## DIY Project-1

## Building a Prediction model for banking loan approval info for a Nationalized Bank.

## Loading dataset

from google.colab import drive  
drive.mount('/content/drive')

Mounted at /content/drive

cd drive/MyDrive/data

/content/drive/MyDrive/data

ls

athlete\_events.csv lda\_10\_iter.pkl tennis/  
'BBC News Train.csv' ML\_Project\_1.zip titanic\_train.csv  
 Dig-MNIST.csv MNIST/ twitter/  
 groceries.csv pokemon.csv vgsales.csv  
'groceries - groceries.csv' project1.csv wineQualityReds.csv  
 housing.csv Sentiment/ wineQualityWhites.csv

%matplotlib inline  
import numpy as np  
import pandas as pd  
import sklearn as sk  
import matplotlib.pyplot as plt   
import seaborn as sns

data=pd.read\_csv('project1.csv')  
data.head()

APP\_ID CIBIL\_SCORE\_VALUE NEW\_CUST ... LTV TENURE STATUS  
0 12345 0 YES ... 0.767104 12 0  
1 12347 0 NO ... 0.619077 24 0  
2 12349 0 YES ... 0.848949 36 0  
3 12351 2 NO ... 0.515646 12 0  
4 12353 2 NO ... 0.614123 24 1  
  
[5 rows x 18 columns]

## Data Preprocessing

data.shape

(13299, 18)

data.describe()

APP\_ID CIBIL\_SCORE\_VALUE ... TENURE STATUS  
count 13299.0000 13299.000000 ... 13299.000000 13299.000000  
mean 25643.0000 1.037898 ... 24.792240 0.377171  
std 7678.4699 0.865391 ... 7.501085 0.484697  
min 12345.0000 0.000000 ... 12.000000 0.000000  
25% 18994.0000 0.000000 ... 18.000000 0.000000  
50% 25643.0000 1.000000 ... 24.000000 0.000000  
75% 32292.0000 2.000000 ... 36.000000 1.000000  
max 38941.0000 2.000000 ... 36.000000 1.000000  
  
[8 rows x 16 columns]

There is no missing values

data.info()

