Project Report
On

“Micro Credit Defaulter Project”

Submitted by
Mr. Pratyush Ghosh

ACKNOWLEDGEMENT

It is my sensual gratification to present this report on **Micro Credit Defaulter Project**. Working on this project was an incredible experience that has given me a very informative knowledge. I would like to express my sincere thanks to MR. SAPNA VERMA for a regular follow up and valuable suggestions provided throughout. And I am also thankful to FlipRobo for providing this opportunity.

OVERVIEW

1. INTRODUCTION

A Microfinance Institution (MFI) is an organization that offers financial services to low income populations. MFS becomes very useful when targeting especially the unbanked poor families living in remote areas with not much sources of income. The Microfinance services (MFS) provided by MFI are Group Loans, Agricultural Loans, Individual Business Loans and so on.

Many microfinance institutions (MFI), experts and donors are supporting the idea of using mobile financial services (MFS) which they feel are more convenient and efficient, and cost saving, than the traditional high-touch model used since long for the purpose of delivering microfinance services. Though, the MFI industry is primarily focusing on low income families and is very useful in such areas, the implementation of MFS has been uneven with both significant challenges and successes.

Today, microfinance is widely accepted as a poverty-reduction tool, representing $70 billion in outstanding loans and a global outreach of 200 million clients.

We are working with one such client that is in Telecom Industry. They are a fixed wireless telecommunications network provider. They have launched various products and have developed its business and organization based on the budget operator model, offering better products at Lower Prices to all value conscious customers through a strategy of disruptive innovation that focuses on the subscriber.

They understand the importance of communication and how it affects a person's life, thus, focusing on providing their services and products to low income families and poor customers that can help them in the need of hour.

They are collaborating with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be defaulter if he deviates from the path of paying back the loaned amount within the time duration of 5 days. For the loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).

The sample data is provided to us from our client database. It is hereby given to you for this exercise. In order to improve the selection of customers for the credit, the client wants some predictions that could help them in further investment and improvement in selection of customers.

2. PROBLEM FRAMING:

Here in this project we need to Build a model which can be used to predict in terms of a probability for each loan transaction, whether the customer will be paying back the loaned amount within 5 days of insurance of loan. In this case, Label '1' indicates that the loan has been payed i.e. Non- defaulter, while, Label '0' indicates that the loan has not been payed i.e. defaulter.

2.1 Data overview

Importing necessary libraries

I have imported some necessary libraries

- Pandas: 'pandas' is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python.
- Numpy: NumPy is the fundamental package for scientific computing in Python. NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data
- Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Seaborn : is a Python data visualization library based on matplotlib
- StandardScaler : to scale the numerical data and bring to normal scale
- Evaluation metrics like accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score, and from sklearn.metrics
- To check cross validation score for different folds cross_val_score from sklearn.model_selection.

```
#import Necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

#import required accuracy metrics
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import KFold, cross_val_score

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Importing Libraries

Loading the dataset

In [2]:

```
#loading the data set
df = pd.read_csv(r"C:\Users\Asus\Desktop\flipnwork\abhi micro\Data file.csv")
df
```

Out[2]:

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	mec
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	...	6.0	
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	...	12.0	
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	...	6.0	
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	...	6.0	
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	...	6.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
209588	209589	1	22758I85348	404.0	151.872333	151.872333	1089.19	1089.19	1.0	0.0	...	6.0	
209589	209590	1	95583I84455	1075.0	36.936000	36.936000	1728.36	1728.36	4.0	0.0	...	6.0	
209590	209591	1	28556I85350	1013.0	11843.111667	11904.350000	5861.83	8893.20	3.0	0.0	...	12.0	
209591	209592	1	59712I82733	1732.0	12488.228333	12574.370000	411.83	984.58	2.0	38.0	...	12.0	
209592	209593	1	65061I85339	1581.0	4489.362000	4534.820000	483.92	631.20	13.0	0.0	...	12.0	

209593 rows × 37 columns

Fig. 1 Loading the Data

Using pandas I have loaded the dataset for this project. By looking at the shape of our data set it is observed that this data set is having 2,09,593 rows and 37 columns. This time we are going handle huge dataset so we will face different problems associated with it.

3. Data Processing

After loading the data, I have ensured that whether there is any null value present in the dataset. And luckily we are not having null values in the data. Observing the data info I recognize that data set has 3 columns with object data type and others are integer and float types.

