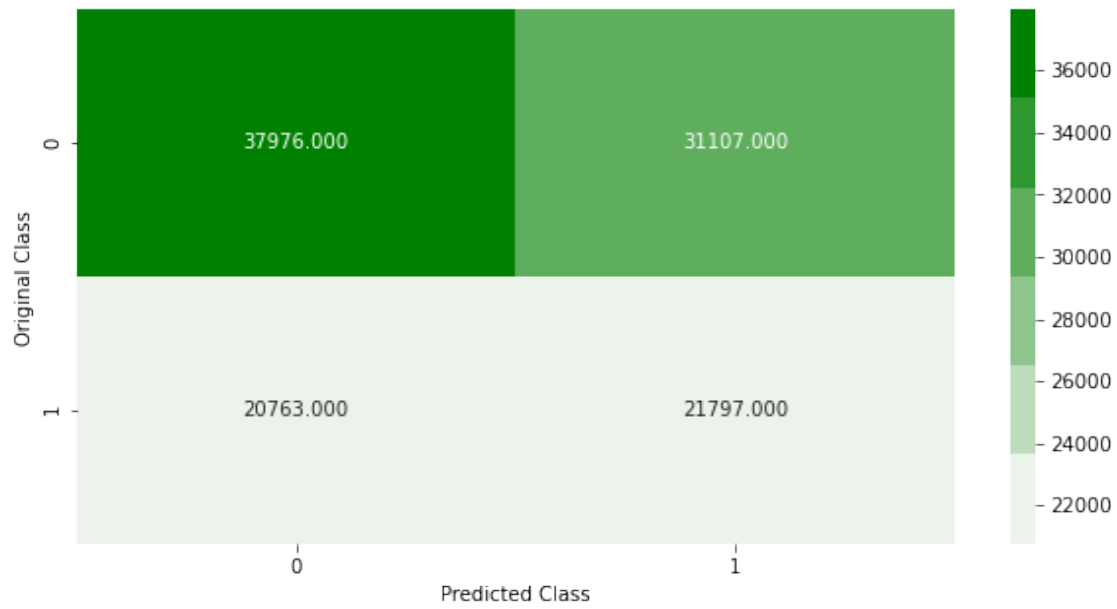
Sum of columns in precision matrix [1. 1.]

----- Recall matrix

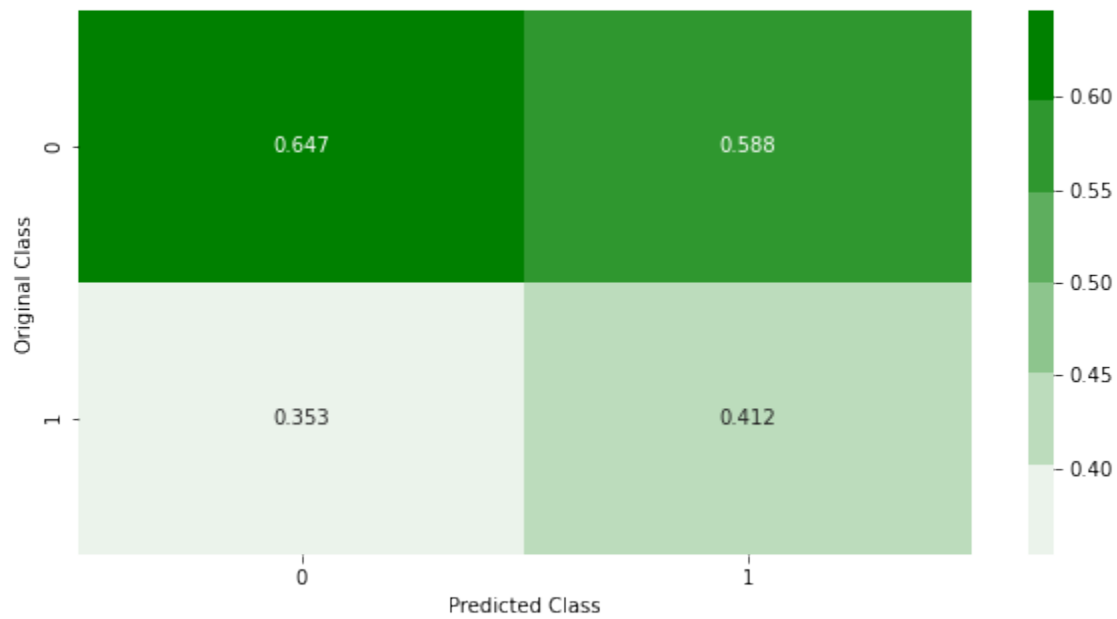


Sum of rows in precision matrix [1. 1.]
 Confusion Matrix for Best Thresold for the CV Data
 best train threshold : 0.37095551706245133
 The Weighted Recall Score: 0.5353940685936422
 The Weighted Precision Score: 0.5571220570902515
 The Weighted F1 Score: 0.5417661263065336

```
=====
=====
=====
=====
----- Confusion matrix
-----
```

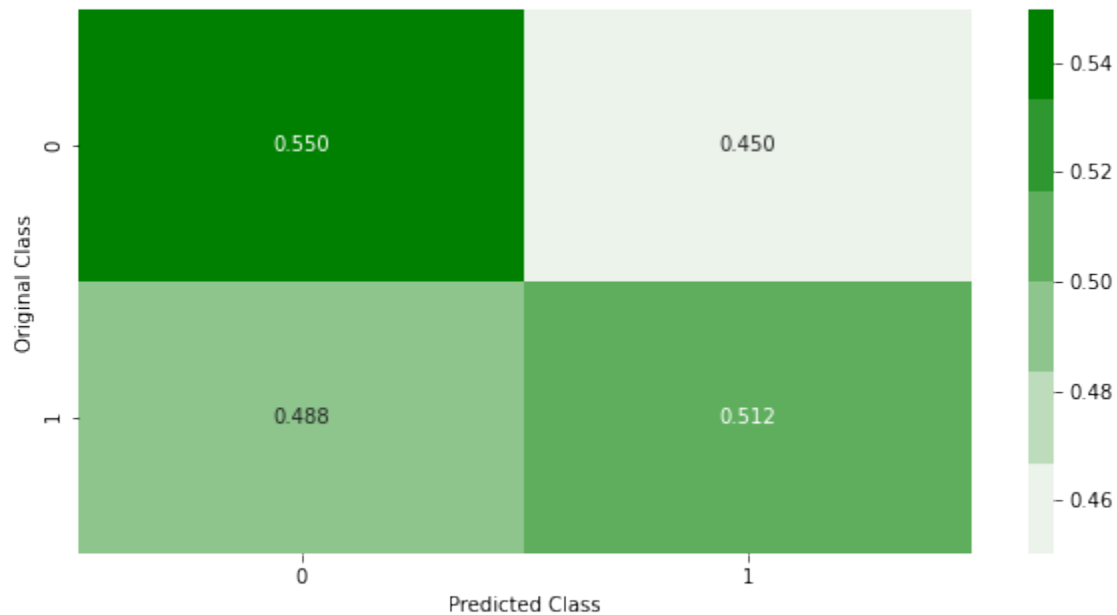


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.4 Random Forests after dropping the features with high VIF Values

10.4.1 Running Random Forests on the final dataset post dropping the variables with large VIF and dummy variable trap

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(class_weight='balanced')
param = { 'n_estimators': [10,50,100], 'max_depth' : [
    ↳ [2,6,10,14], 'min_samples_split': [5,50,100,250], 'criterion' : ['gini']}

rf_tune = RandomizedSearchCV(rf,param,cv=10,n_jobs=-1,verbose=1)
rf_tune.fit(train_fin4, train_y)
print('best parameter :',rf_tune.best_params_)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits
 best parameter : {'n_estimators': 100, 'min_samples_split': 5, 'max_depth': 14, 'criterion': 'gini'}

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf_best = [
    ↳ RandomForestClassifier(max_depth=14,min_samples_split=5,criterion='gini',n_estimators=100)
rf_best.fit(train_fin4,train_y)

sig_clf = CalibratedClassifierCV(rf_best, method="sigmoid")
```

```

sig_clf.fit(train_fin4,train_y)

train_y_pred = rf_best.predict_proba(train_fin4)
train_y_pred = train_y_pred[:,1]
cv_y_pred = rf_best.predict_proba(cv_fin4)
cv_y_pred = cv_y_pred[:,1]

```

10.4.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```

[:]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)

```

Confusion Matrix for the Train Data

The Weighted Recall Score: 0.7929117178122929

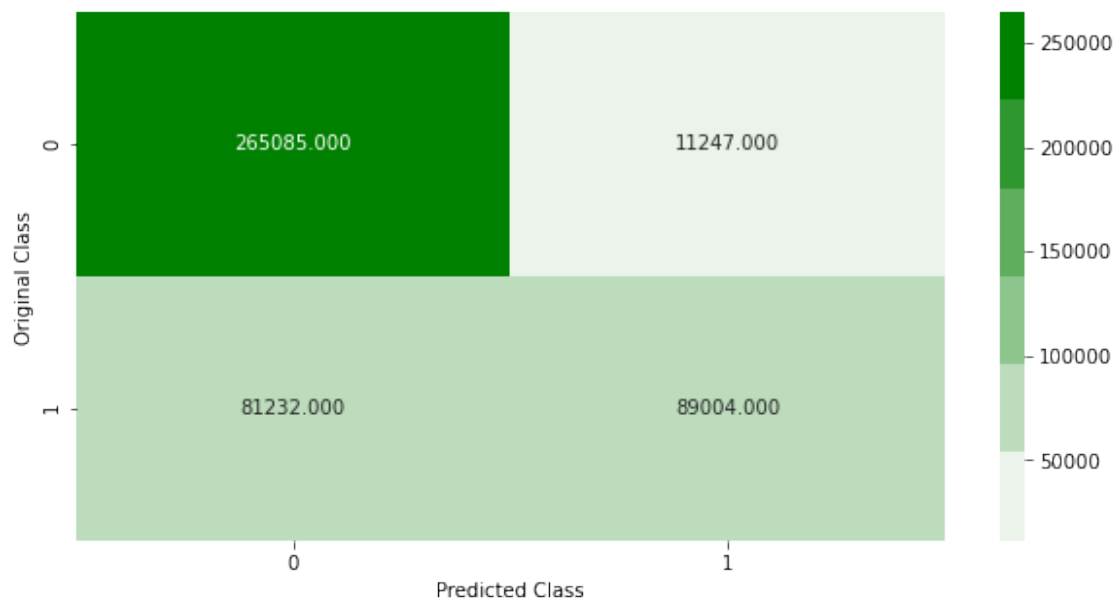
The Weighted Precision Score: 0.8120894300185671

The Weighted F1 Score: 0.7777592525689894

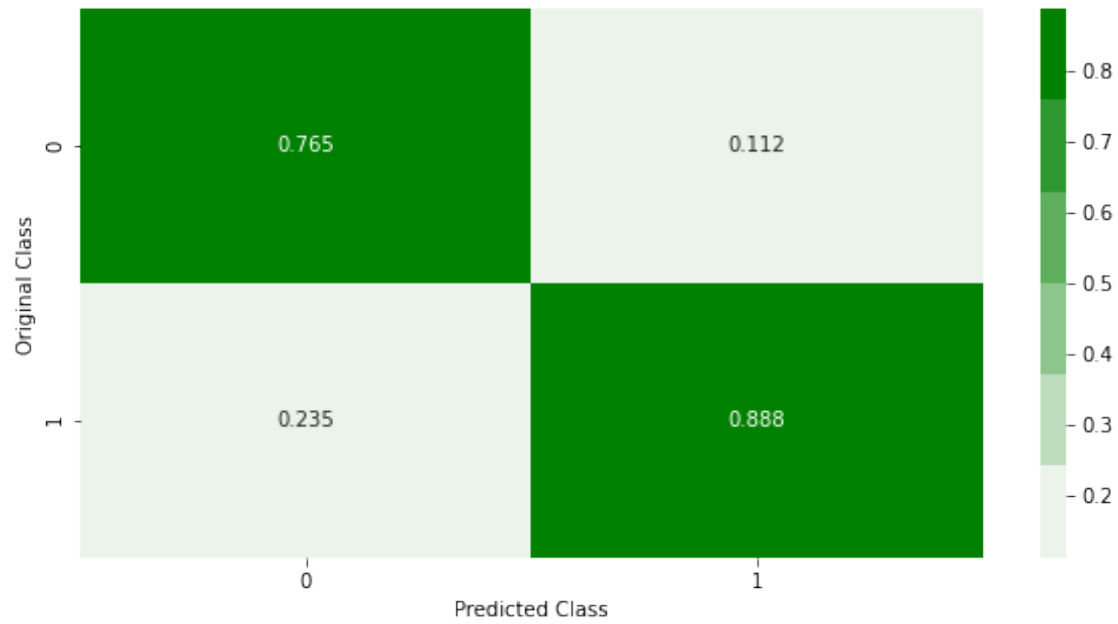
```

=====
=====
=====
----- Confusion matrix
-----

```

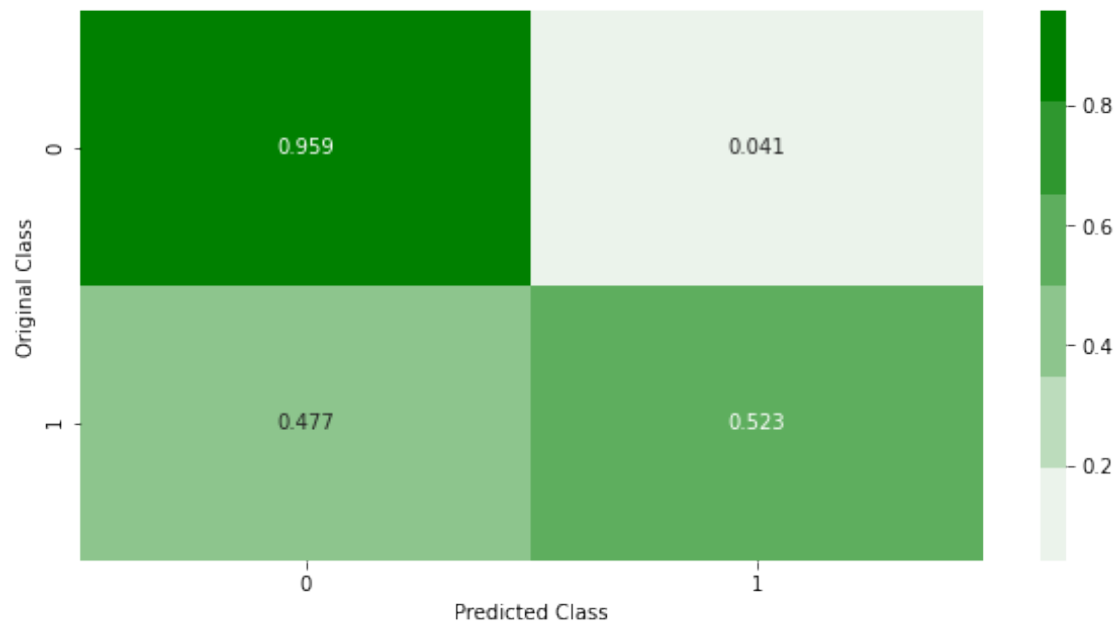


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.7681090619205861
The Weighted Precision Score: 0.7663916040694735
The Weighted F1 Score: 0.7670109433672729

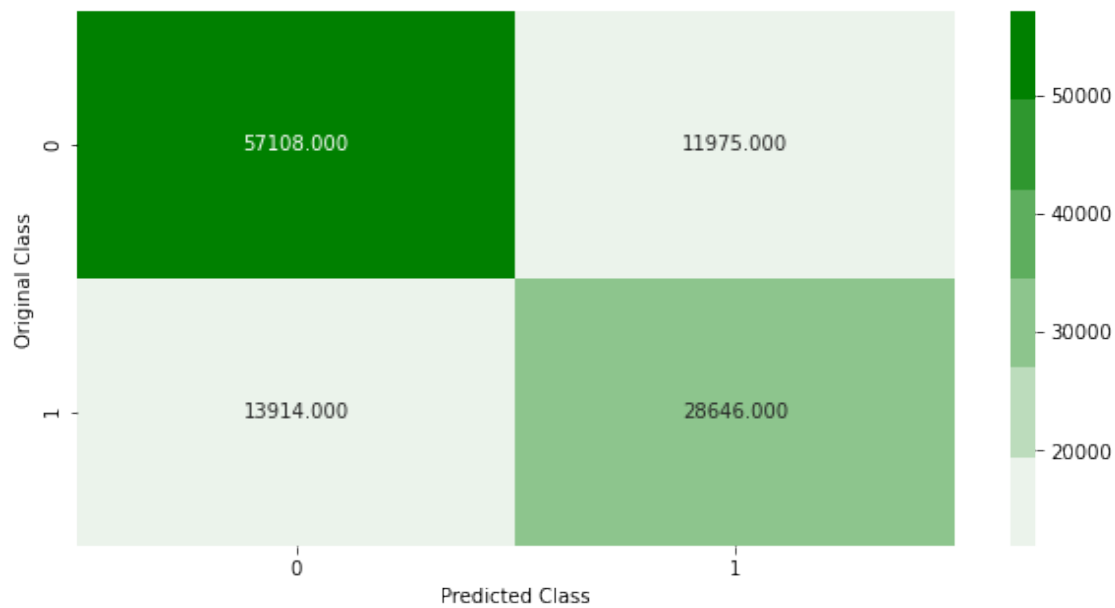
=====

=====

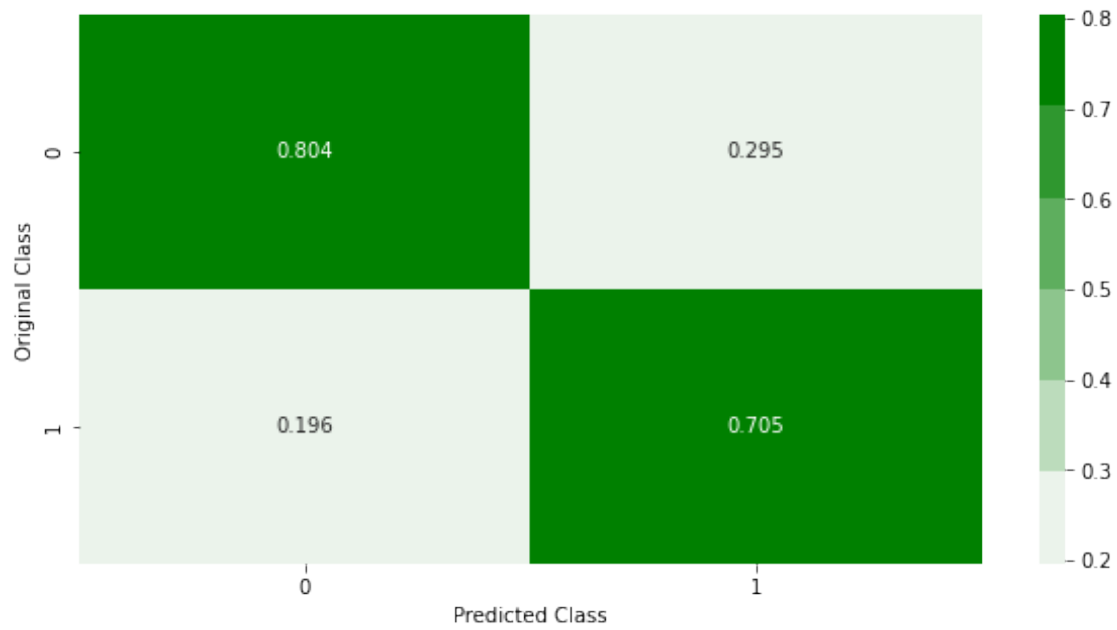
=====

=====

----- Confusion matrix

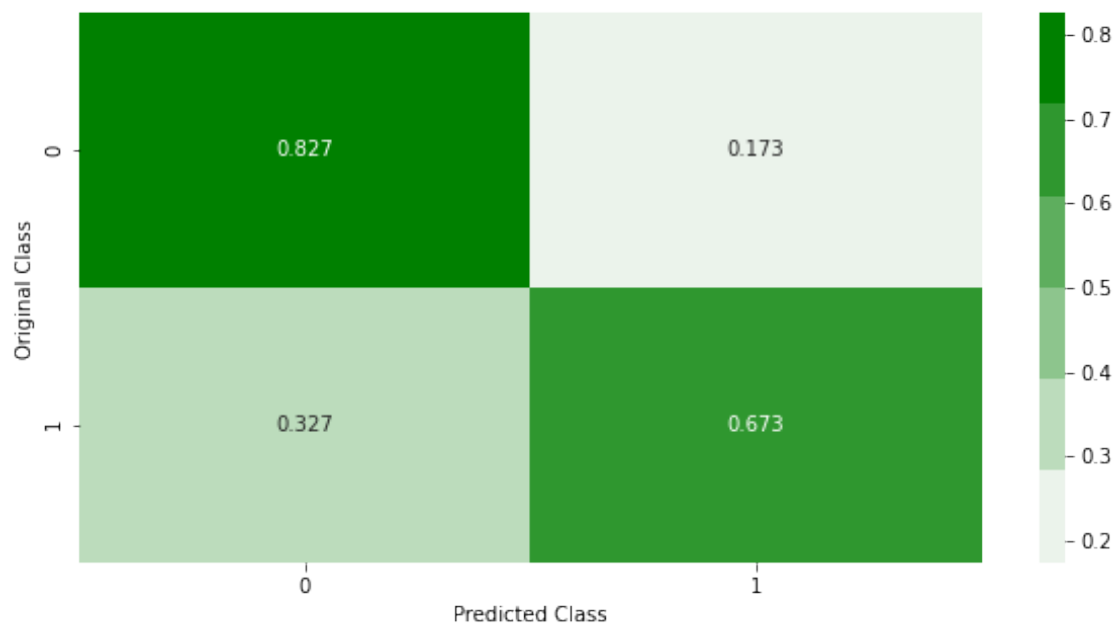


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.4.3 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

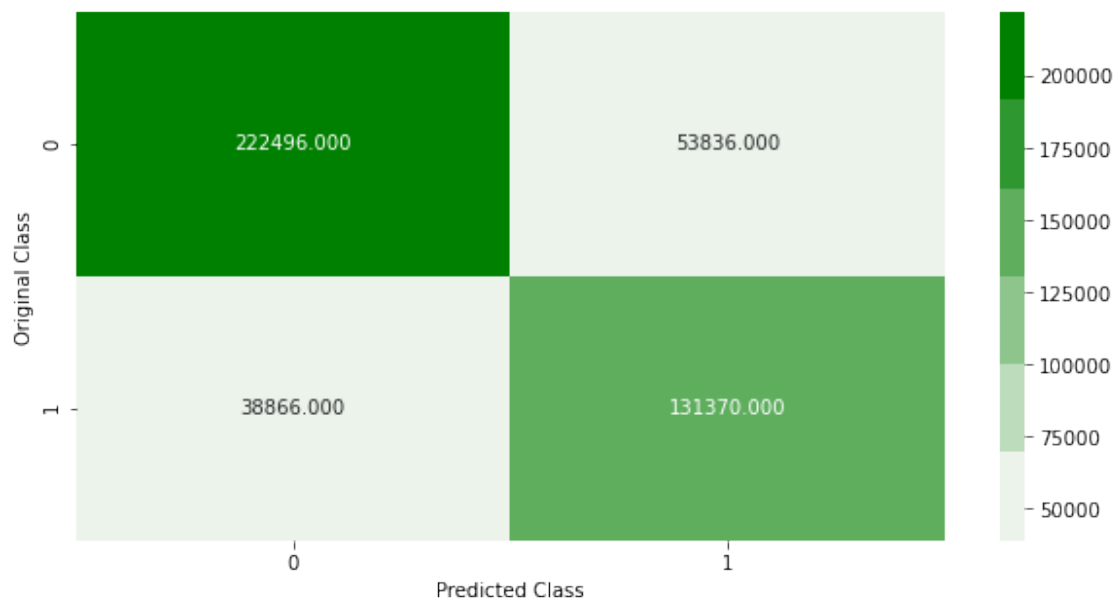
best train threshold : 0.3971542059215449

The Weighted Recall Score: 0.7924123537736694

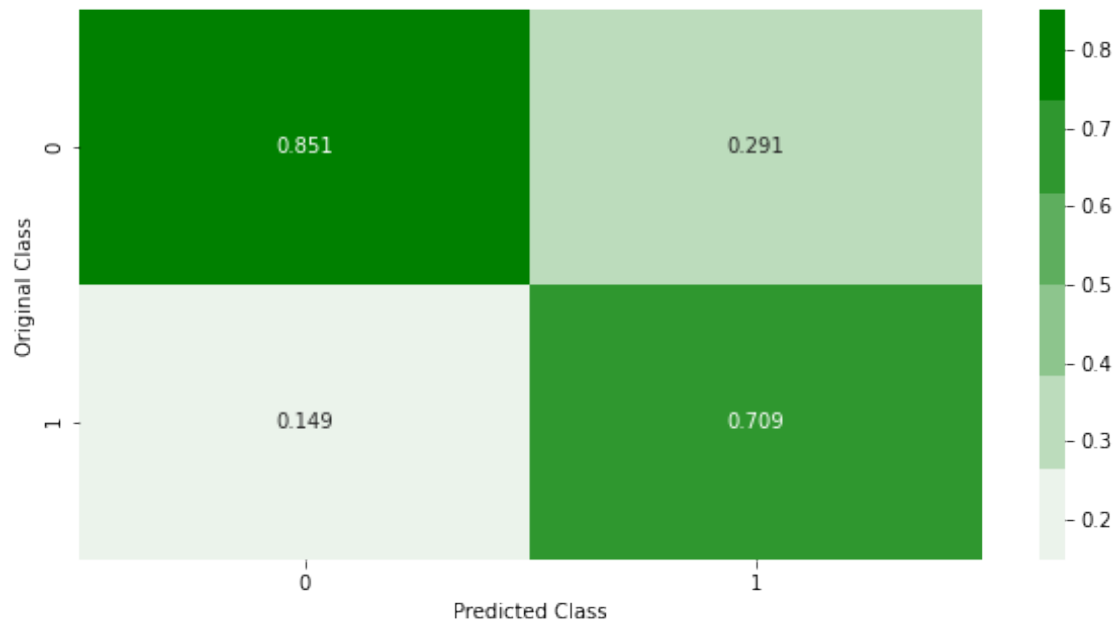
The Weighted Precision Score: 0.7971717235167012

The Weighted F1 Score: 0.7938940585399948

```
=====
=====
=====
----- Confusion matrix
-----
```

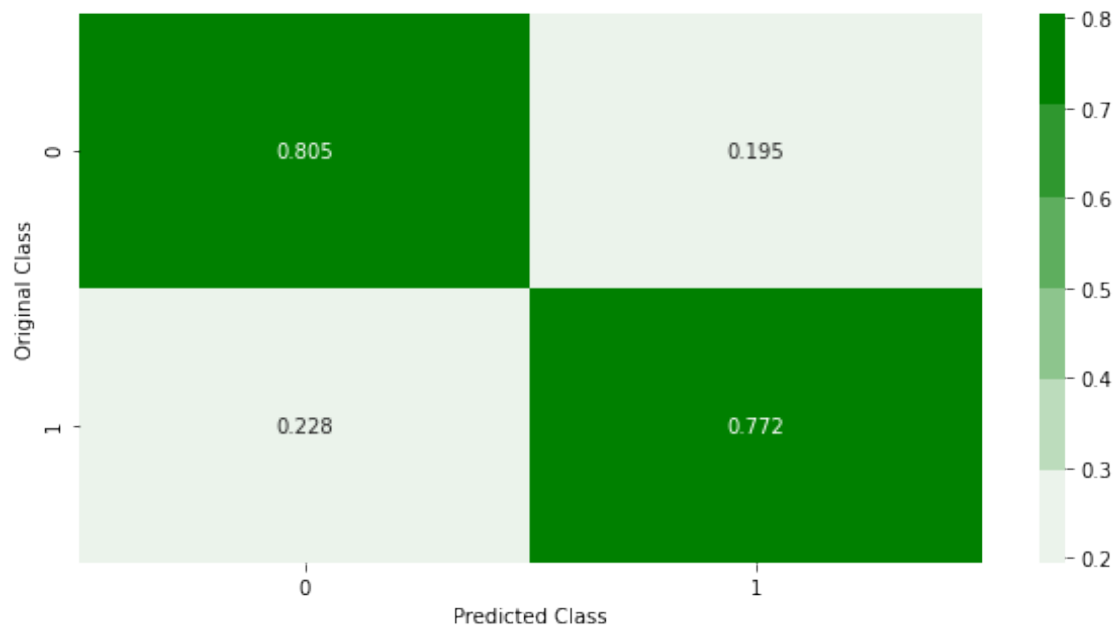


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Threshold for the CV Data

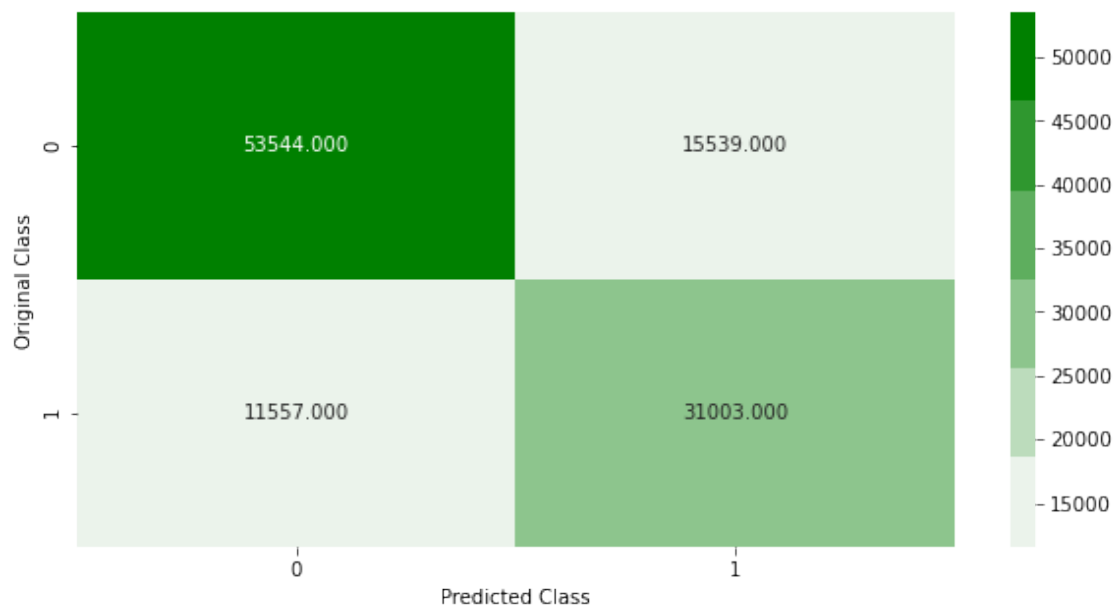
best train threshold : 0.4818554096188539
The Weighted Recall Score: 0.7572978153578818
The Weighted Precision Score: 0.7628742606101553
The Weighted F1 Score: 0.7591198638616238

=====

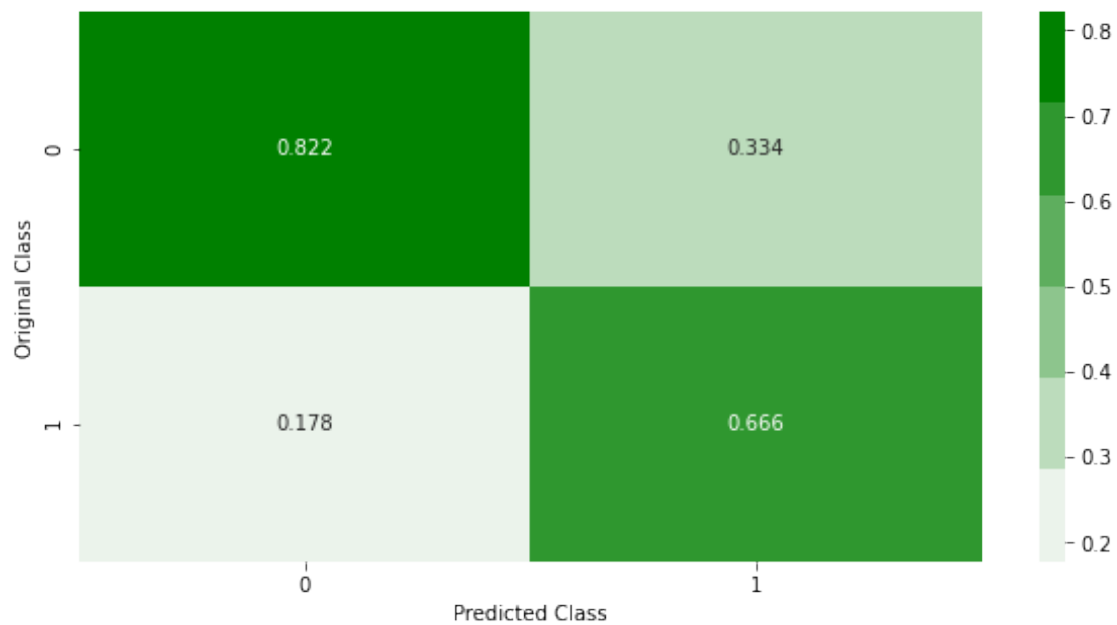
=====

=====

----- Confusion matrix

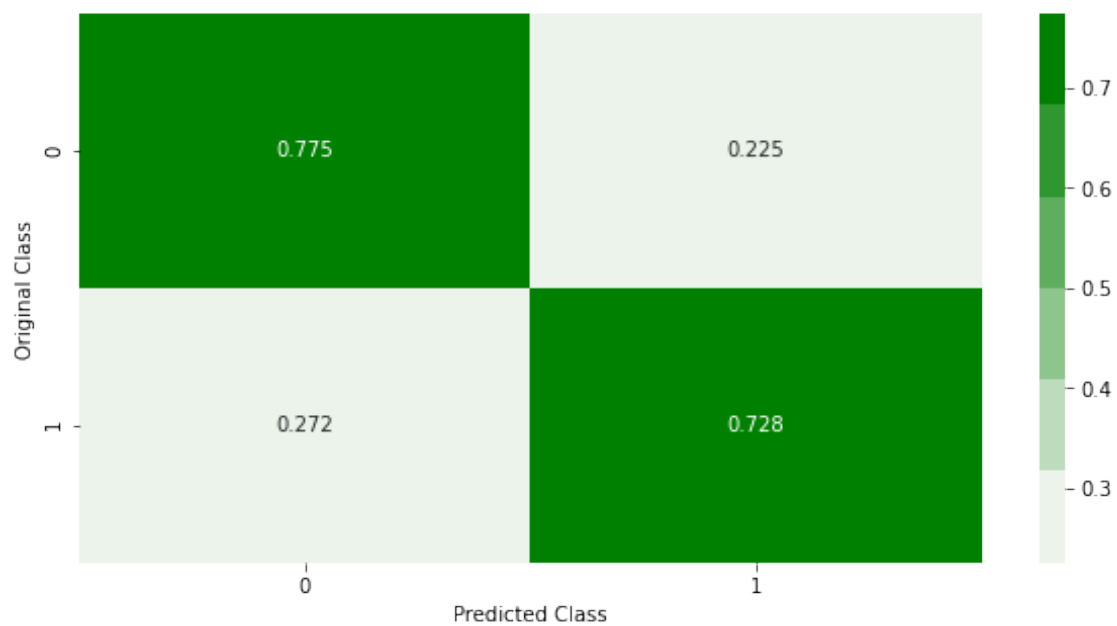


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.5 XGBoost after dropping the features with high VIF Values

10.5.1 Running XGBoost on the final dataset post dropping the variables with large VIF and dummy variable trap

```
[ ]: pip install xgboost
```

Requirement already satisfied: xgboost in /opt/conda/lib/python3.7/site-packages (1.4.2)

Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from xgboost) (1.7.0)

Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from xgboost) (1.19.5)

Note: you may need to restart the kernel to use updated packages.

```
[ ]: from xgboost import XGBClassifier
xgb_tuned = XGBClassifier(subsample=1,n_estimators= 100,max_depth=
    ↳3,colsample_bytree=1,learning_rate= 0.1)
xgb_tuned.fit(train_fin4,train_y)

train_y_pred = xgb_tuned.predict_proba(train_fin4)
train_y_pred = train_y_pred[:,1]
cv_y_pred = xgb_tuned.predict_proba(cv_fin4)
cv_y_pred = cv_y_pred[:,1]
```

[15:12:56] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

10.5.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)
```

Confusion Matrix for the Train Data

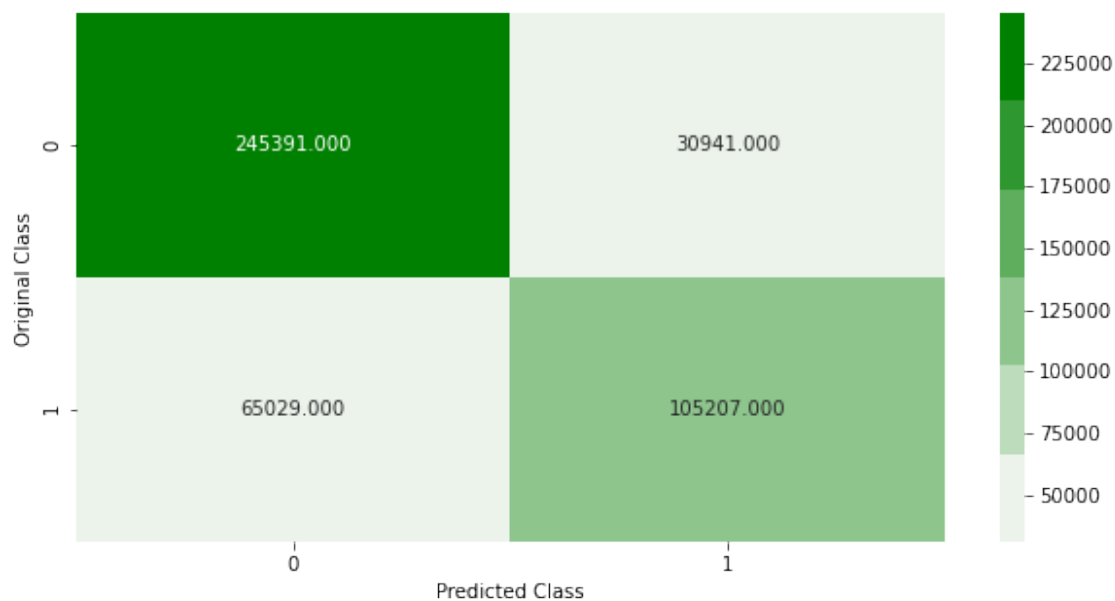
The Weighted Recall Score: 0.7850943193421831

The Weighted Precision Score: 0.7837376560479234

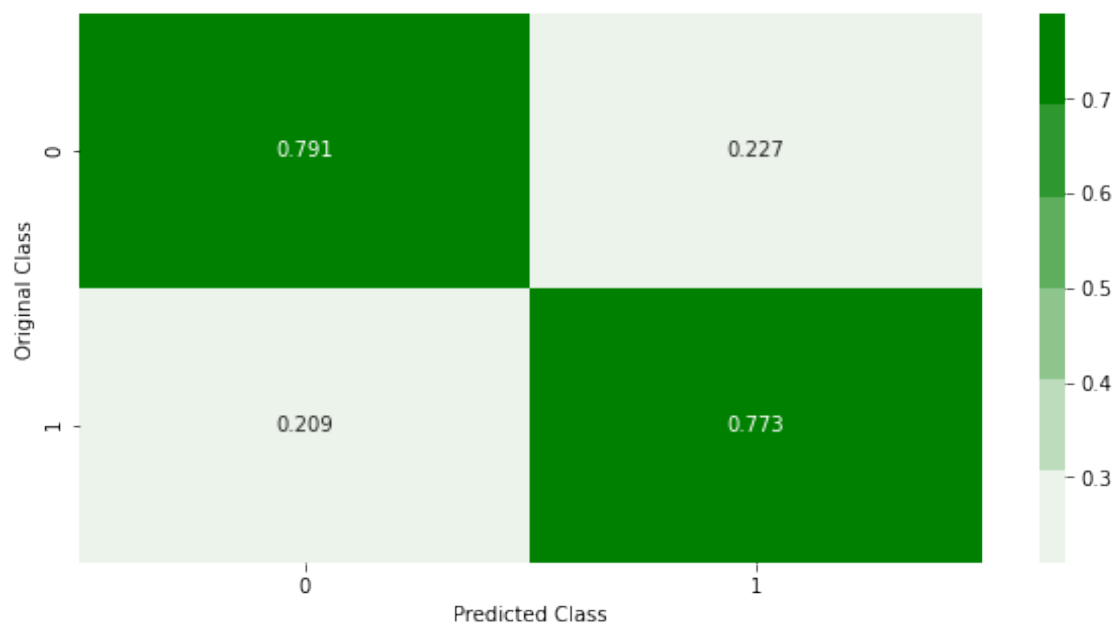
The Weighted F1 Score: 0.7793818063055133