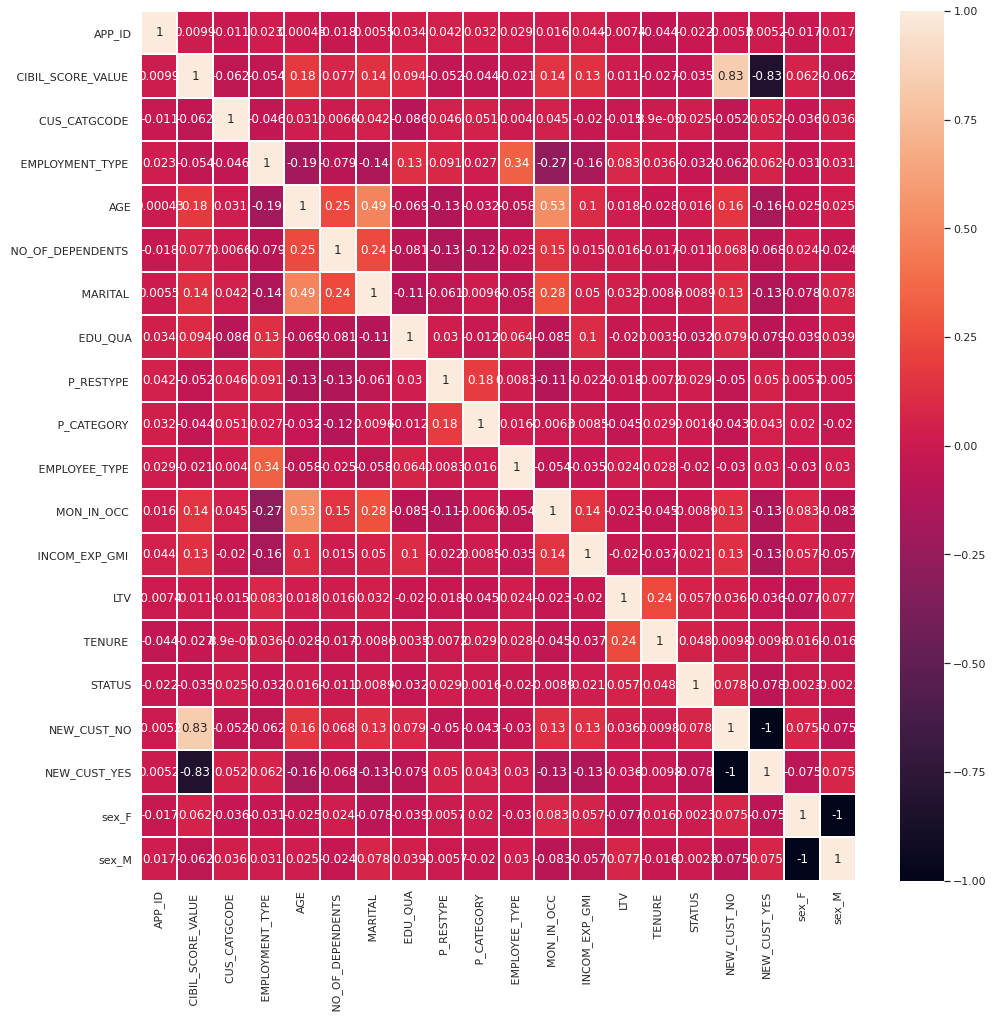
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 13299 entries, 0 to 13298  
Data columns (total 18 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 APP\_ID 13299 non-null int64   
 1 CIBIL\_SCORE\_VALUE 13299 non-null int64   
 2 NEW\_CUST 13299 non-null object   
 3 CUS\_CATGCODE 13299 non-null int64   
 4 EMPLOYMENT\_TYPE 13299 non-null int64   
 5 AGE 13299 non-null int64   
 6 SEX 13299 non-null object   
 7 NO\_OF\_DEPENDENTS 13299 non-null int64   
 8 MARITAL 13299 non-null int64   
 9 EDU\_QUA 13299 non-null int64   
 10 P\_RESTYPE 13299 non-null int64   
 11 P\_CATEGORY 13299 non-null int64   
 12 EMPLOYEE\_TYPE 13299 non-null int64   
 13 MON\_IN\_OCC 13299 non-null int64   
 14 INCOM\_EXP\_GMI 13299 non-null int64   
 15 LTV 13299 non-null float64  
 16 TENURE 13299 non-null int64   
 17 STATUS 13299 non-null int64   
dtypes: float64(1), int64(15), object(2)  
memory usage: 1.8+ MB

Categorical data like CUS\_CATGCODE,SEX need to be converted

dfNewCustDummies = pd.get\_dummies(data['NEW\_CUST'], prefix = 'NEW\_CUST')  
dfSexDummies = pd.get\_dummies(data[' SEX '], prefix = 'sex')  
data = pd.concat([data, dfNewCustDummies], axis=1)  
data = pd.concat([data, dfSexDummies], axis=1)  
data.drop(' SEX ', inplace=True, axis=1)  
data.drop('NEW\_CUST', inplace=True, axis=1)