```
#let's see info about data
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 37 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            209593 non-null  int64
 1   label                 209593 non-null  int64
 2   msisdn                209593 non-null  object
 3   aon                   209593 non-null  float64
 4   daily_decr30          209593 non-null  float64
 5   daily_decr90          209593 non-null  float64
 6   rental30              209593 non-null  float64
 7   rental90              209593 non-null  float64
 8   last_rech_date_ma     209593 non-null  float64
 9   last_rech_date_da     209593 non-null  float64
 10  last_rech_amt_ma      209593 non-null  int64
 11  cnt_ma_rech30         209593 non-null  int64
 12  fr_ma_rech30          209593 non-null  float64
 13  sumamnt_ma_rech30     209593 non-null  float64
 14  medianamnt_ma_rech30  209593 non-null  float64
 15  medianmarechprebal30  209593 non-null  float64
 16  cnt_ma_rech90         209593 non-null  int64
 17  fr_ma_rech90          209593 non-null  int64
 18  sumamnt_ma_rech90     209593 non-null  int64
 19  medianamnt_ma_rech90  209593 non-null  float64
 20  medianmarechprebal90  209593 non-null  float64
 21  cnt_da_rech30         209593 non-null  float64
 22  fr_da_rech30          209593 non-null  float64
 23  cnt_da_rech90         209593 non-null  int64
 24  fr_da_rech90          209593 non-null  int64
 25  cnt_loans30           209593 non-null  int64
 26  amnt_loans30          209593 non-null  int64
 27  maxamnt_loans30       209593 non-null  float64
 28  medianamnt_loans30    209593 non-null  float64
 29  cnt_loans90           209593 non-null  float64
 30  amnt_loans90          209593 non-null  int64
 31  maxamnt_loans90       209593 non-null  int64
 32  medianamnt_loans90    209593 non-null  float64
 33  payback30             209593 non-null  float64
 34  payback90             209593 non-null  float64
 35  pcircle               209593 non-null  object
 36  pdate                 209593 non-null  object
dtypes: float64(21), int64(13), object(3)
memory usage: 59.2+ MB
```

Data cleaning steps:

While analyzing the data I found that we don't need columns named 'Unnamed 0', and 'msisdn'. Because column 'Unnamed 0' contains index numbers and column 'msisdn' contains contact numbers of customers. So these two columns are no way contributing to label prediction. So I dropped both these columns.

```
df.drop(columns = ['Unnamed: 0', 'msisdn'] , inplace = True )
```

Column 'aon' contains age on cellular network in days, for better understanding I have converted this data into years, by dividing this column by 365

```
df['aon'] = df['aon']/365
```

Looking at the data description and distribution plots I came to know that most of the entries are negative and very large values which are unrealistic. I decided to drop negative data and as we don't want to lose much data so I am using percentile method to replace large number. And have replaced all unrealistic entries with suitable action.

```
num = df._get_numeric_data()

for col in num.columns:
    if col != 'label':
        df = df.loc[df[col] >= 0]

for col in df.columns:
    if col != 'pdate':
      percentile=df[col].quantile([0.01,0.96]).values
      df[col][df[col] <= percentile[0]] = percentile[0]
      df[col][df[col] >= percentile[1]] = percentile[1]
```

Code used to replace unrealistic data

Column pcircle has single unique entry throughout its length and. So I decided to drop this column.