```
=====
=====
=====
=====
```

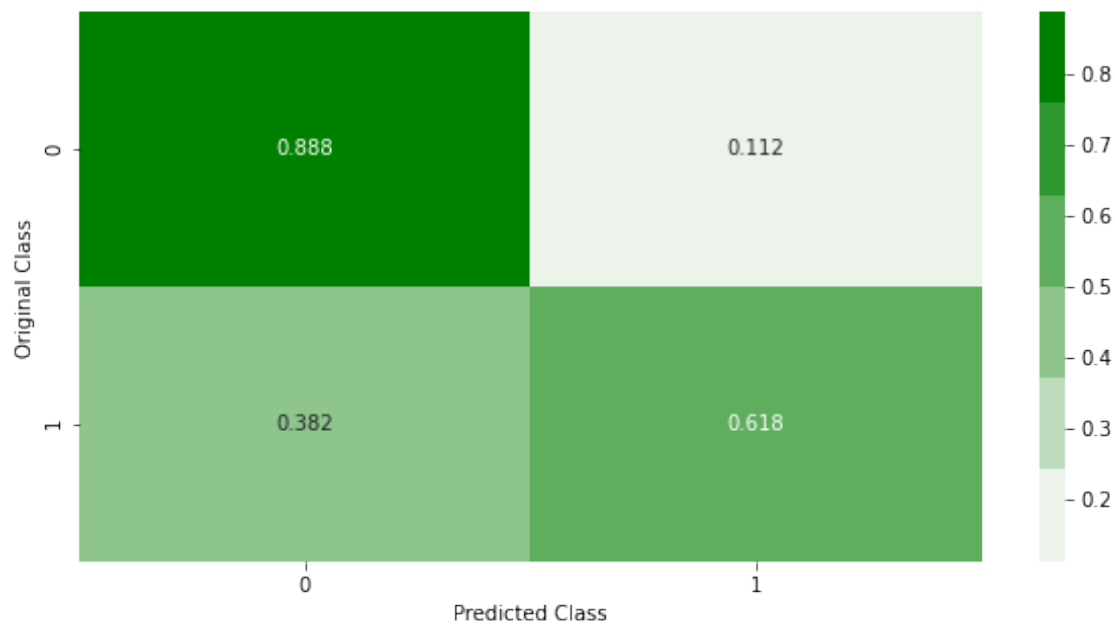
----- Confusion matrix



----- Precision matrix

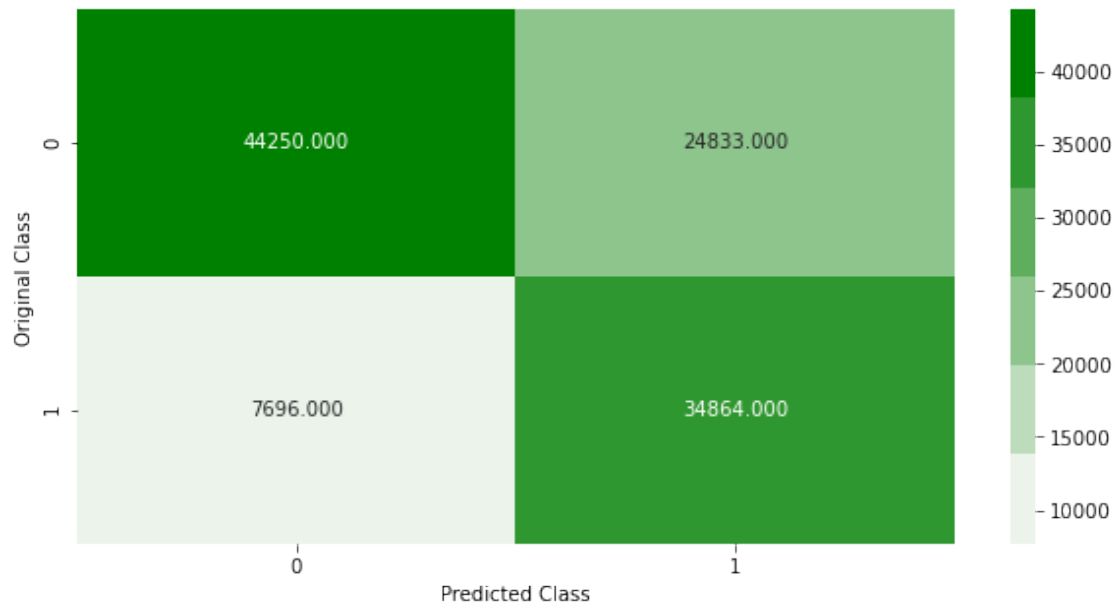


Sum of columns in precision matrix [1. 1.]
----- Recall matrix

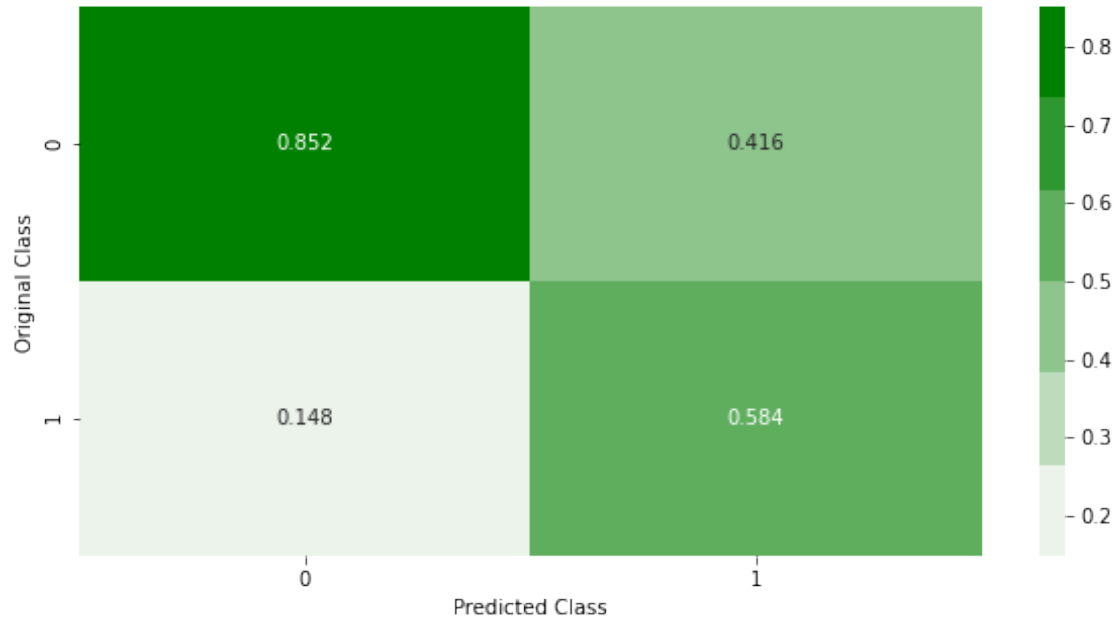


Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.7086337701423286
The Weighted Precision Score: 0.7497452249562168
The Weighted F1 Score: 0.712420567529374
=====

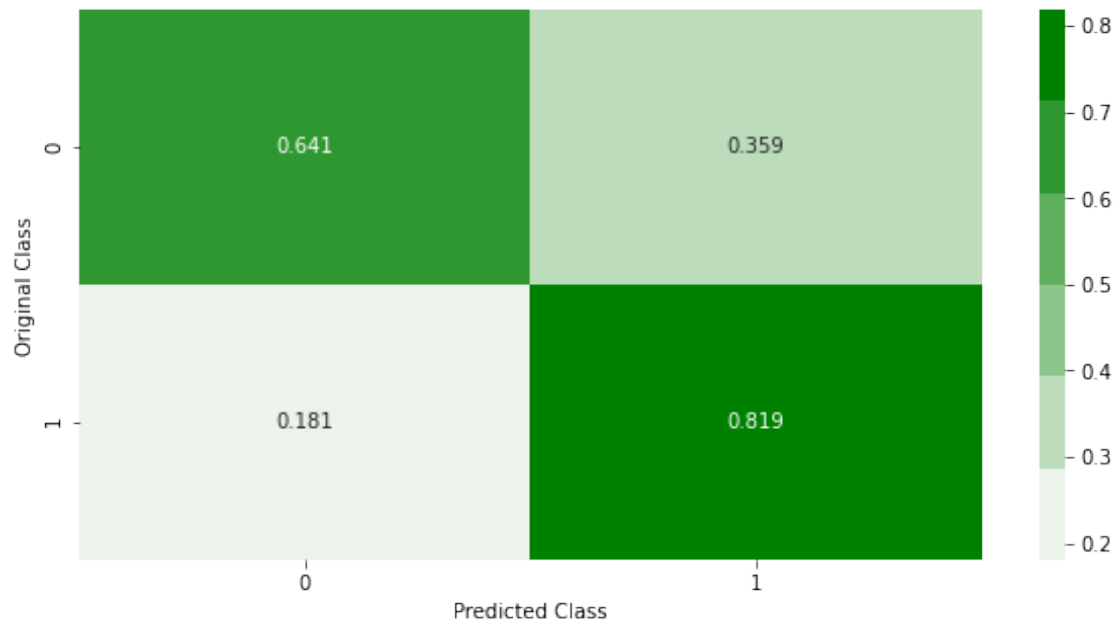
----- Confusion matrix



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.5.3 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

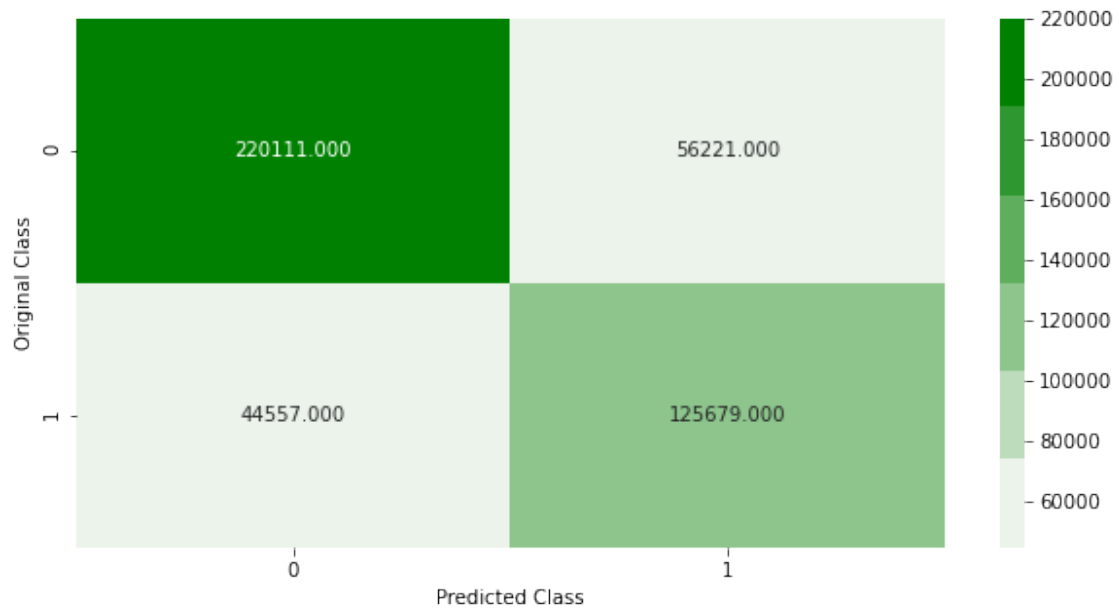
best train threshold : 0.43001717

The Weighted Recall Score: 0.7743277619533867

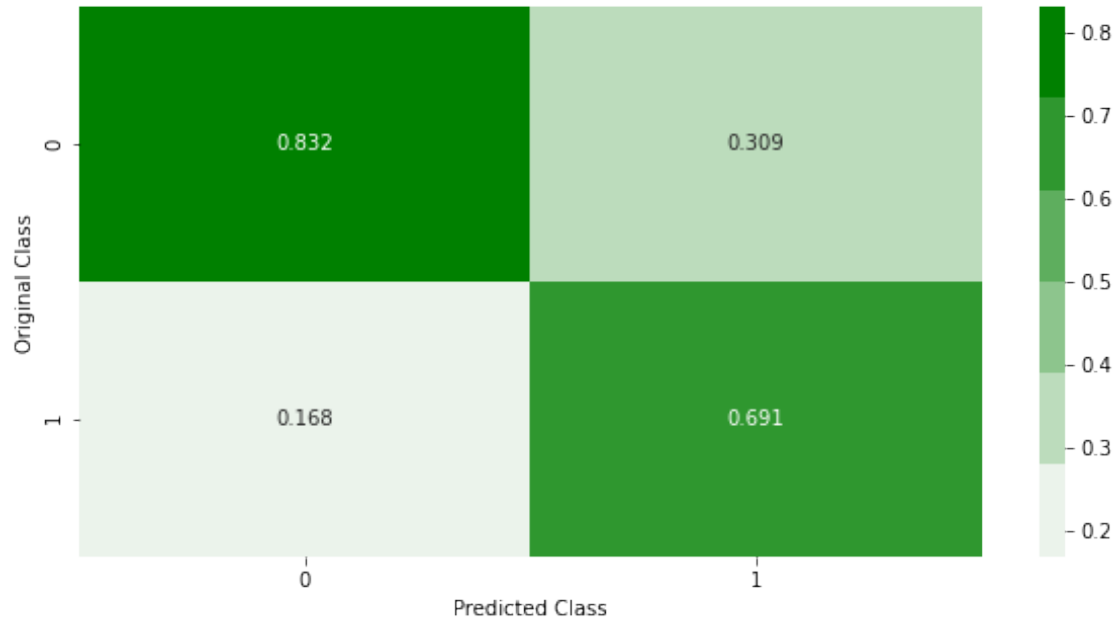
The Weighted Precision Score: 0.7780034096596982

The Weighted F1 Score: 0.7756325415082669

```
=====
=====
=====
----- Confusion matrix
-----
```

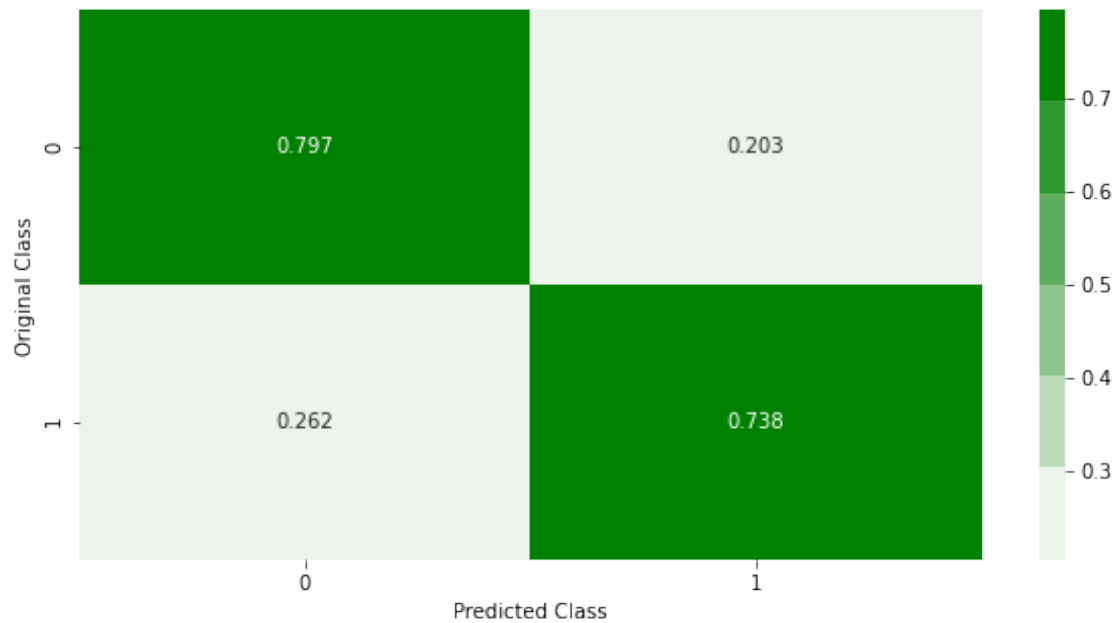


----- Precision matrix



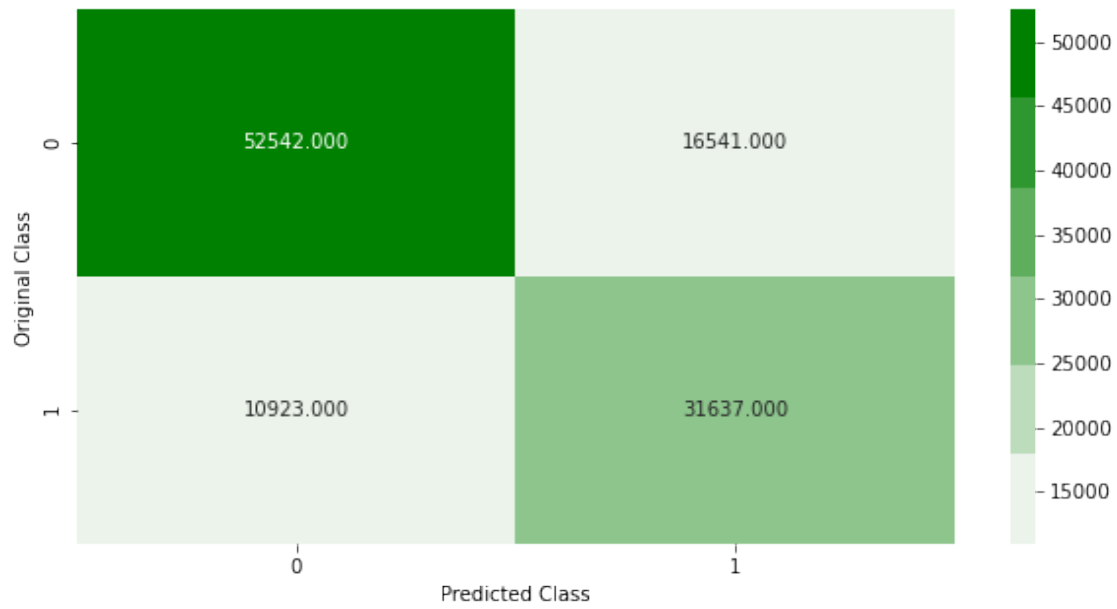
Sum of columns in precision matrix [1. 1.]

----- Recall matrix

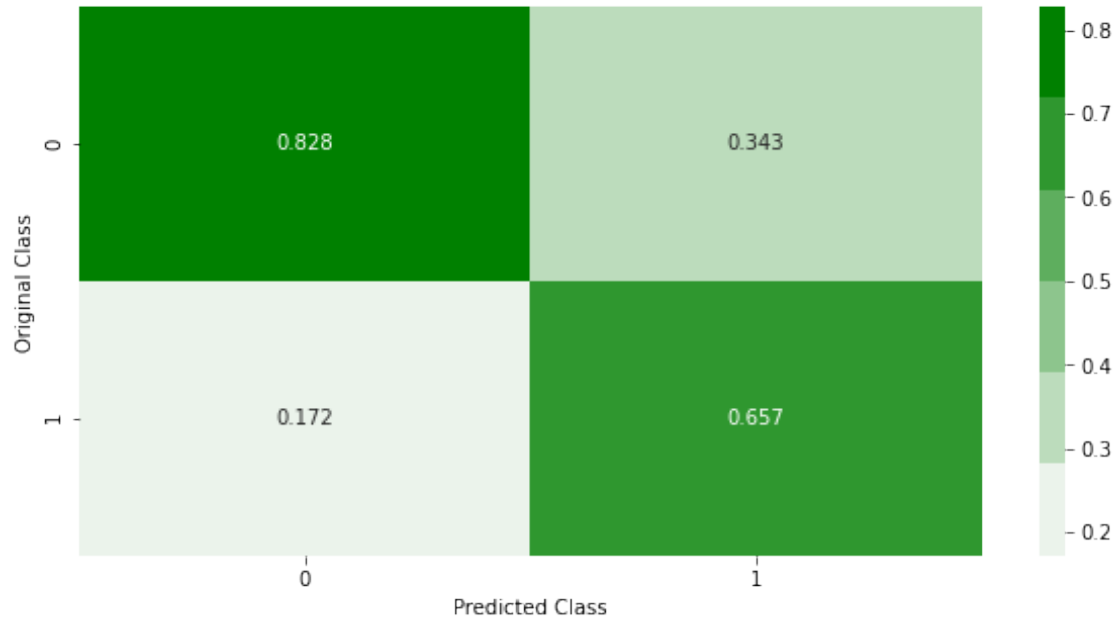


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.5688667
The Weighted Recall Score:  0.7540015943677615
The Weighted Precision Score:  0.7626175938652925
The Weighted F1 Score:  0.7564037537399892
```

```
=====
=====
=====
----- Confusion matrix
-----
```

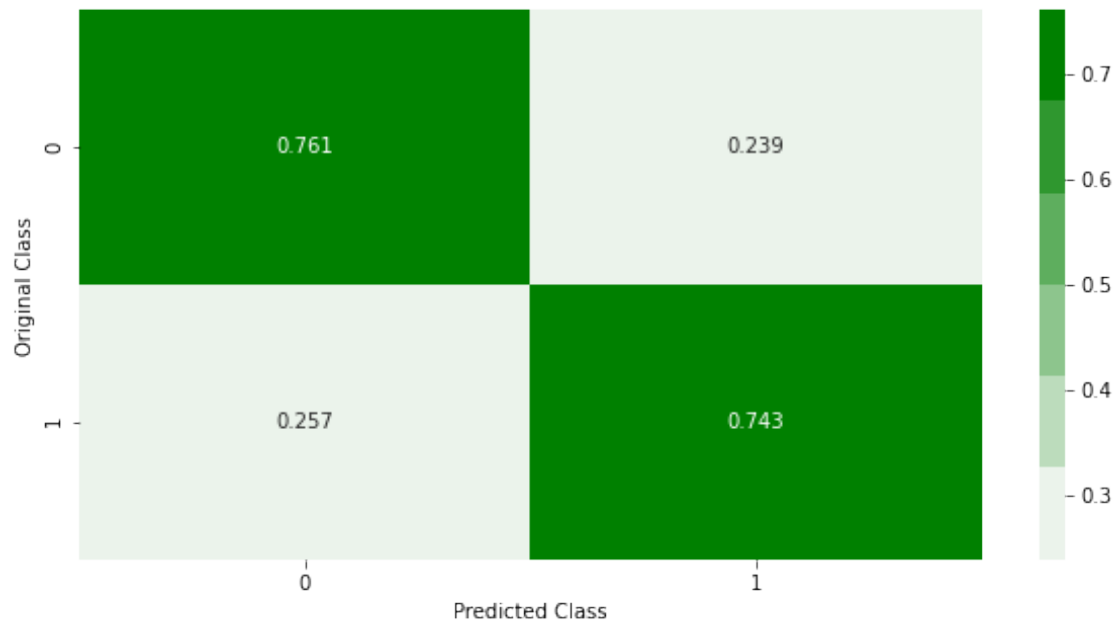


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.6 lightGBM Classifier after dropping the features with high VIF Values

```
[ ]: import lightgbm as lgb
      clf= lgb.LGBMClassifier()
      param_grid ={'num_leaves': [300,400,500,1000], 'reg_alpha': [0.01,1,10]}

      lgb_tune = RandomizedSearchCV(clf,param_grid,cv=5,scoring='f1')
      lgb_tune.fit(train_fin32, train_y)

      print('best parameter :',lgb_tune.best_params_)
```

```
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.6s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.4s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.4s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 21.1s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 21.8s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 21.2s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 20.9s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 28.0s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 45.2s
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 45.5s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 41.0s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 40.2s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 40.5s
```

```

[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 41.2s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 40.8s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 41.9s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 41.1s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 40.9s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 44.3s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 41.5s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 42.4s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 42.1s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 41.7s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 40.2s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 37.5s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.6s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.4s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 20.9s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 21.1s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 22.4s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 20.9s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 22.3s
[CV] END ...class_weight={0: 0.33, 1: 0.66999999999999999}; total time= 41.7s
[CV] END ...class_weight={0: 0.33, 1: 0.66999999999999999}; total time= 41.1s
[CV] END ...class_weight={0: 0.33, 1: 0.66999999999999999}; total time= 40.6s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 40.8s
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 40.5s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 41.7s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 40.8s
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 40.4s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 41.0s
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 40.6s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 41.5s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 41.1s
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 41.3s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 42.3s
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 42.2s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 40.0s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 39.8s
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 33.4s
best parameter : {'reg_alpha': 0.01, 'num_leaves': 1000}

```

```

[ ]: lgb_best= lgb.LGBMClassifier(reg_alpha= 0.01,num_leaves=1000)
lgb_best.fit(train_fin32,train_y)

```

```

[ ]: LGBMClassifier(num_leaves=1000, reg_alpha=0.01)

```

```

[ ]: train_y_pred= lgb_best.predict_proba(train_fin4)
train_y_pred= train_y_pred[:,1]

cv_y_pred= lgb_best.predict_proba(cv_fin4)

```

```
cv_y_pred= cv_y_pred[:,1]
```

10.6.1 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)
```

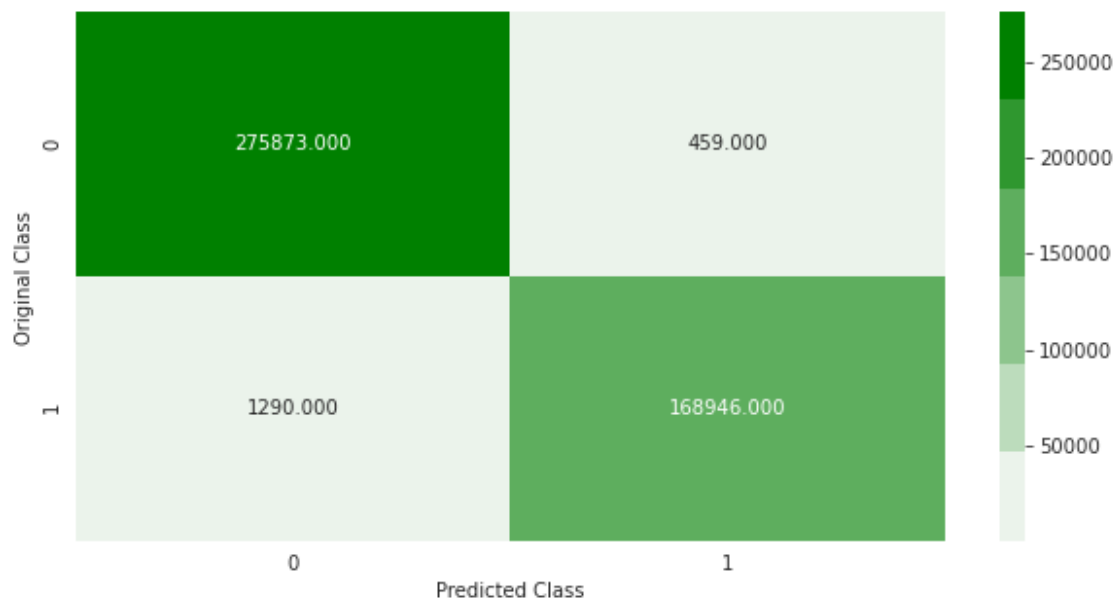
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.9960834632127694

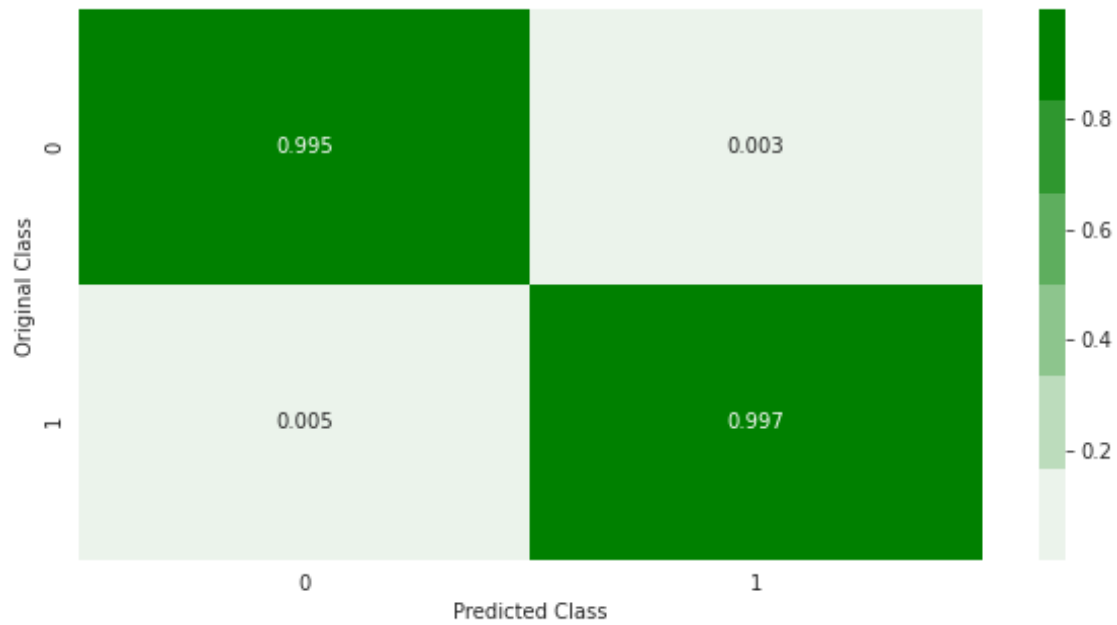
The Weighted Precision Score: 0.9960870822439472

The Weighted F1 Score: 0.9960816119980144

```
=====
=====
----- Confusion matrix
-----
```

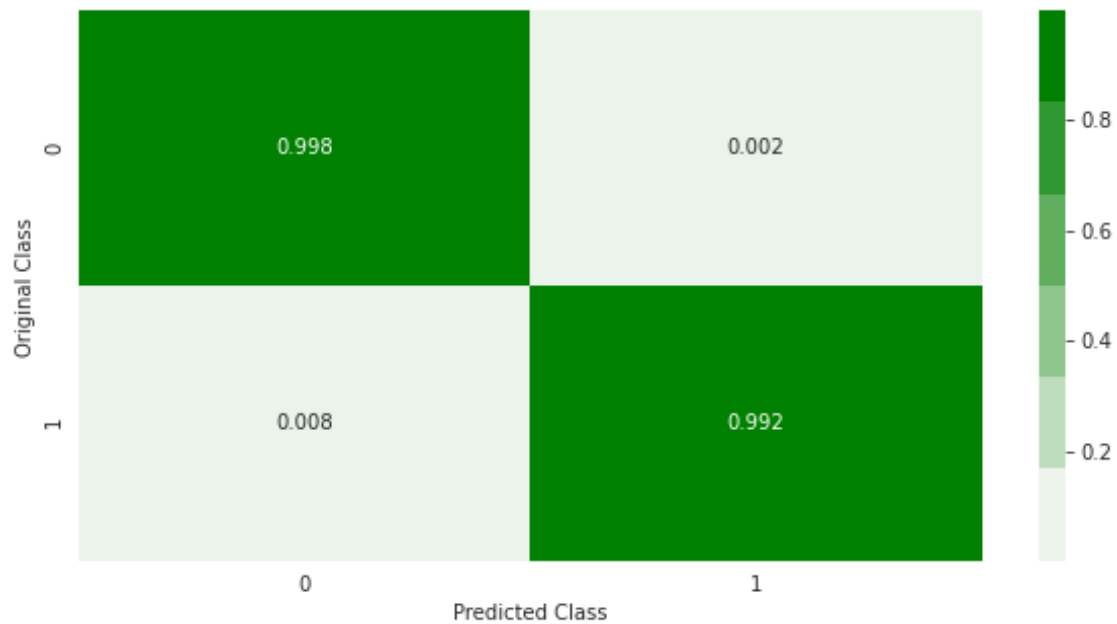


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

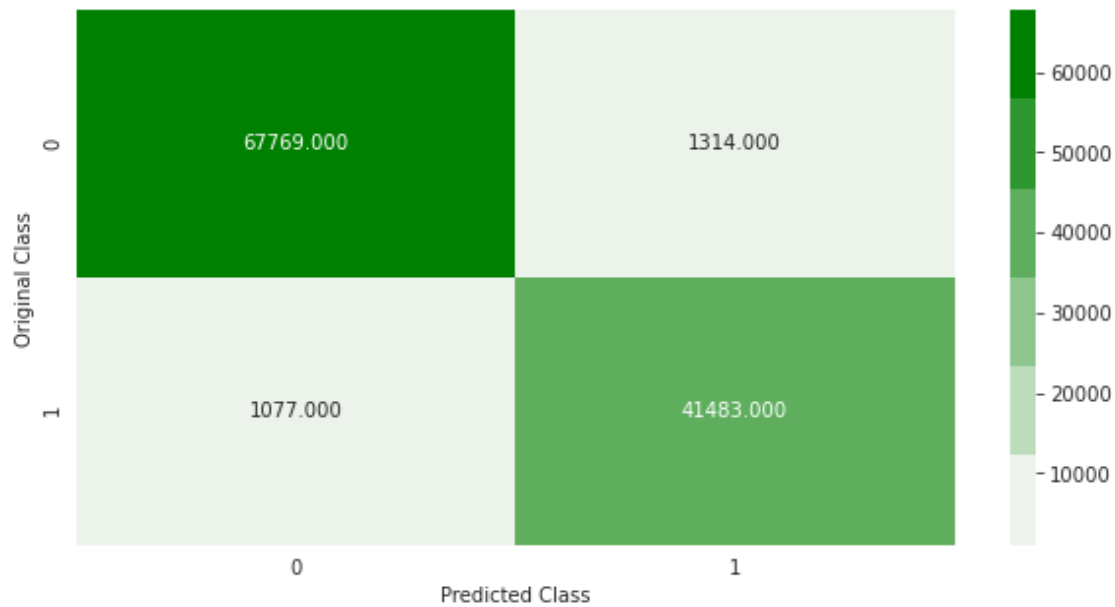
The Weighted Recall Score: 0.9785835206864738
The Weighted Precision Score: 0.9786154895146161
The Weighted F1 Score: 0.9785948531987279

=====

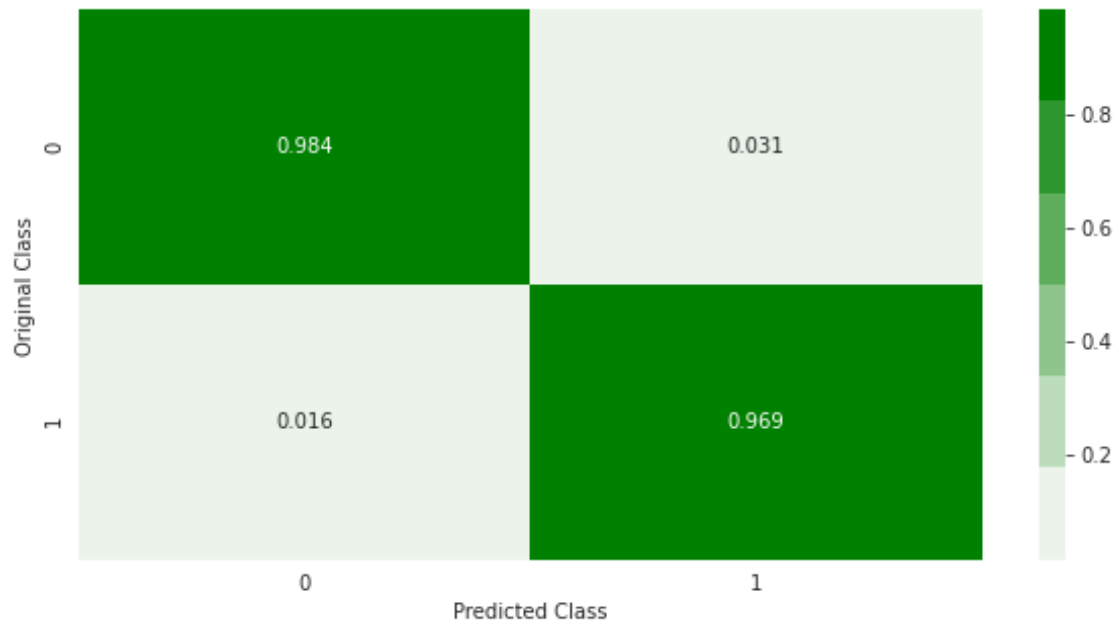
=====

=====

----- Confusion matrix

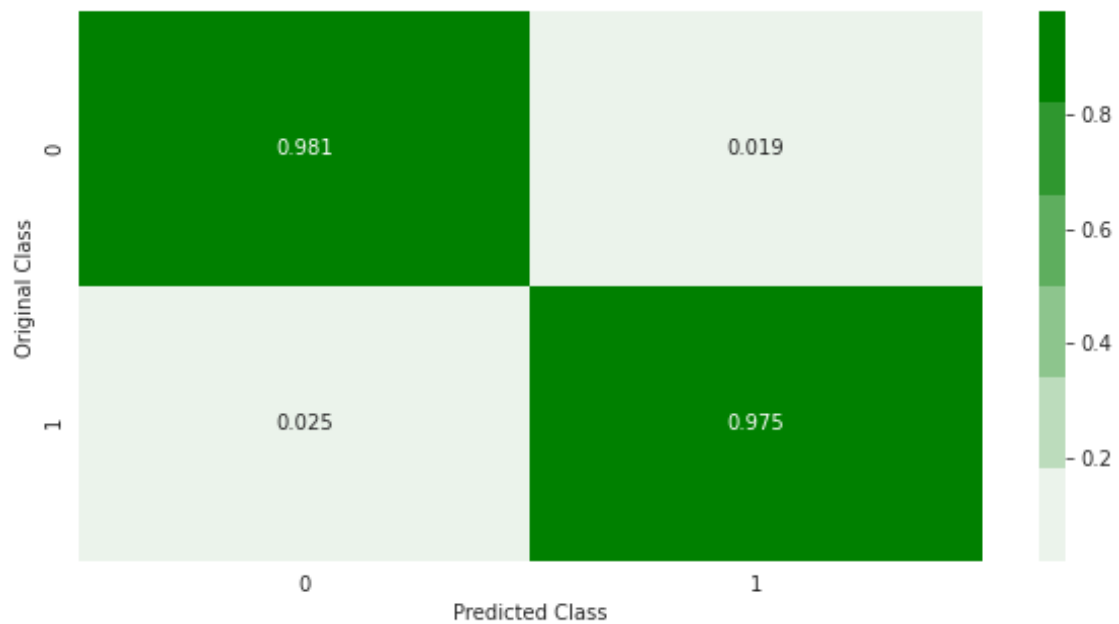


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.6.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

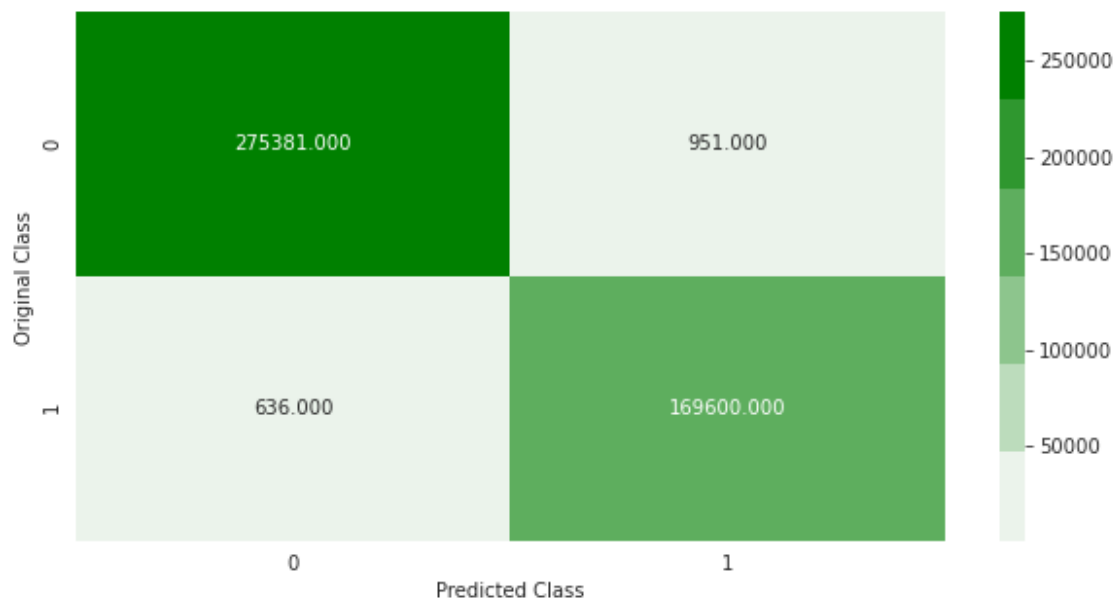
best train threshold : 0.4051453756255439

The Weighted Recall Score: 0.9964462299134734

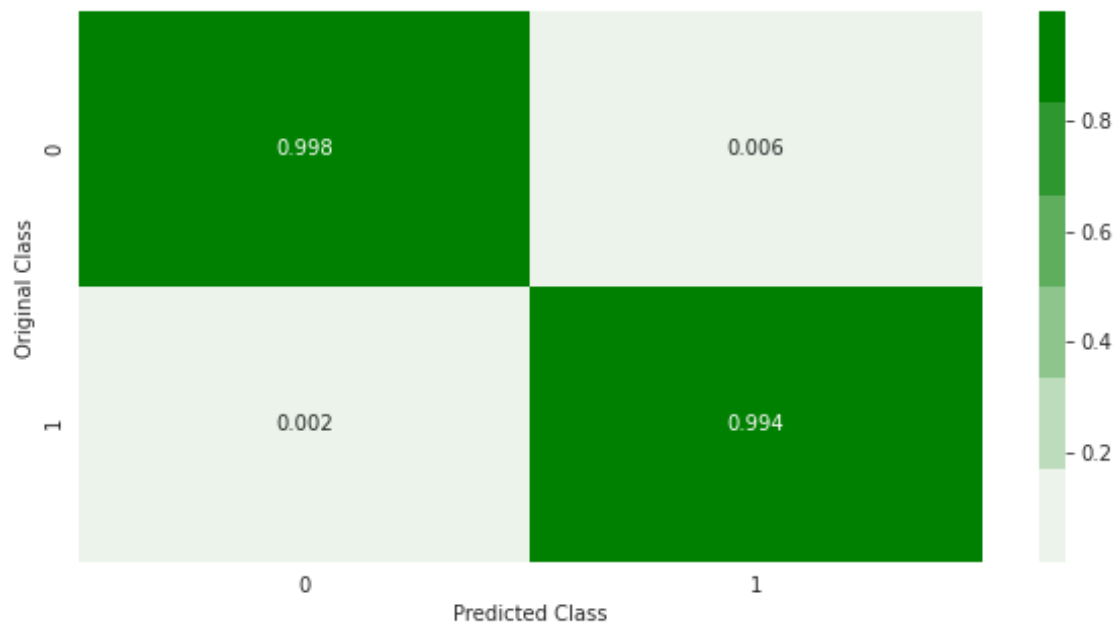
The Weighted Precision Score: 0.9964485378021324

The Weighted F1 Score: 0.9964468590008319