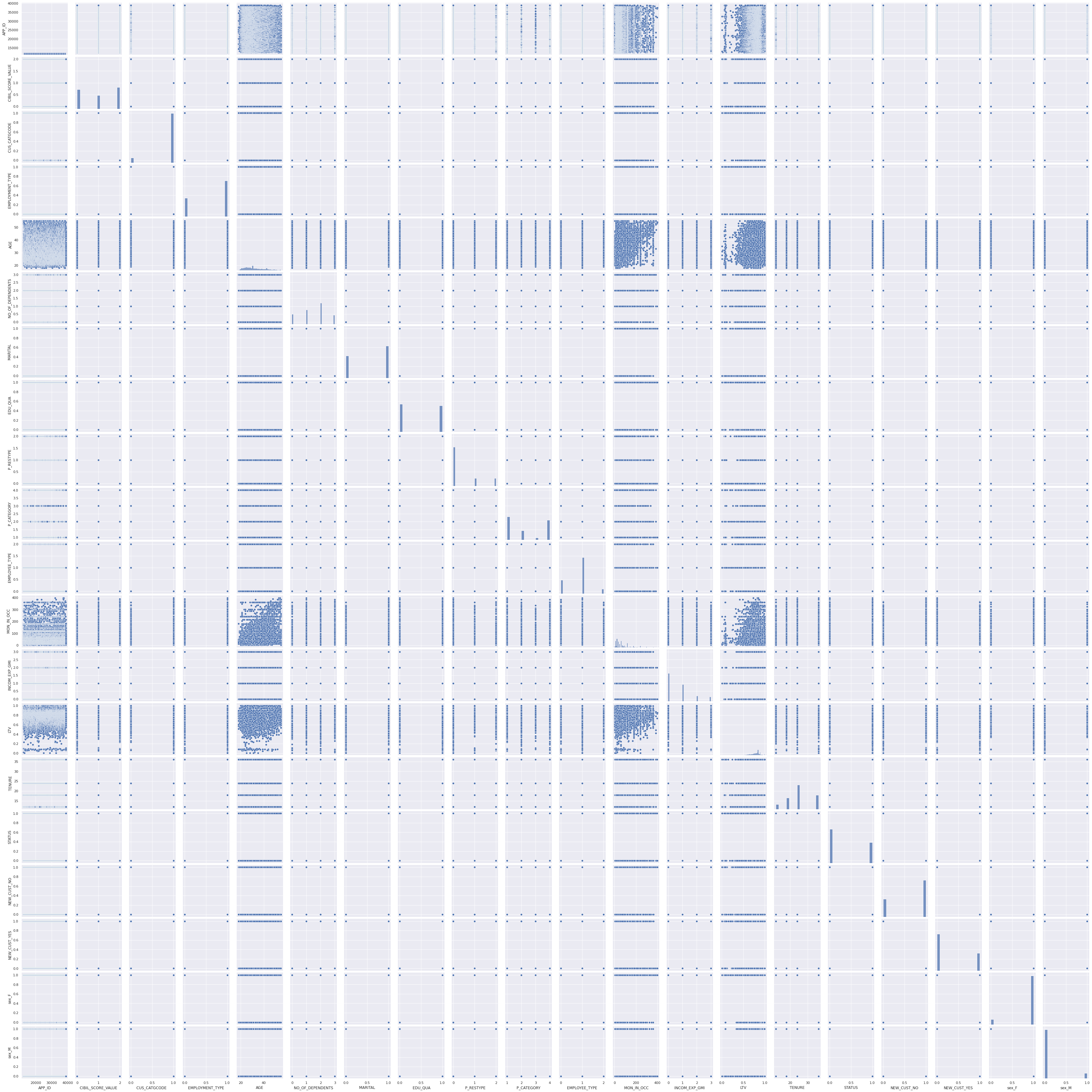
## Exploratory data analysis

sns.set(rc={'figure.figsize':(16,16)})  
ax=sns.heatmap(data.corr(),annot=True,linewidths=1)



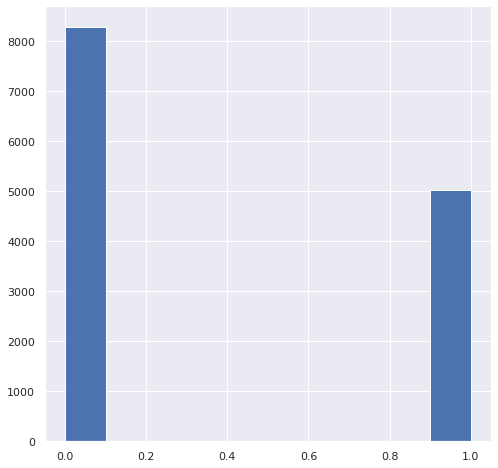
sns.pairplot(data)

<seaborn.axisgrid.PairGrid at 0x7f6e2898bf10>



sns.set(rc={'figure.figsize':(8,8)})  
data['STATUS'].hist()

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f6e1d609690>



Clearly,this is an imbalanced dataset

## Method 1: Undersampling

# Class count  
count\_class\_0, count\_class\_1 = data.STATUS.value\_counts()  
   
# Divide by class  
df\_class\_0 = data[data['STATUS'] == 0]  
df\_class\_1 = data[data['STATUS'] == 1]

# Undersample 0-class and concat the DataFrames of both class  
df\_class\_0\_under = df\_class\_0.sample(count\_class\_1)  
df\_test\_under = pd.concat([df\_class\_0\_under, df\_class\_1], axis=0)  
  
print('Random under-sampling:')  
print(df\_test\_under.STATUS.value\_counts())

Random under-sampling:  
1 5016  
0 5016  
Name: STATUS, dtype: int64

X = df\_test\_under.drop('STATUS',axis='columns')  
y = df\_test\_under['STATUS']  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=15, stratify=y)