```
df.drop(columns = ['pcircle'], inplace = True)
```

Now I will create new columns for day, months, and year using column 'pdate '

```
df['Month'] = pd.DatetimeIndex(df['pdate']).month
df['Day'] = pd.DatetimeIndex(df['pdate']).day
df['Year'] = pd.DatetimeIndex(df['pdate']).year
```

As we have derived separate columns for day, month and year using column pdate, I will drop this column

```
df.drop(columns = 'pdate', inplace = True)
```

Checking the value counts for column **'Year'**

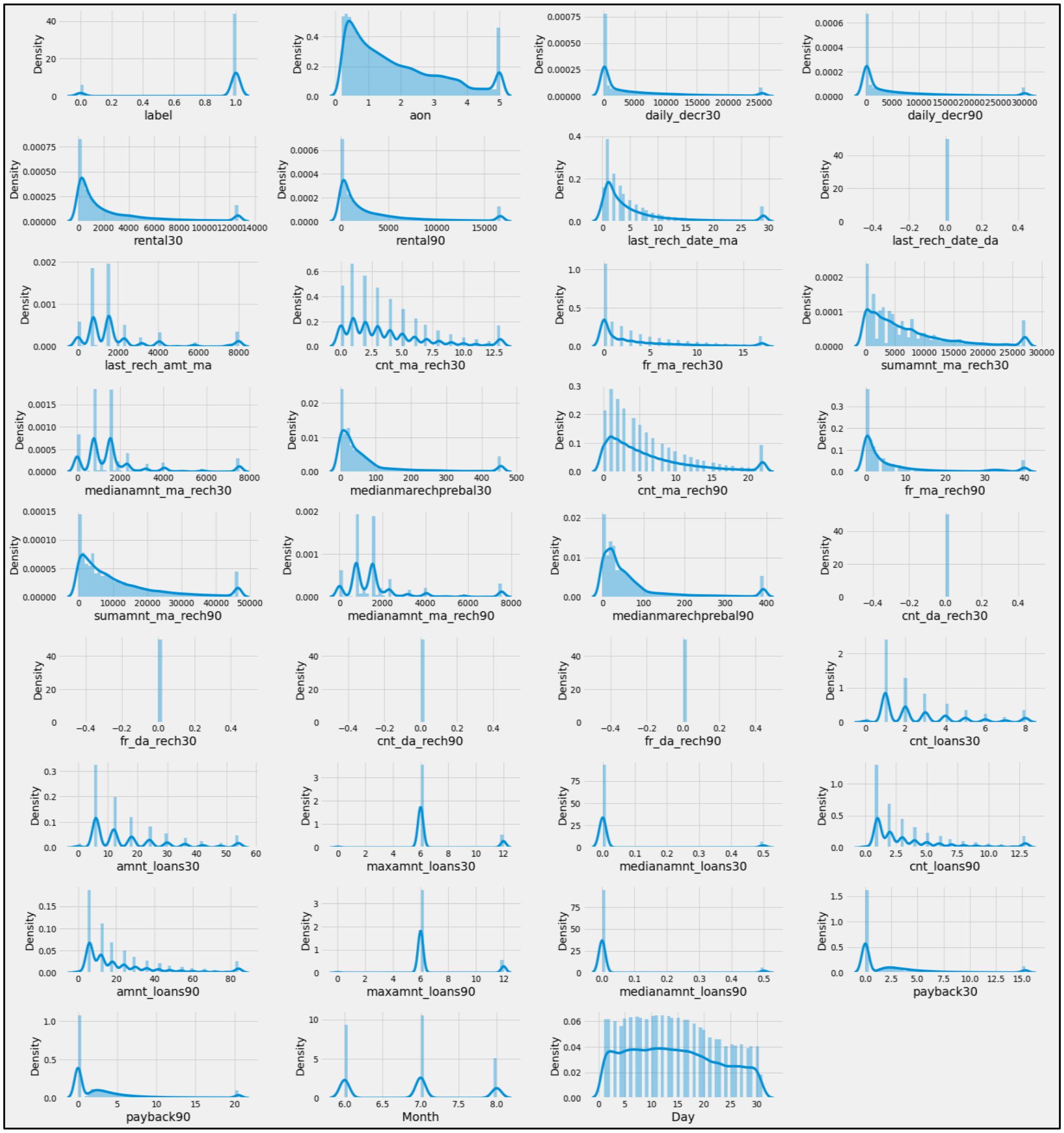
```
df['Year'].value_counts()
```

```
2016    197333
```

Looking at the value count for Year column we came to know that all the data is collected from the year 2016. So we can drop this column.