```
=====
=====
=====
----- Confusion matrix
-----
```

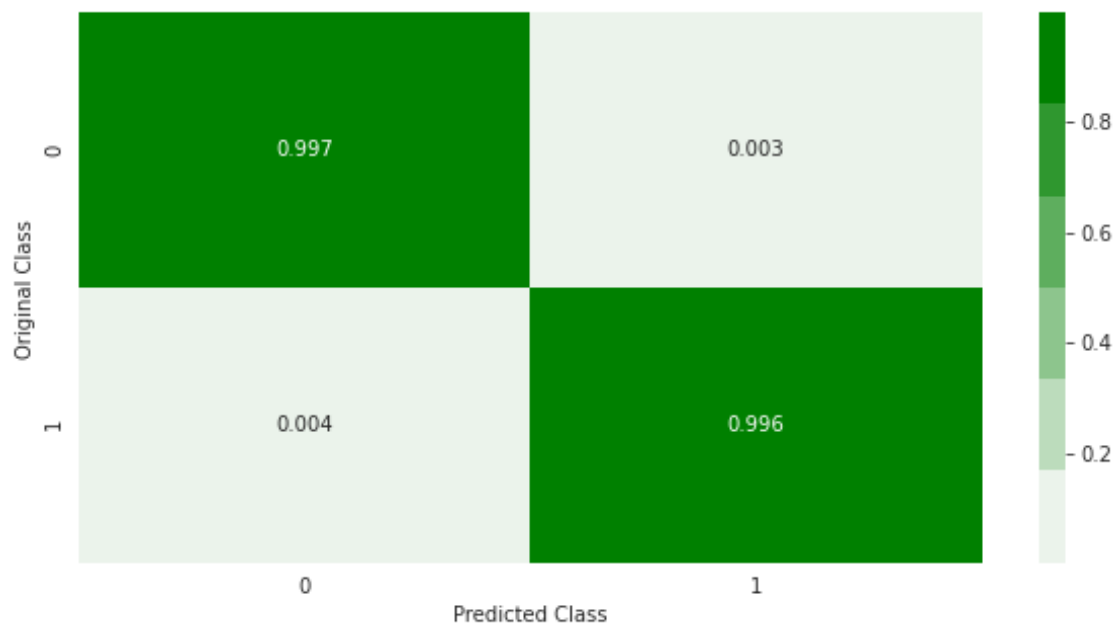


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Thresold for the CV Data

best train threshold : 0.5238812301828405
The Weighted Recall Score: 0.9791119908995638
The Weighted Precision Score: 0.9791135398213046
The Weighted F1 Score: 0.9791127441050557

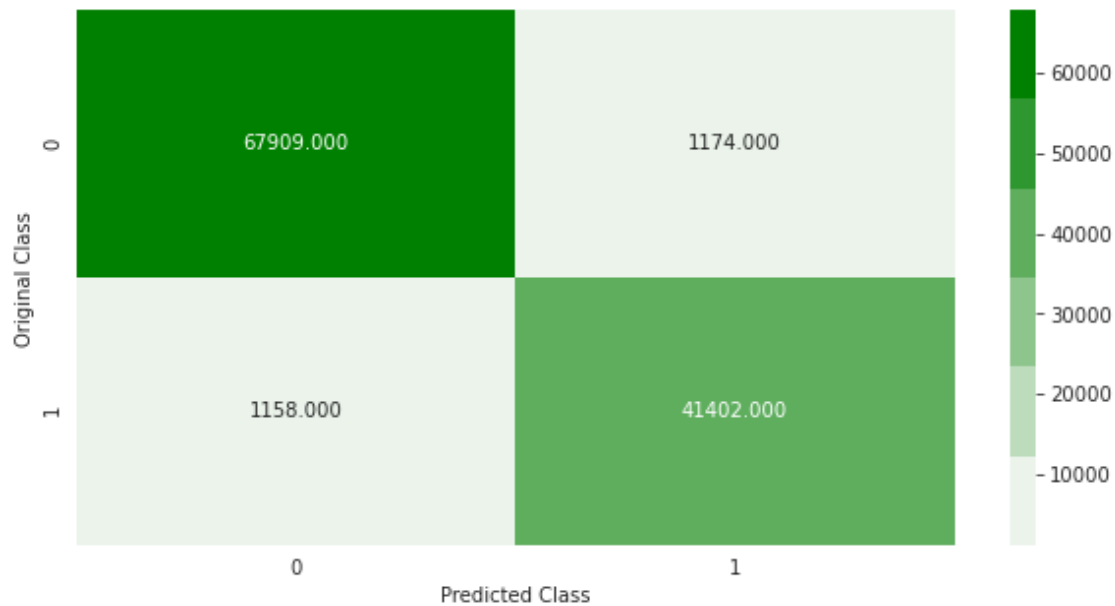
=====

=====

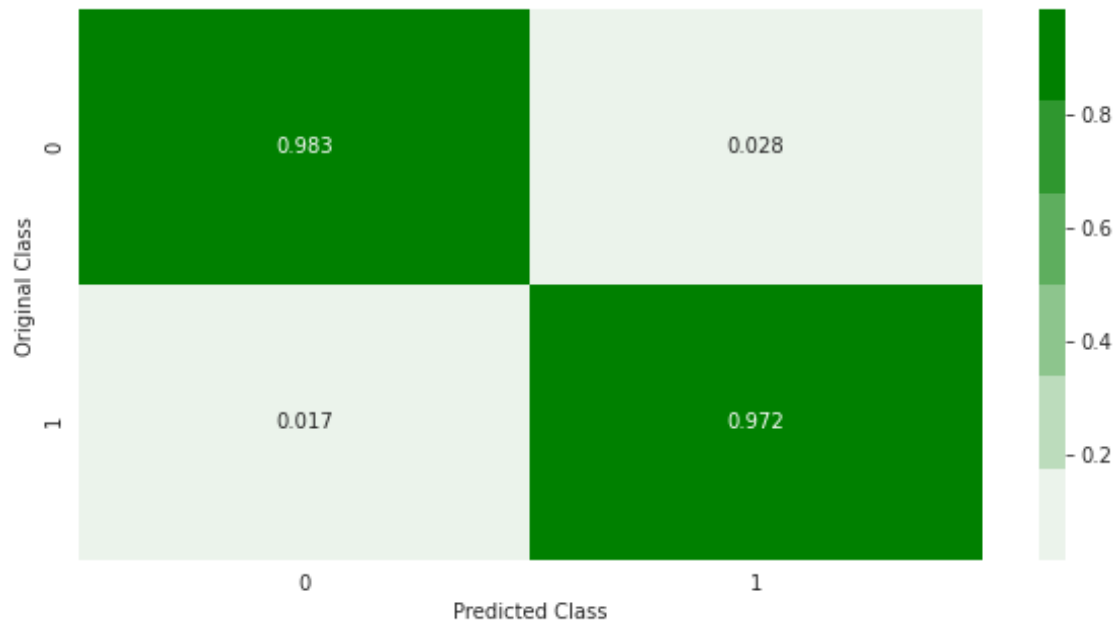
=====

=====

----- Confusion matrix

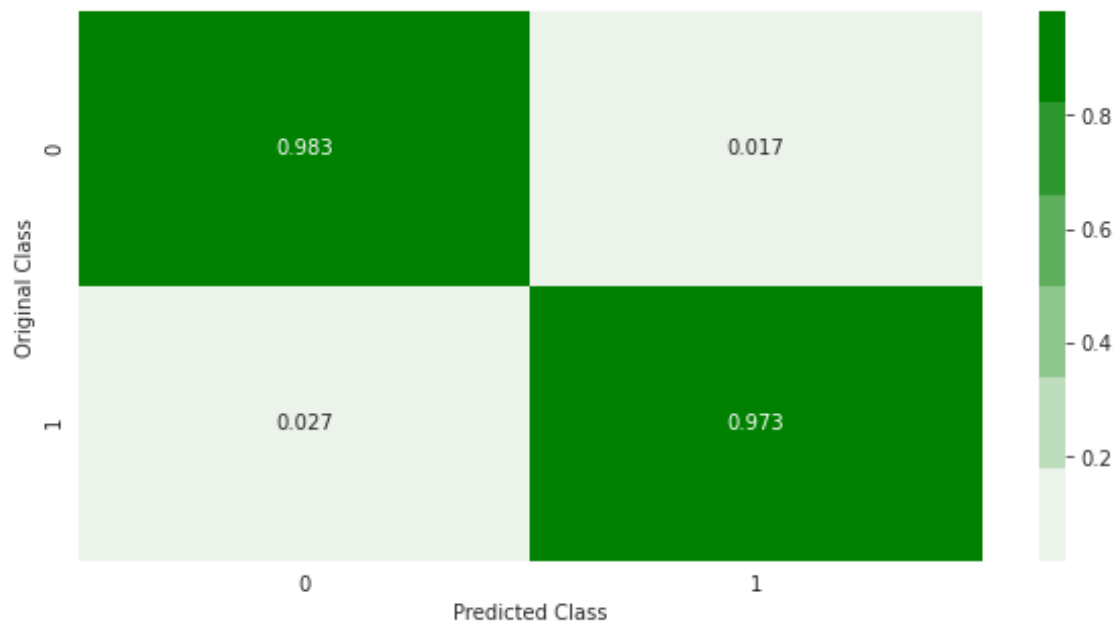


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]:
```

10.7 lightGBM Classifier after dropping the features with high VIF Values and after adjusting for Class Weights

```
[ ]: import lightgbm as lgb
from sklearn.model_selection import GridSearchCV, StratifiedKFold

lgb_cw= lgb.LGBMClassifier(reg_alpha= 0.01,num_leaves=1000)

weights = np.linspace(0.0,0.99,10)

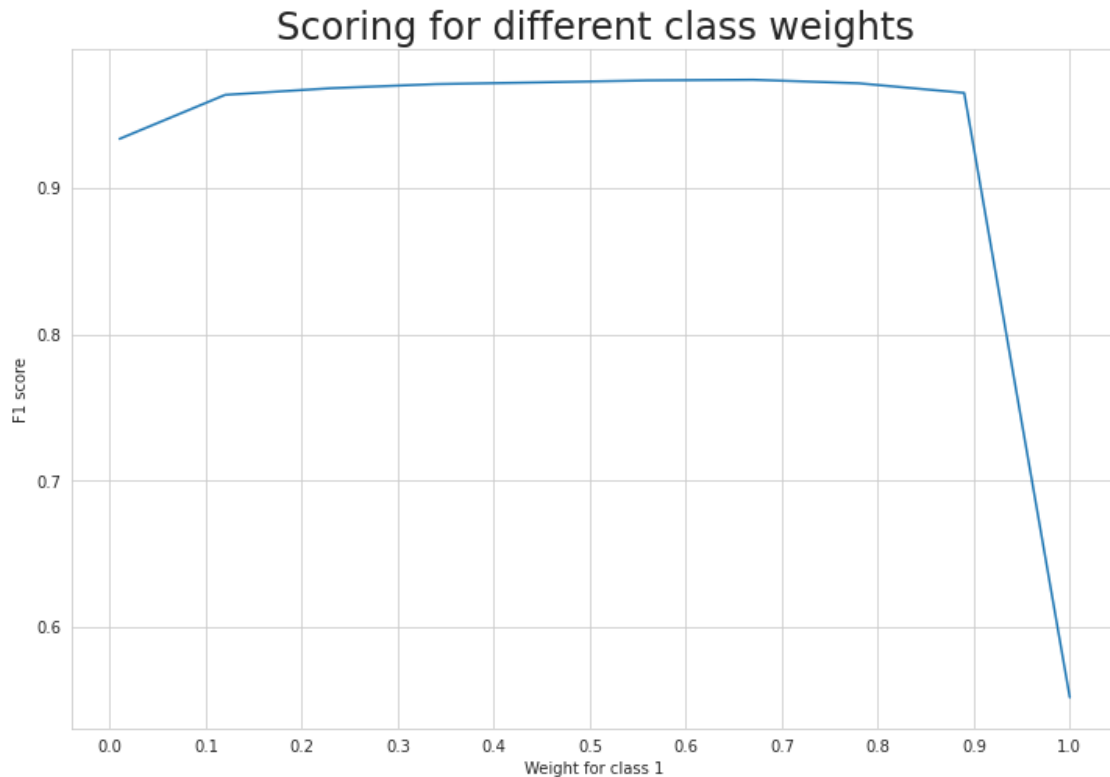
param_grid = {'class_weight': [{0:x, 1:1.0-x} for x in weights]}

gridsearch = GridSearchCV(estimator= lgb_cw,
                           param_grid= param_grid,
                           cv=StratifiedKFold(),
                           n_jobs=-1,
                           scoring='f1',
                           verbose=2).fit(train_fin32, train_y)

sns.set_style('whitegrid')
plt.figure(figsize=(12,8))
weigh_data = pd.DataFrame({'score': gridsearch.cv_results_['mean_test_score'],
                           'weight': (1- weights)})
sns.lineplot(weigh_data['weight'], weigh_data['score'])
plt.xlabel('Weight for class 1')
plt.ylabel('F1 score')
plt.xticks([round(i/10,1) for i in range(0,11,1)])
plt.title('Scoring for different class weights', fontsize=24)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
[ ]: Text(0.5, 1.0, 'Scoring for different class weights')
```



```
[ ]: print(gridsearch.best_params_)
```

```
{'class_weight': {0: 0.33, 1: 0.6699999999999999}}
```

```
[ ]: lgb_cw = lgb.LGBMClassifier(reg_alpha= 0.01,num_leaves=1000,class_weight={0:0.
    ↳33, 1: 0.66999})
lgb_cw.fit(train_fin32, train_y)
```

```
[ ]: LGBMClassifier(class_weight={0: 0.33, 1: 0.66999}, num_leaves=1000,
    reg_alpha=0.01)
```

```
[ ]: train_y_pred_cw32= lgb_cw.predict_proba(train_fin32)
train_y_pred_cw32= train_y_pred_cw32[:,1]

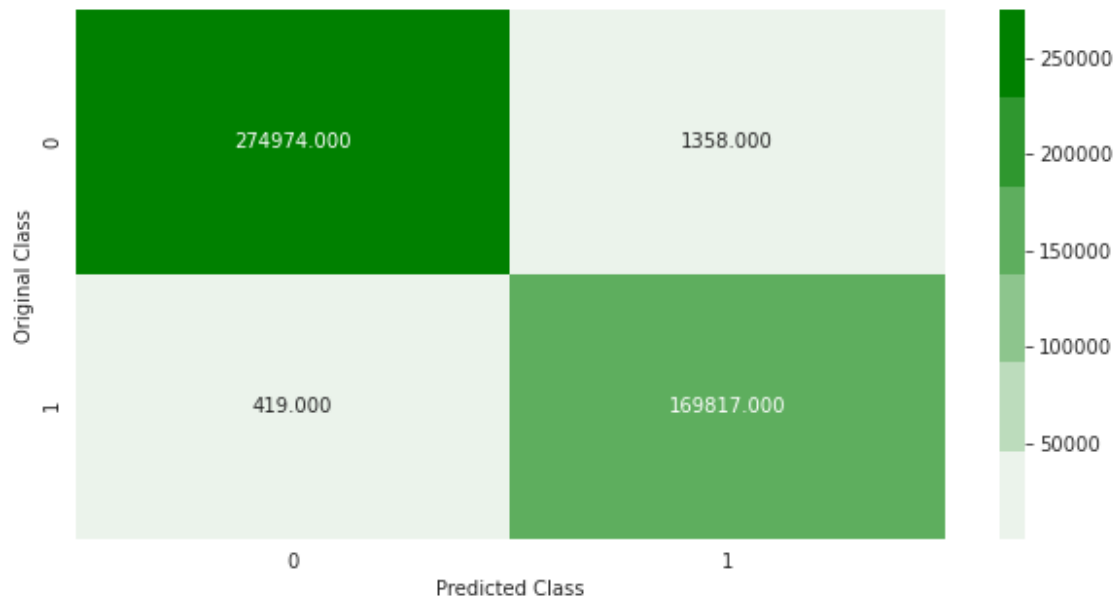
cv_y_pred_cw32= lgb_cw.predict_proba(cv_fin32)
cv_y_pred_cw32= cv_y_pred_cw32[:,1]
```

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y, train_y_pred_cw32)

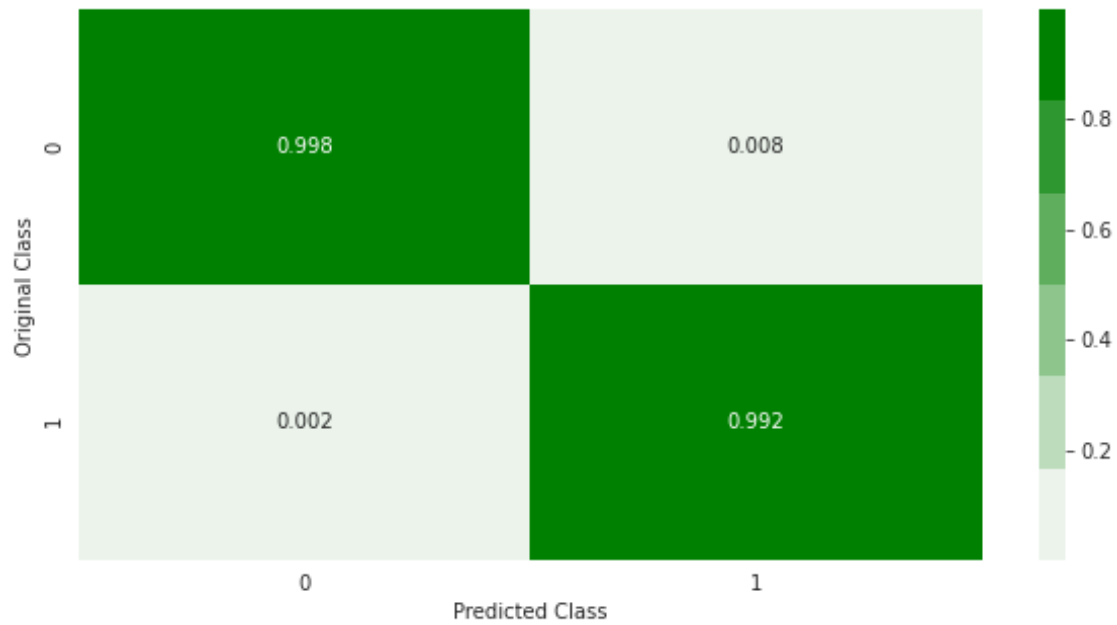
print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred_cw32)
```


Confusion Matrix for the Train Data
The Weighted Recall Score: 0.9960207627953638
The Weighted Precision Score: 0.9960342452014659
The Weighted F1 Score: 0.9960228487431603

=====
=====
=====
----- Confusion matrix

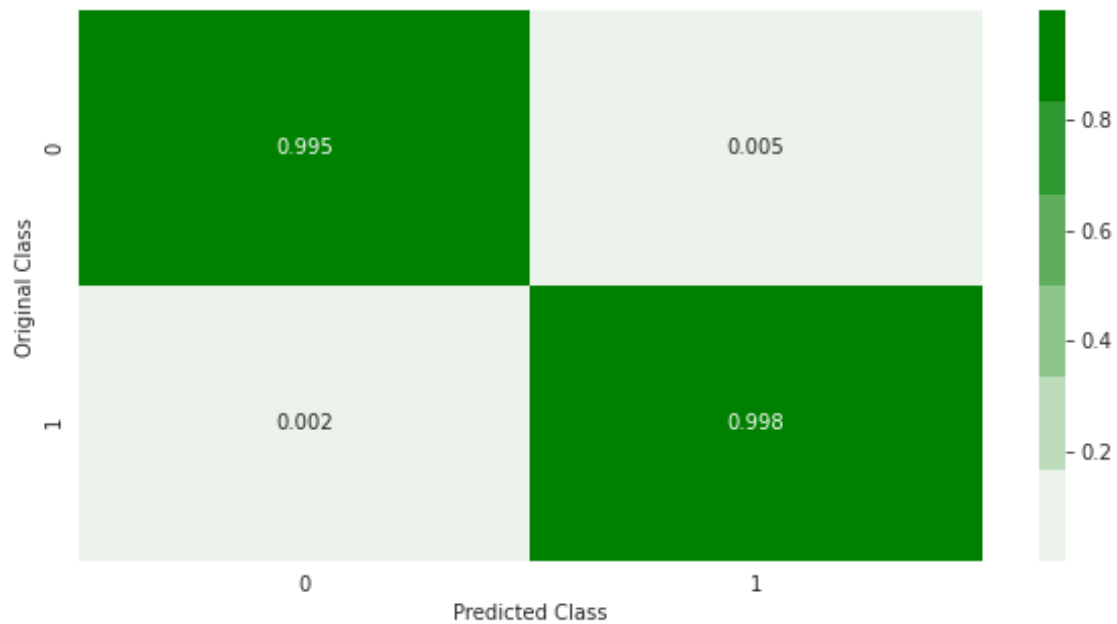


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

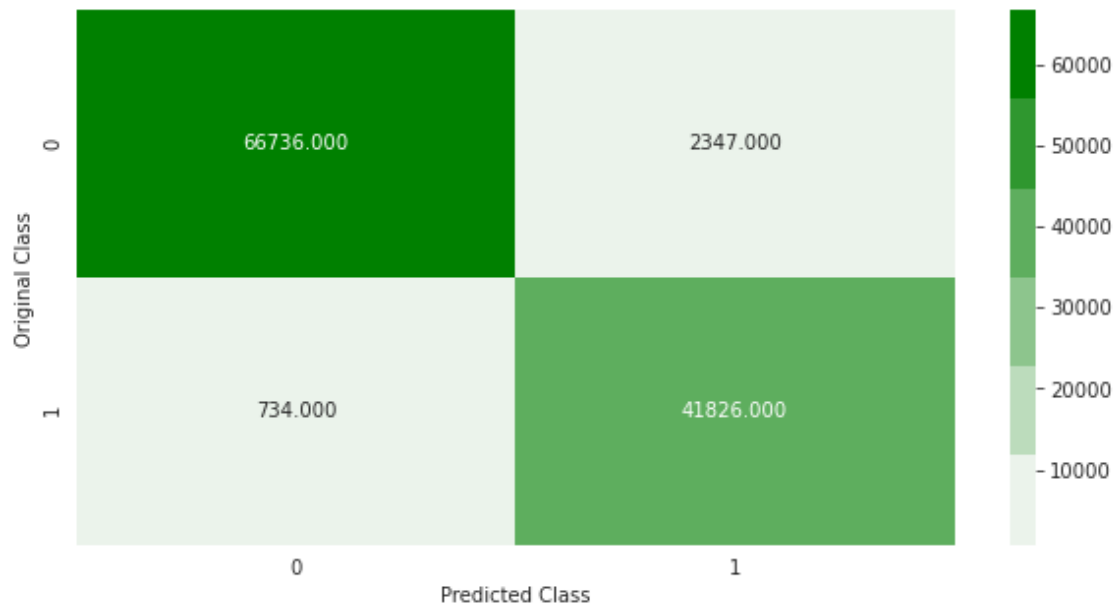
The Weighted Recall Score: 0.9724031063299983
The Weighted Precision Score: 0.9730135722037025
The Weighted F1 Score: 0.9724967293575186

=====

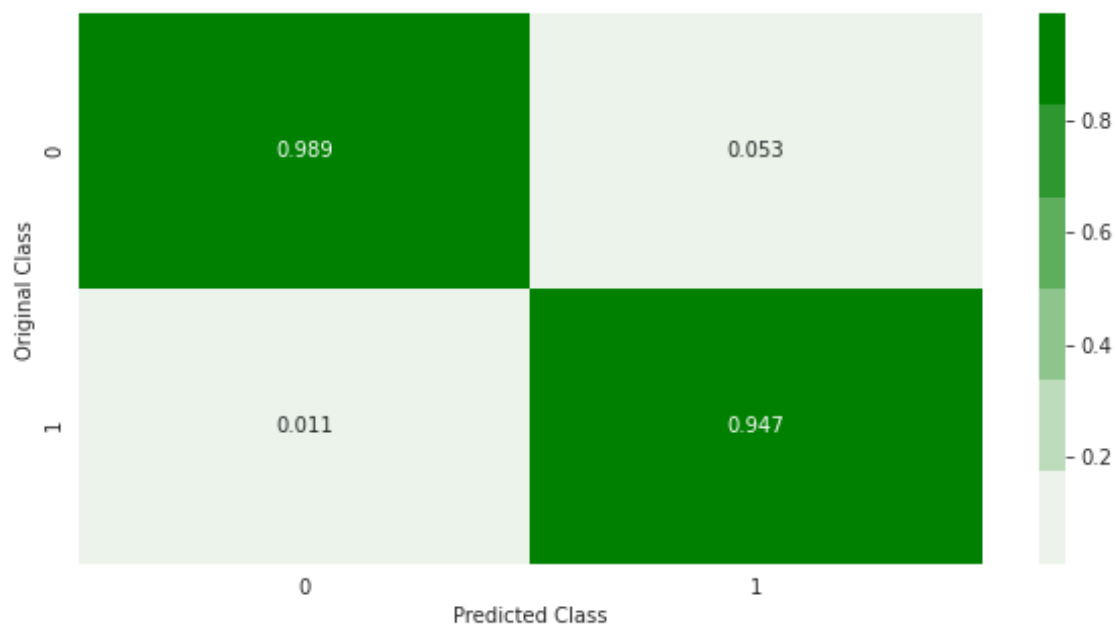
=====

=====

----- Confusion matrix

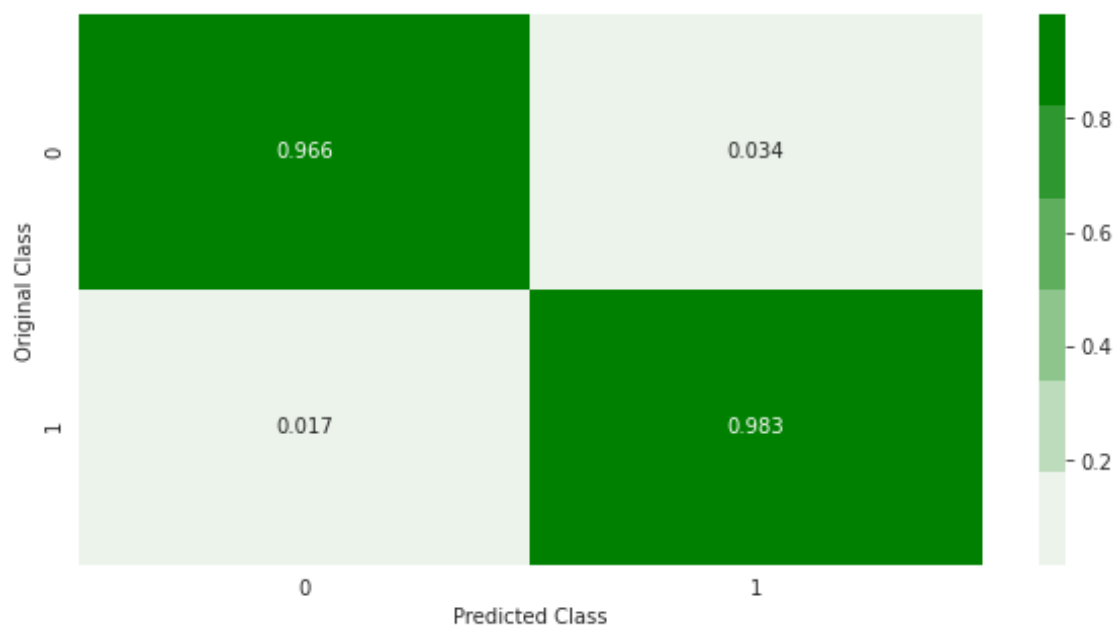


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.6s

```

[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.7s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 34.9s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 33.9s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 34.7s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 1.4min
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 1.2min
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 1.3min
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 1.3min
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 1.3min
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 1.4min
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 1.2min
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 1.4min
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 1.2min
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 1.2min
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 1.2min
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 1.2min
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 1.2min
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 1.2min
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 1.4min
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 1.5min
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 1.3min
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 1.2min
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 1.2min

```

```

[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
best_thresholds(train_y,train_y_pred_cw32)

print("Confusion Matrix for Best Thresold for the CV Data")
best_thresholds(cv_y,cv_y_pred_cw32)

```

```

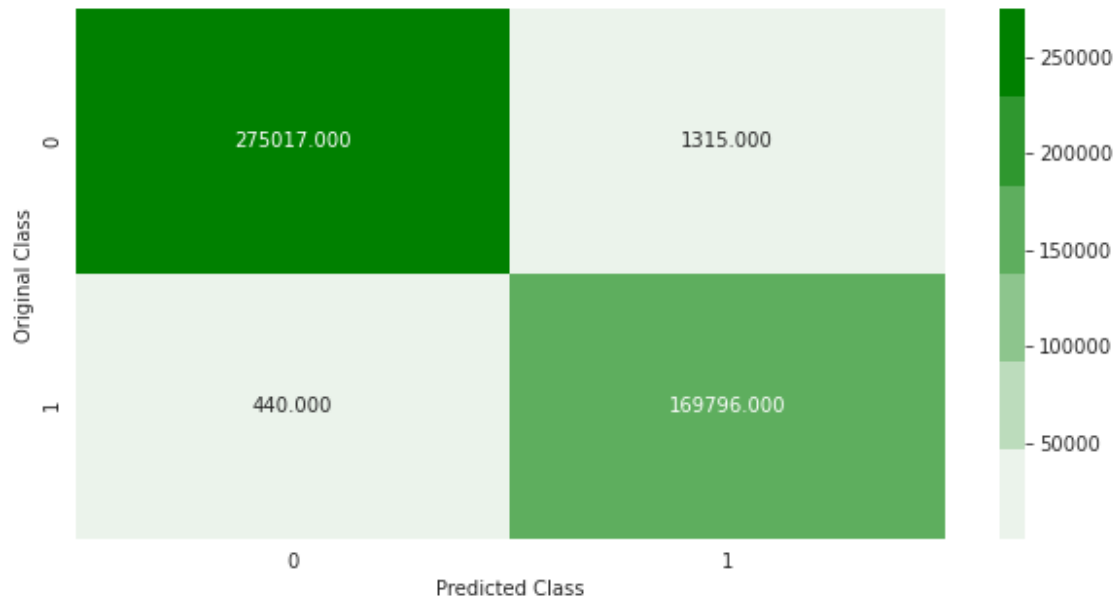
Confusion Matrix for Best Thresold for the Train Data
best train threshold : 0.505046076308342
The Weighted Recall Score: 0.9960700274090396
The Weighted Precision Score: 0.9960819556234891
The Weighted F1 Score: 0.9960719484245465

```

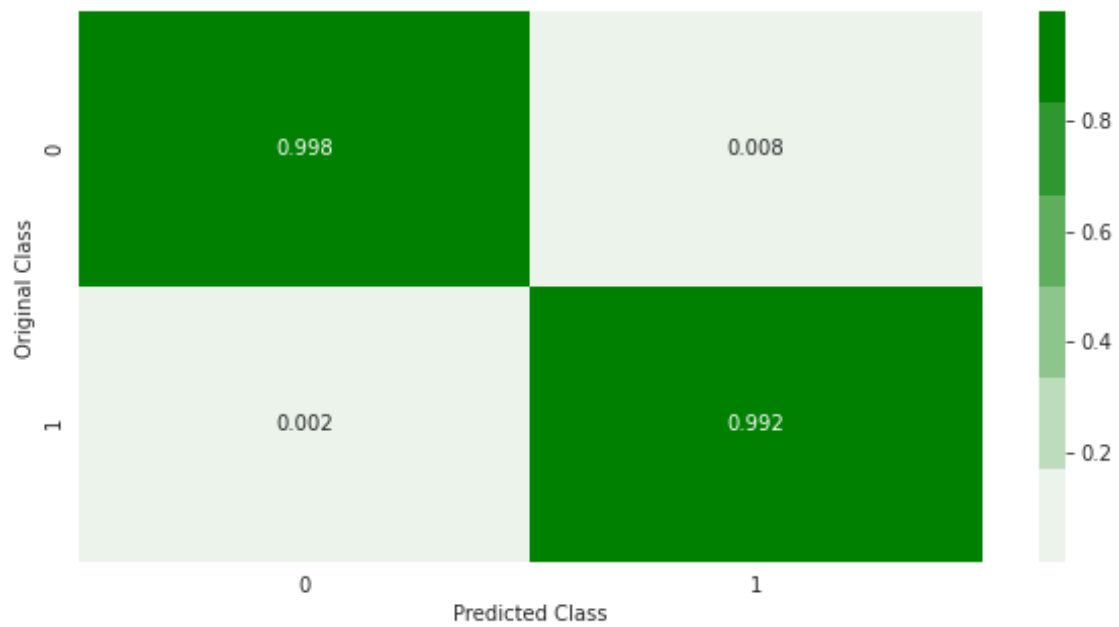
```

=====
=====
=====
----- Confusion matrix
-----

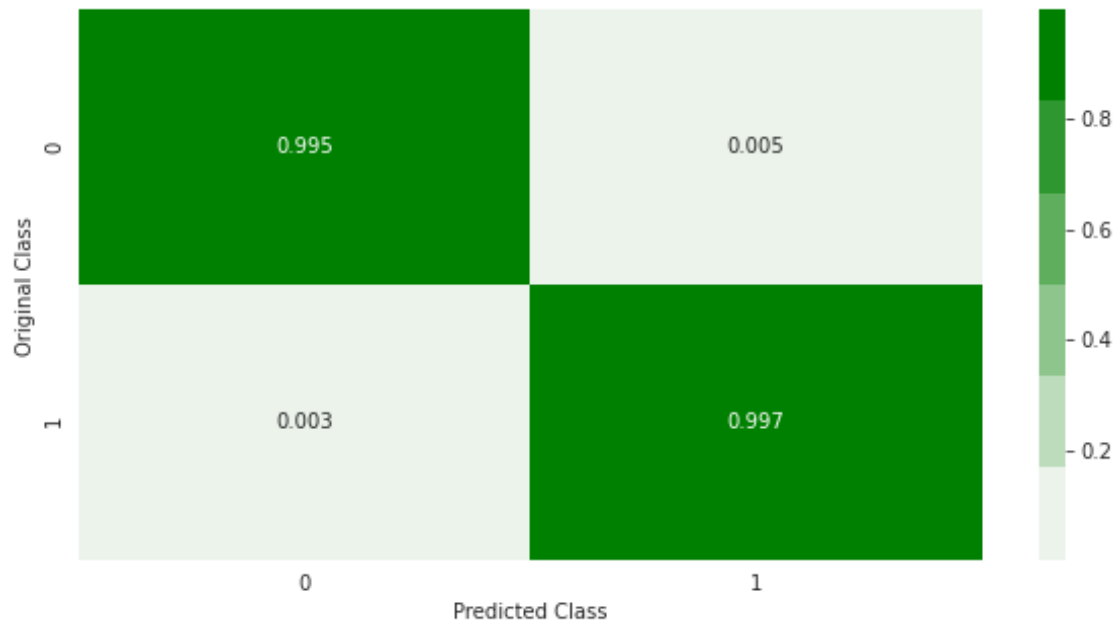
```



----- Precision matrix

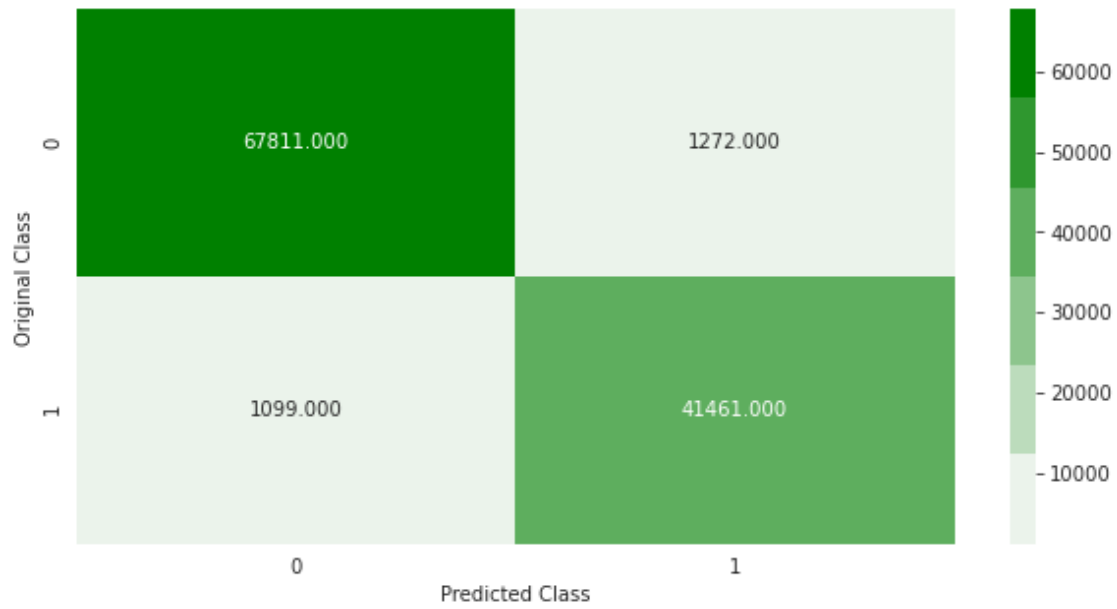


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

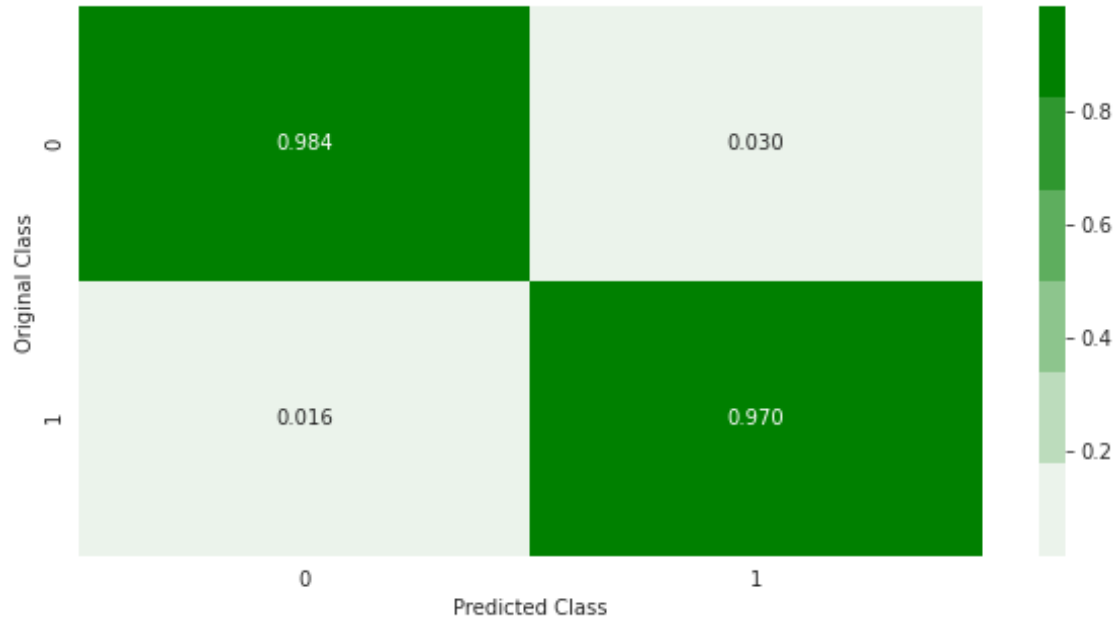


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.6336851249758755
The Weighted Recall Score:  0.9787626631315891
The Weighted Precision Score:  0.9787840750786295
The Weighted F1 Score:  0.9787708885184319
```

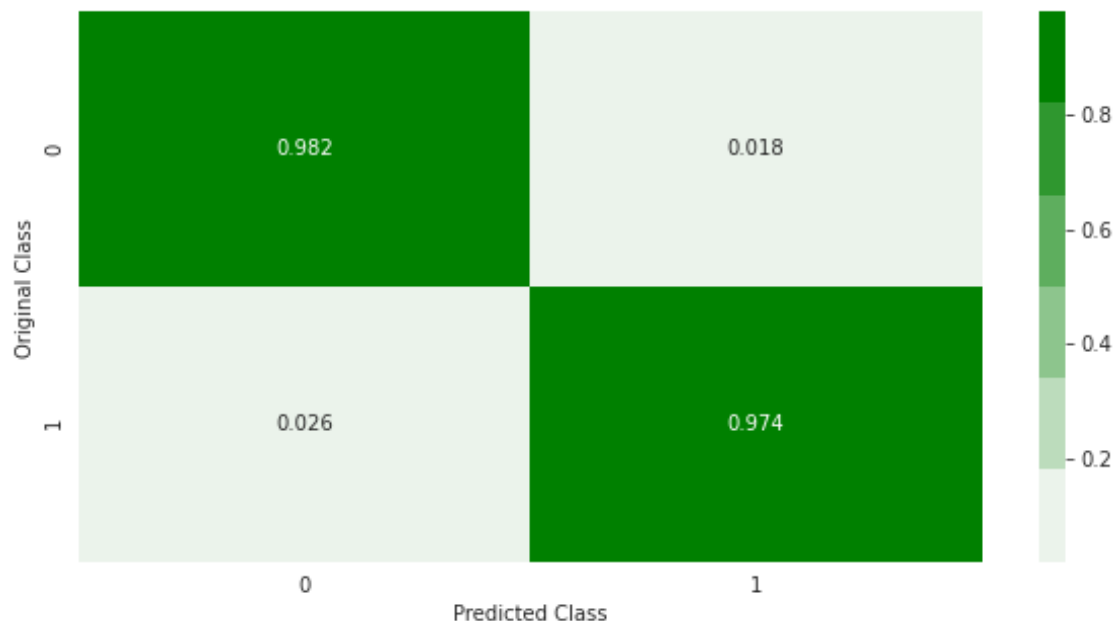
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.6s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.3s
[CV] END ...class_weight={0: 0.0, 1: 1.0}; total time= 6.9s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 34.5s
[CV] END ...class_weight={0: 0.11, 1: 0.89}; total time= 33.9s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 37.8s
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 1.2min
[CV] END ...class_weight={0: 0.22, 1: 0.78}; total time= 1.1min
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 1.2min
[CV] END ...class_weight={0: 0.33, 1: 0.6699999999999999}; total time= 1.5min
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 1.3min
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 1.2min
[CV] END ...class_weight={0: 0.44, 1: 0.56}; total time= 1.2min
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 1.2min
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 1.2min
[CV] END ...class_weight={0: 0.55, 1: 0.44999999999999996}; total time= 1.2min
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 1.2min
[CV] END ...class_weight={0: 0.66, 1: 0.33999999999999997}; total time= 1.2min
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 1.2min
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 1.2min
[CV] END ...class_weight={0: 0.77, 1: 0.22999999999999998}; total time= 1.3min
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 1.5min
[CV] END ...class_weight={0: 0.88, 1: 0.12}; total time= 1.3min
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 1.2min
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 1.1min
[CV] END ...class_weight={0: 0.99, 1: 0.010000000000000009}; total time= 54.5s
```

10.8 Stacking Classifier after dropping the features with high VIF Values

```
[ ]: from sklearn.ensemble import StackingClassifier
      estimators=[('rf',
                  RandomForestClassifier(max_depth=14,min_samples_split=5,criterion='gini',n_estimators=100)),
                  LGBMClassifier())]

      stk_clf = StackingClassifier(estimators=estimators,
                                  final_estimator=LogisticRegression())

      stk_clf.fit(train_fin32, train_y)

[ ]: StackingClassifier(estimators=[('rf',
                                   RandomForestClassifier(max_depth=14,
                                                           min_samples_split=5)),
                                   ('lgb', LGBMClassifier())],
                       final_estimator=LogisticRegression())

[ ]: train_y_pred_stk= stk_clf.predict_proba(train_fin32)
      train_y_pred_stk= train_y_pred_stk[:,1]

      cv_y_pred_stk= stk_clf.predict_proba(cv_fin32)
      cv_y_pred_stk= cv_y_pred_stk[:,1]