### KNeighbors Classifier

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance'],  
 'metric':['euclidean','manhattan','minkowski']  
 }]

from sklearn.model\_selection import GridSearchCV  
from sklearn.neighbors import KNeighborsClassifier  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'metric': 'manhattan', 'n\_neighbors': 17, 'weights': 'uniform'}  
Best cross-validation score: 0.52  
Test set score: 0.51

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[497 507]  
 [472 531]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.51 0.50 0.50 1004  
 1 0.51 0.53 0.52 1003  
  
 accuracy 0.51 2007  
 macro avg 0.51 0.51 0.51 2007  
weighted avg 0.51 0.51 0.51 2007

from sklearn.preprocessing import StandardScaler  
scaler=StandardScaler().fit(X\_train)  
X\_train\_scaled=scaler.transform(X\_train)  
X\_test\_scaled=scaler.transform(X\_test)

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance'],  
 'metric':['euclidean','manhattan','minkowski']  
 }]

from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import GridSearchCV  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'metric': 'manhattan', 'n\_neighbors': 17, 'weights': 'uniform'}  
Best cross-validation score: 0.52  
Test set score: 0.51

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[497 507]  
 [472 531]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.51 0.50 0.50 1004  
 1 0.51 0.53 0.52 1003  
  
 accuracy 0.51 2007  
 macro avg 0.51 0.51 0.51 2007  
weighted avg 0.51 0.51 0.51 2007

This seems to reduce the false positive error but is not recommended as it leaves a part of data unused

# Method2: Oversampling

# Oversample 1-class and concat the DataFrames of both classes  
df\_class\_1\_over = df\_class\_1.sample(count\_class\_0, replace=True)  
df\_test\_over = pd.concat([df\_class\_0, df\_class\_1\_over], axis=0)  
   
print('Random over-sampling:')  
print(df\_test\_over.STATUS.value\_counts())

Random over-sampling:  
1 8283  
0 8283  
Name: STATUS, dtype: int64

X = df\_test\_over.drop('STATUS',axis='columns')  
y = df\_test\_over['STATUS']  
   
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=15, stratify=y)

# Number of classes in training Data  
y\_train.value\_counts()

1 6626  
0 6626  
Name: STATUS, dtype: int64

### KNeighborsClassifier

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance'],  
 'metric':['euclidean','manhattan','minkowski']  
 }]

from sklearn.neighbors import KNeighborsClassifier  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'metric': 'euclidean', 'n\_neighbors': 1, 'weights': 'uniform'}  
Best cross-validation score: 0.71  
Test set score: 0.74

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[1086 571]  
 [ 284 1373]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.79 0.66 0.72 1657  
 1 0.71 0.83 0.76 1657  
  
 accuracy 0.74 3314  
 macro avg 0.75 0.74 0.74 3314  
weighted avg 0.75 0.74 0.74 3314

### LogisticRegression

param\_grid = [{  
 'C': [0.001, 0.01, 0.1, 1, 10, 100],   
 }]  
  
from sklearn.model\_selection import GridSearchCV  
  
from sklearn.linear\_model import LogisticRegression  
grid\_search = GridSearchCV(LogisticRegression(max\_iter=14000), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'C': 10}  
Best cross-validation score: 0.54  
Test set score: 0.56

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[968 689]  
 [763 894]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.56 0.58 0.57 1657  
 1 0.56 0.54 0.55 1657  
  
 accuracy 0.56 3314  
 macro avg 0.56 0.56 0.56 3314  
weighted avg 0.56 0.56 0.56 3314

### RidgeClassifier

param\_grid = [{  
 'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],  
 }]

from sklearn.linear\_model import RidgeClassifier  
grid\_search = GridSearchCV(RidgeClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'alpha': 0.1}  
Best cross-validation score: 0.61  
Test set score: 0.61

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[1258 399]  
 [ 905 752]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.58 0.76 0.66 1657  
 1 0.65 0.45 0.54 1657  
  
 accuracy 0.61 3314  
 macro avg 0.62 0.61 0.60 3314  
weighted avg 0.62 0.61 0.60 3314