```
df.drop(columns = 'Year', inplace = True)
```

Check the distribution plots for all numerical data



After replacing negative and large values we can see the distribution plots looks quite good, but now some of the columns having only 0's throughout so I will delete those columns.

```
df.drop(columns =
['last_rech_date_da','fr_ma_rech30','fr_ma_rech90','fr_da_rech30','fr_d
a_rech90','cnt_da_rech30','cnt_da_rech90','medianamnt_loans30','mediana
mnt_loans90'], inplace = True)
```

Dropping unwanted columns

4. Visualization

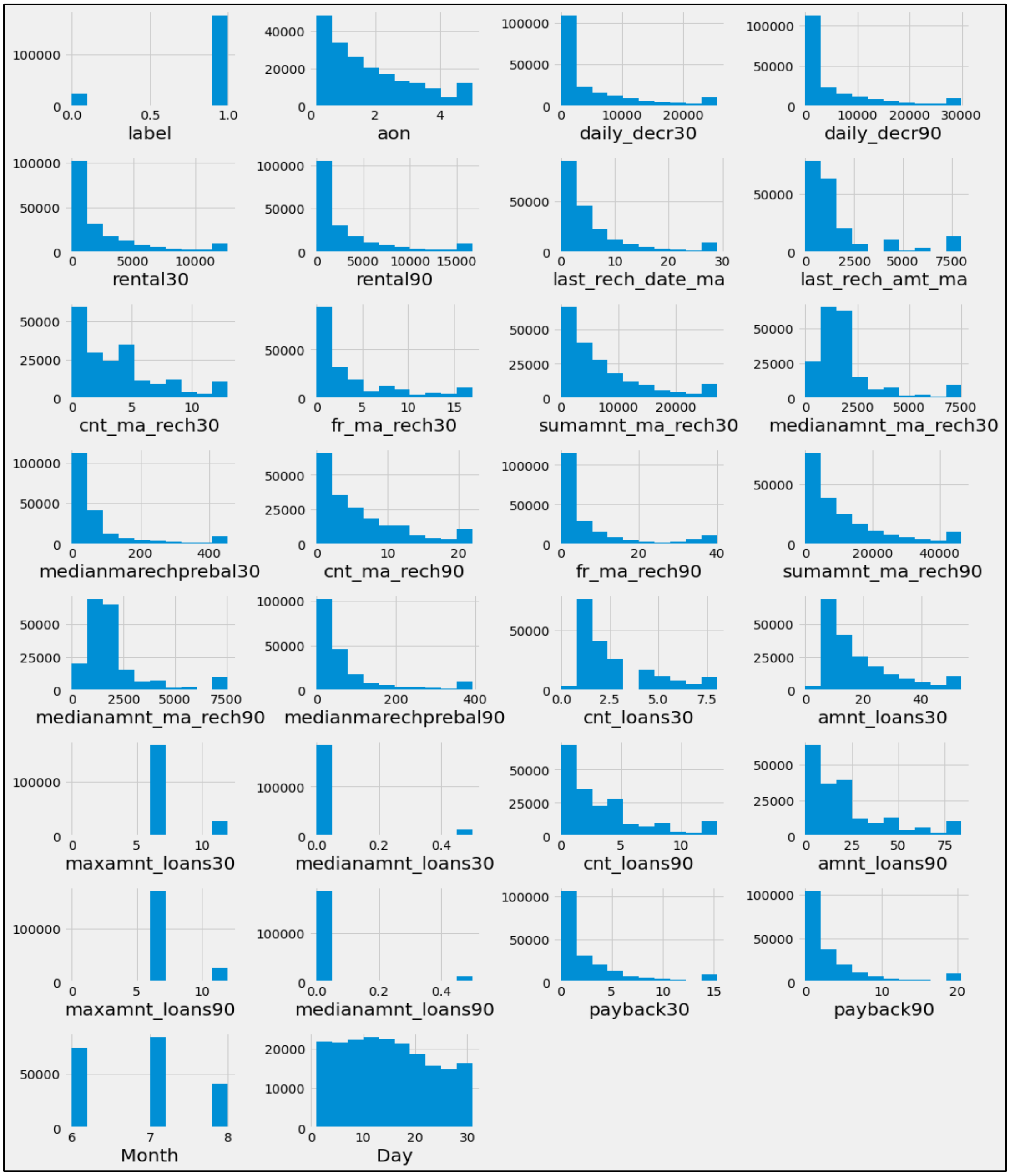


Fig. 3 Histograms for processed data

Looking at the above hist plots we can say now we got better range of our data than earlier.

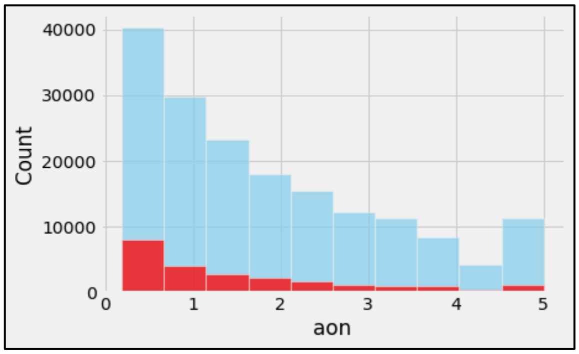


Fig. 4

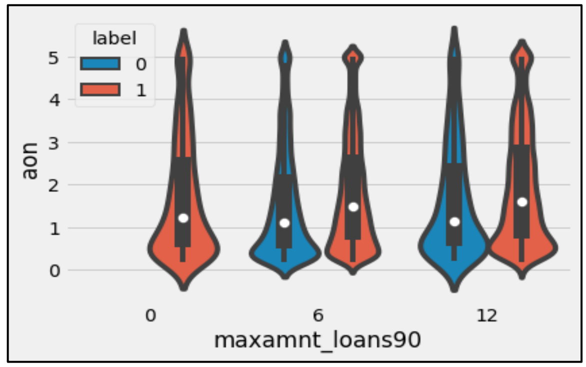


Fig. 5

Looking at above hist plot for age of customers on cellular network in years; we can say that large numbers of customers are there who are using cellular network since around last 1 year and the rate of not paying back the credit amount in these people is higher than others.