[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y, train_y_pred_stk)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred_stk)
```

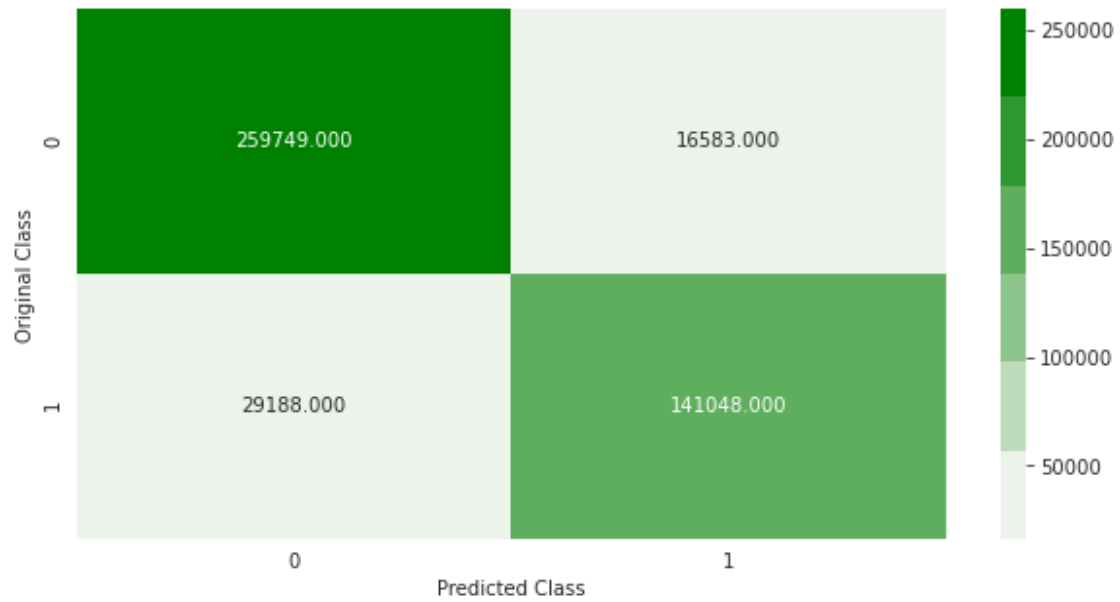
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.89750497124738

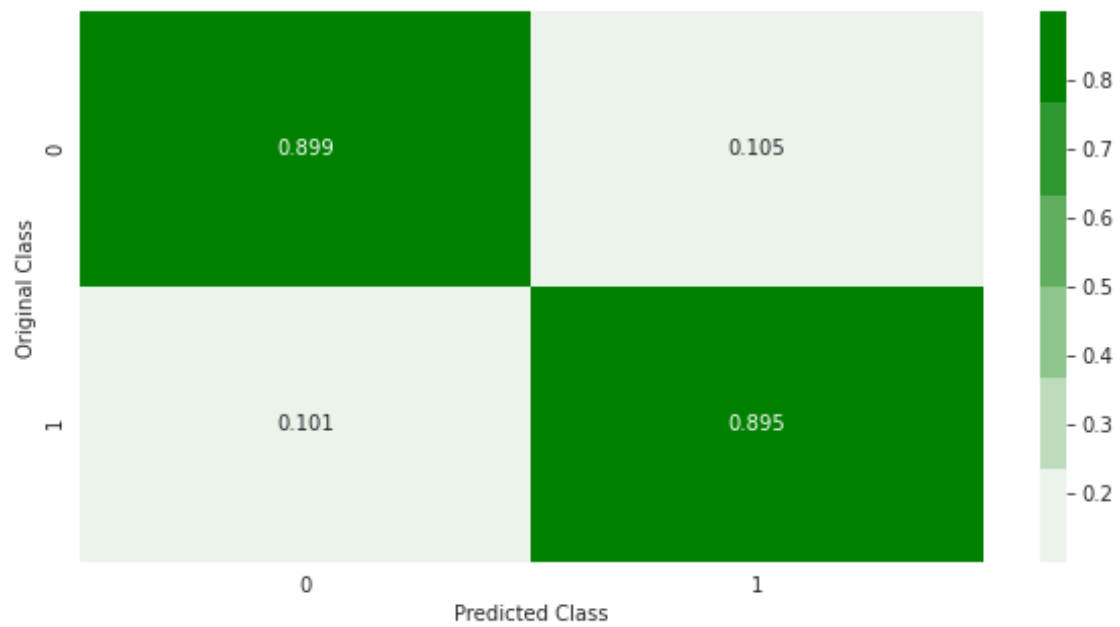
The Weighted Precision Score: 0.8973869051673463

The Weighted F1 Score: 0.896677510971728

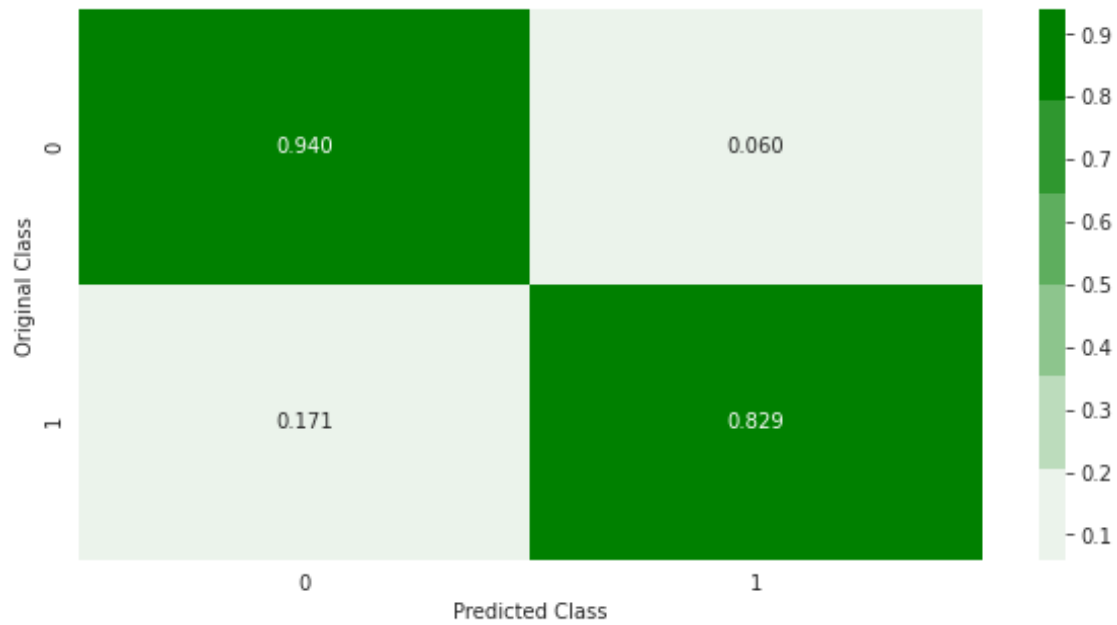
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

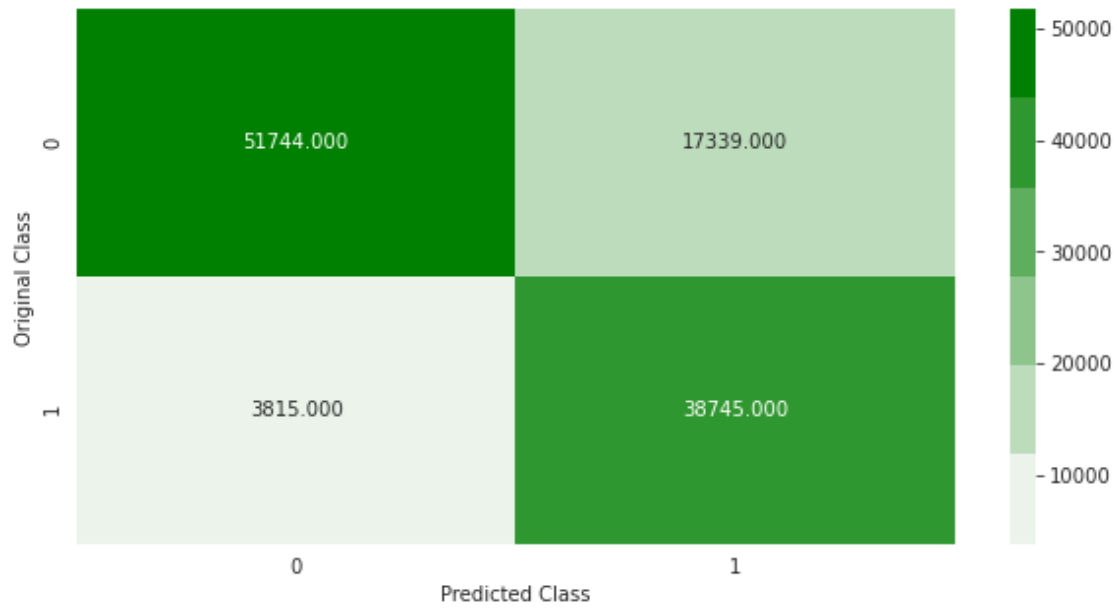
Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.8105210358016176

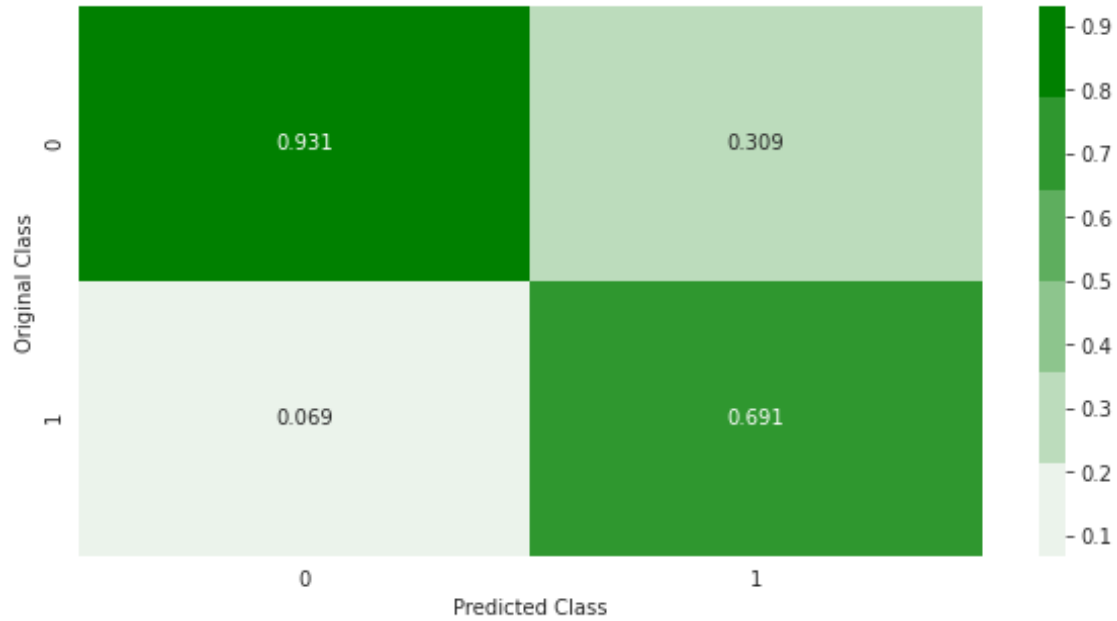
The Weighted Precision Score: 0.8396537297467751

The Weighted F1 Score: 0.8132302355352053

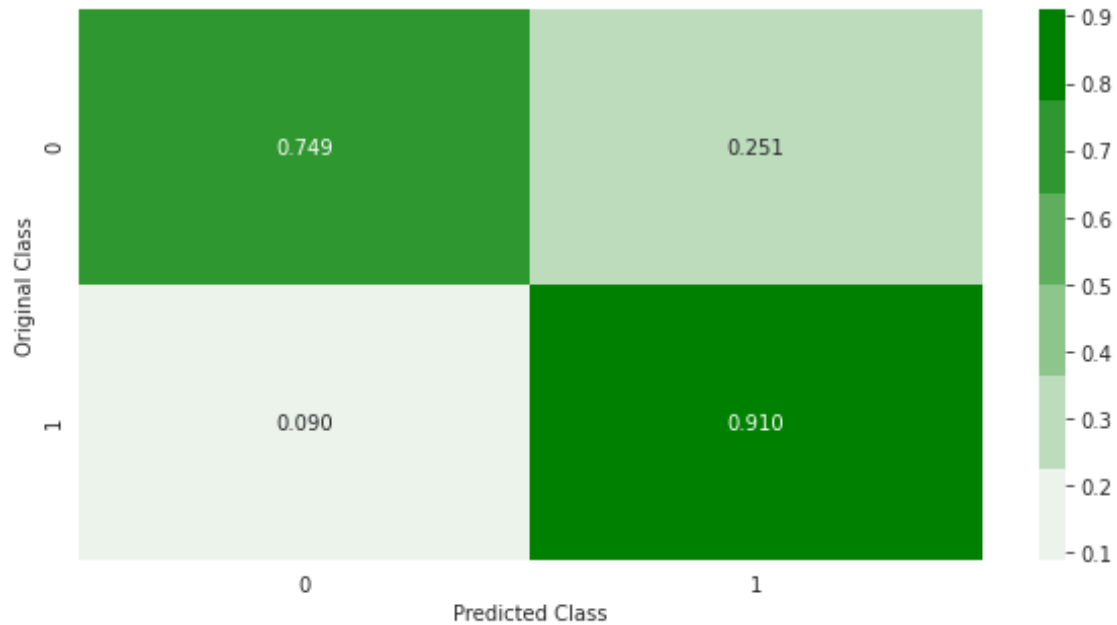
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



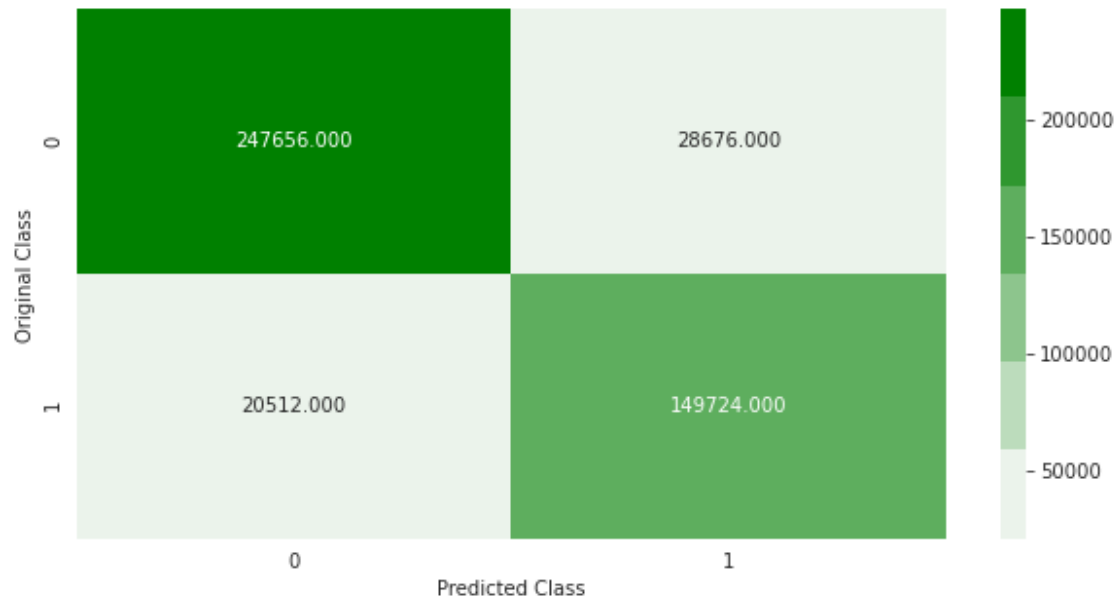
Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y,train_y_pred_stk)

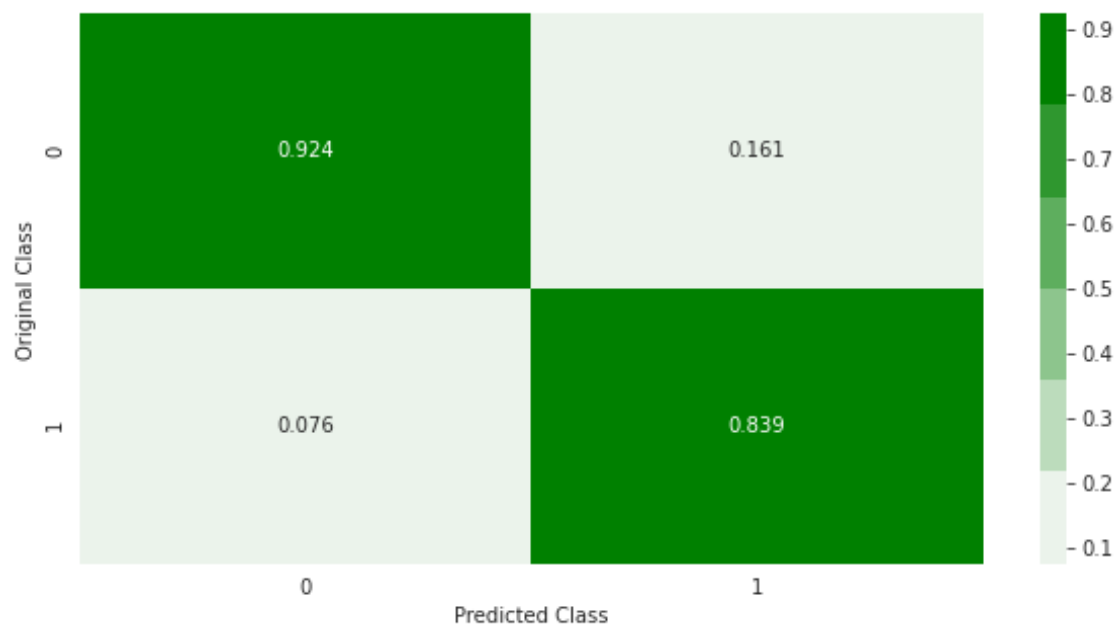
      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred_stk)
```

Confusion Matrix for Best Thresold for the Train Data
 best train threshold : 0.29770238037226315
 The Weighted Recall Score: 0.8898532810232708
 The Weighted Precision Score: 0.8913935200304479
 The Weighted F1 Score: 0.8903171860417844

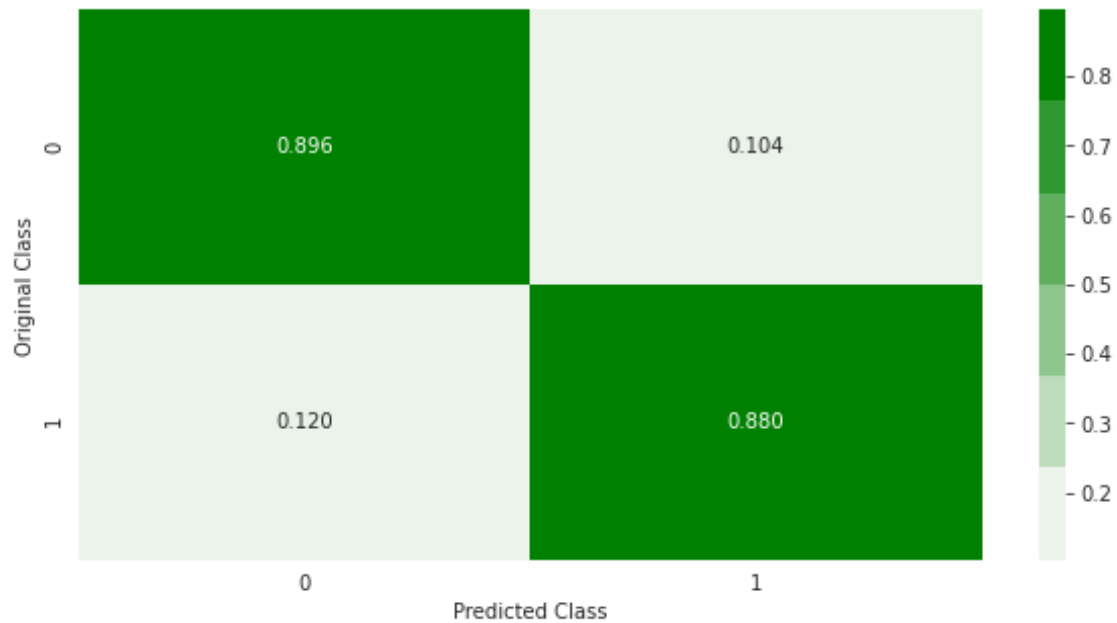
```
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix

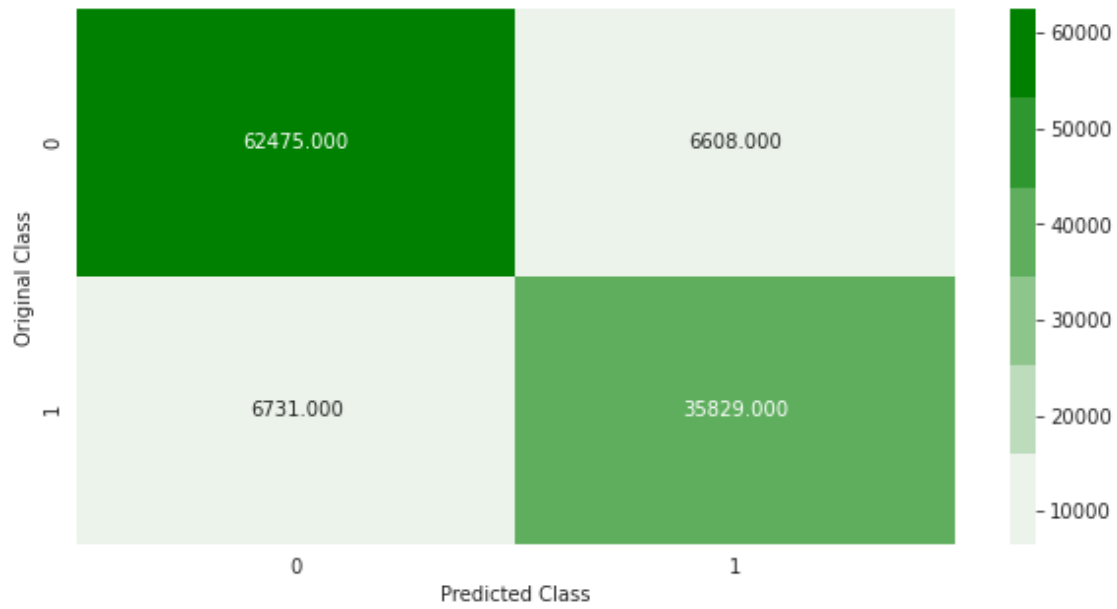


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

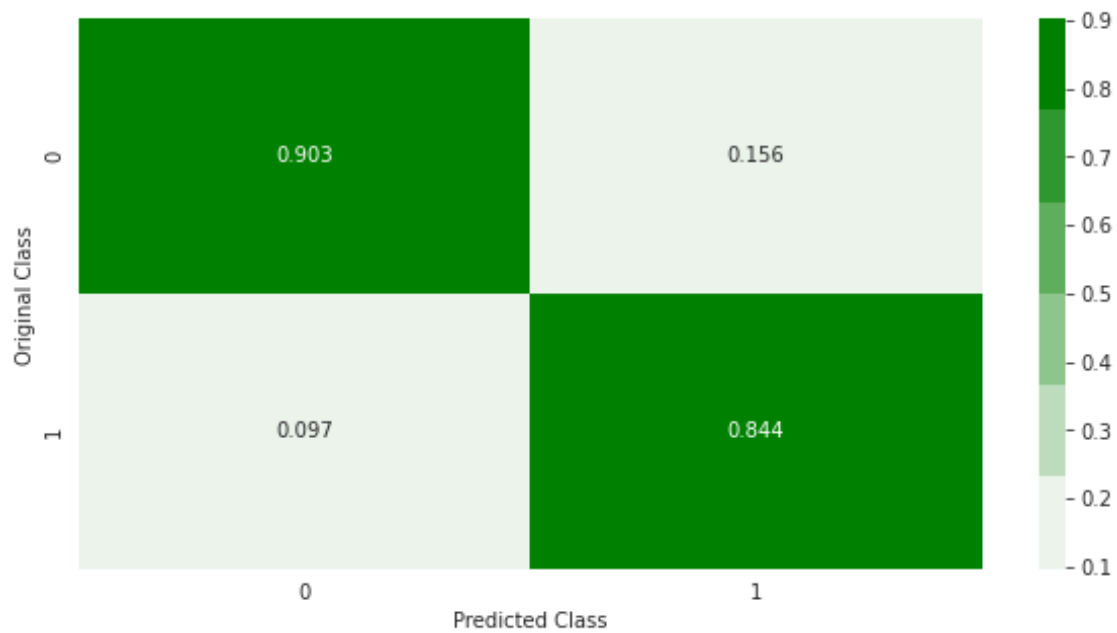


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.7852369844435586
The Weighted Recall Score:  0.8805209462303951
The Weighted Precision Score:  0.8804565472347925
The Weighted F1 Score:  0.8804876313946836
```

```
=====
=====
=====
----- Confusion matrix
-----
```

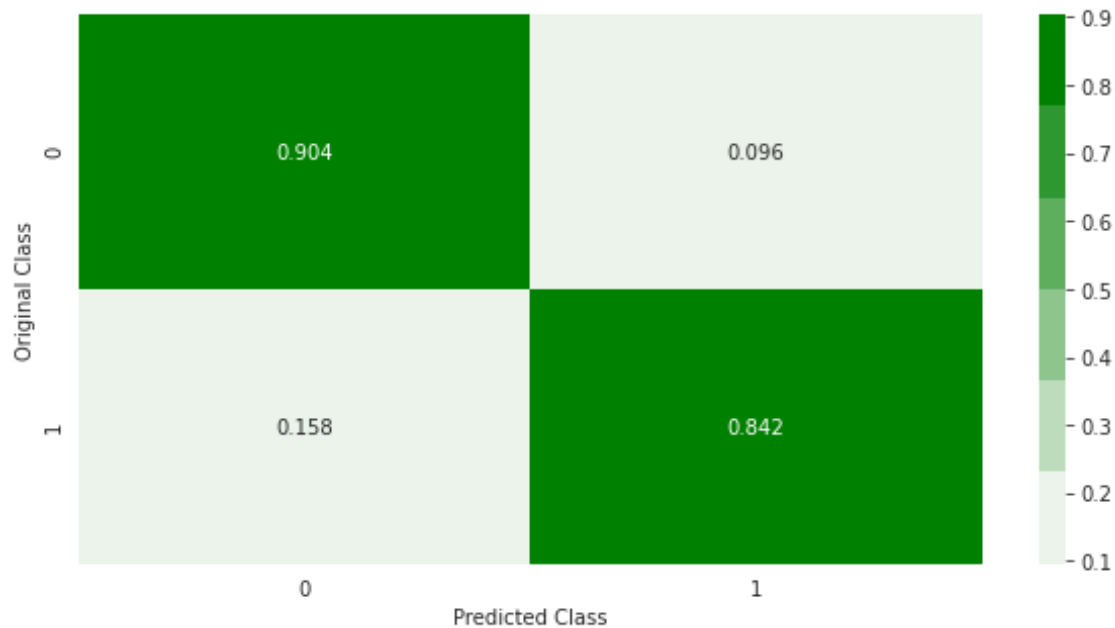



----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

10.9 Observations on Experiment 2:

1. The performance of all the supervised models in experiment 2 is below par than all the supervised models in Experiment 1.
2. Although the lightGBM out performed Random Forests classifier and logistic regression in experiment 2 there has not been any improvement in the scores compared to experiment 1.

10.10 Next Steps:

1. Correcting for Class Imbalance using SMOTE Oversampling techniques to see if the performance of the models might improve.

11 Experiment 3:

11.1 Using SMOTE Oversampling technique in order to create a dataset corrected for Data imbalance

```
[ ]: pip install imblearn
```

```
Requirement already satisfied: imblearn in /opt/conda/lib/python3.7/site-packages (0.0)
```

```
Requirement already satisfied: imbalanced-learn in /opt/conda/lib/python3.7/site-packages (from imblearn) (0.8.0)
```

Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.19.5)
 Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.0.1)
 Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.7.0)
 Requirement already satisfied: scikit-learn>=0.24 in /opt/conda/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (0.24.2)
 Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn>=0.24->imbalanced-learn->imblearn) (2.2.0)
 Note: you may need to restart the kernel to use updated packages.

```
[ ]: from imblearn.over_sampling import SMOTE
smt= SMOTE(random_state=20)
train_x_smt, train_y_smt = smt.fit_resample(train_fin4,train_y)
```

```
[ ]: train_y_smt[:4]
```

```
[ ]: 0    0
      1    1
      2    1
      3    1
      Name: PotentialFraud, dtype: int64
```

```
[ ]: with open('/home/megha_murthy_n/train_smt_v2.pkl','wb') as tr_smt:
      pickle.dump(train_x_smt,tr_smt)
      with open('/home/megha_murthy_n/train_smt_y_v2.pkl','wb') as tr_smt_y:
        pickle.dump(train_y_smt,tr_smt_y)
```

```
[ ]: with open('/home/megha_murthy_n/train_smt_v2.pkl','rb') as tr_smt:
      train_x_smt= pd.read_pickle(tr_smt)
      with open('/home/megha_murthy_n/train_smt_y_v2.pkl','rb') as tr_smt_y:
        train_y_smt= pd.read_pickle(tr_smt_y)
```

```
[ ]: print(train_x_smt.shape)
      print(cv_fin4.shape)
      print(train_y_smt.shape)
      print(cv_y.shape)
```

```
(552664, 32)
(111643, 32)
(552664,)
(111643,)
```

11.2 Since Random Forests and lightGBM have outperformed in the experiments 2 and 3.

11.3 Using Random Forests and lightGBM in this case to see if there has been an improvement in the Precision and Recall scores

11.4 Random Forests post performing SMOTE Oversampling

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(class_weight='balanced')
param = {'n_estimators': [100,180,250], 'max_depth' : [
    → [25,35,50], 'min_samples_split': [100,150], 'criterion' : ['gini']}

rf_tune = RandomizedSearchCV(rf,param,cv=5,n_jobs=-1,verbose=1)
rf_tune.fit(train_x_smt, train_y_smt)
print('best parameter : ',rf_tune.best_params_)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

best parameter : {'n_estimators': 250, 'min_samples_split': 100, 'max_depth': 35, 'criterion': 'gini'}

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf_best = [
    → RandomForestClassifier(max_depth=35,min_samples_split=100,criterion='gini',n_estimators=250)
rf_best.fit(train_x_smt,train_y_smt)

#sig_clf = CalibratedClassifierCV(rf_best, method="sigmoid")
#sig_clf.fit(train_x_smt,train_y_smt)

train_y_pred = rf_best.predict_proba(train_x_smt)
train_y_pred = train_y_pred[:,1]
cv_y_pred = rf_best.predict_proba(cv_fin4)
cv_y_pred = cv_y_pred[:,1]
```

11.4.1 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Threshold= 0.5

```
[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y_smt, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)
```

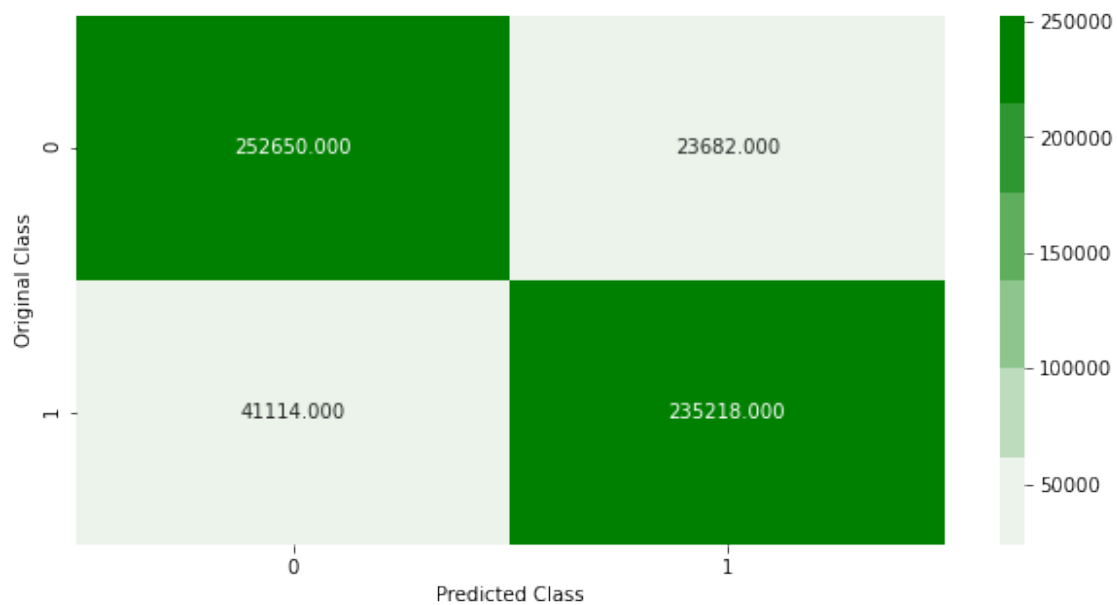
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8827569734956502

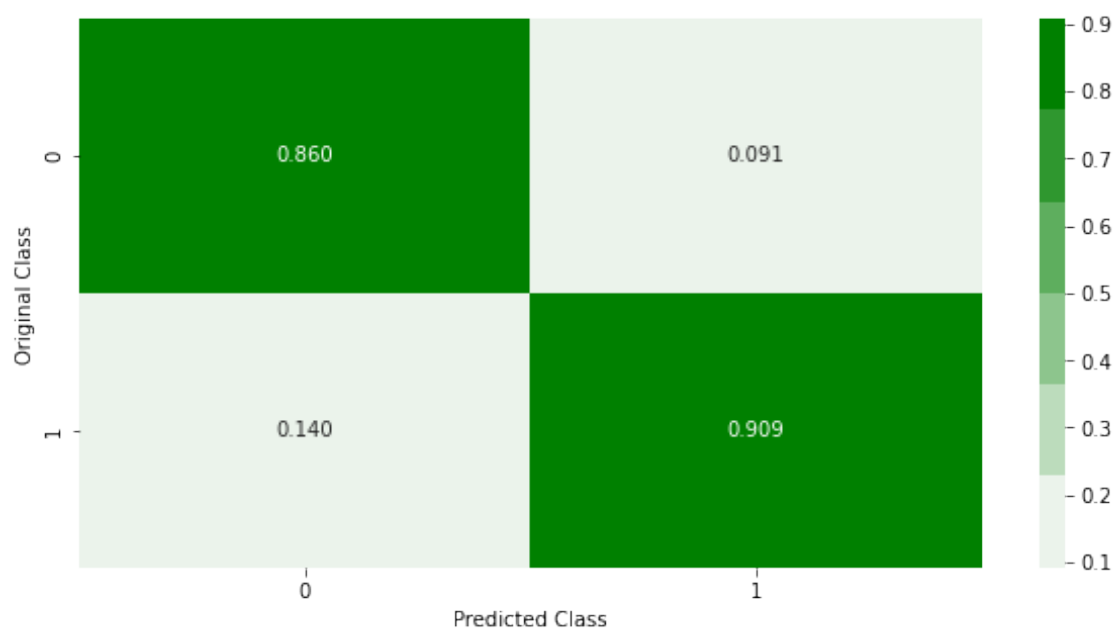
The Weighted Precision Score: 0.8842862531929249

The Weighted F1 Score: 0.8826402142203844

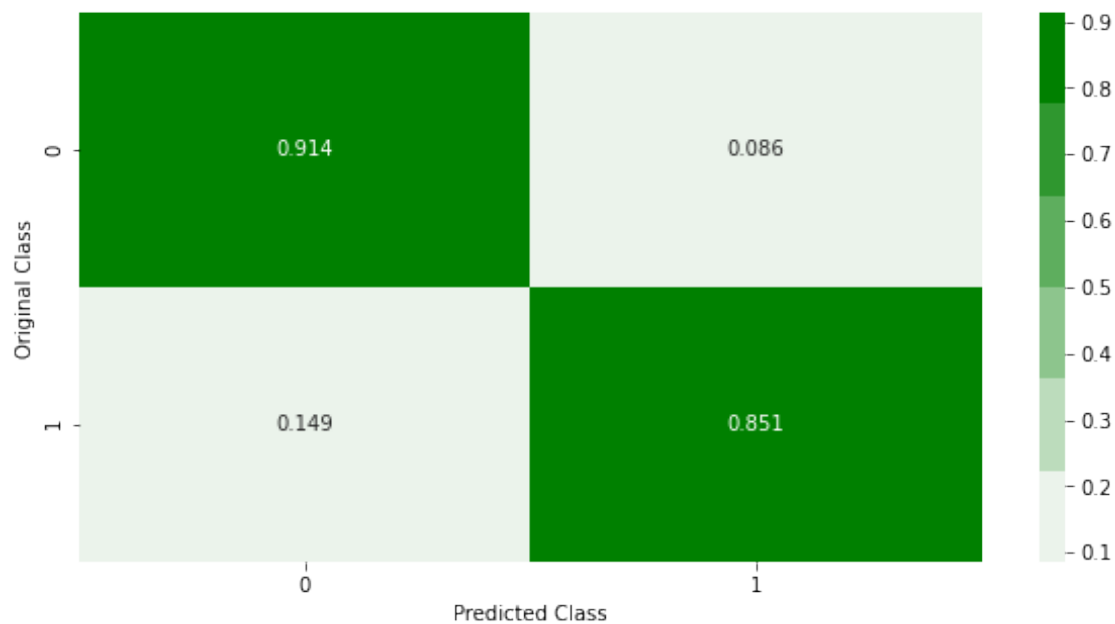
```
=====
=====
=====
----- Confusion matrix
-----
```