### SVC (Support Vector Classifier)

param\_grid = [{  
 'kernel':['poly', 'rbf', 'sigmoid'],  
 'C':[50, 10, 1.0, 0.1, 0.01],  
 'gamma':['scale']  
 }]  
   
from sklearn.svm import SVC  
grid\_search = GridSearchCV(SVC(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'C': 50, 'gamma': 'scale', 'kernel': 'poly'}  
Best cross-validation score: 0.52  
Test set score: 0.52

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[ 511 1146]  
 [ 461 1196]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.53 0.31 0.39 1657  
 1 0.51 0.72 0.60 1657  
  
 accuracy 0.52 3314  
 macro avg 0.52 0.52 0.49 3314  
weighted avg 0.52 0.52 0.49 3314

### RandomForestClassifier

param\_grid = [{  
 'n\_estimators':[10, 100, 1000],  
 'max\_features':['sqrt', 'log2']  
 }]  
from sklearn.ensemble import RandomForestClassifier  
grid\_search = GridSearchCV(RandomForestClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))  
pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'max\_features': 'log2', 'n\_estimators': 1000}  
Best cross-validation score: 0.80  
Test set score: 0.83  
[[1356 301]  
 [ 261 1396]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.84 0.82 0.83 1657  
 1 0.82 0.84 0.83 1657  
  
 accuracy 0.83 3314  
 macro avg 0.83 0.83 0.83 3314  
weighted avg 0.83 0.83 0.83 3314

### GradientBoostingClassifier

param\_grid=[  
 {  
 'n\_estimators': [10, 100, 1000],  
 'learning\_rate' : [0.001, 0.01, 0.1],  
 'subsample' : [0.5, 0.7, 1.0],  
 'max\_depth' : [3, 7, 9]  
 }  
]  
from sklearn.ensemble import GradientBoostingClassifier  
grid\_search = GridSearchCV(GradientBoostingClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))  
pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'learning\_rate': 0.1, 'max\_depth': 9, 'n\_estimators': 1000, 'subsample': 0.7}  
Best cross-validation score: 0.78  
Test set score: 0.82  
[[1346 311]  
 [ 294 1363]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.82 0.81 0.82 1657  
 1 0.81 0.82 0.82 1657  
  
 accuracy 0.82 3314  
 macro avg 0.82 0.82 0.82 3314  
weighted avg 0.82 0.82 0.82 3314

### BaggingClassifier

param\_grid=[  
 {  
 'n\_estimators': [10, 100, 1000]  
 }  
]  
from sklearn.ensemble import BaggingClassifier  
grid\_search = GridSearchCV(BaggingClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))  
pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'n\_estimators': 1000}  
Best cross-validation score: 0.78  
Test set score: 0.82  
[[1330 327]  
 [ 265 1392]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.83 0.80 0.82 1657  
 1 0.81 0.84 0.82 1657  
  
 accuracy 0.82 3314  
 macro avg 0.82 0.82 0.82 3314  
weighted avg 0.82 0.82 0.82 3314

Scaling the data to train better

from sklearn.preprocessing import StandardScaler  
scaler=StandardScaler().fit(X\_train)  
X\_train\_scaled=scaler.transform(X\_train)  
X\_test\_scaled=scaler.transform(X\_test)

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance']  
 }]

from sklearn.model\_selection import GridSearchCV  
   
from sklearn.neighbors import KNeighborsClassifier  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'n\_neighbors': 1, 'weights': 'uniform'}  
Best cross-validation score: 0.71  
Test set score: 0.76

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[1093 564]  
 [ 234 1423]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.82 0.66 0.73 1657  
 1 0.72 0.86 0.78 1657  
  
 accuracy 0.76 3314  
 macro avg 0.77 0.76 0.76 3314  
weighted avg 0.77 0.76 0.76 3314

param\_grid = [{  
 'C': [0.001, 0.01, 0.1, 1, 10, 100],   
 }]  
   
from sklearn.model\_selection import GridSearchCV  
   
from sklearn.linear\_model import LogisticRegression  
grid\_search = GridSearchCV(LogisticRegression(max\_iter=14000), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'C': 0.1}  
Best cross-validation score: 0.61  
Test set score: 0.62  
[[1255 402]  
 [ 862 795]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.59 0.76 0.67 1657  
 1 0.66 0.48 0.56 1657  
  
 accuracy 0.62 3314  
 macro avg 0.63 0.62 0.61 3314  
weighted avg 0.63 0.62 0.61 3314