Fig. 5 represents the maximum amount of loan taken by the user in last 90 days. This violin plot tells us that when the loan taken in last 90 days is very less or near to zero they will pay back the credit amount within 5 days

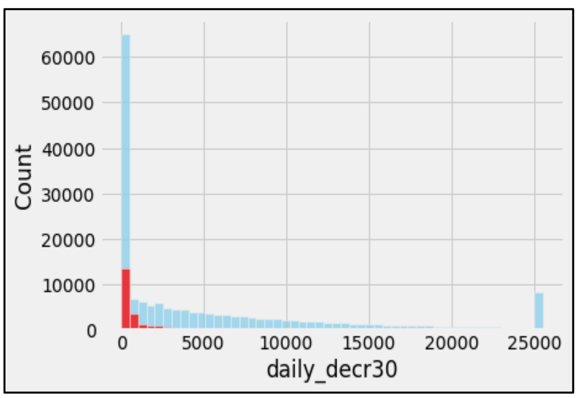


Fig. 6

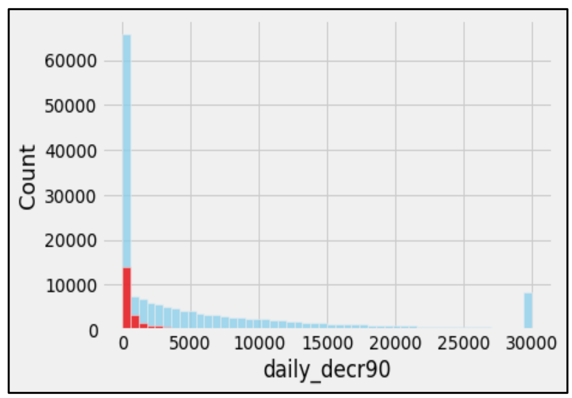


Fig. 7

Both of the above plots represents daily amount spent from main account, averaged over last 30 and 90 days respectively. Looking at these plots we can conclude that the customers will pay back the loan amount when the daily amount spent from main account is above 2500.

Most customers who didn't spent any amount from the main account has higher ratio of not to paying back the loan amount within estimated time.

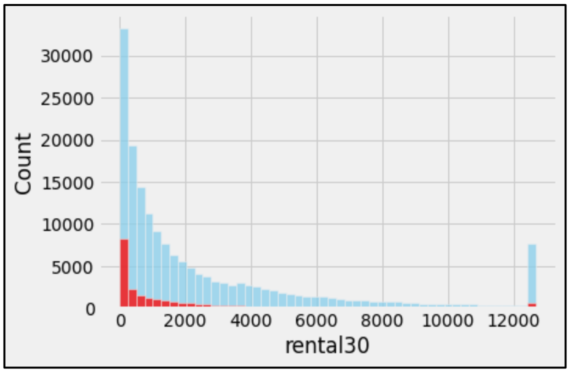


Fig. 8

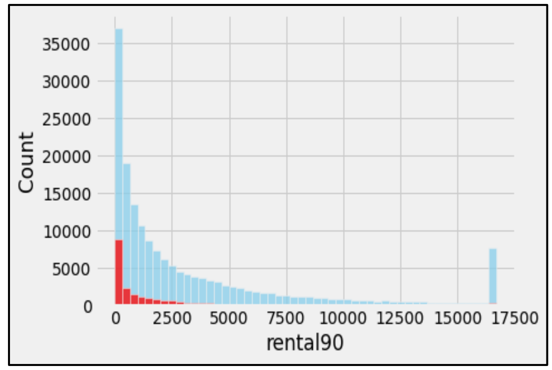


Fig. 9