```
----- Precision matrix
-----
```

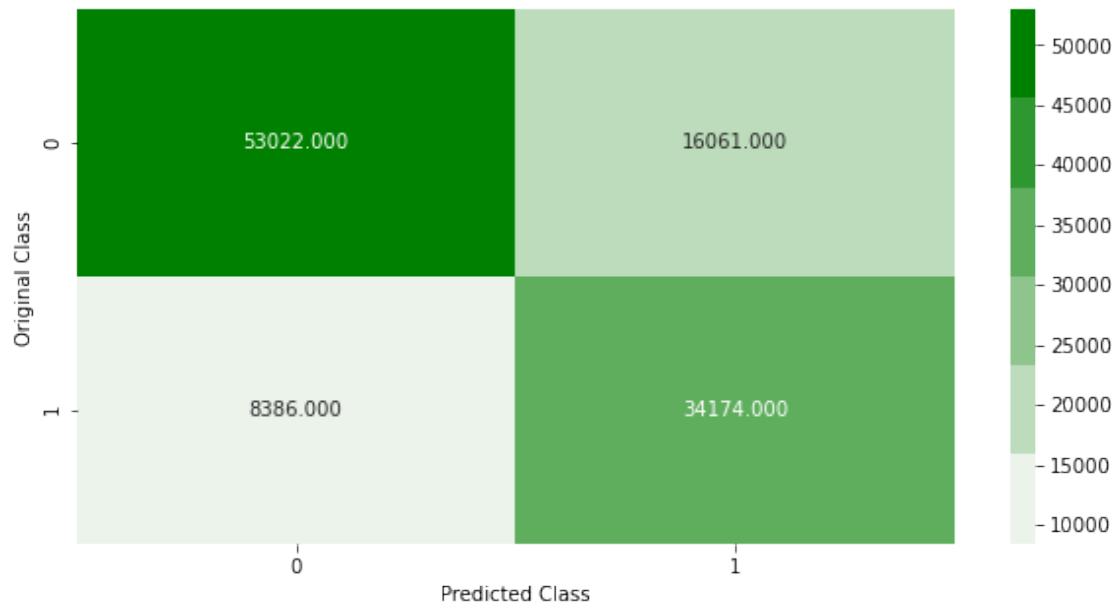


Sum of columns in precision matrix [1. 1.]
----- Recall matrix

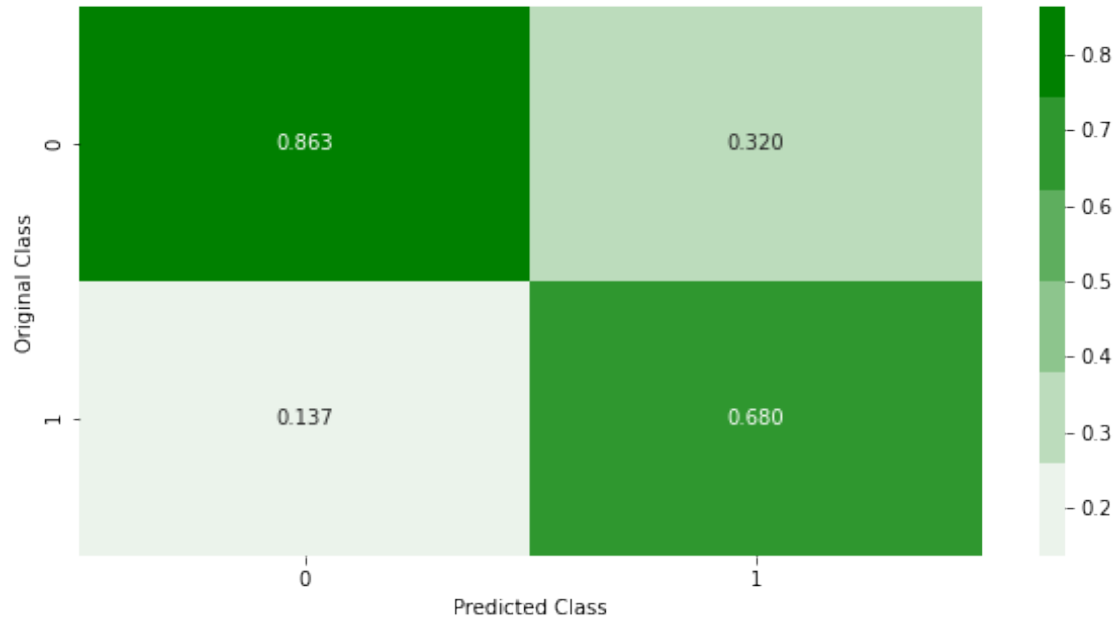


Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.7810252322133945
The Weighted Precision Score: 0.7936164117649752
The Weighted F1 Score: 0.7836412012324007
=====

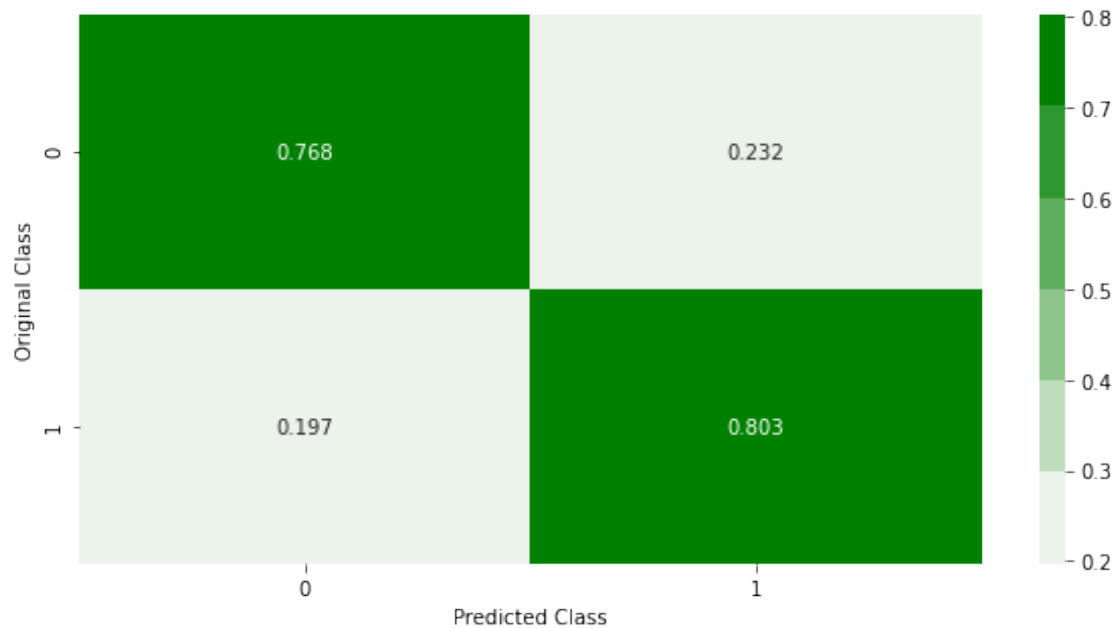
----- Confusion matrix



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

11.4.2 Confusion Matrix, Precision Matrix and Recall Matrix for Train and CV Datasets with Best Threshold obtained using ROC curves

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_smt,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

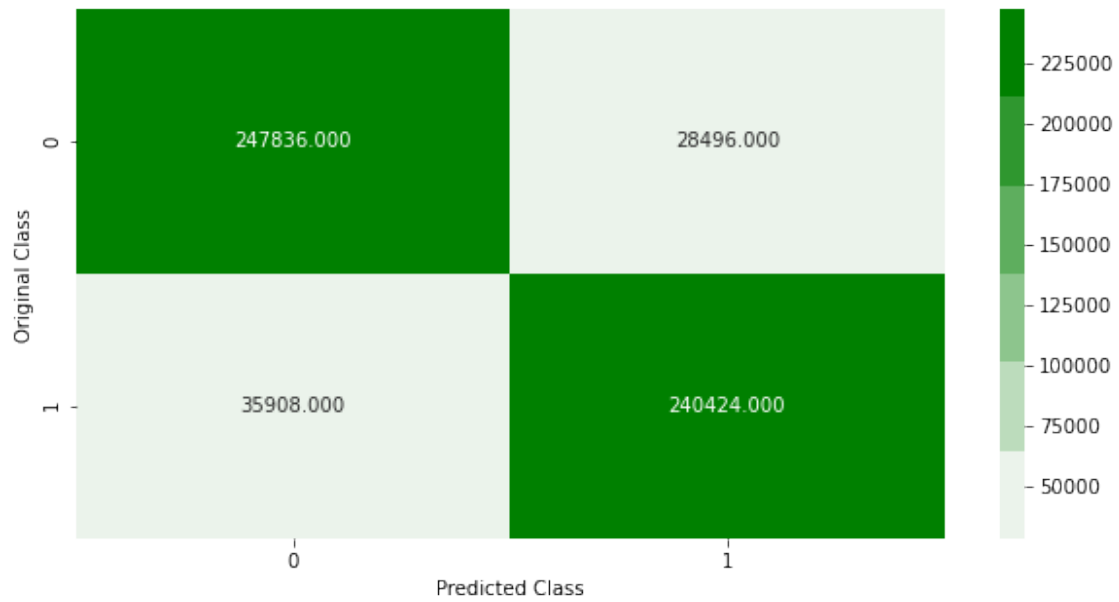
best train threshold : 0.48293765523885707

The Weighted Recall Score: 0.8834662652172025

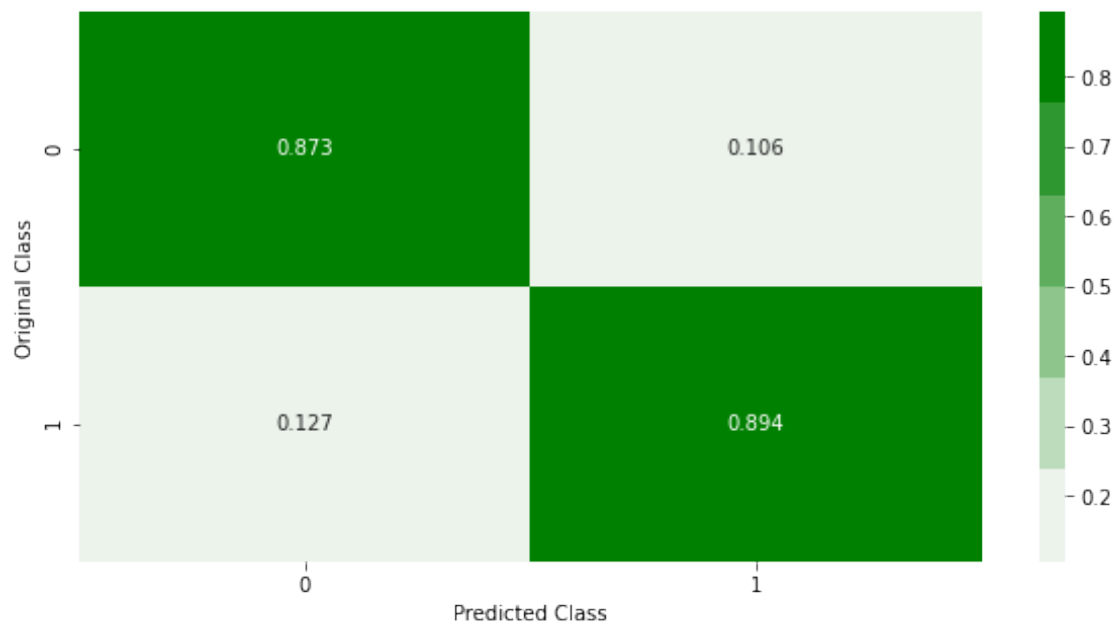
The Weighted Precision Score: 0.883742353639881

The Weighted F1 Score: 0.8834453010190287

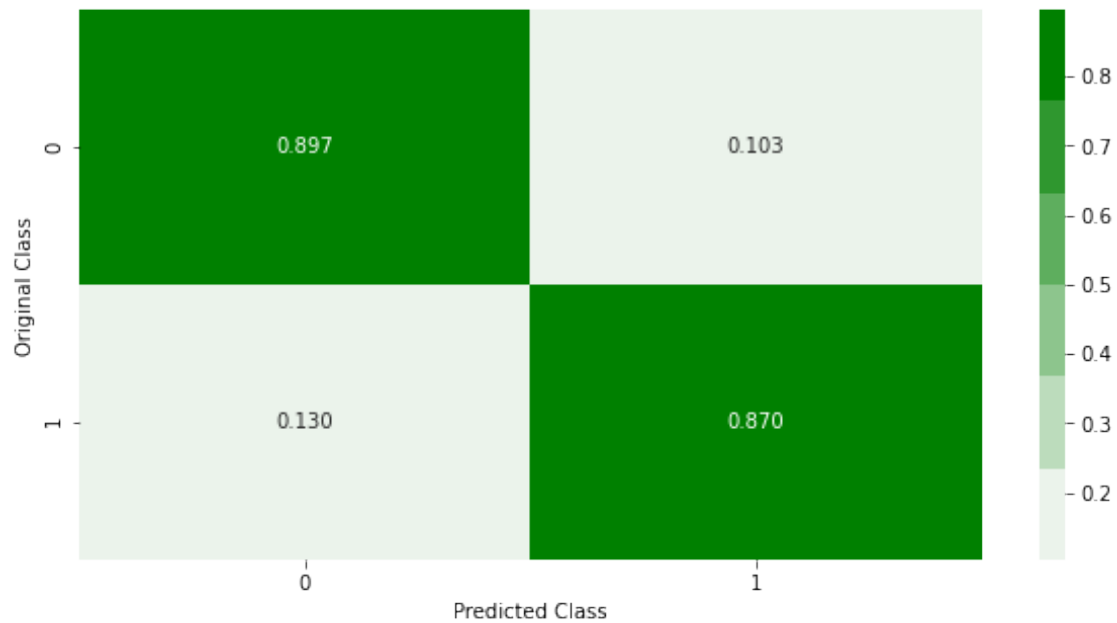
```
=====
=====
=====
----- Confusion matrix
-----
```

----- Precision matrix

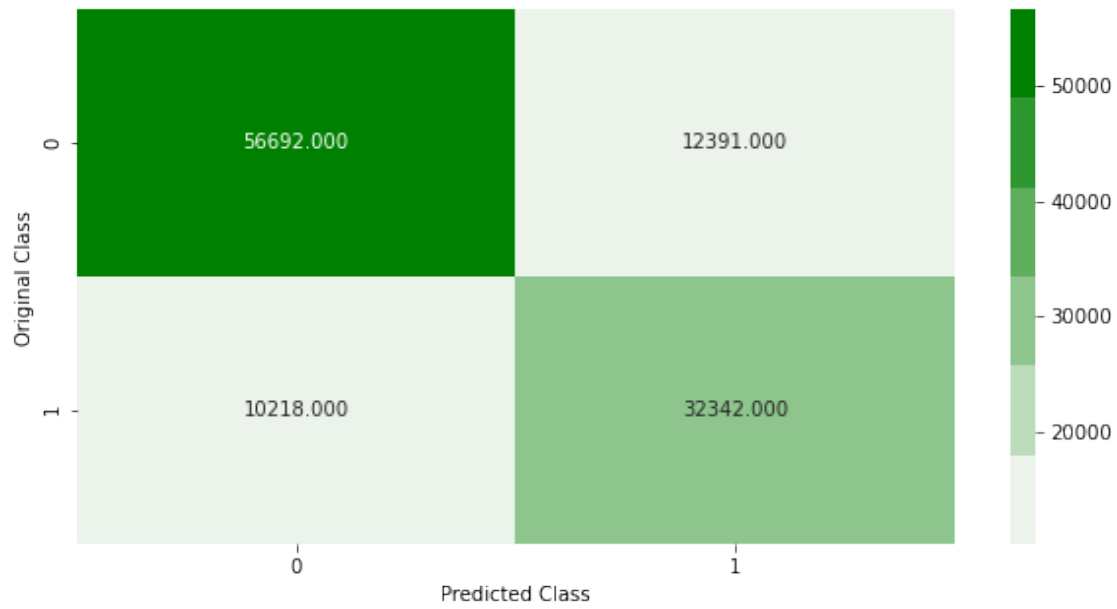


Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix

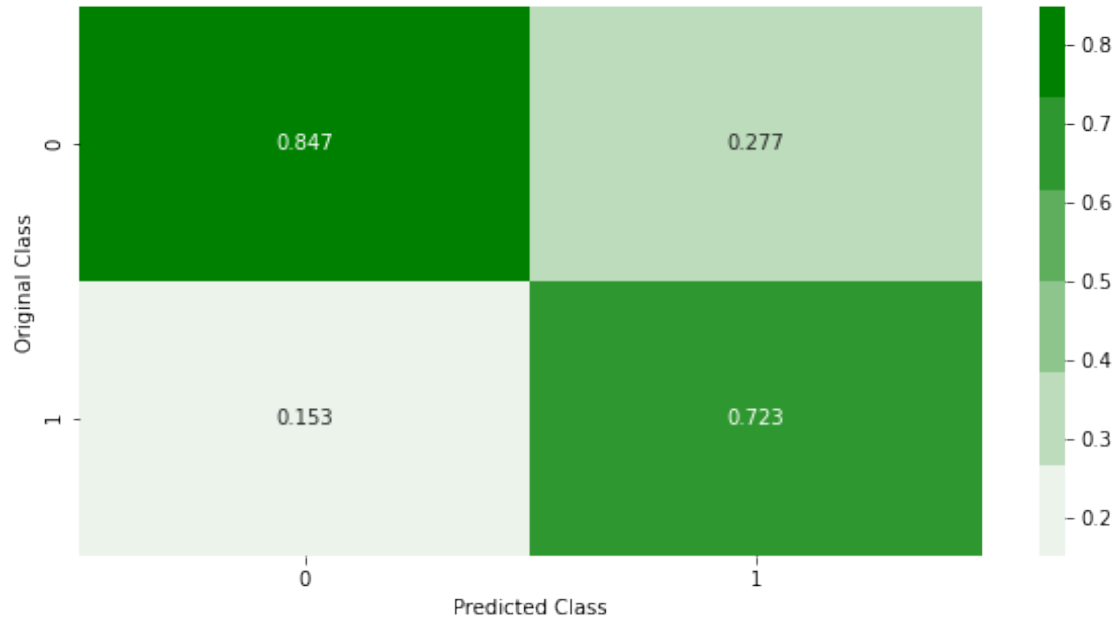


```
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.524356881273391
The Weighted Recall Score:  0.7974884229194844
The Weighted Precision Score:  0.7999075135089689
The Weighted F1 Score:  0.798391058760542
```

```
=====
=====
=====
----- Confusion matrix
-----
```

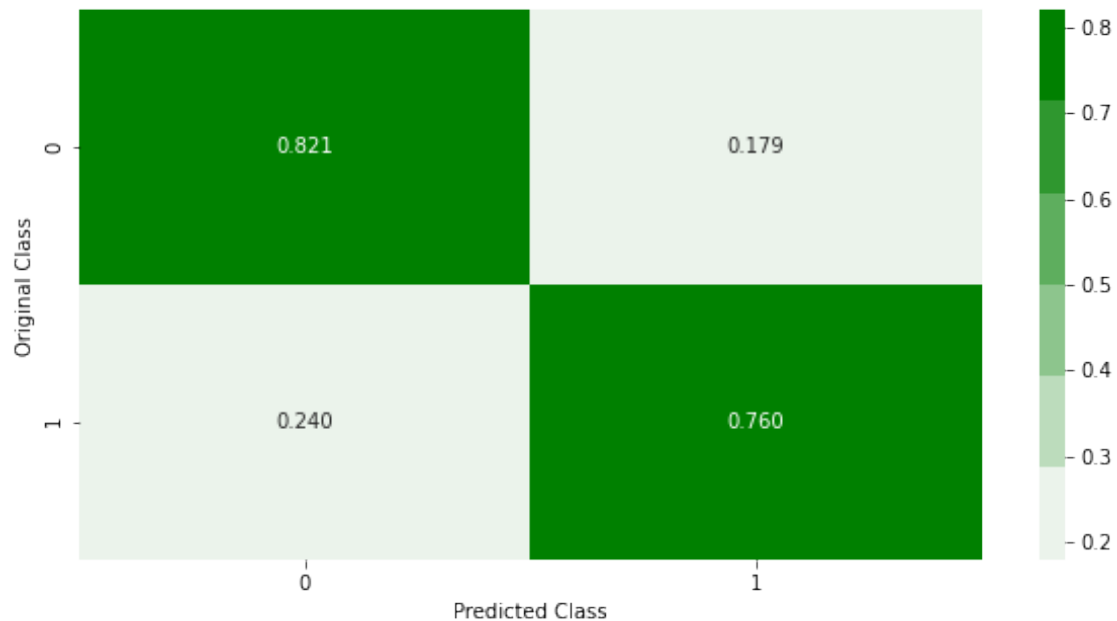


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf_best = \
    RandomForestClassifier(max_depth=35,min_samples_split=100,criterion='gini',n_estimators=150)
rf_best.fit(train_x_smt,train_y_smt)

sig_clf = CalibratedClassifierCV(rf_best, method="sigmoid")
sig_clf.fit(train_x_smt,train_y_smt)

train_y_pred = rf_best.predict(train_x_smt)
cv_y_pred = rf_best.predict(cv_x_smt)

print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y_smt, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y_smt, cv_y_pred)
```

Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8807286163021293

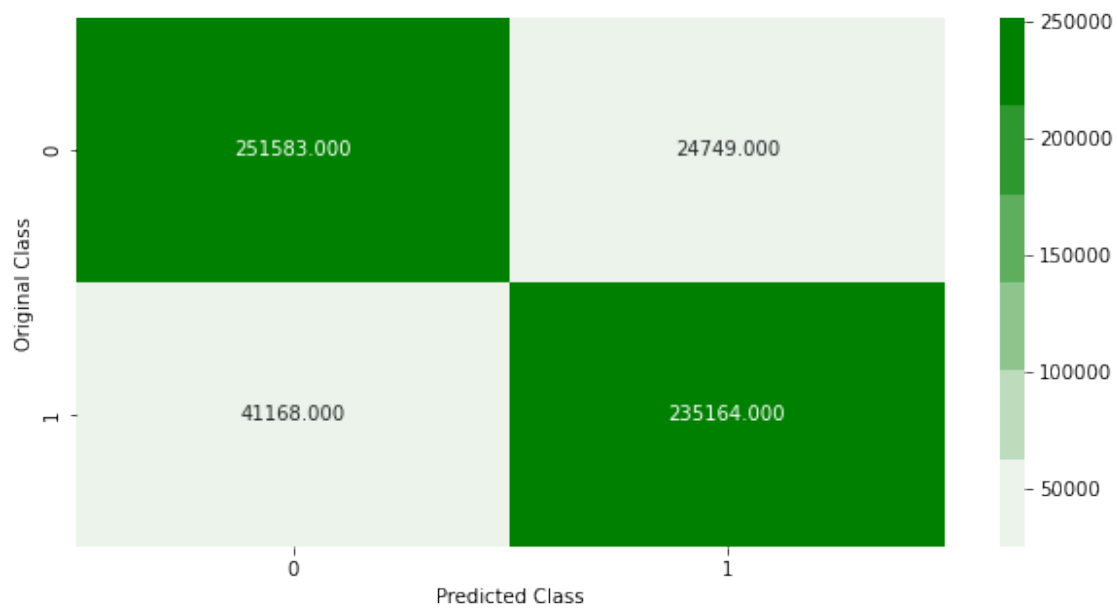
The Weighted Precision Score: 0.8820775249436567

The Weighted F1 Score: 0.8806232526549775

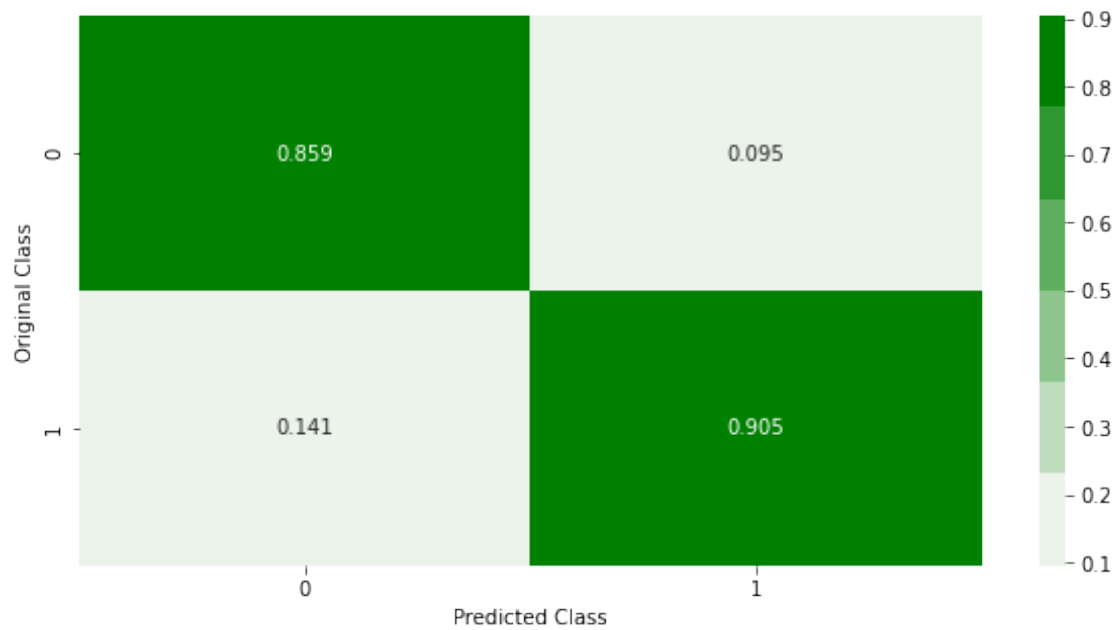
```
=====
=====
=====
```

=====

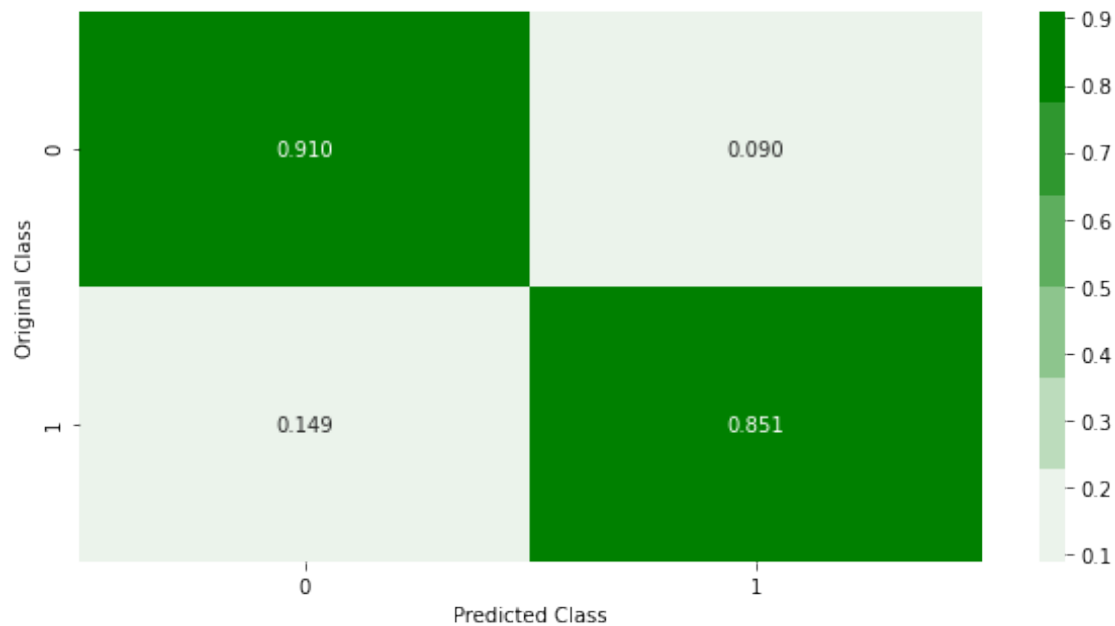
----- Confusion matrix



----- Precision matrix

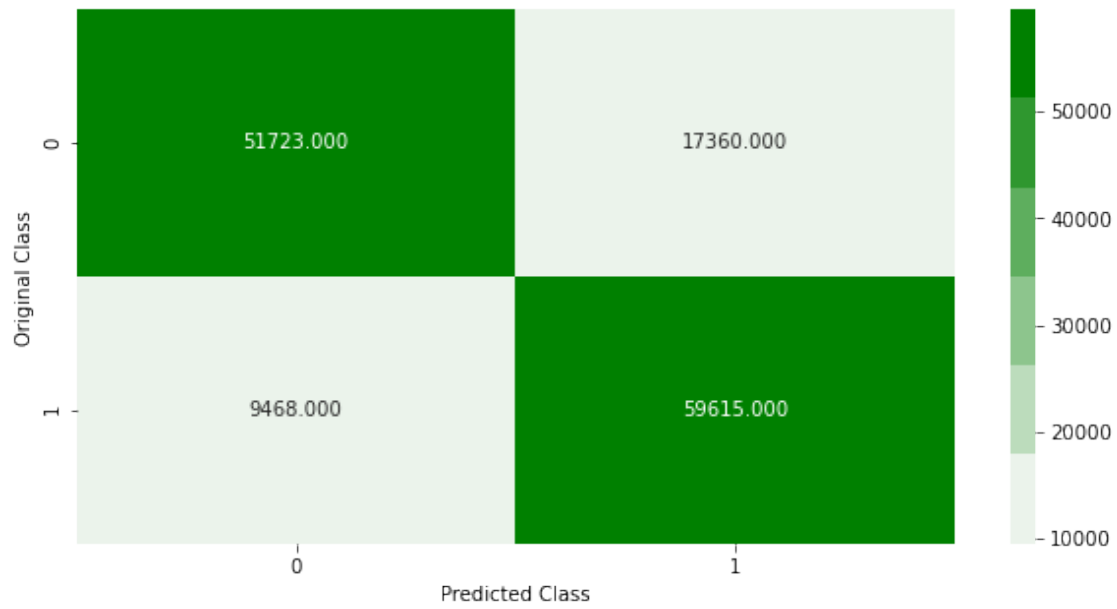


Sum of columns in precision matrix [1. 1.]
----- Recall matrix

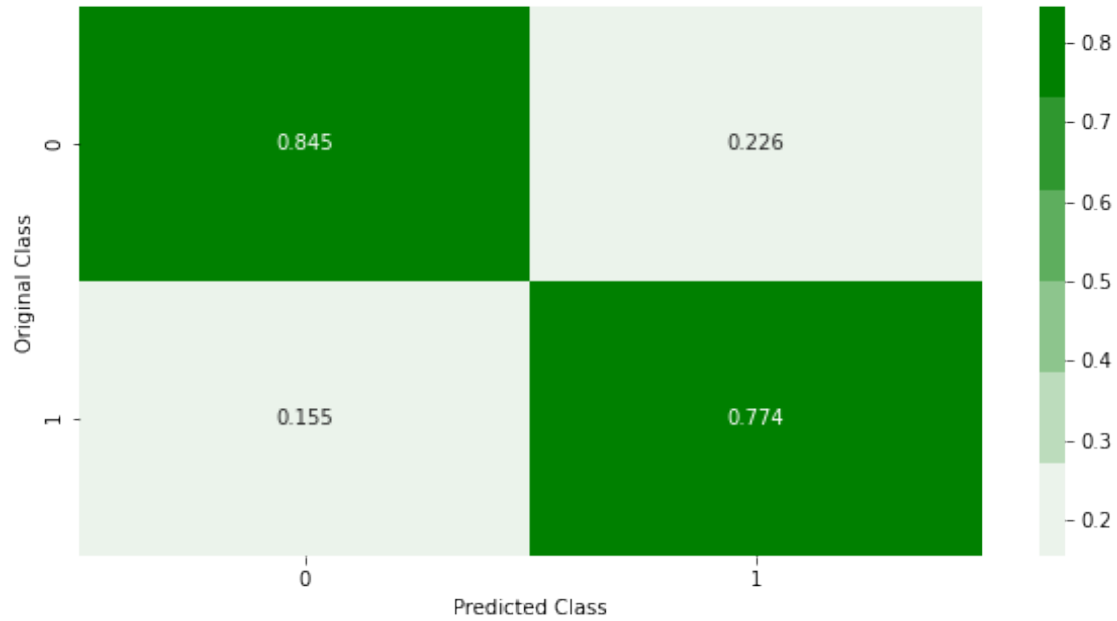


Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.8058277723897341
The Weighted Precision Score: 0.8098717973398548
The Weighted F1 Score: 0.805192180760999
=====

----- Confusion matrix

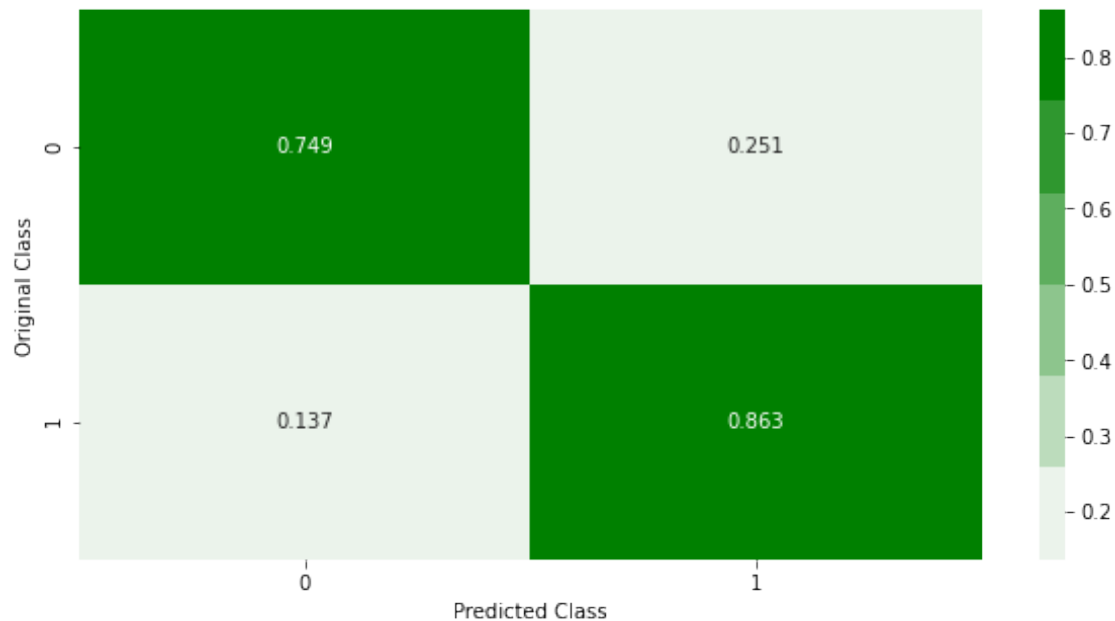


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

11.5 lightGBM post performing SMOTE Oversampling

```
[ ]: pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /opt/conda/lib/python3.7/site-
packages (3.2.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages
(from lightgbm) (1.19.5)
Requirement already satisfied: wheel in /opt/conda/lib/python3.7/site-packages
(from lightgbm) (0.36.2)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages
(from lightgbm) (1.7.0)
Requirement already satisfied: scikit-learn!=0.22.0 in /opt/conda/lib/python3.7
/site-packages (from lightgbm) (0.24.2)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-
packages (from scikit-learn!=0.22.0->lightgbm) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7
/site-packages (from scikit-learn!=0.22.0->lightgbm) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

```
[ ]: import lightgbm as lgb
     clf = lgb.LGBMClassifier()
     clf.fit(train_x_smt,train_y_smt)
```



```

[:]: LGBMClassifier()
[:]: train_y_pred= clf.predict_proba(train_x_smt)
      train_y_pred= train_y_pred[:,1]

      cv_y_pred= clf.predict_proba(cv_fin4)
      cv_y_pred= cv_y_pred[:,1]

[:]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y_smt, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)

```

Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8995284657585803

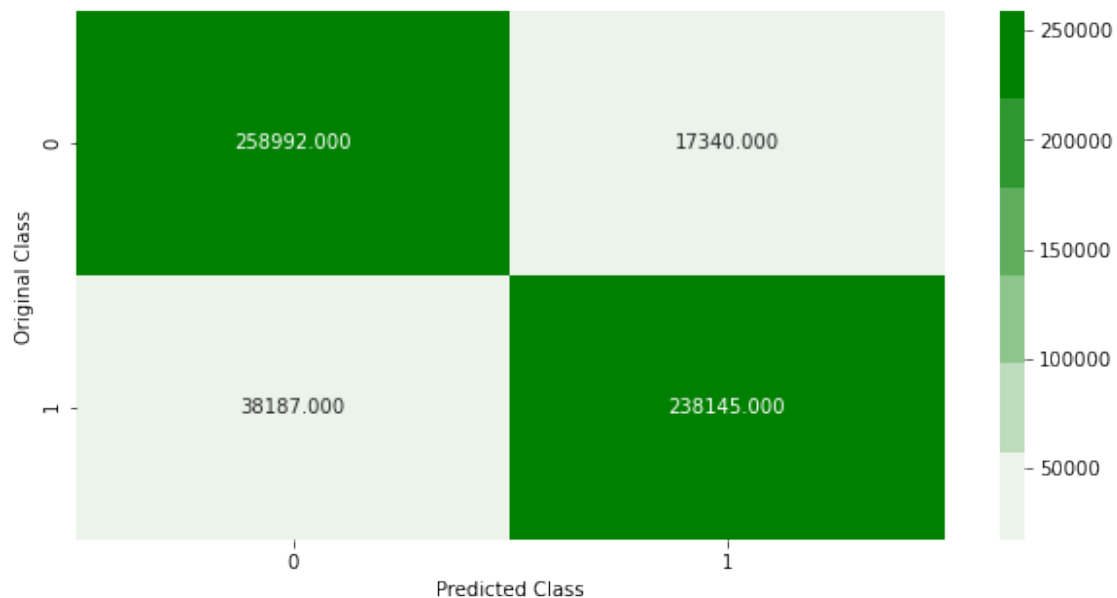
The Weighted Precision Score: 0.9018153876742483

The Weighted F1 Score: 0.8993853042721387

```

=====
=====
=====
----- Confusion matrix
-----

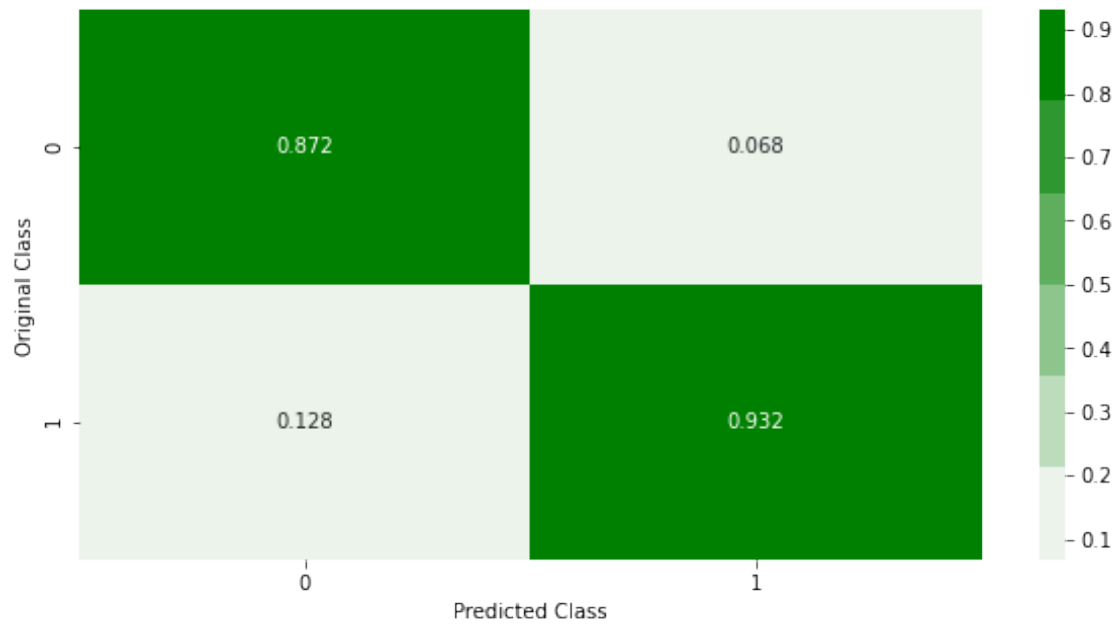
```



```

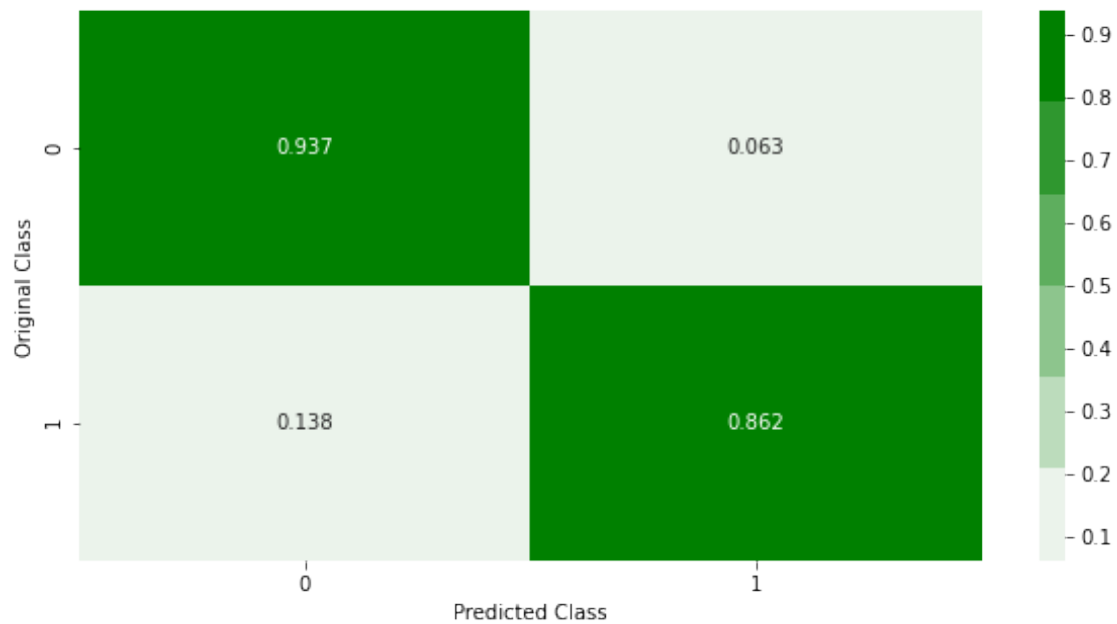
----- Precision matrix
-----

```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

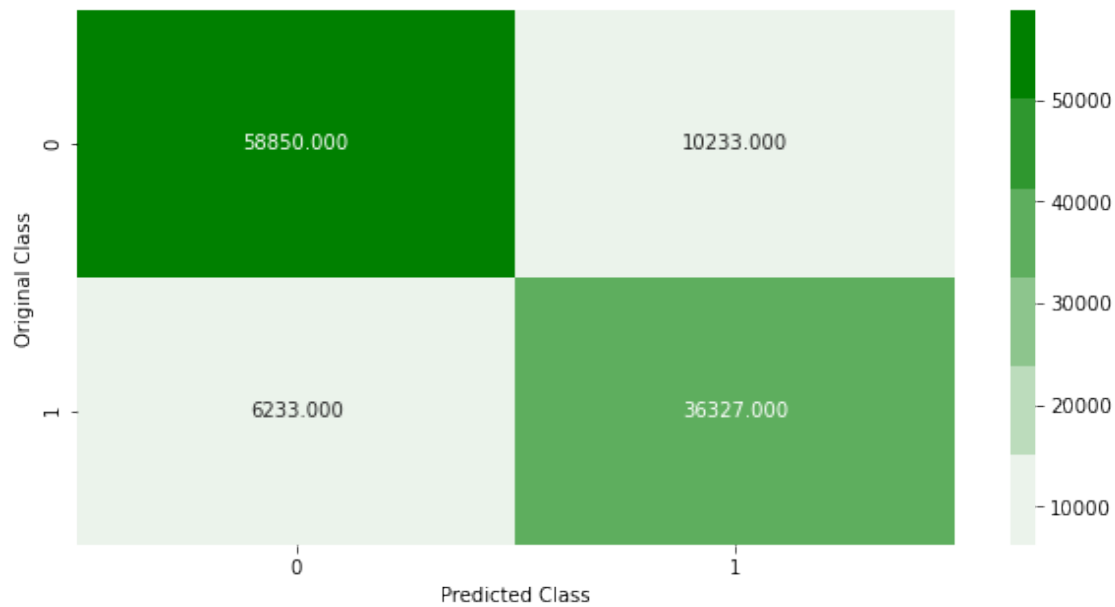
The Weighted Recall Score: 0.8525120249366284
The Weighted Precision Score: 0.8569551485032544
The Weighted F1 Score: 0.8536233092688301

=====

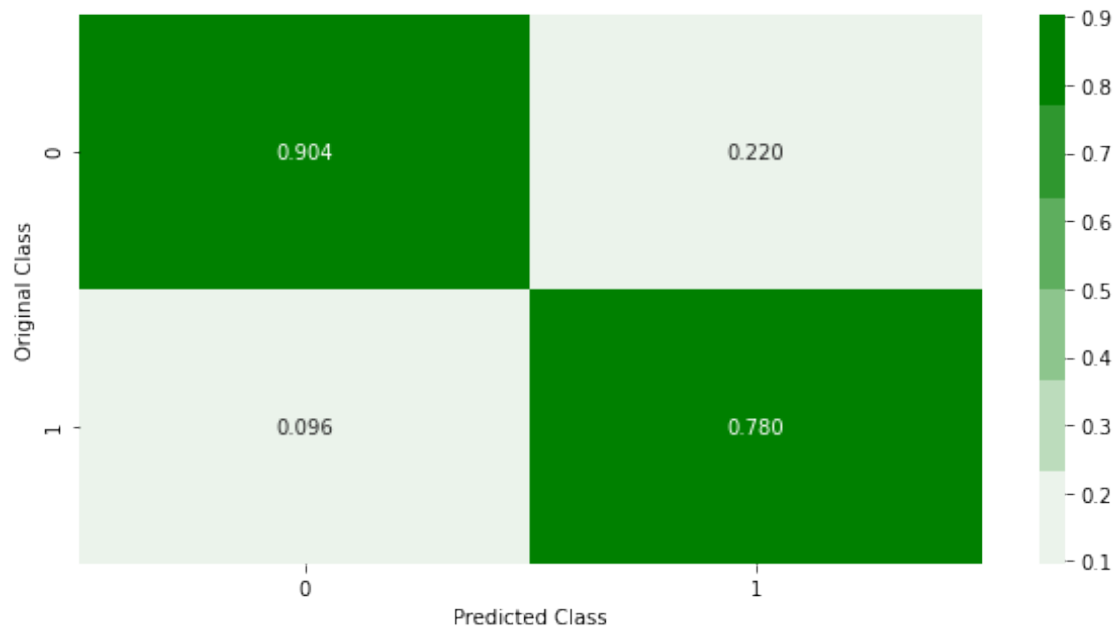
=====

=====

----- Confusion matrix

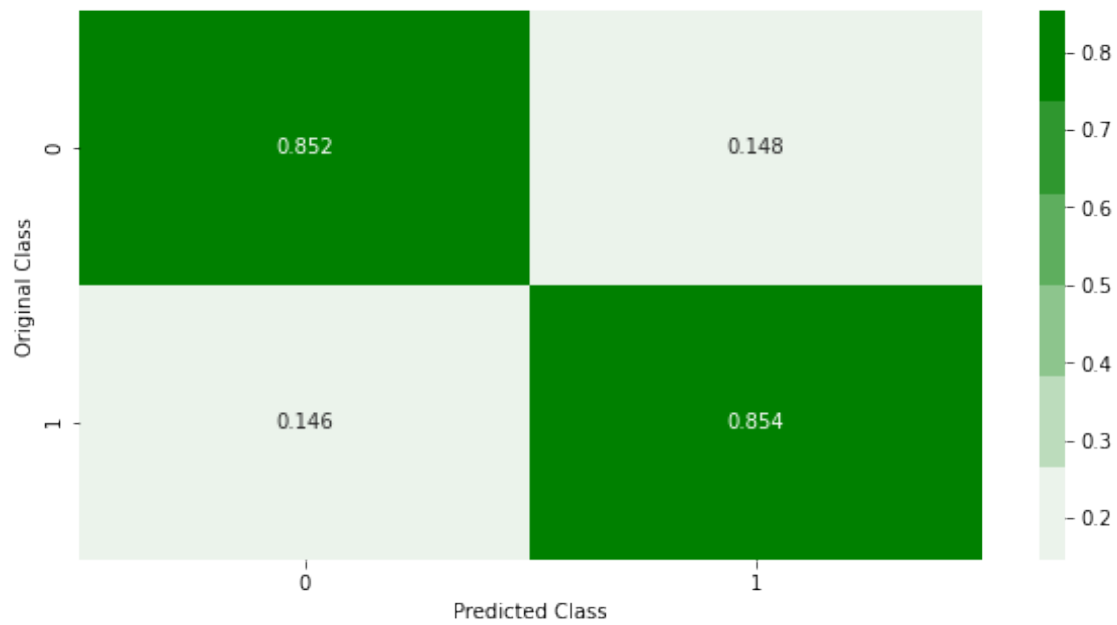


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_smt,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

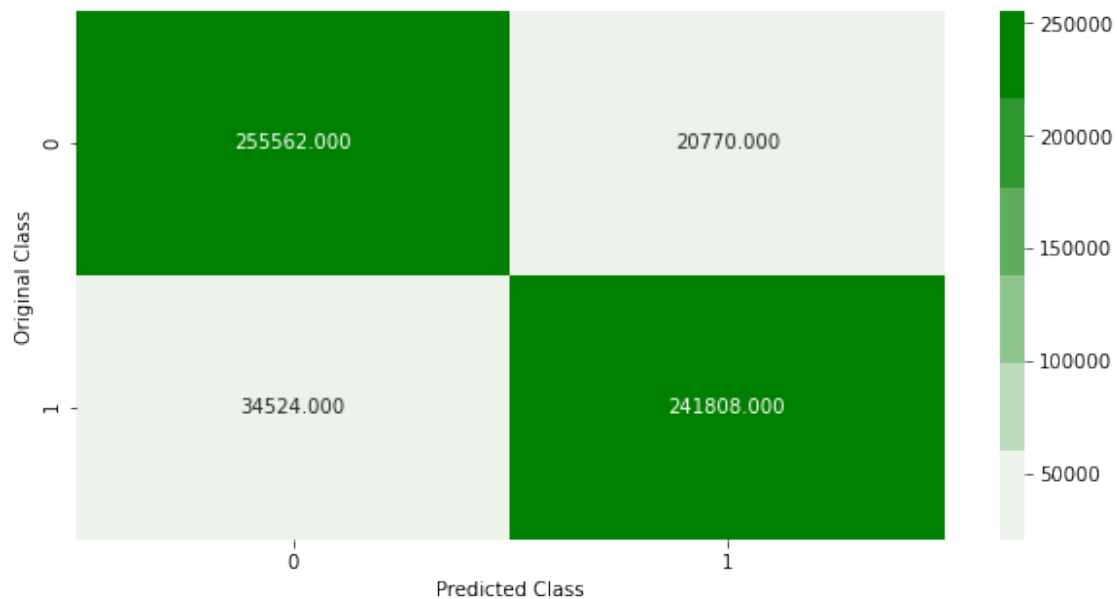
best train threshold : 0.47394922472574996

The Weighted Recall Score: 0.8999500600726662

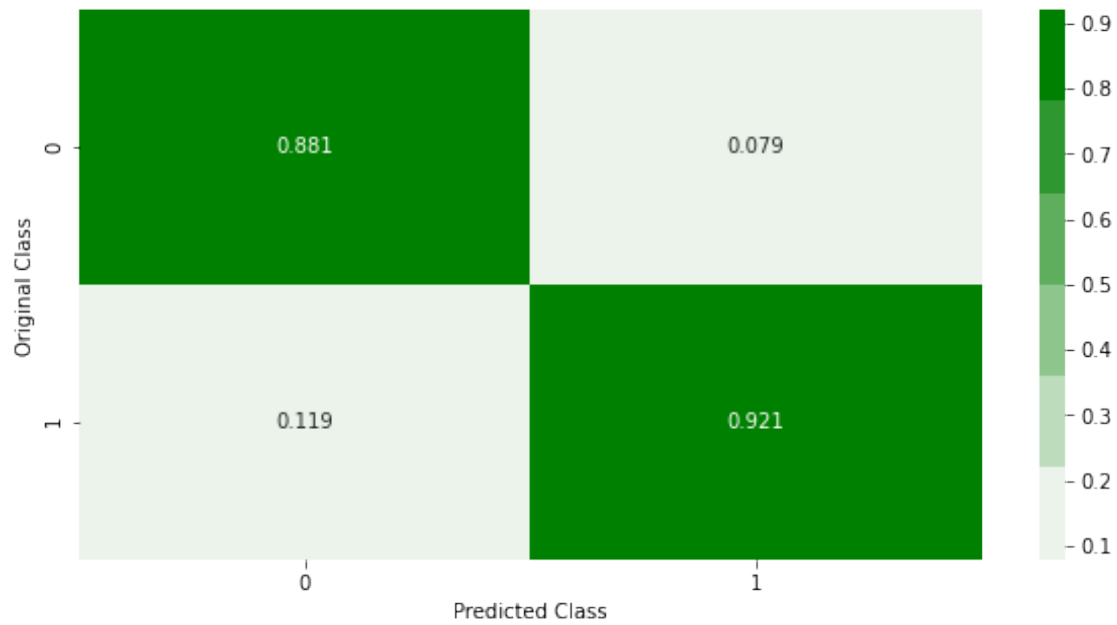
The Weighted Precision Score: 0.9009433561049958

The Weighted F1 Score: 0.899888055804829