param\_grid = [{  
 'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],  
 }]  
from sklearn.linear\_model import RidgeClassifier  
grid\_search = GridSearchCV(RidgeClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'alpha': 0.1}  
Best cross-validation score: 0.60  
Test set score: 0.61  
[[1657 0]  
 [1657 0]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.50 1.00 0.67 1657  
 1 0.00 0.00 0.00 1657  
  
 accuracy 0.50 3314  
 macro avg 0.25 0.50 0.33 3314  
weighted avg 0.25 0.50 0.33 3314

param\_grid = [{  
 'kernel':['poly', 'rbf', 'sigmoid'],  
 'C':[50, 10, 1.0, 0.1, 0.01],  
 'gamma':['scale']  
 }]  
   
from sklearn.svm import SVC  
grid\_search = GridSearchCV(SVC(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
   
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'C': 50, 'gamma': 'scale', 'kernel': 'rbf'}  
Best cross-validation score: 0.69  
Test set score: 0.71  
[[1115 542]  
 [ 433 1224]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.72 0.67 0.70 1657  
 1 0.69 0.74 0.72 1657  
  
 accuracy 0.71 3314  
 macro avg 0.71 0.71 0.71 3314  
weighted avg 0.71 0.71 0.71 3314

# Method3: SMOTE

pip install imbalanced-learn

X = data.drop('STATUS',axis='columns')  
y = data['STATUS']

from imblearn.over\_sampling import SMOTE  
   
smote = SMOTE(sampling\_strategy='minority')  
X\_sm, y\_sm = smote.fit\_sample(X, y)  
   
unique, counts = np.unique(y\_sm, return\_counts=True)  
print(unique,counts)

[0 1] [8283 8283]

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sm, y\_sm, test\_size=0.2, random\_state=15, stratify=y\_sm)

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance'],  
 'metric':['euclidean','manhattan','minkowski']  
 }]

from sklearn.neighbors import KNeighborsClassifier  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test, y\_test)))

Best parameters: {'metric': 'manhattan', 'n\_neighbors': 1, 'weights': 'uniform'}  
Best cross-validation score: 0.65  
Test set score: 0.67

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[ 980 677]  
 [ 416 1241]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.70 0.59 0.64 1657  
 1 0.65 0.75 0.69 1657  
  
 accuracy 0.67 3314  
 macro avg 0.67 0.67 0.67 3314  
weighted avg 0.67 0.67 0.67 3314

from sklearn.preprocessing import StandardScaler  
scaler=StandardScaler().fit(X\_train)  
X\_train\_scaled=scaler.transform(X\_train)  
X\_test\_scaled=scaler.transform(X\_test)

### KNeighborsClassifier

param\_grid = [{  
 'n\_neighbors':list(range(1,21,2)),  
 'weights':['uniform','distance'],  
 'metric':['euclidean','manhattan','minkowski']  
 }]

from sklearn.neighbors import KNeighborsClassifier  
grid\_search = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))

Best parameters: {'metric': 'manhattan', 'n\_neighbors': 19, 'weights': 'distance'}  
Best cross-validation score: 0.67  
Test set score: 0.69

pred=grid\_search.predict(X\_test)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

[[ 152 1505]  
 [ 117 1540]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.57 0.09 0.16 1657  
 1 0.51 0.93 0.66 1657  
  
 accuracy 0.51 3314  
 macro avg 0.54 0.51 0.41 3314  
weighted avg 0.54 0.51 0.41 3314

### LogisticRegression

param\_grid = [{  
 'C': [0.001, 0.01, 0.1, 1, 10, 100],   
 }]  
   
from sklearn.model\_selection import GridSearchCV  
   
from sklearn.linear\_model import LogisticRegression  
grid\_search = GridSearchCV(LogisticRegression(max\_iter=14000), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'C': 1}  
Best cross-validation score: 0.63  
Test set score: 0.63  
[[1275 382]  
 [ 831 826]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.61 0.77 0.68 1657  
 1 0.68 0.50 0.58 1657  
  
 accuracy 0.63 3314  
 macro avg 0.64 0.63 0.63 3314  
weighted avg 0.64 0.63 0.63 3314