Fig. 8 and Fig. 9 represent main account balance over last 30 and 90 days respectively. Most of the customers having average main account balance in both the cases in the range of 0 to 2500, the customers who are having account balance over 2500 are most likely to pay back the credit amount within estimated time. And customers with balance amount 0 are having higher rate of not paying back the credit amount within 5 days.

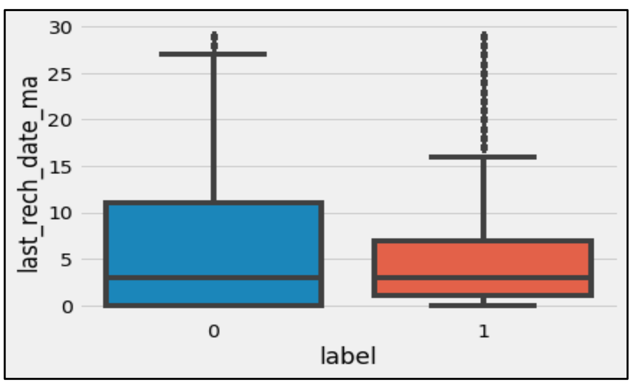


Fig. 10

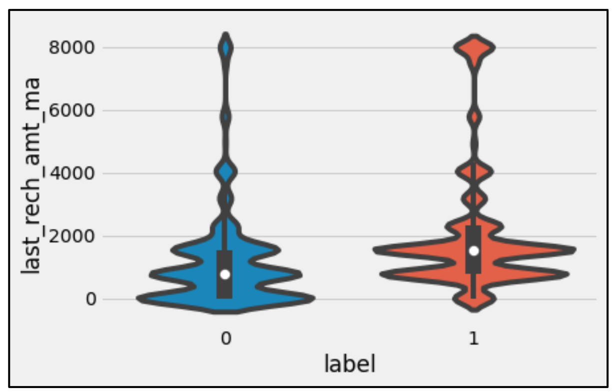


Fig. 11

Fig. 10 represents the box plot for number of days till last recharge of main account, as we see in this box plot more number of people who recharge their main account 6 to 12 days back are most likely not paying the loan amount.

Fig. 11 shows the violin plot for amount of last recharge of main account. Looking at this graph we can say that if the amount of last recharge of main account is around 2000 then more number of people will pay back the loan amount.

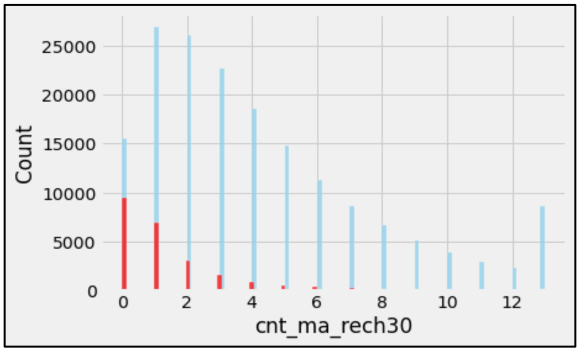