```
=====
=====
=====
----- Confusion matrix
-----
```

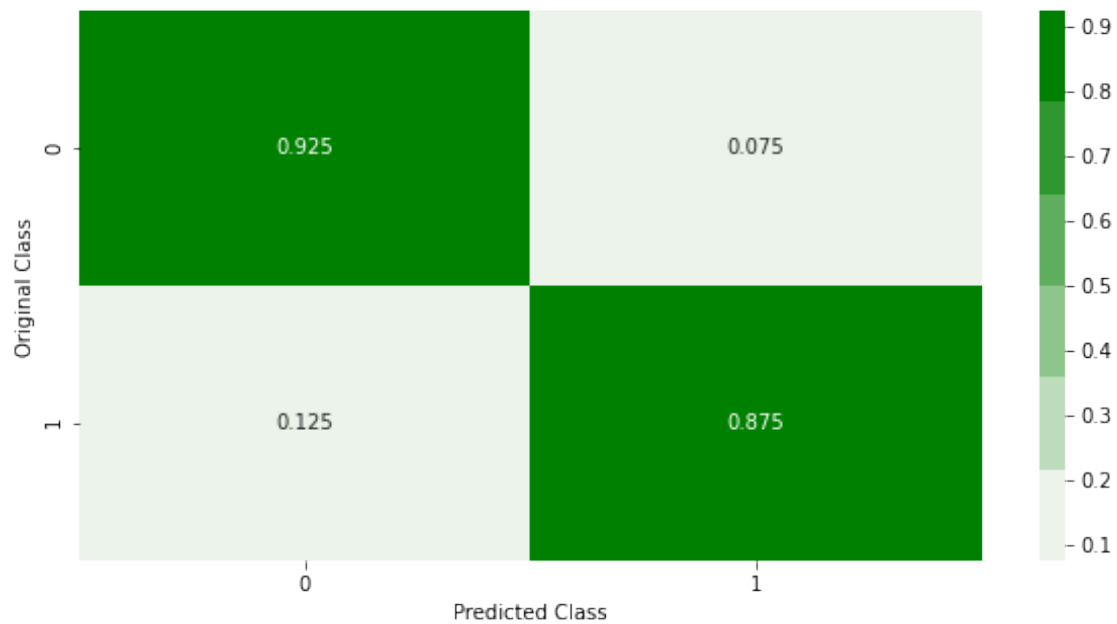


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix

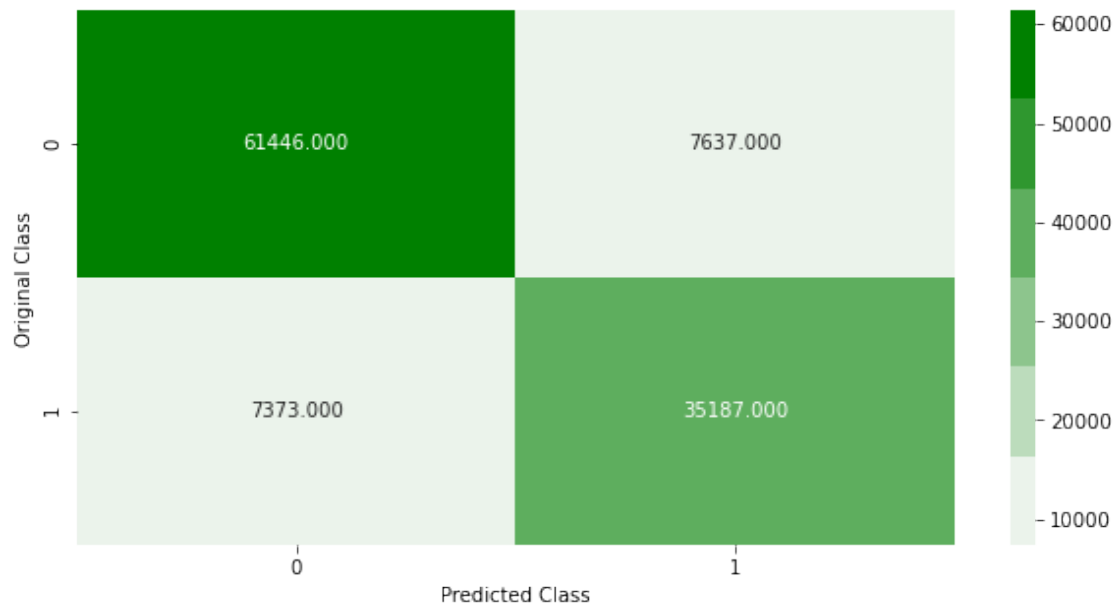


Sum of rows in precision matrix [1. 1.]

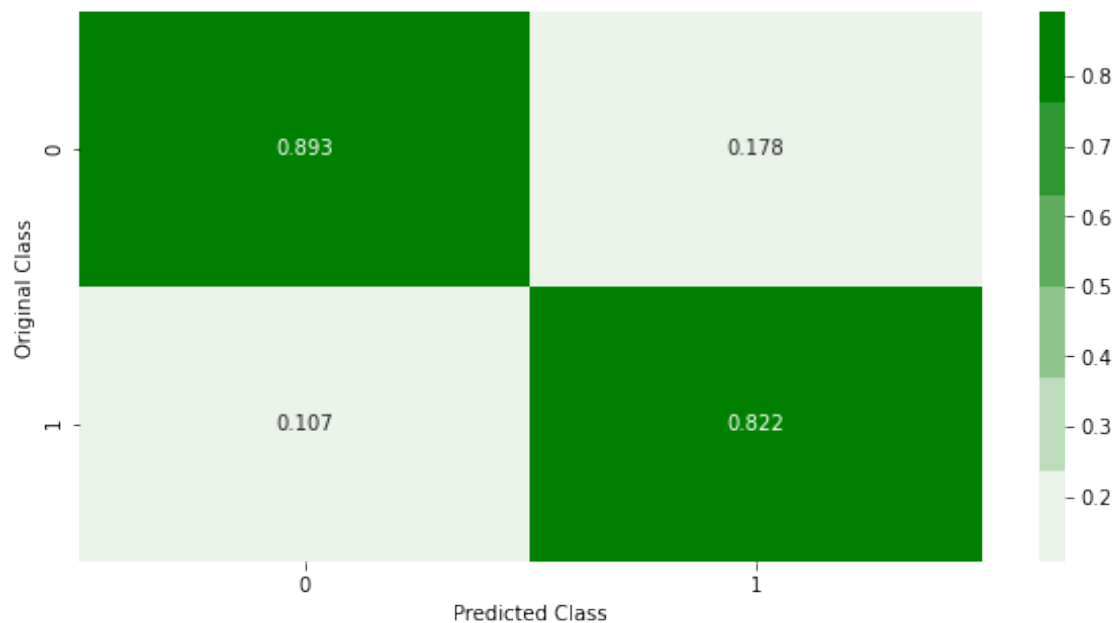
Confusion Matrix for Best Thresold for the CV Data

best train threshold : 0.539677800278026
The Weighted Recall Score: 0.8655535949410174
The Weighted Precision Score: 0.8657219565541188
The Weighted F1 Score: 0.8656327510145012

=====
=====
=====
----- Confusion matrix

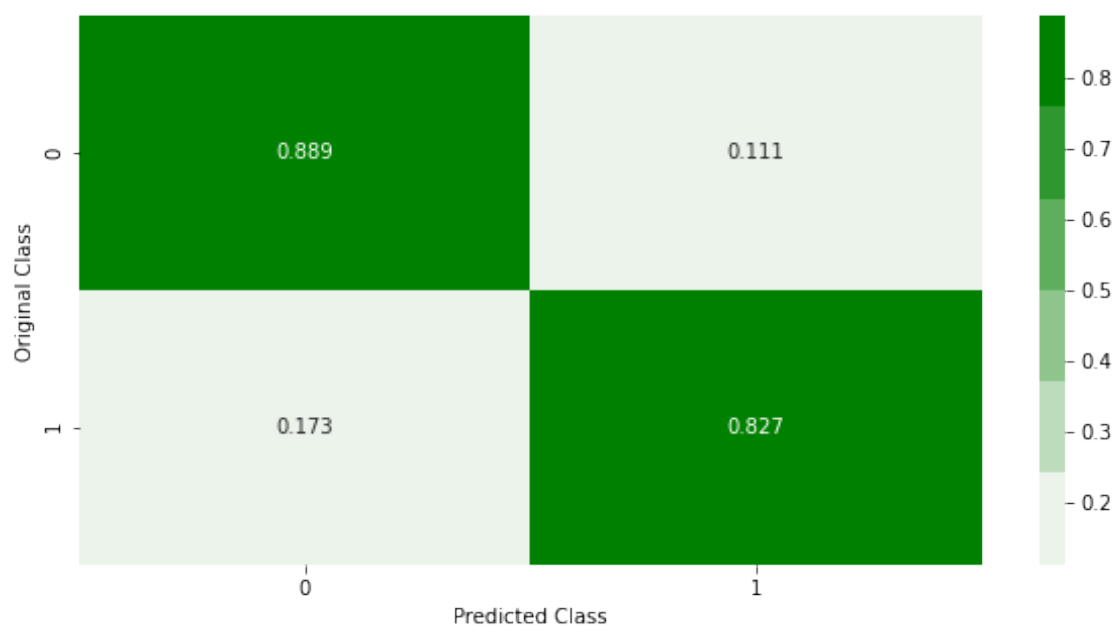


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

11.6 Observations on Experiment 3:

1. The performance of all the supervised models in experiment 3 has further decreased compared to experiments 1 and 2.
2. lightGBM out performed Random Forests and xgboost in experiment 3 there has not been any improvement in the scores compared to experiments 1 and 2.

12 Exprimment 4:

12.1 Using PCA in order to create a dataset of completely Independent Features to chcek for improvment in Model Performance

```
[ ]: from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA

pca= IncrementalPCA(n_components=32,batch_size=50)

train_pca= pca.fit_transform(train_x_smt)
cv_pca= pca.transform(cv_x_smt)

[ ]: with open('/home/megha_murthy_n/train_pca.pkl','wb') as tr_pca:
    pickle.dump(train_x_smt,tr_pca)
with open('/home/megha_murthy_n/cv_pca.pkl','wb') as cv_pca:
    pickle.dump(cv_x_smt,cv_pca)

[ ]: with open('/home/megha_murthy_n/train_pca.pkl','rb') as tr_pca:
    train_pca= pd.read_pickle(tr_pca)
with open('/home/megha_murthy_n/cv_pca.pkl','rb') as cv_pca:
    cv_pca= pd.read_pickle(cv_pca)
```

12.2 Random Forests post performing PCA

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf_best =_
    ↳RandomForestClassifier(max_depth=35,min_samples_split=100,criterion='gini',n_estimators=150)
rf_best.fit(train_pca,train_y_smt)

sig_clf = CalibratedClassifierCV(rf_best, method="sigmoid")
sig_clf.fit(train_pca,train_y_smt)

train_y_pred = sig_clf.predict_proba(train_pca)
train_y_pred = train_y_pred[:,1]
cv_y_pred = sig_clf.predict_proba(cv_pca)
cv_y_pred = cv_y_pred[:,1]

[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y_smt, train_y_pred)
```

```
print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y_smt, cv_y_pred)
```

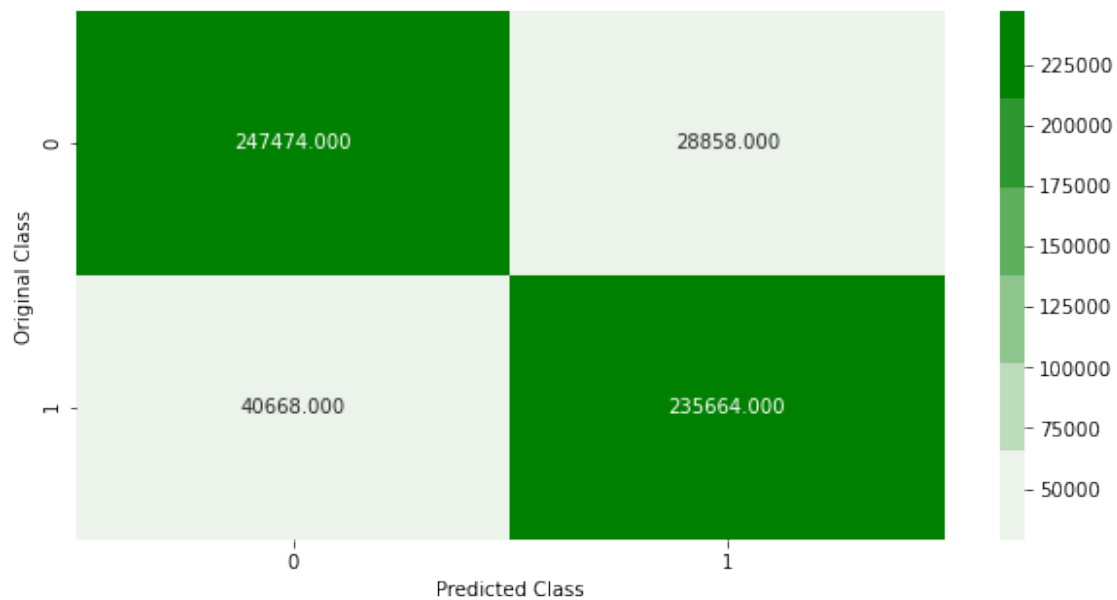
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8741984279779396

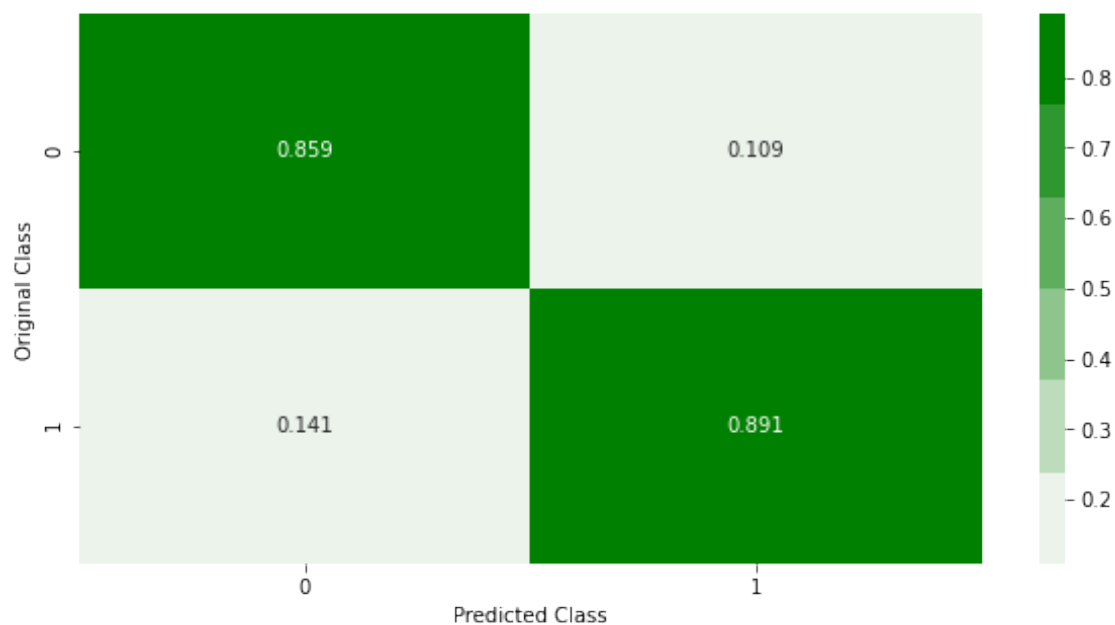
The Weighted Precision Score: 0.8748831801049444

The Weighted F1 Score: 0.8741409552416481

```
=====
=====
=====
----- Confusion matrix
-----
```

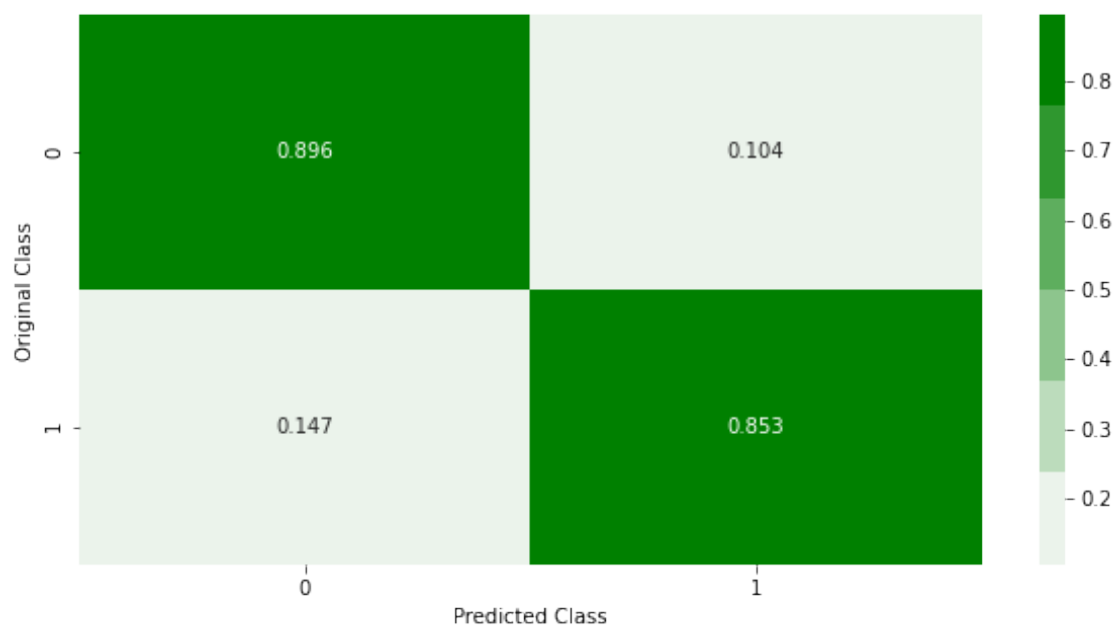


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.8011884255171315
The Weighted Precision Score: 0.8067195598677002
The Weighted F1 Score: 0.8002880643304527

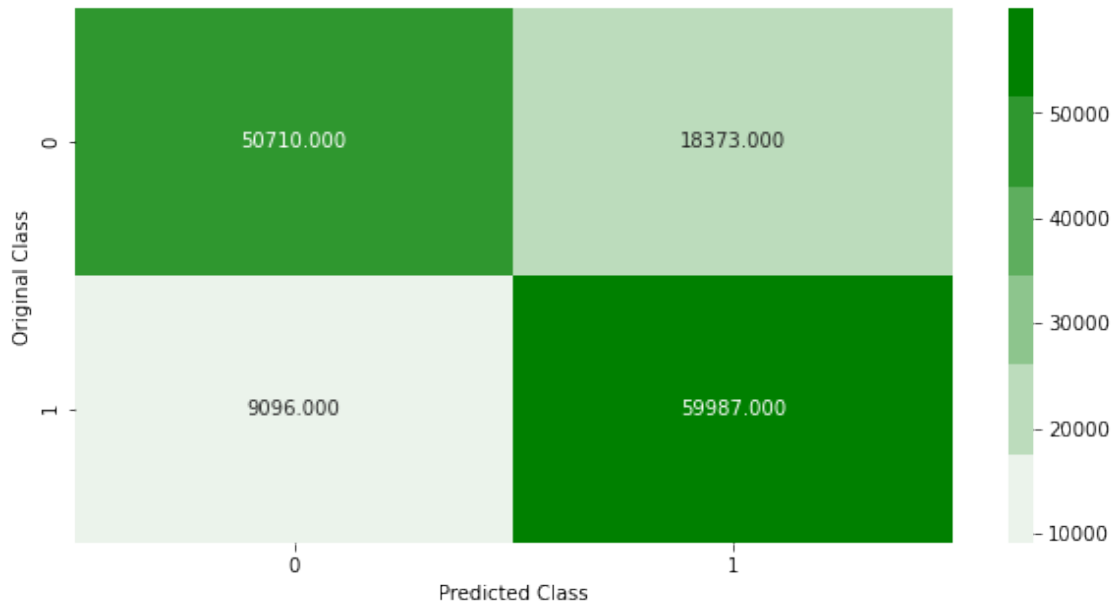
=====

=====

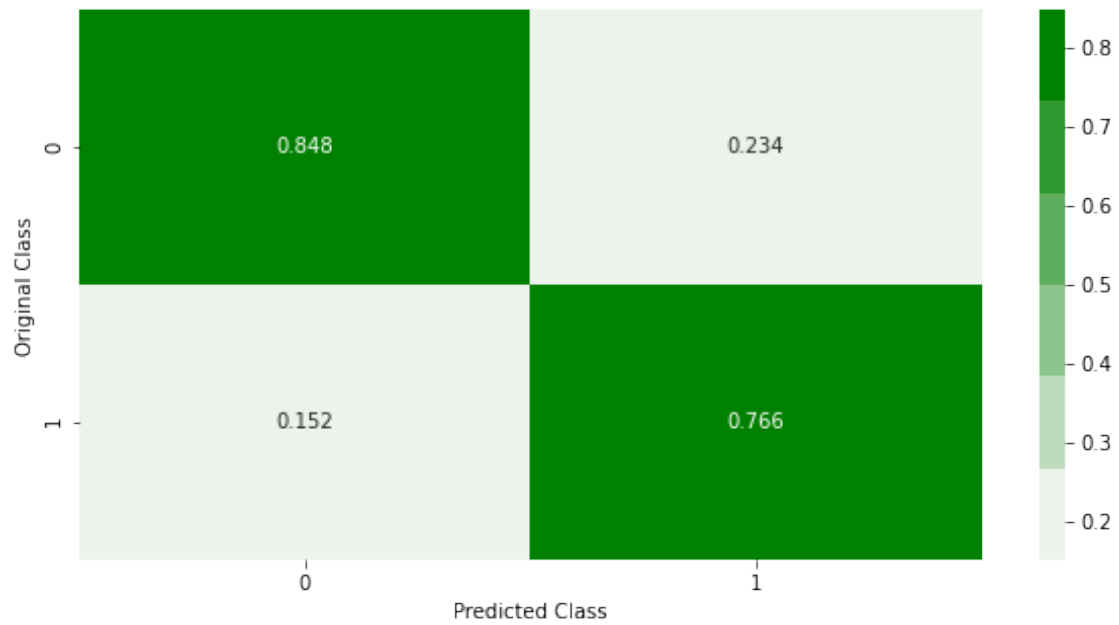
=====

=====

----- Confusion matrix

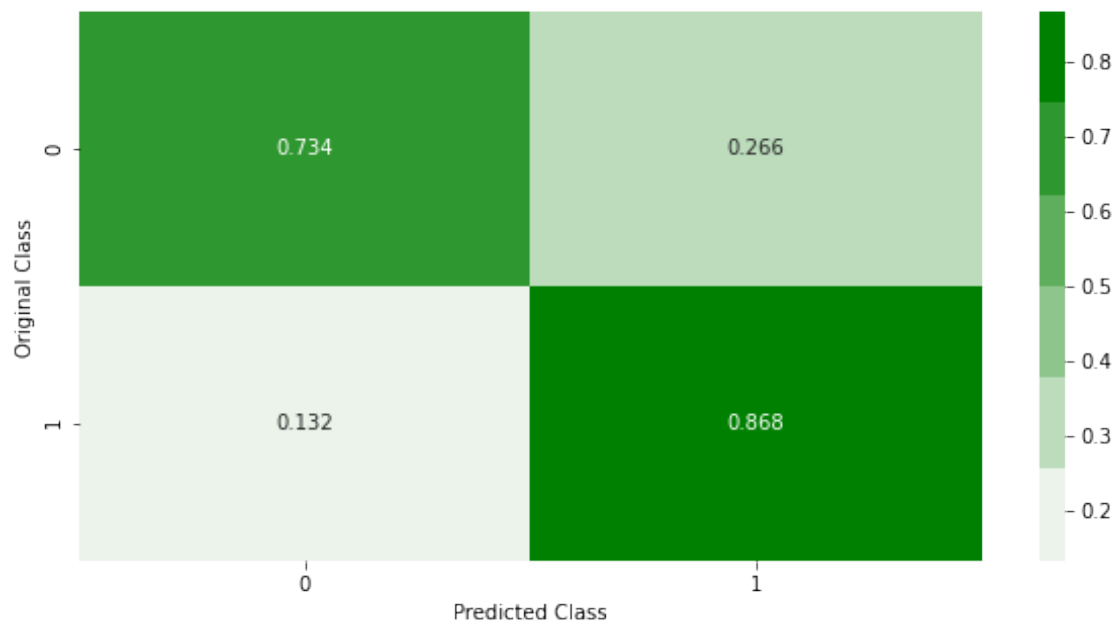


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_smt,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y_smt,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

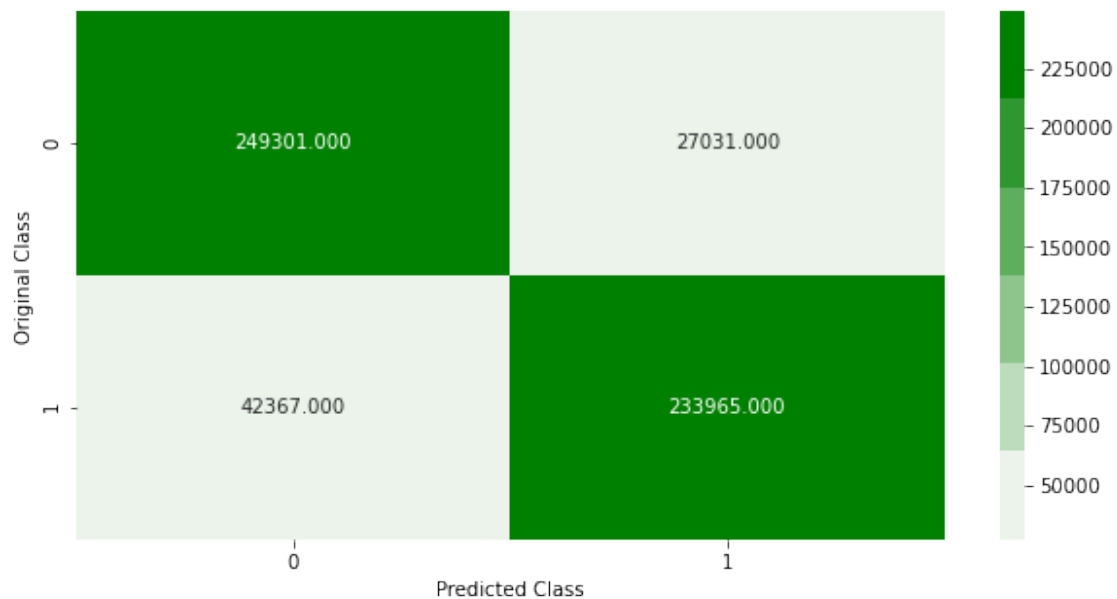
best train threshold : 0.5133249045110312

The Weighted Recall Score: 0.8744300334380383

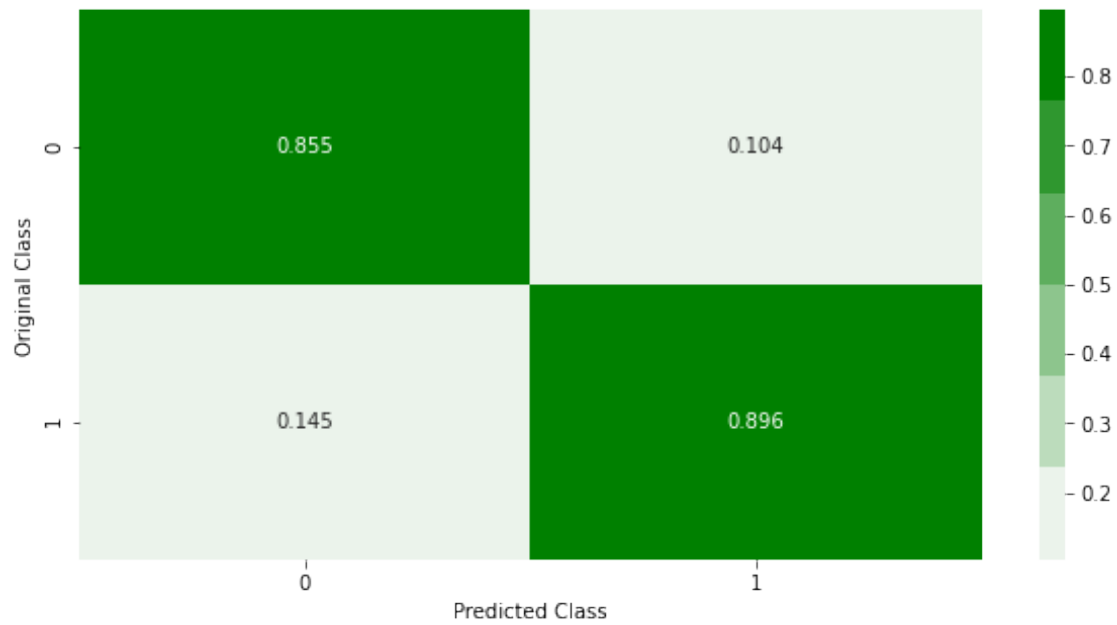
The Weighted Precision Score: 0.8755868706266607

The Weighted F1 Score: 0.8743332675758569

```
=====
=====
=====
----- Confusion matrix
-----
```

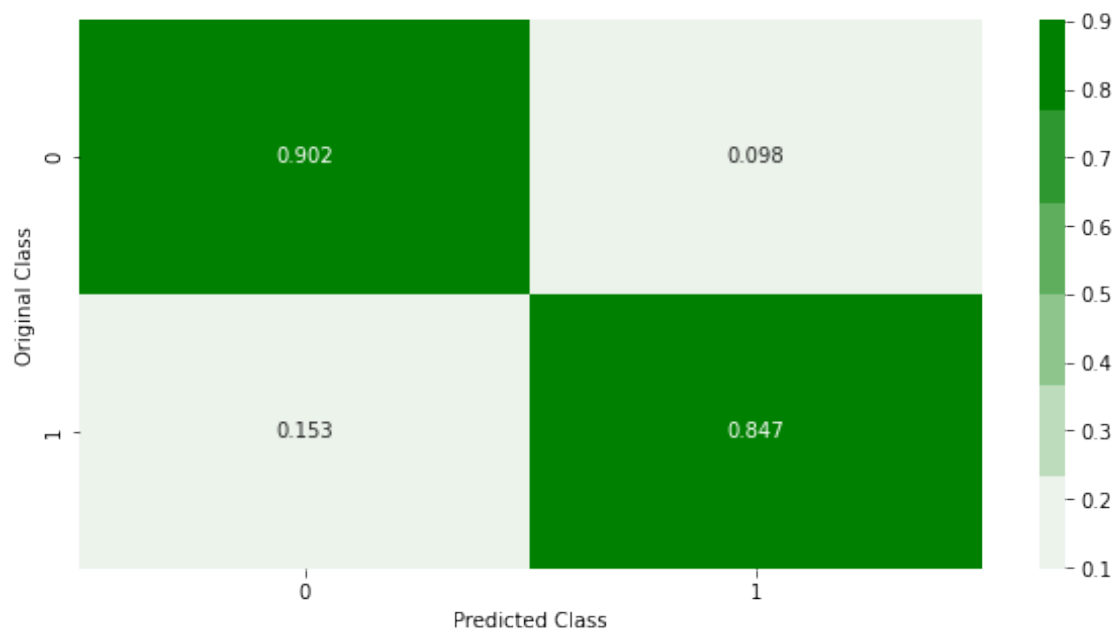


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



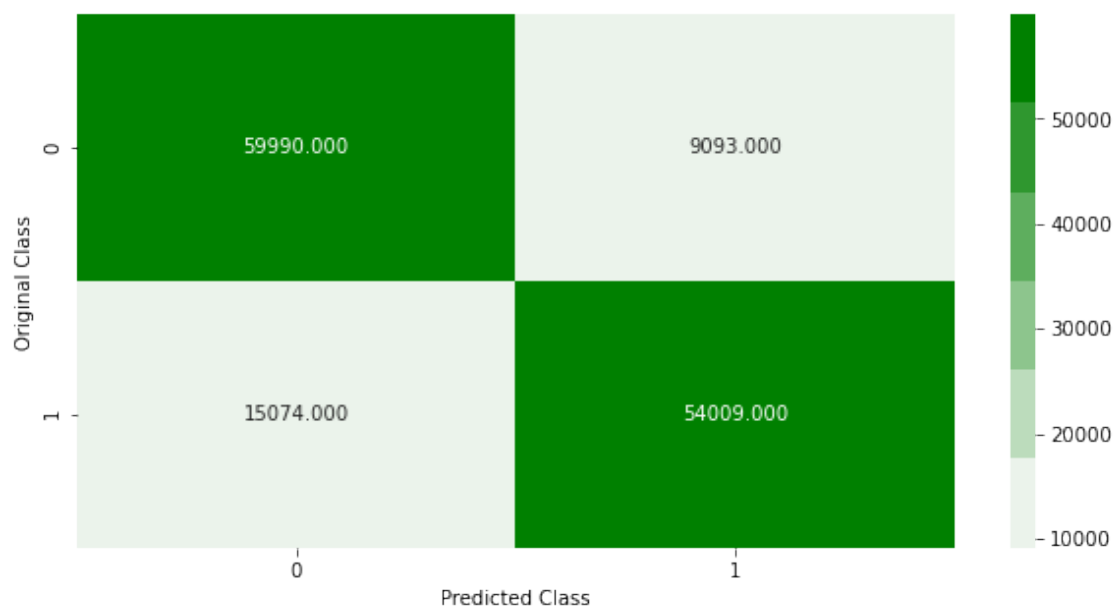
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.6443011761864068
The Weighted Recall Score: 0.8250872139310684
The Weighted Precision Score: 0.8275423336005343
The Weighted F1 Score: 0.8247588303923098

=====

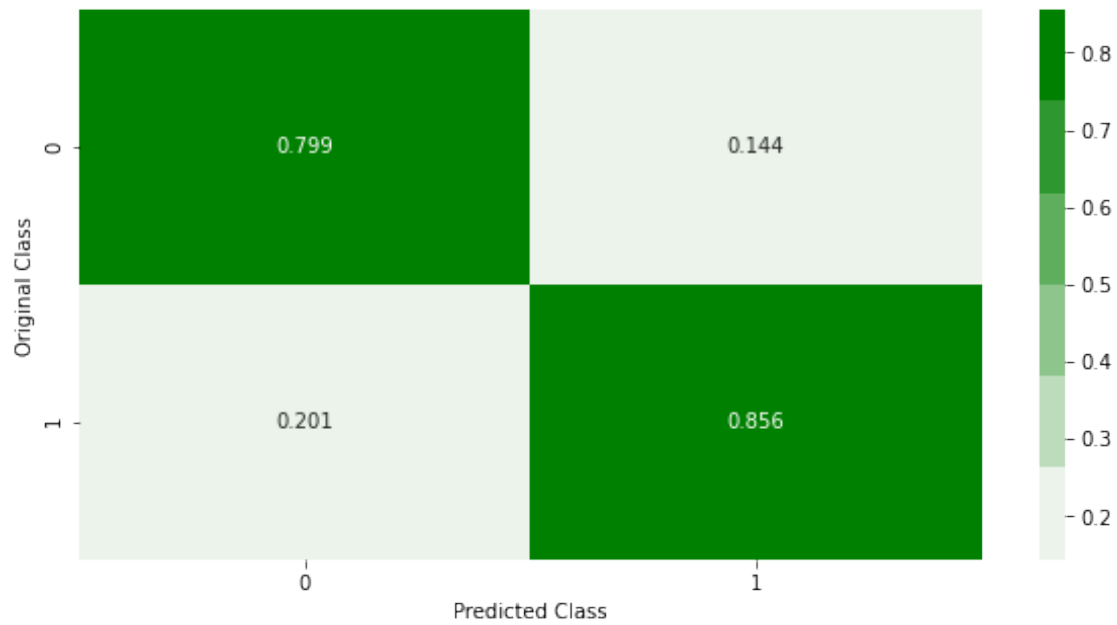
=====

=====

----- Confusion matrix

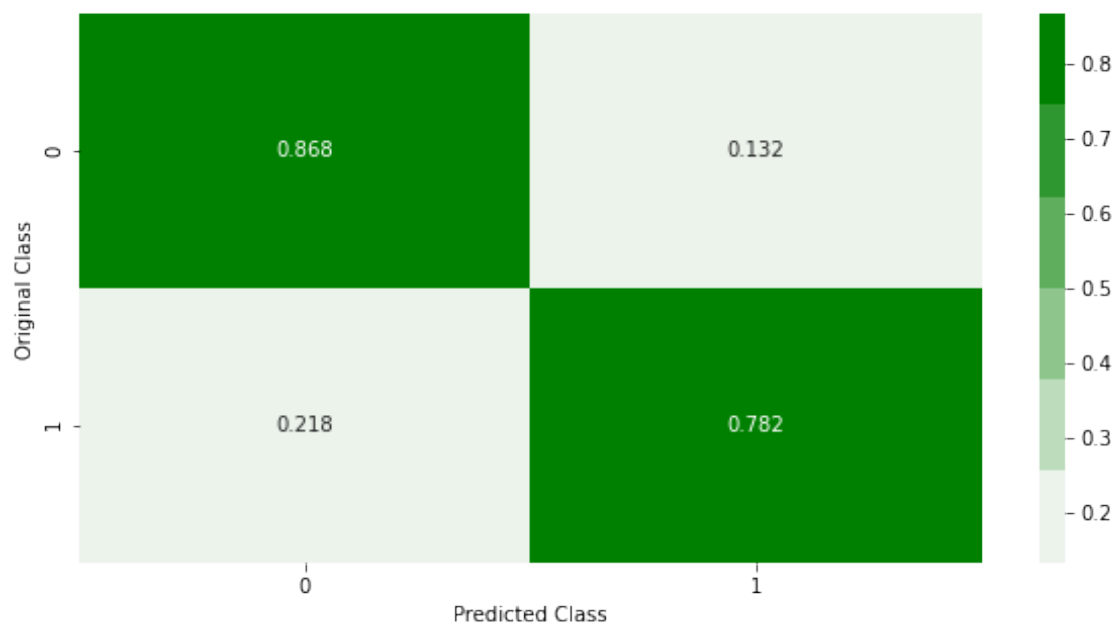


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

12.3 lightGBM Classifier post performing PCA

```
[ ]: import lightgbm as lgb
      clf = lgb.LGBMClassifier()
      clf.fit(train_pca,train_y_smt)

[ ]: LGBMClassifier()

[ ]: train_y_pred= clf.predict_proba(train_pca)
      train_y_pred= train_y_pred[:,1]

      cv_y_pred= clf.predict_proba(cv_pca)
      cv_y_pred= cv_y_pred[:,1]

[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y_smt, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y_smt, cv_y_pred)
```

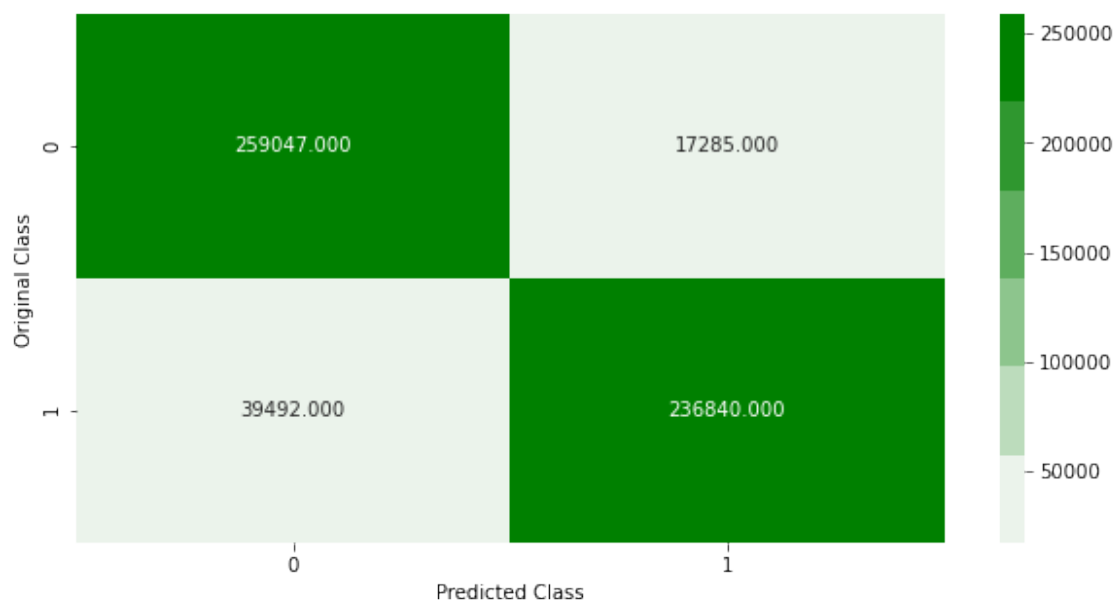
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8972666936873037