### RidgeClassifier

param\_grid = [{  
 'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],  
 }]  
from sklearn.linear\_model import RidgeClassifier  
grid\_search = GridSearchCV(RidgeClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'alpha': 0.1}  
Best cross-validation score: 0.63  
Test set score: 0.63  
[[1277 380]  
 [ 838 819]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.60 0.77 0.68 1657  
 1 0.68 0.49 0.57 1657  
  
 accuracy 0.63 3314  
 macro avg 0.64 0.63 0.63 3314  
weighted avg 0.64 0.63 0.63 3314

### SVC(Support Vector Classifier)

param\_grid = [{  
 'kernel':['poly', 'rbf', 'sigmoid'],  
 'C':[50, 10, 1.0, 0.1, 0.01],  
 'gamma':['scale']  
 }]  
   
from sklearn.svm import SVC  
grid\_search = GridSearchCV(SVC(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
   
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}  
Best cross-validation score: 0.70  
Test set score: 0.70  
[[1284 373]  
 [ 616 1041]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.68 0.77 0.72 1657  
 1 0.74 0.63 0.68 1657  
  
 accuracy 0.70 3314  
 macro avg 0.71 0.70 0.70 3314  
weighted avg 0.71 0.70 0.70 3314

### RandomForestClassifier

param\_grid = [{  
 'n\_estimators':[10, 100, 1000],  
 'max\_features':['sqrt', 'log2']  
 }]  
from sklearn.ensemble import RandomForestClassifier  
grid\_search = GridSearchCV(RandomForestClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'max\_features': 'sqrt', 'n\_estimators': 100}  
Best cross-validation score: 0.73  
Test set score: 0.72  
[[1372 285]  
 [ 639 1018]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.68 0.83 0.75 1657  
 1 0.78 0.61 0.69 1657  
  
 accuracy 0.72 3314  
 macro avg 0.73 0.72 0.72 3314  
weighted avg 0.73 0.72 0.72 3314

### GradientBoostingClassifier

param\_grid=[  
 {  
 'n\_estimators': [10, 100, 1000],  
 'learning\_rate' :[0.001, 0.01, 0.1],  
 'subsample': [0.5, 0.7, 1.0],  
 'max\_depth' : [3, 7, 9]  
 }  
]  
from sklearn.ensemble import GradientBoostingClassifier  
grid\_search = GridSearchCV(GradientBoostingClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'learning\_rate': 0.1, 'max\_depth': 9, 'n\_estimators': 1000, 'subsample': 1.0}  
Best cross-validation score: 0.79  
Test set score: 0.82  
[[1342 315]  
 [ 285 1372]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.82 0.81 0.82 1657  
 1 0.81 0.83 0.82 1657  
  
 accuracy 0.82 3314  
 macro avg 0.82 0.82 0.82 3314  
weighted avg 0.82 0.82 0.82 3314

### BaggingClassifier

param\_grid=[  
 {  
 'n\_estimators': [10, 100, 1000]  
 }  
]  
from sklearn.ensemble import BaggingClassifier  
grid\_search = GridSearchCV(BaggingClassifier(), param\_grid, cv=5)  
grid\_search.fit(X\_train\_scaled, y\_train)  
print("Best parameters: {}".format(grid\_search.best\_params\_))  
print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))  
print("Test set score: {:.2f}".format(grid\_search.score(X\_test\_scaled, y\_test)))  
pred=grid\_search.predict(X\_test\_scaled)  
from sklearn.metrics import confusion\_matrix,classification\_report  
confusion=confusion\_matrix(y\_test,pred)  
print(confusion)  
print("Classification Report: \n", classification\_report(y\_test, pred))

Best parameters: {'n\_estimators': 1000}  
Best cross-validation score: 0.79  
Test set score: 0.83  
[[1328 329]  
 [ 248 1409]]  
Classification Report:   
 precision recall f1-score support  
  
 0 0.84 0.80 0.82 1657  
 1 0.81 0.85 0.83 1657  
  
 accuracy 0.83 3314  
 macro avg 0.83 0.83 0.83 3314  
weighted avg 0.83 0.83 0.83 3314

## Conclusion

Clearly the best model is random forest classifier with oversampled data

Best parameters: {'max\_features': 'log2', 'n\_estimators': 1000}

Best cross-validation score: 0.80

Test set score: 0.83

Or

Bagging classifier with oversampled data by SMOTE technique

Best parameters: {'n\_estimators': 1000}

Best cross-validation score: 0.79

Test set score: 0.83