Fig 12

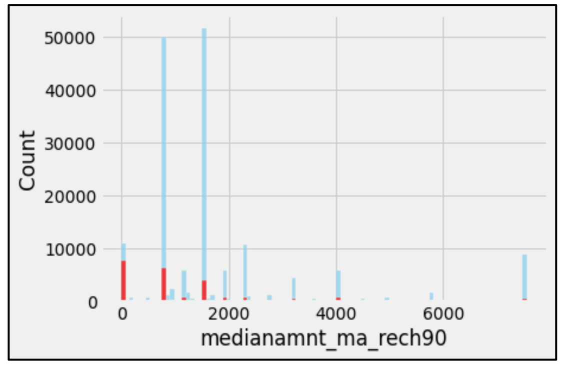


Fig 13

Fig. 12 shows histogram for number of times main account got recharged in last 30 days shows that when people didn't recharge their main account or recharges only once in 30 days the rate of not paying back the credit amount is higher compared to others.

And looking at Fig13 histogram we can see that when median of amount of recharges done in main account over last 90 days at user level is 0, the ratio of customers to not to pay back the credit amount within 5 days is high.

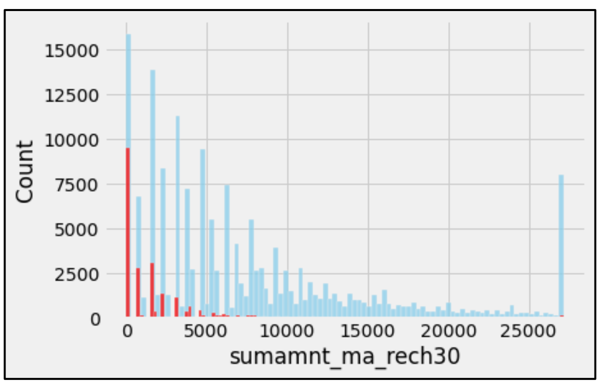


Fig 14

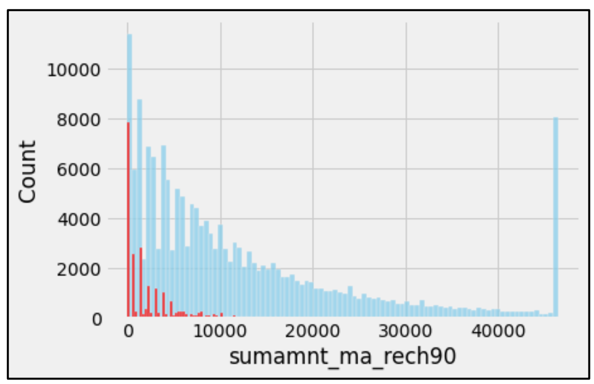


Fig 15

Both of the above figures represents total amount of recharge in main account over last 30 and 90 days respectively. Looking at these plots we can say that customers who makes total amount of recharge in main account above 10000 are mostly paying back the credit amount within 5 days. And it is observed that customers who are recharging their accounts with very less amount are most likely to not pay back the credit amount.