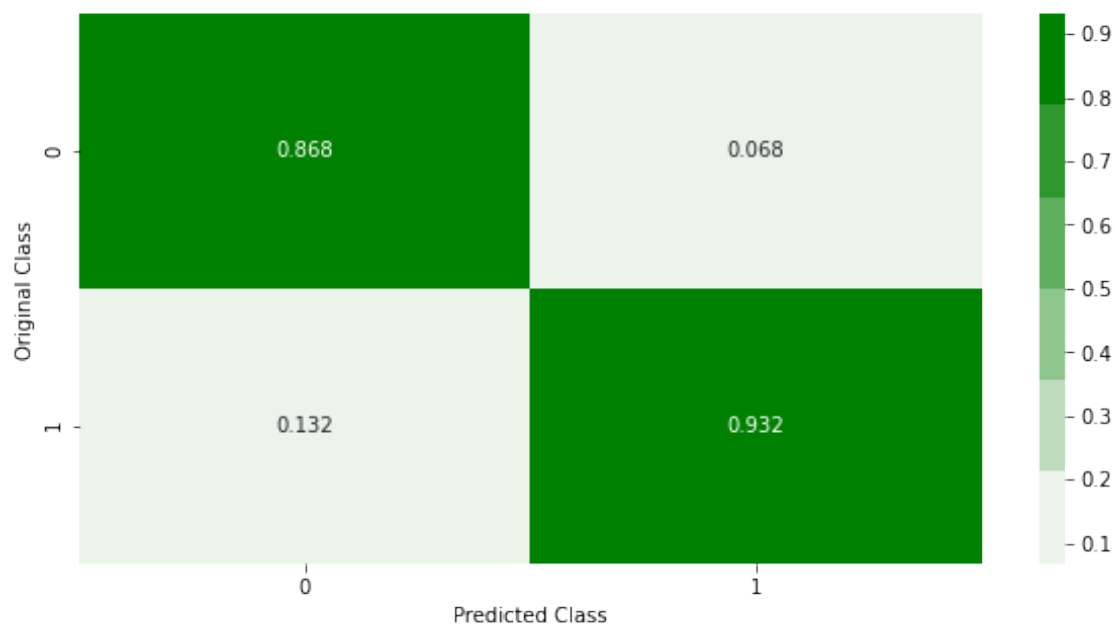
The Weighted Precision Score: 0.8998490340036648

The Weighted F1 Score: 0.8971005551184618

```
=====
=====
=====
----- Confusion matrix
-----
```

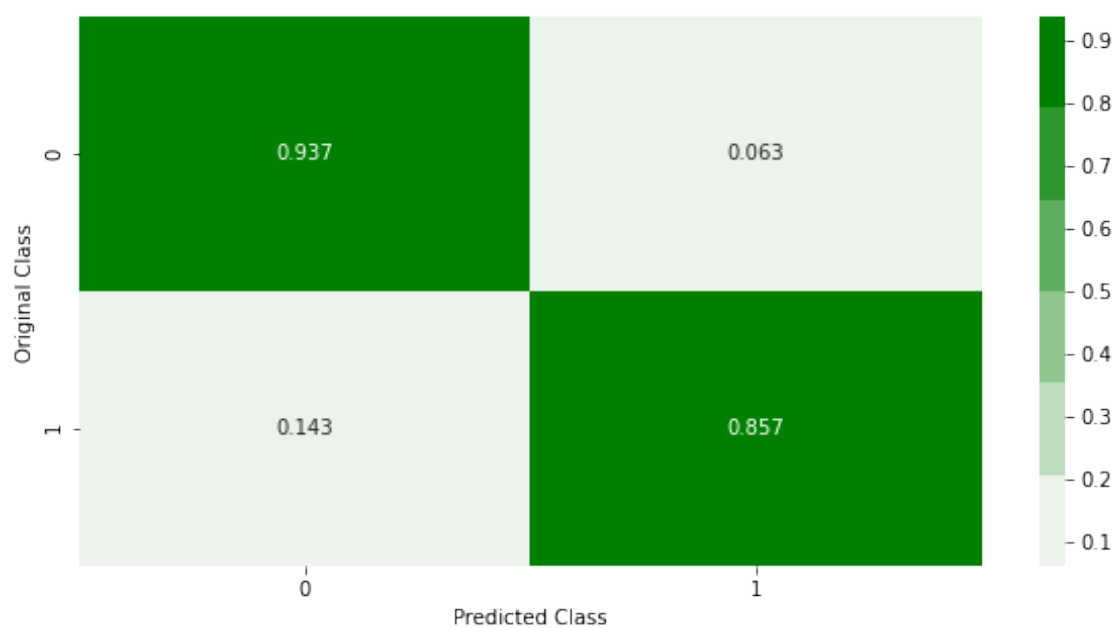


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



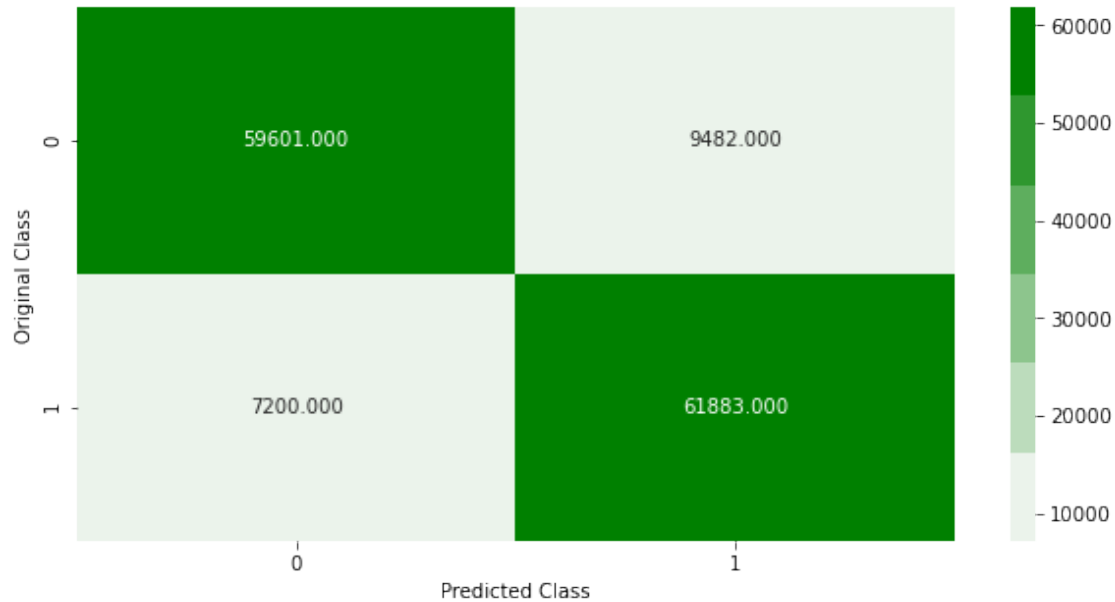
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.879261178582285
The Weighted Precision Score: 0.879675465707076
The Weighted F1 Score: 0.8792282332167454

=====

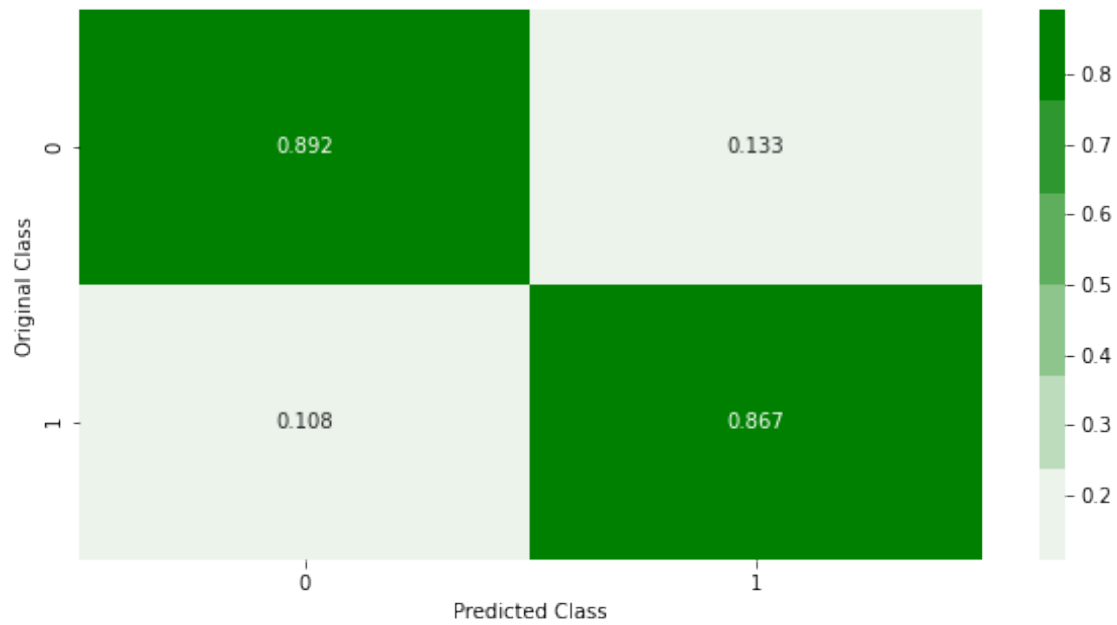
=====

=====

----- Confusion matrix

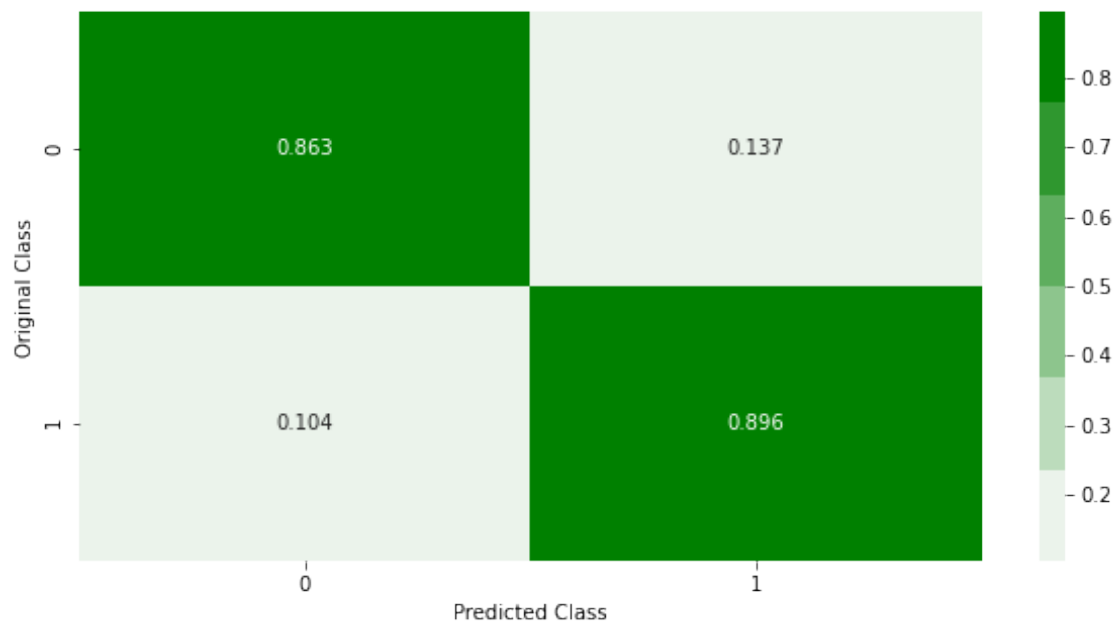


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_smt,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y_smt,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

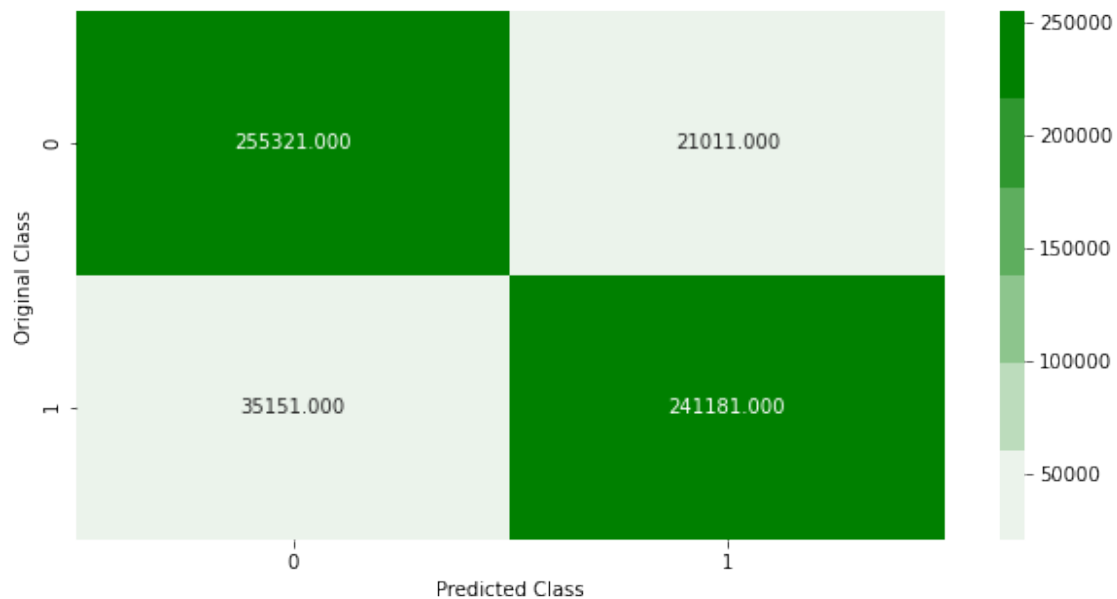
best train threshold : 0.4728681458144896

The Weighted Recall Score: 0.8983794855463717

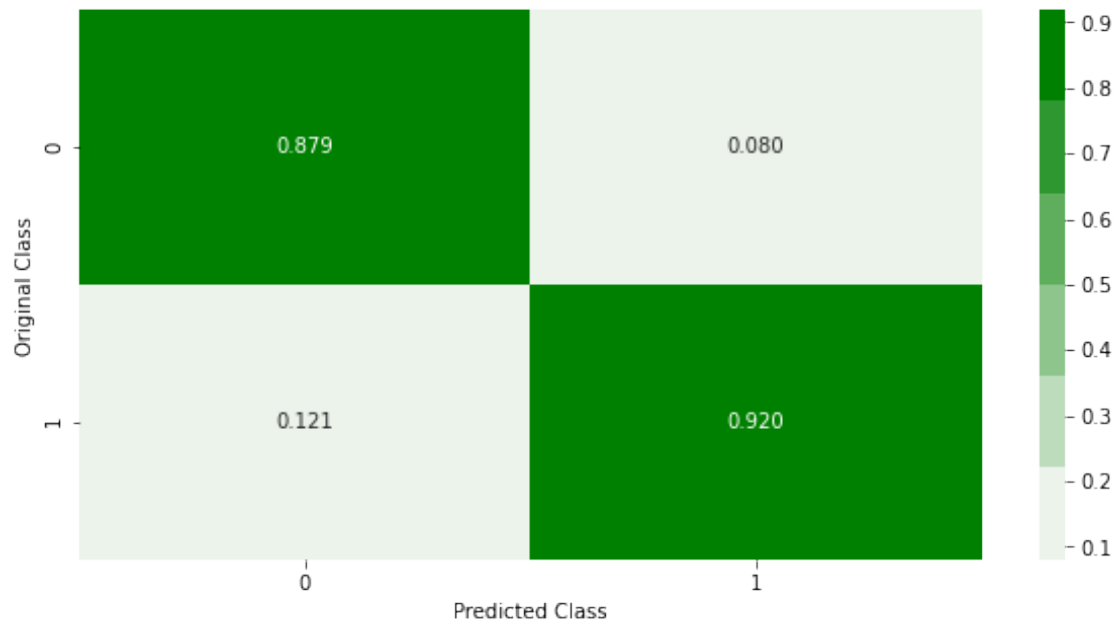
The Weighted Precision Score: 0.8994253419840094

The Weighted F1 Score: 0.8983129211131099

```
=====
=====
=====
----- Confusion matrix
-----
```

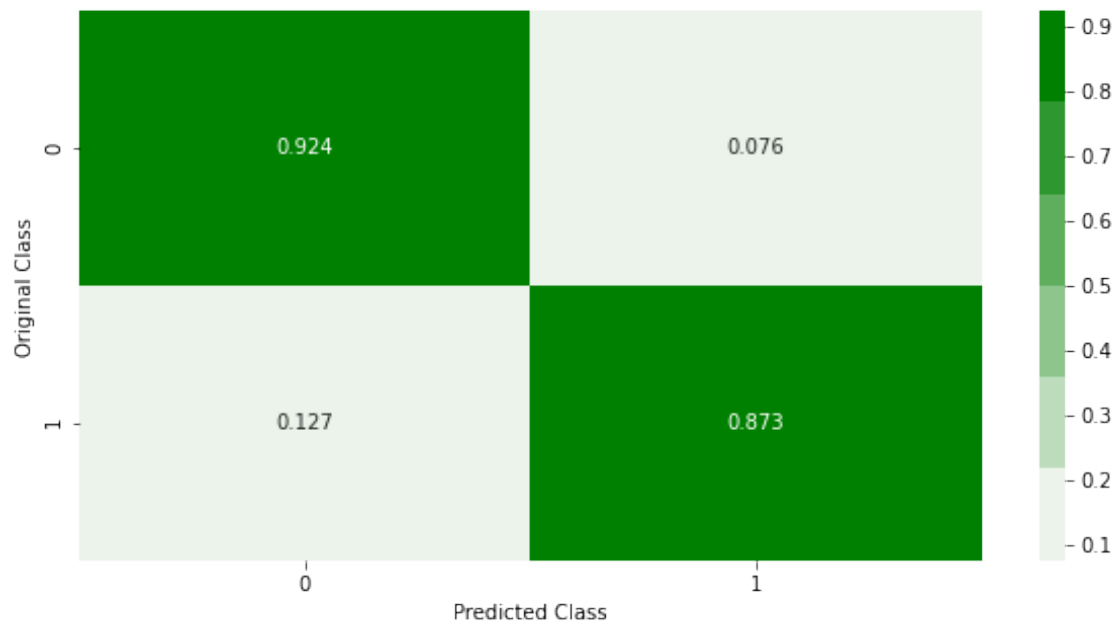


```
----- Precision matrix
-----
```



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Thresold for the CV Data

best train threshold : 0.5713607369482622
The Weighted Recall Score: 0.8883299798792756
The Weighted Precision Score: 0.890430384859955
The Weighted F1 Score: 0.888179589338091

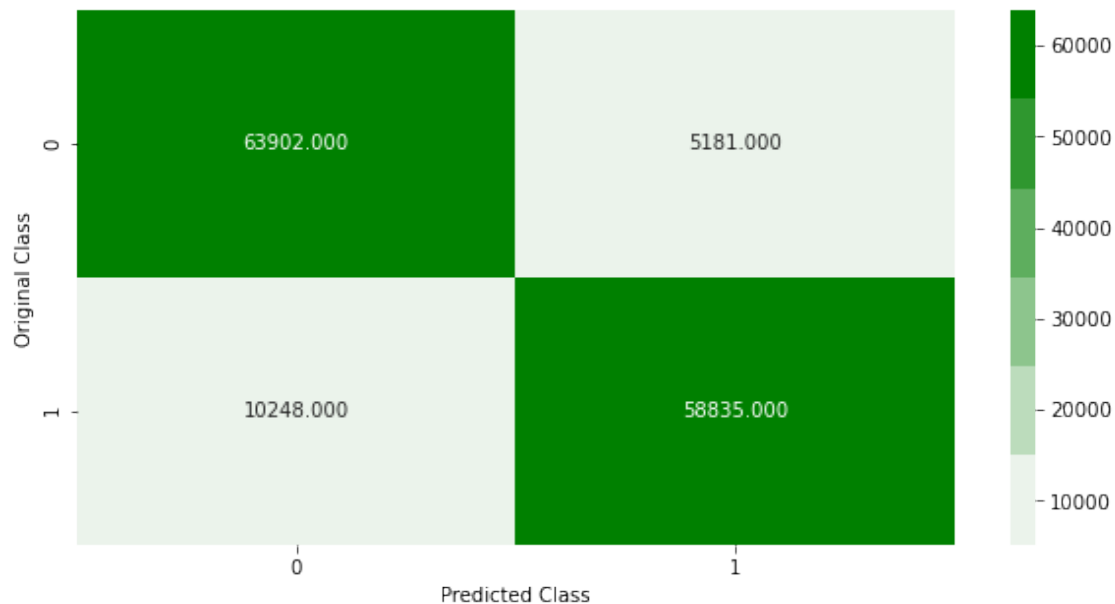
=====

=====

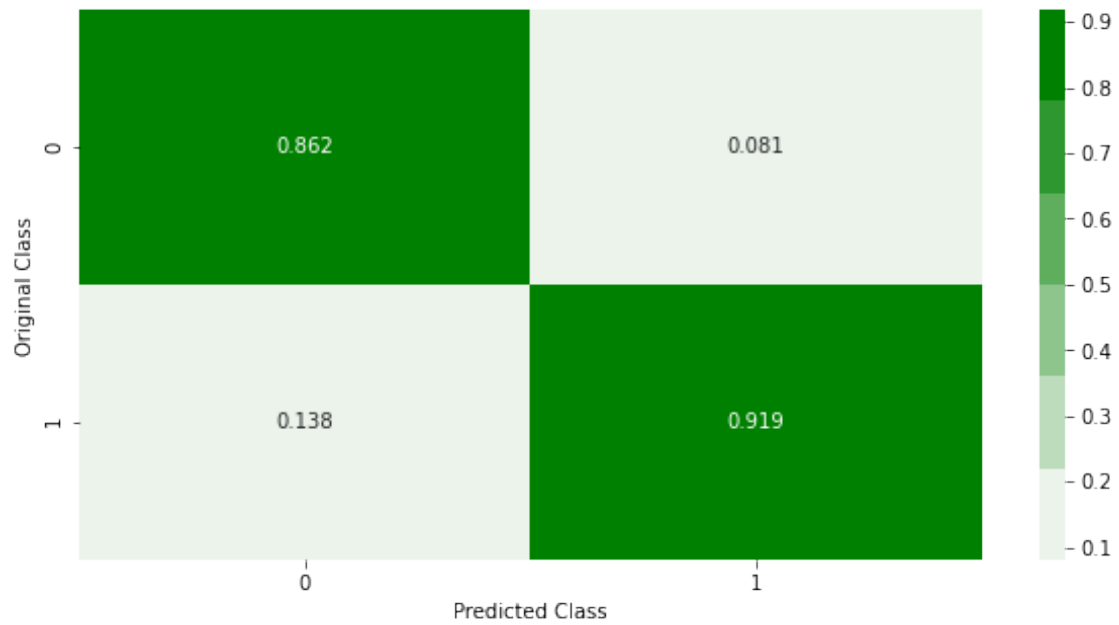
=====

=====

----- Confusion matrix

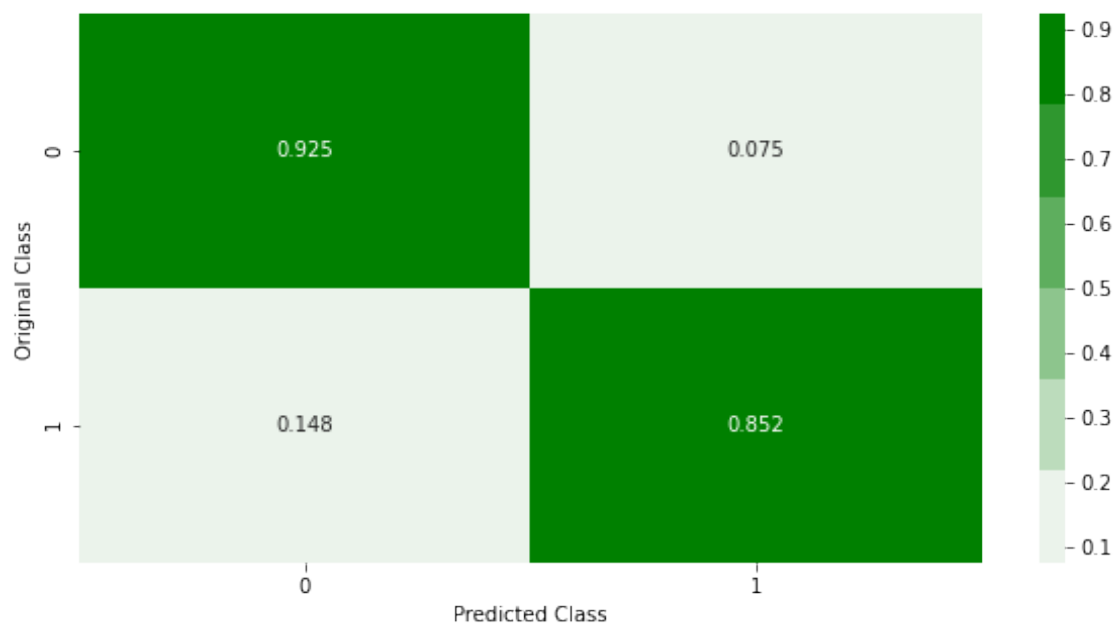


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

12.4 Experiment 5

12.4.1 Using SMOTE Undersampling technique in order to create a dataset corrected for Data imbalance

```
[ ]: from imblearn.under_sampling import RandomUnderSampler
und_sampler= RandomUnderSampler(sampling_strategy='majority')
train_x_us,train_y_us= und_sampler.fit_resample(train_fin4,train_y)
```

```
[ ]: print(train_x_us.shape)
print(cv_fin4.shape)
print(train_y_us.shape)
print(cv_y.shape)
```

```
(340472, 32)
(111643, 32)
(340472,)
(111643,)
```

```
[ ]: print(train_y_us.value_counts())
#print(cv_y_us.value_counts())
```

```
0    170236
1    170236
Name: PotentialFraud, dtype: int64
```

```
[ ]: with open('/home/megha_murthy_n/train_x_us.pkl','wb') as tr_us_x:
    pickle.dump(train_x_us,tr_us_x)
with open('/home/megha_murthy_n/train_y_us.pkl','wb') as tr_us_y:
    pickle.dump(train_y_us,tr_us_y)
```

```
[ ]: with open('/home/megha_murthy_n/train_x_us.pkl','rb') as tr_us_x:
    train_x_us= pd.read_pickle(tr_us_x)
with open('/home/megha_murthy_n/train_y_us.pkl','rb') as tr_us_y:
    train_y_us= pd.read_pickle(tr_us_y)
```

```
[ ]: print(train_x_us.shape)
print(cv_fin32.shape)
print(train_y_us.shape)
print(cv_y.shape)
```

```
(340472, 32)
(111643, 32)
(340472,)
(111643,)
```

12.5 Random Forests post performing UnderSampling

```
[ ]: from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(class_weight='balanced')
param = {'n_estimators': [100,180,250], 'max_depth' : [
    ↳[25,35,40], 'min_samples_split': [100,150], 'criterion' : ['gini']]

rf_tune = RandomizedSearchCV(rf,param,cv=5,n_jobs=-1,verbose=1)
rf_tune.fit(train_x_us, train_y_us)
print('best parameter : ',rf_tune.best_params_)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
best parameter : {'n_estimators': 250, 'min_samples_split': 100, 'max_depth': 40, 'criterion': 'gini'}

```
[ ]:
[ ]: from sklearn.ensemble import RandomForestClassifier
rf_best = [
    ↳RandomForestClassifier(max_depth=40,min_samples_split=100,criterion='gini',n_estimators=250)
rf_best.fit(train_x_us,train_y_us)

#sig_clf = CalibratedClassifierCV(rf_best, method="sigmoid")
#sig_clf.fit(train_x_us,train_y_us)

train_y_pred = rf_best.predict_proba(train_x_us)
train_y_pred = train_y_pred[:,1]
cv_y_pred = rf_best.predict_proba(cv_fin32)
cv_y_pred = cv_y_pred[:,1]

[ ]: print("Confusion Matrix for the Train Data")
plot_confusion_matrix(train_y_us, train_y_pred)

print("Confusion Matrix for the Cross Validate Data")
plot_confusion_matrix(cv_y, cv_y_pred)
```

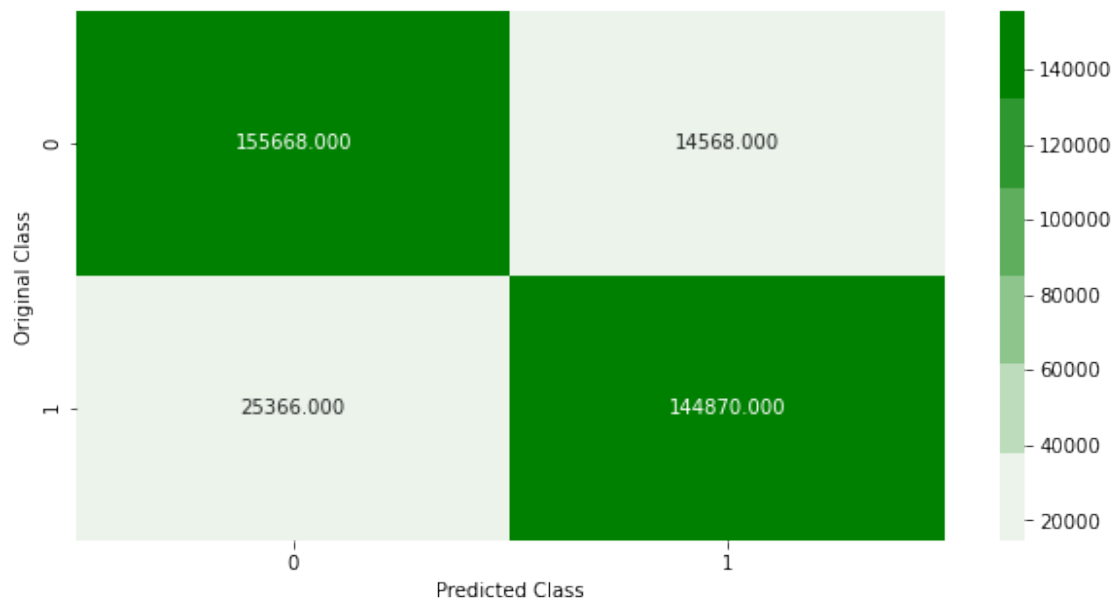
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8827098851006837

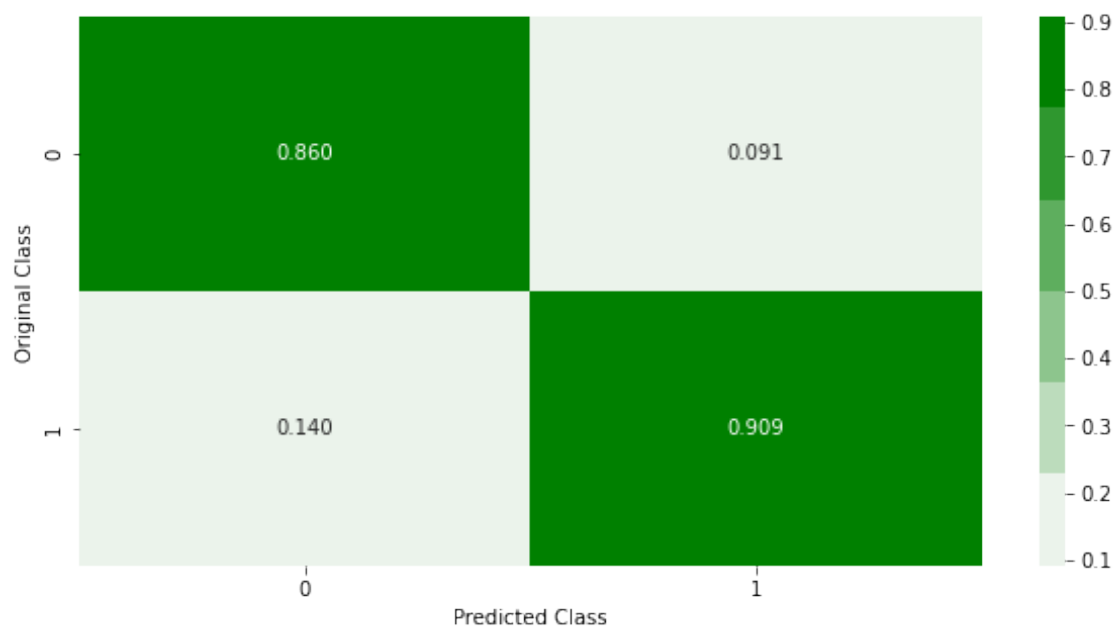
The Weighted Precision Score: 0.8842558667722954

The Weighted F1 Score: 0.882591792605276

```
=====
=====
=====
----- Confusion matrix
-----
```

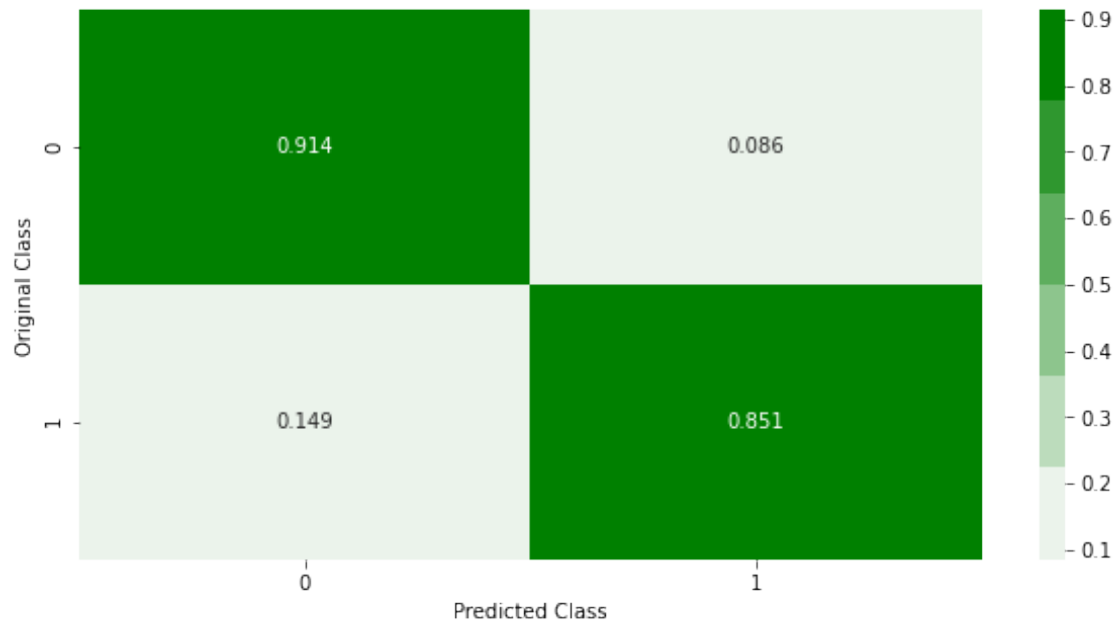


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

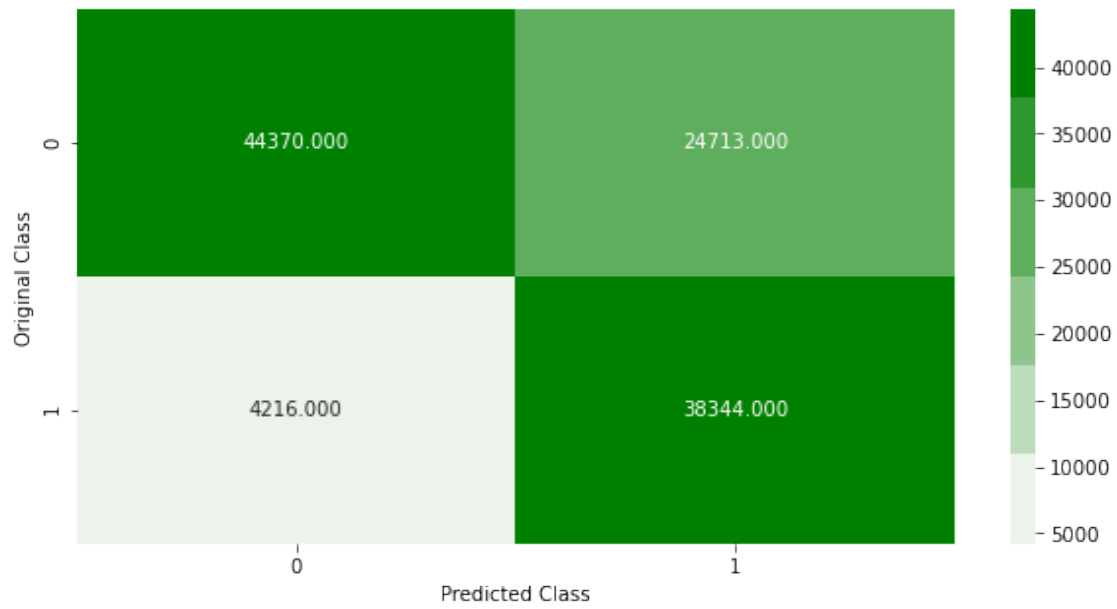
Confusion Matrix for the Cross Validate Data

The Weighted Recall Score: 0.7408794102630707

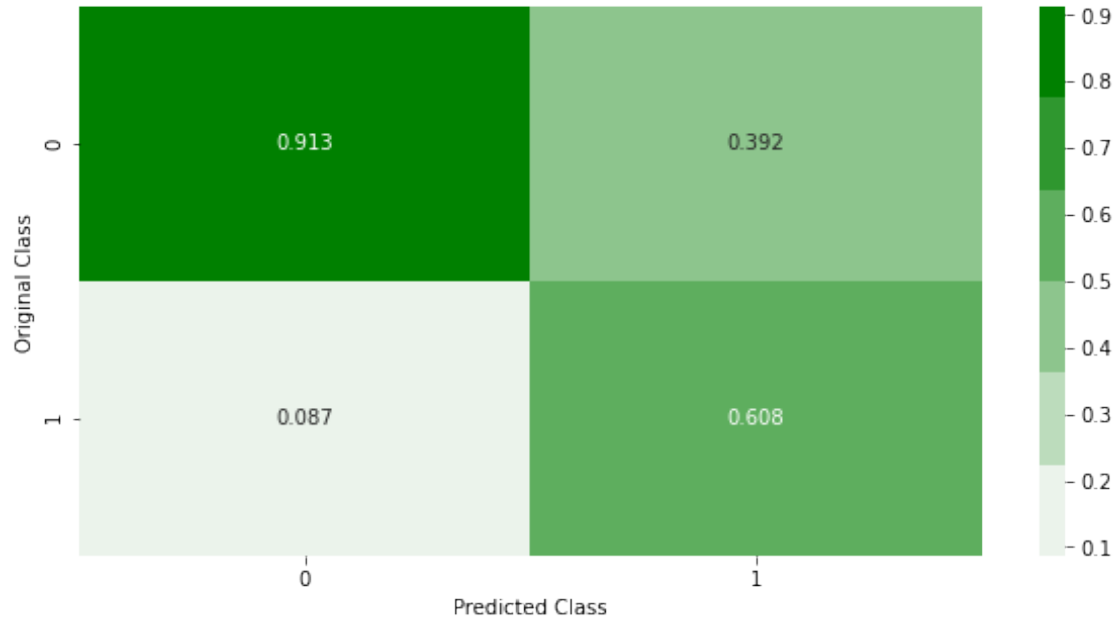
The Weighted Precision Score: 0.7969015602169743

The Weighted F1 Score: 0.7434546962329205

```
=====
=====
----- Confusion matrix
-----
```

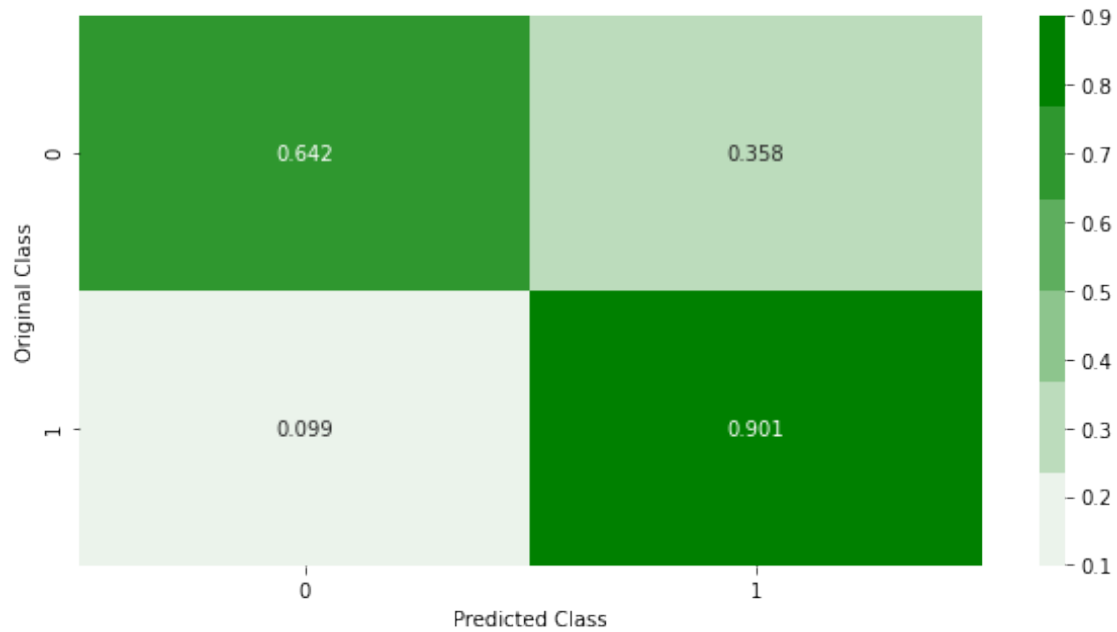


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



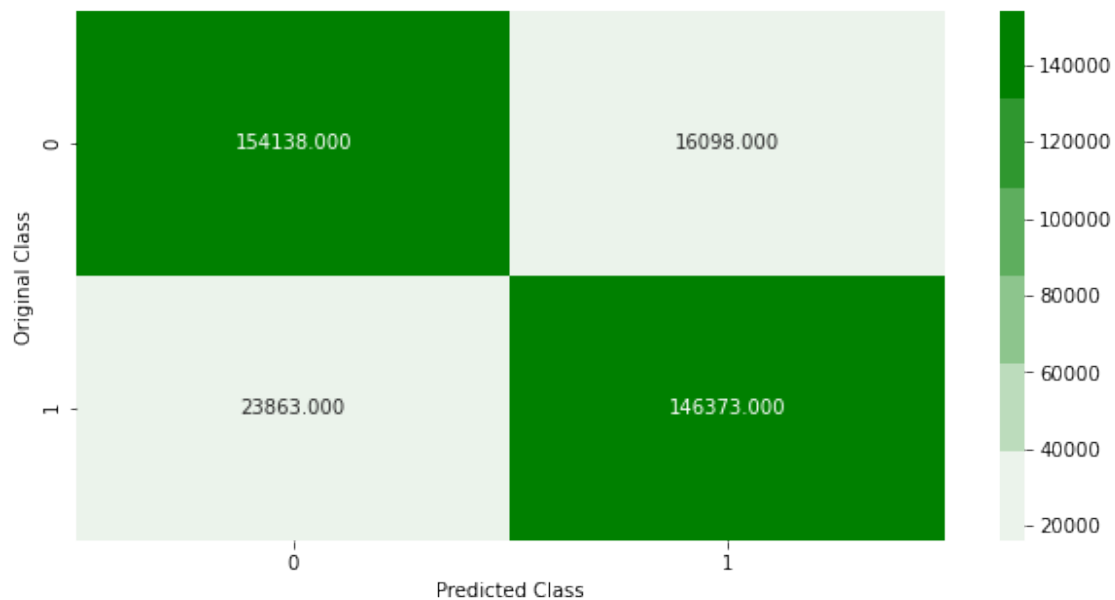
Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_us,train_y_pred)

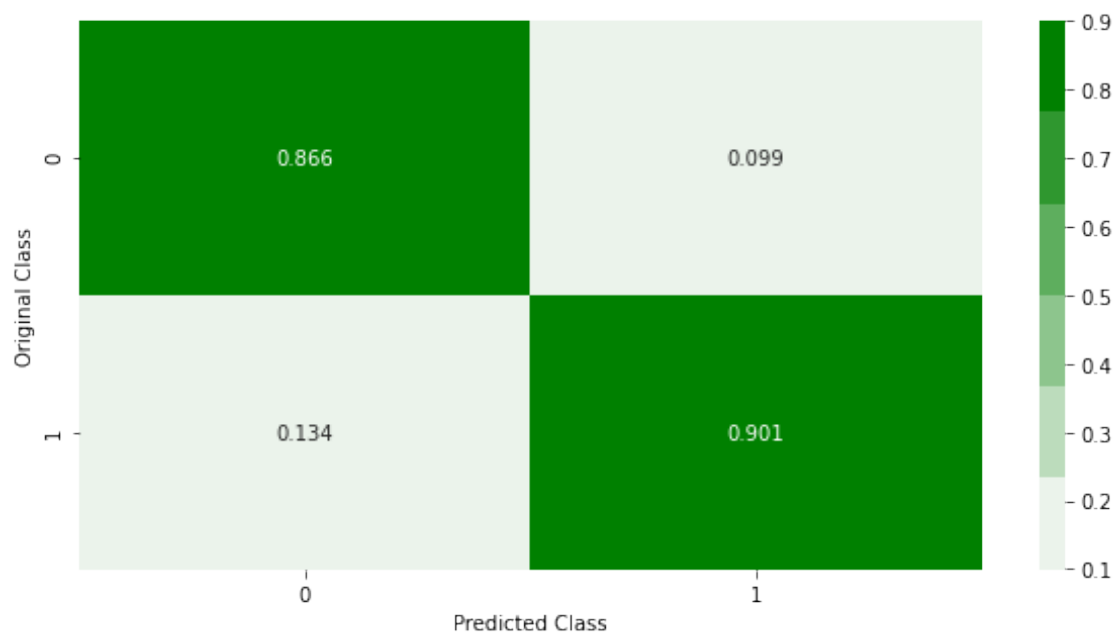
      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

```
Confusion Matrix for Best Thresold for the Train Data
best train threshold : 0.49182137275008136
The Weighted Recall Score: 0.8826305834253625
The Weighted Precision Score: 0.8834283288176092
The Weighted F1 Score: 0.8825695031459896
```

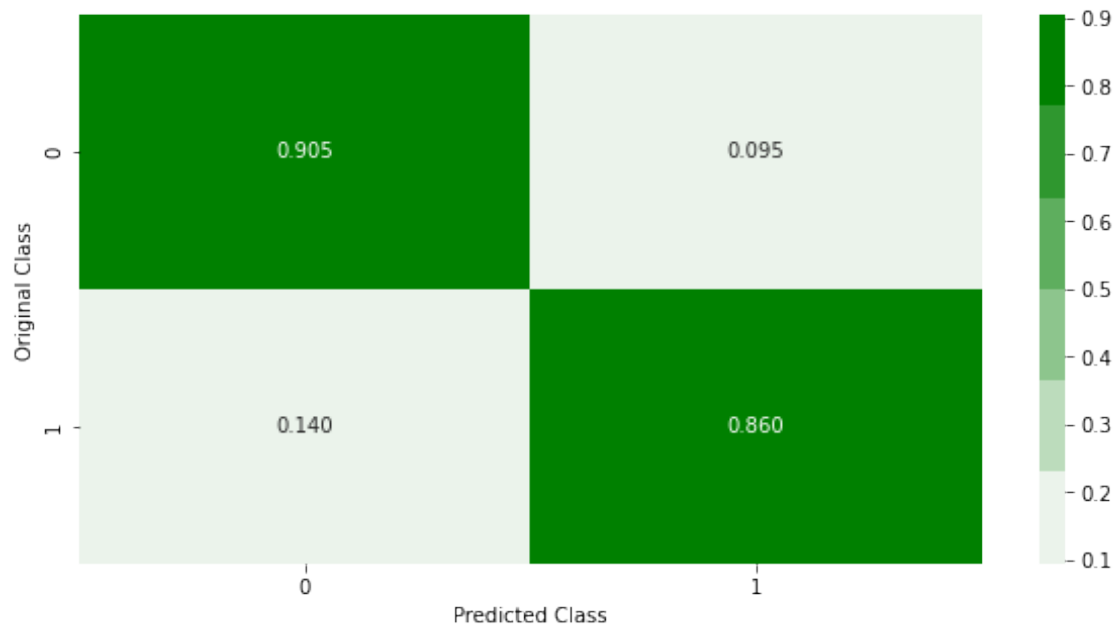
```
=====
=====
=====
=====
----- Confusion matrix
-----
```



----- Precision matrix



Sum of columns in precision matrix [1. 1.]
 ----- Recall matrix



Sum of rows in precision matrix [1. 1.]

Confusion Matrix for Best Thresold for the CV Data

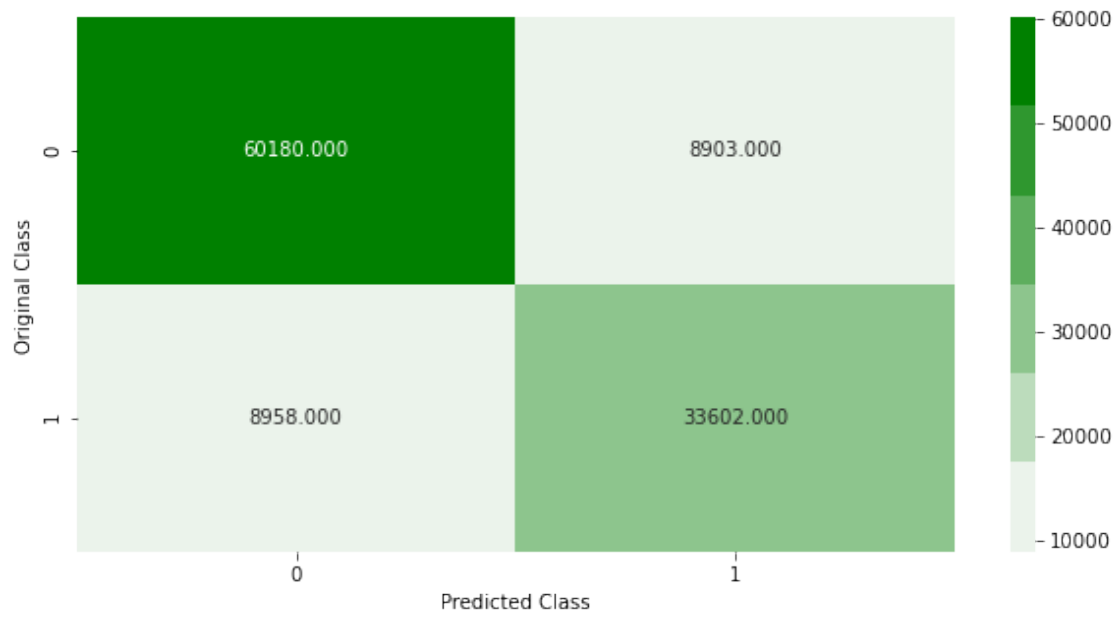
best train threshold : 0.5800254404741126

The Weighted Recall Score: 0.8400168393898408

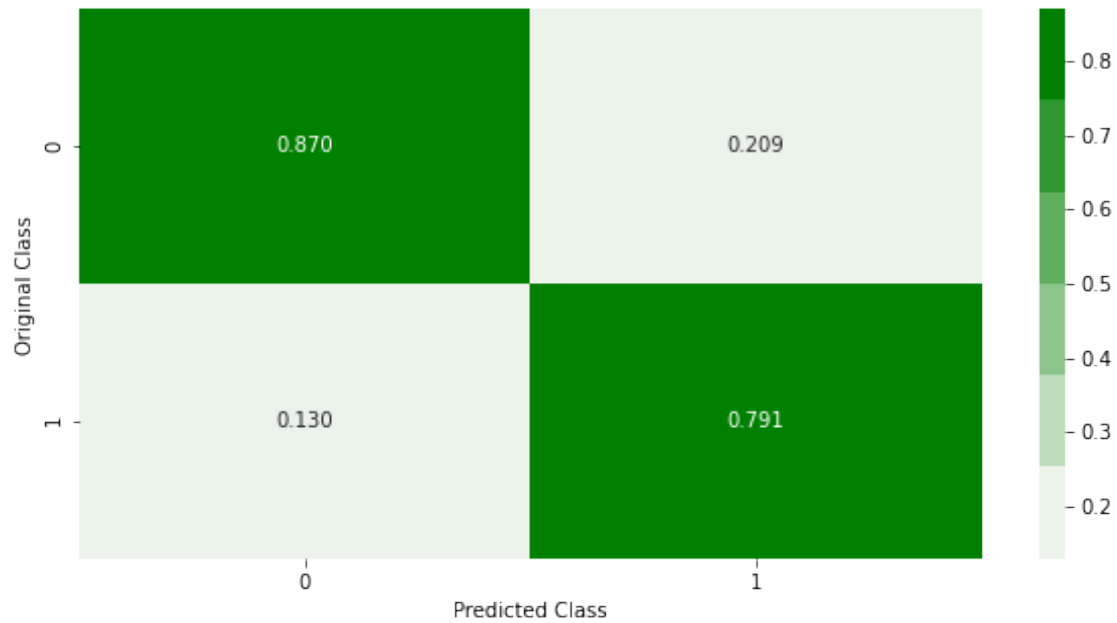
The Weighted Precision Score: 0.8399774818691778

The Weighted F1 Score: 0.8399969494109606

```
=====
=====
=====
----- Confusion matrix
-----
```

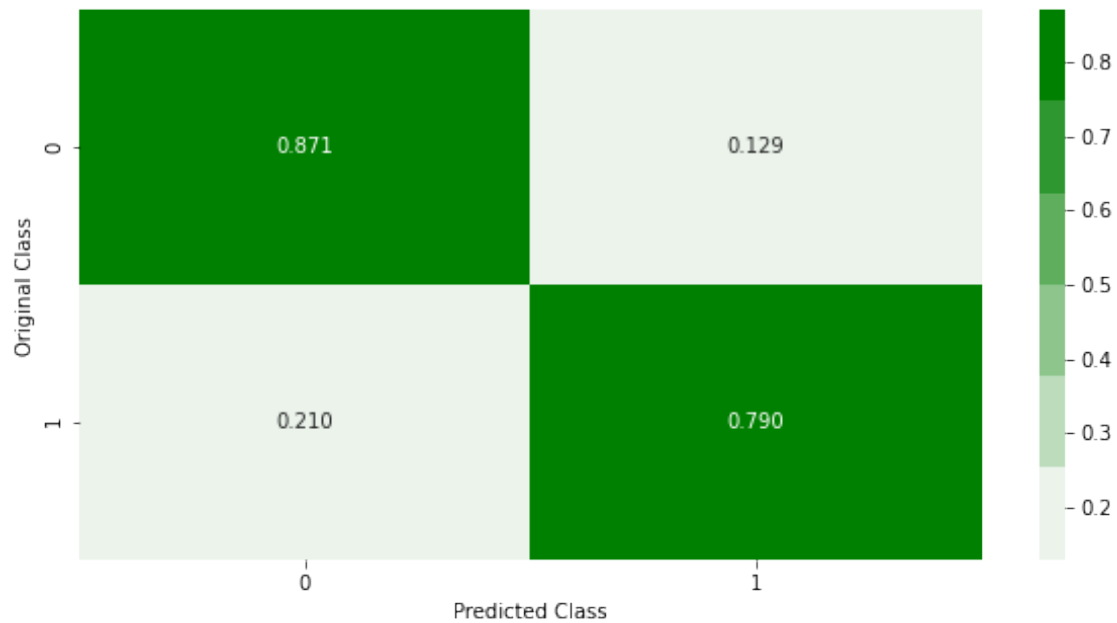


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

12.6 lightGBM post performing UnderSampling

```
[ ]: import lightgbm as lgb
      clf = lgb.LGBMClassifier()
      clf.fit(train_x_us,train_y_us)

[ ]: LGBMClassifier()

[ ]: train_y_pred= clf.predict_proba(train_x_us)
      train_y_pred= train_y_pred[:,1]

      cv_y_pred= clf.predict_proba(cv_fin32)
      cv_y_pred= cv_y_pred[:,1]

[ ]: print(len(train_y_pred))
      print(len(cv_y_pred))

340472
85120

[ ]: print(train_y_us.shape)
      print(cv_y_us.shape)

(340472,)
(138166,)
```

```
[ ]: print("Confusion Matrix for the Train Data")
      plot_confusion_matrix(train_y_us, train_y_pred)

      print("Confusion Matrix for the Cross Validate Data")
      plot_confusion_matrix(cv_y, cv_y_pred)
```

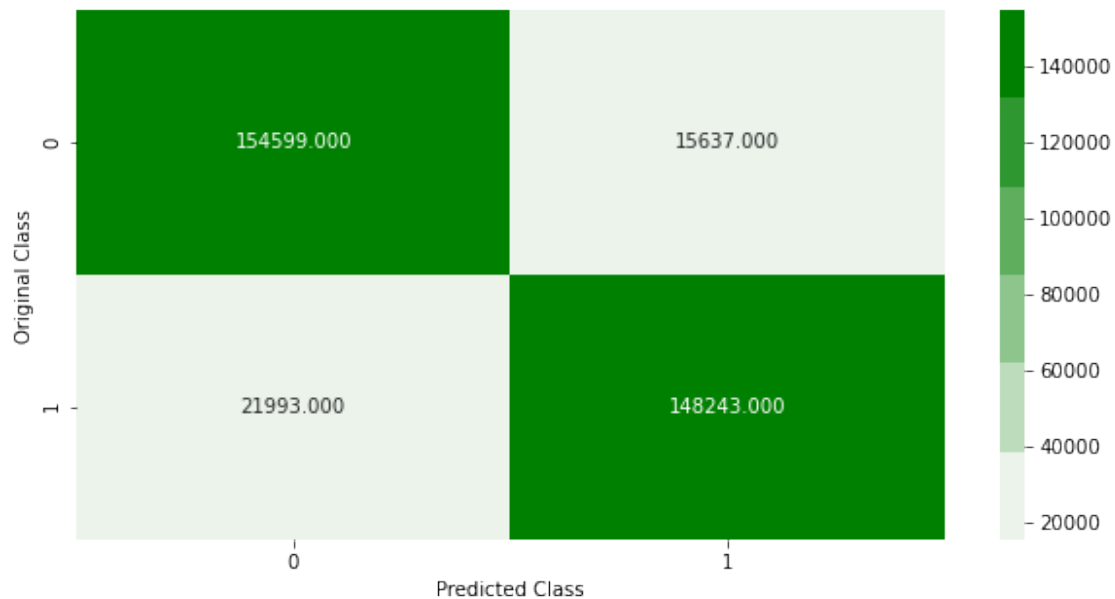
Confusion Matrix for the Train Data

The Weighted Recall Score: 0.8894769613947696

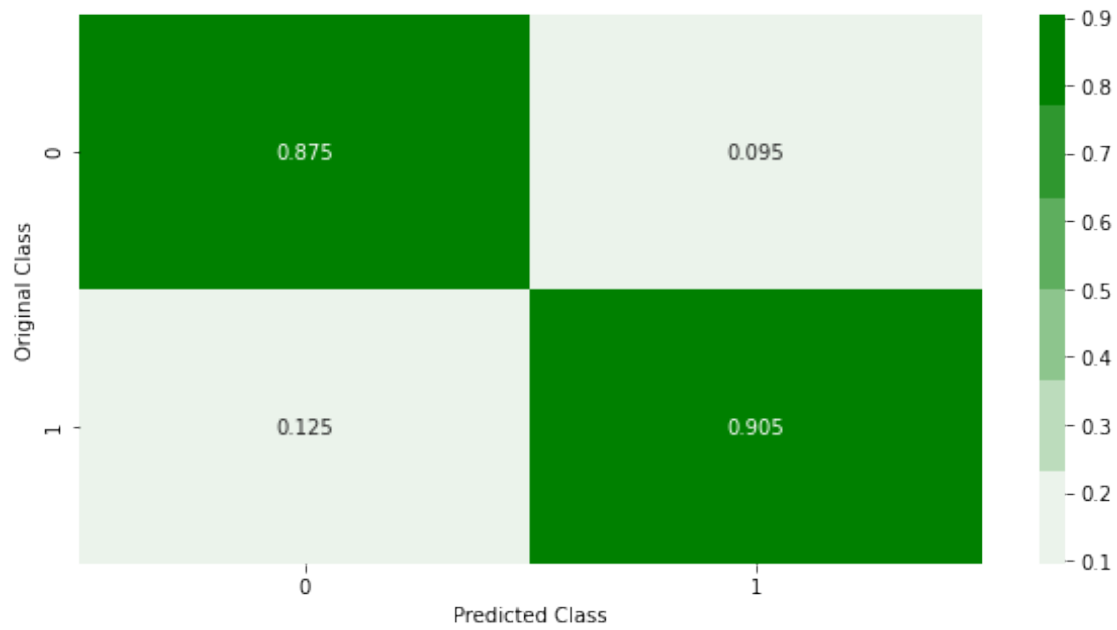
The Weighted Precision Score: 0.8900206529277153

The Weighted F1 Score: 0.8894384304934193

```
=====
=====
=====
----- Confusion matrix
-----
```

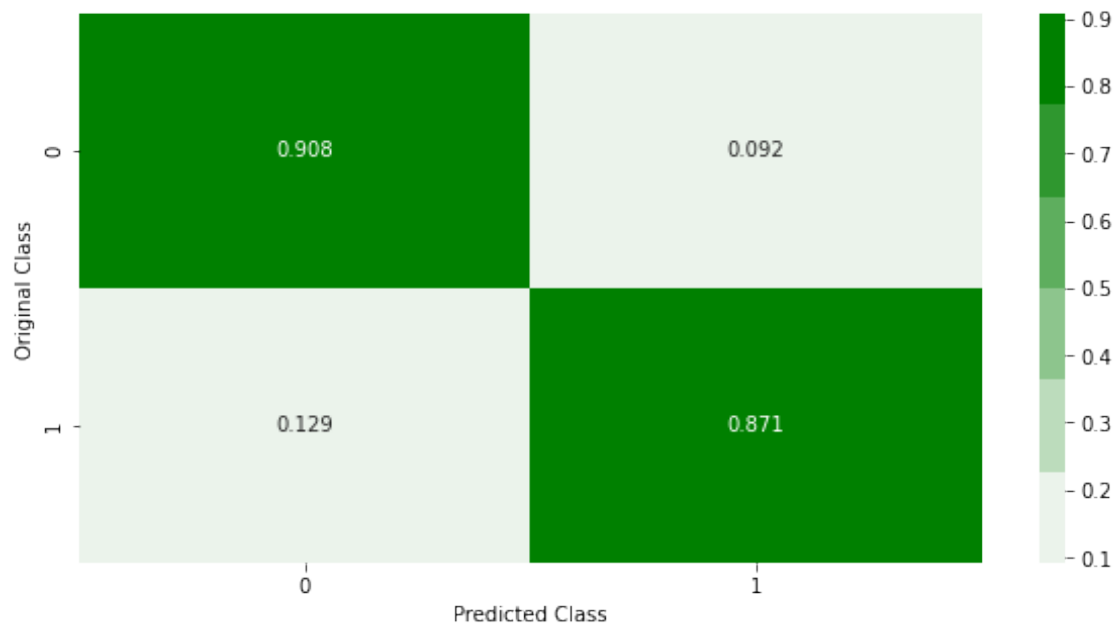


```
----- Precision matrix
-----
```



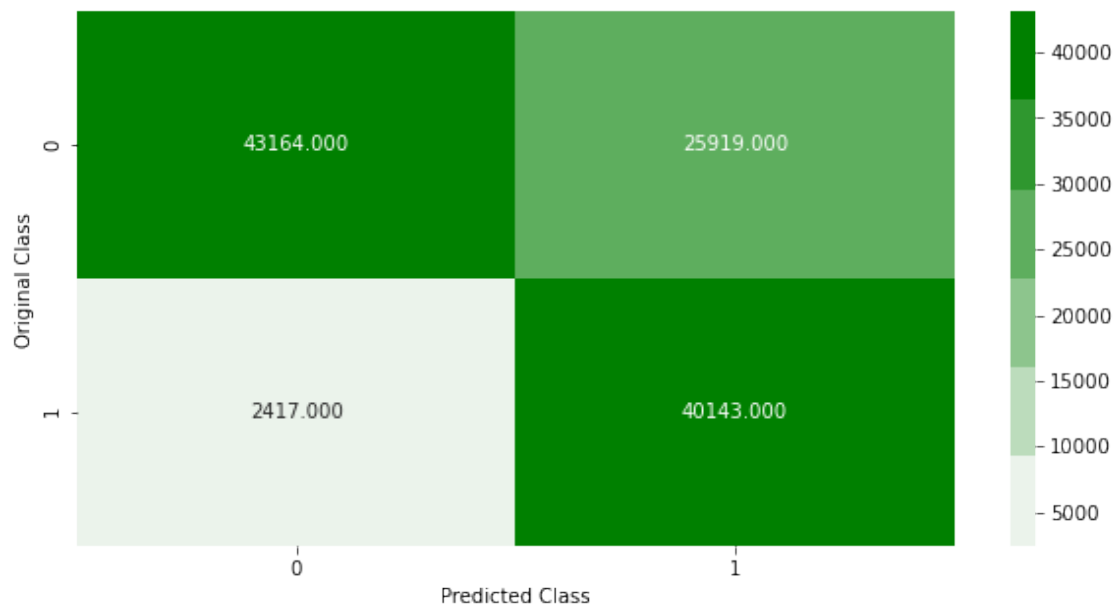
Sum of columns in precision matrix [1. 1.]

----- Recall matrix

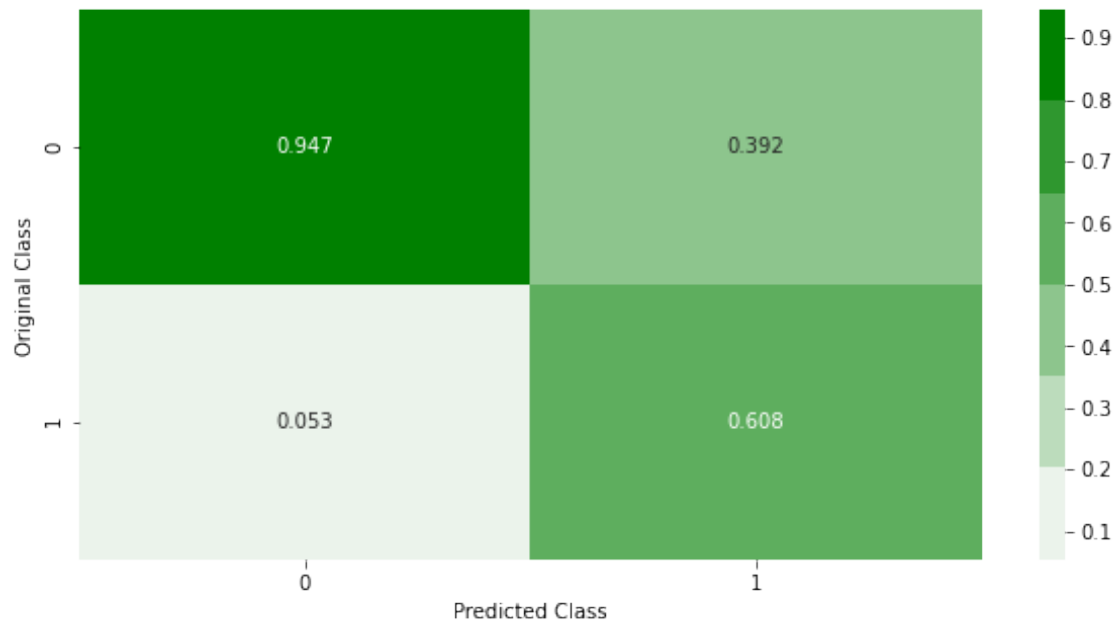


Sum of rows in precision matrix [1. 1.]
Confusion Matrix for the Cross Validate Data
The Weighted Recall Score: 0.7461909837607373
The Weighted Precision Score: 0.817620718841406
The Weighted F1 Score: 0.7476378144433122

=====
=====
=====
----- Confusion matrix

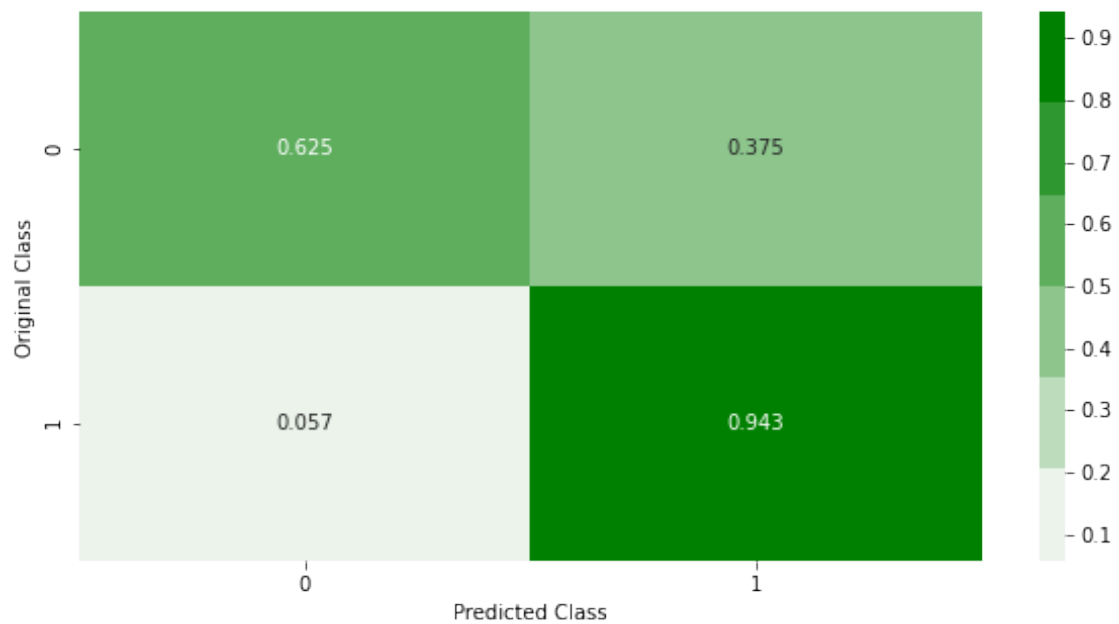


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[ ]: print("Confusion Matrix for Best Thresold for the Train Data")
      best_thresholds(train_y_us,train_y_pred)

      print("Confusion Matrix for Best Thresold for the CV Data")
      best_thresholds(cv_y,cv_y_pred)
```

Confusion Matrix for Best Thresold for the Train Data

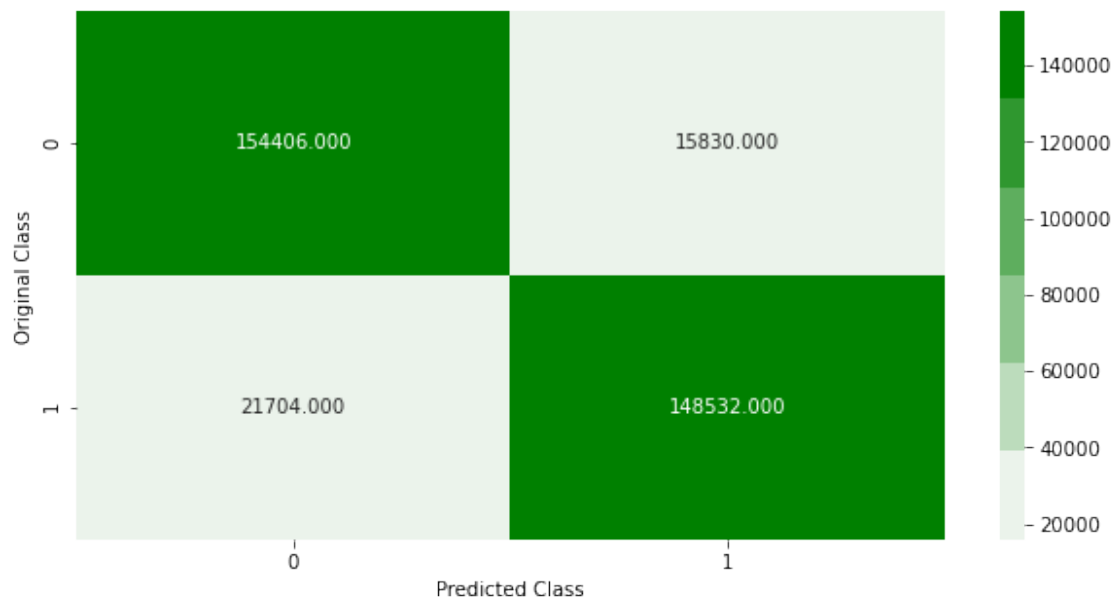
best train threshold : 0.49719696790766693

The Weighted Recall Score: 0.8897589229070232

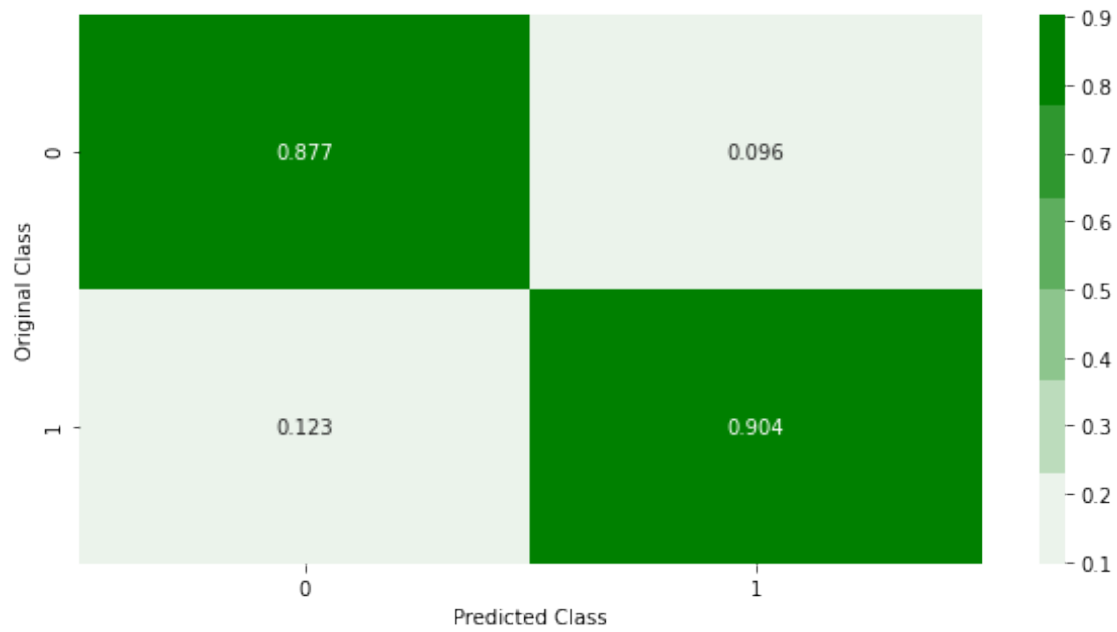
The Weighted Precision Score: 0.8902235221700009

The Weighted F1 Score: 0.8897260999416061

```
=====
=====
=====
----- Confusion matrix
-----
```

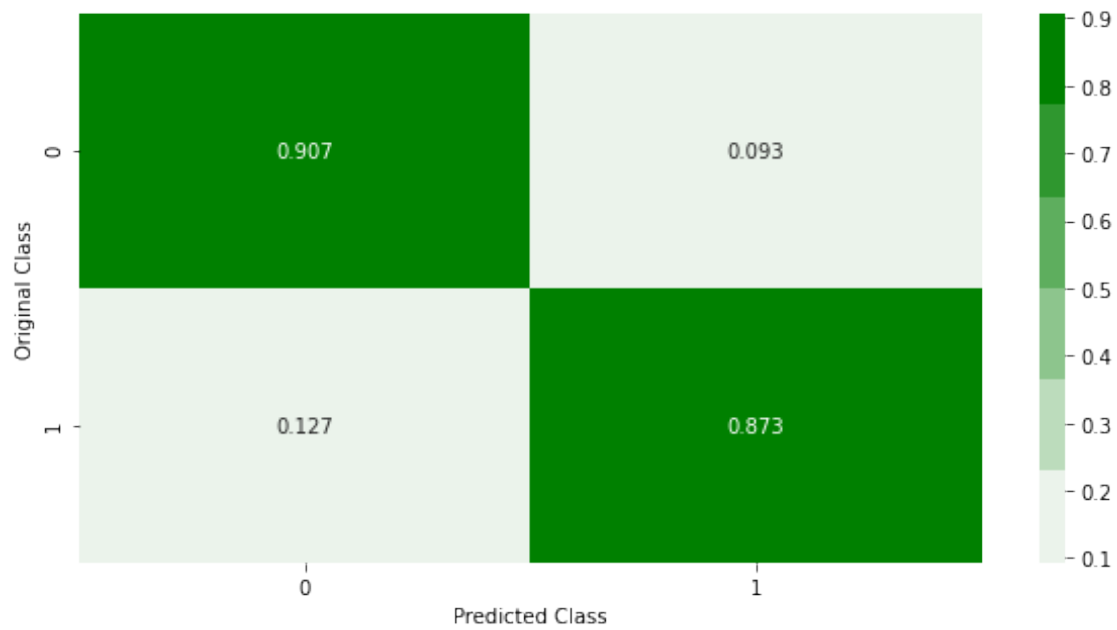


```
----- Precision matrix
-----
```

Sum of columns in precision matrix [1. 1.]

----- Recall matrix



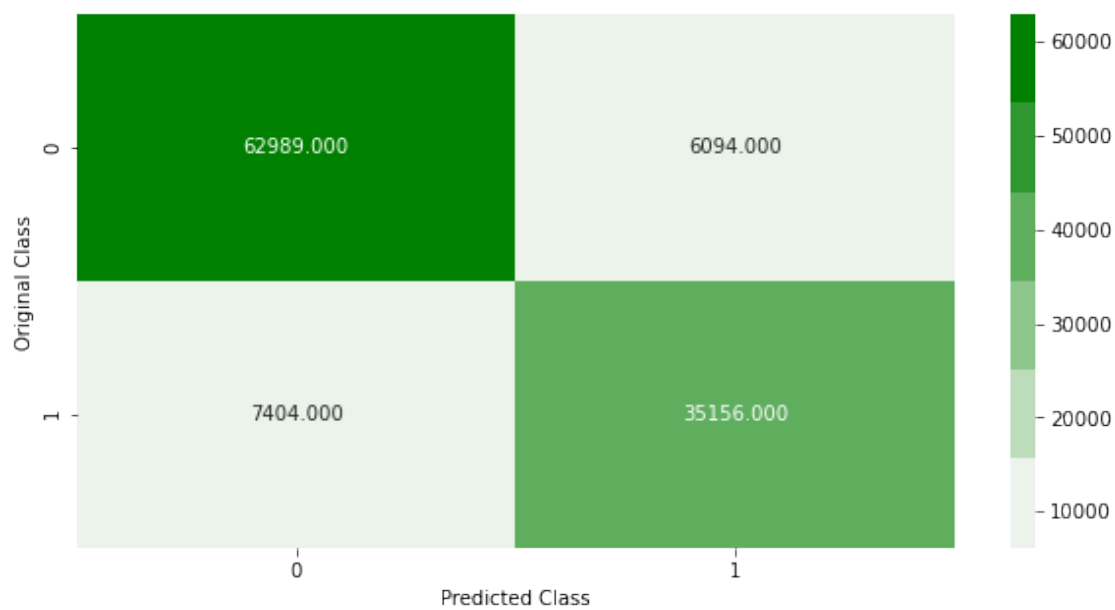
Sum of rows in precision matrix [1. 1.]
Confusion Matrix for Best Thresold for the CV Data
best train threshold : 0.6933070439350091
The Weighted Recall Score: 0.879096763791729
The Weighted Precision Score: 0.8785974609173831
The Weighted F1 Score: 0.8787196486967372

=====

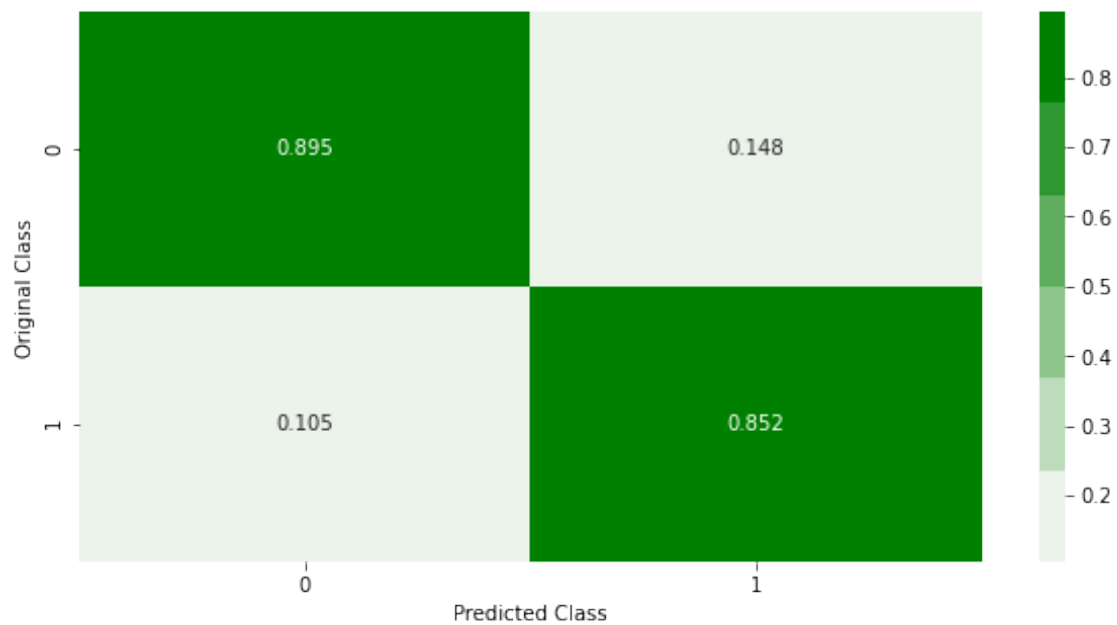
=====

=====

----- Confusion matrix

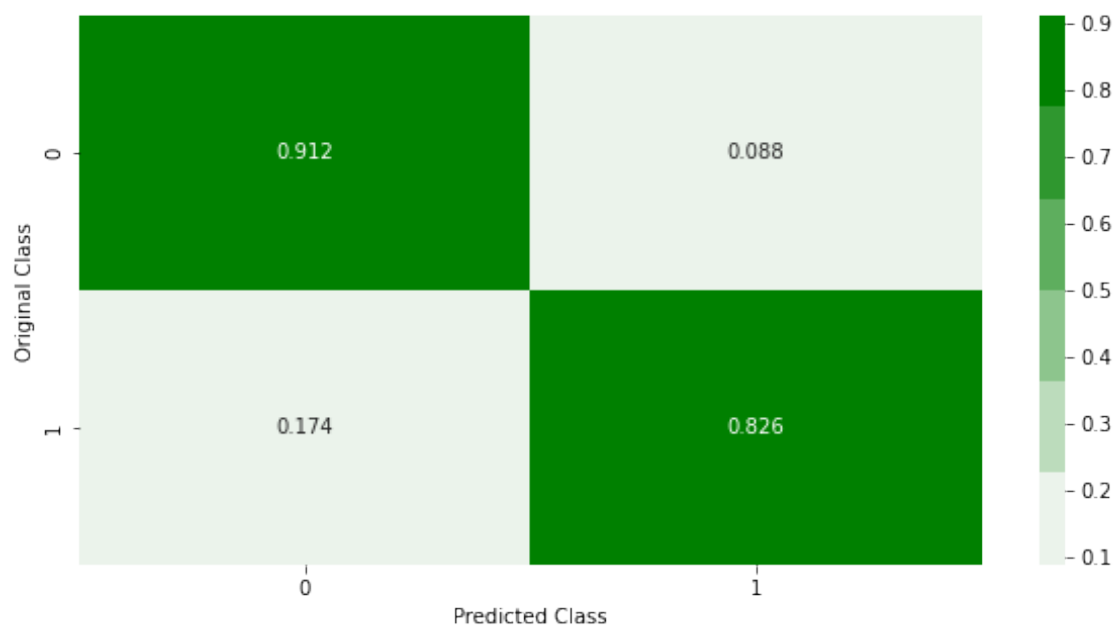


----- Precision matrix



Sum of columns in precision matrix [1. 1.]

----- Recall matrix



Sum of rows in precision matrix [1. 1.]

12.7 Observations

1. The performance of the models has further deteriorated in experiments 3 and 4.
2. In conclusion we have the best performance of Recall Score of 0.910 and Precision score of 0.909 and weighted F1 score of 0.910 in the case of Experiment 1 where we ran lightGBM on Datasets with out correcting for Class Imbalance in the Dataset.

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

12.8 Converting the Python Notebook into a PDF Document

```
[ ]: !wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('HealthInsFraud_v2.2.ipynb')
```

```
--2021-08-13 12:10:06-- https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py
```

```
colab_pdf.py          100%[=====>]    1.82K  --.-KB/s    in 0s
```

```
2021-08-13 12:10:07 (34.6 MB/s) - colab_pdf.py saved [1864/1864]
```

```
Mounted at /content/drive/
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
Extracting templates from packages: 100%
```

```
[ ]:
```