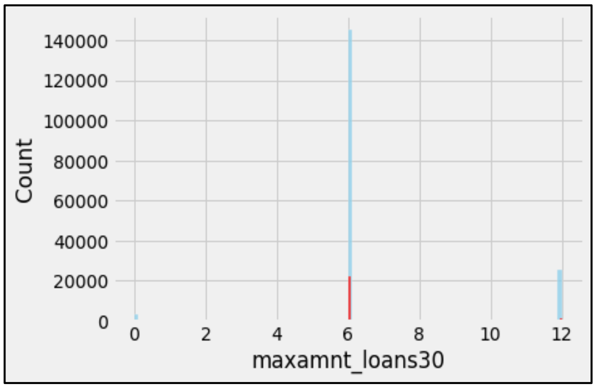


Fig 16

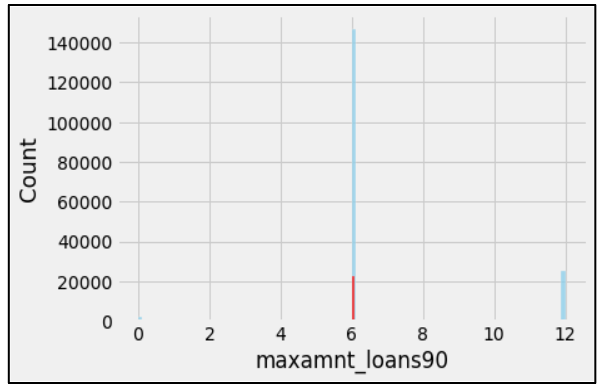


Fig 17

Both of the above plots are representing maximum amount of loan taken by the user in last 30 and 90 days respectively. Looking at these plots we can say that whenever customer takes the loan amount of 6, then only some users may not pay back the loan amount.

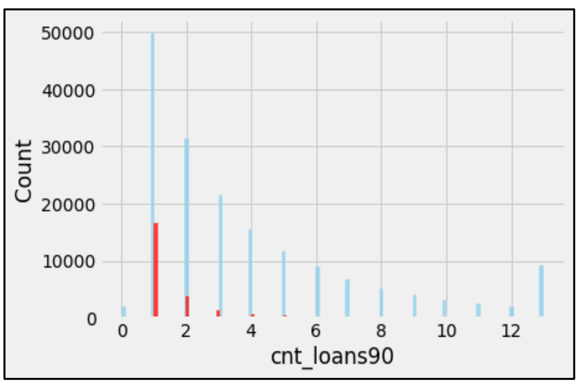


Fig 18

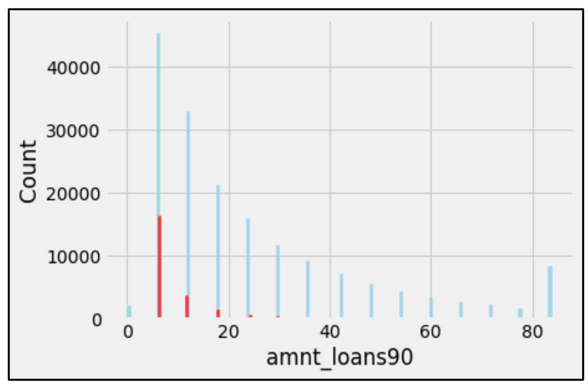


Fig 19

Fig. 18 represents number of loans taken by user in last 90 days, when a person takes loan amount once in last 90 days the chances of not paying back the credit amount within 5 days are higher.

Fig. 19 represents total amount of loans taken by user in last 90 days, looking at this plot we can conclude that when total amount of loans taken by user in last 90 days is below 10; the chances of user not paying back the credit amount are more.

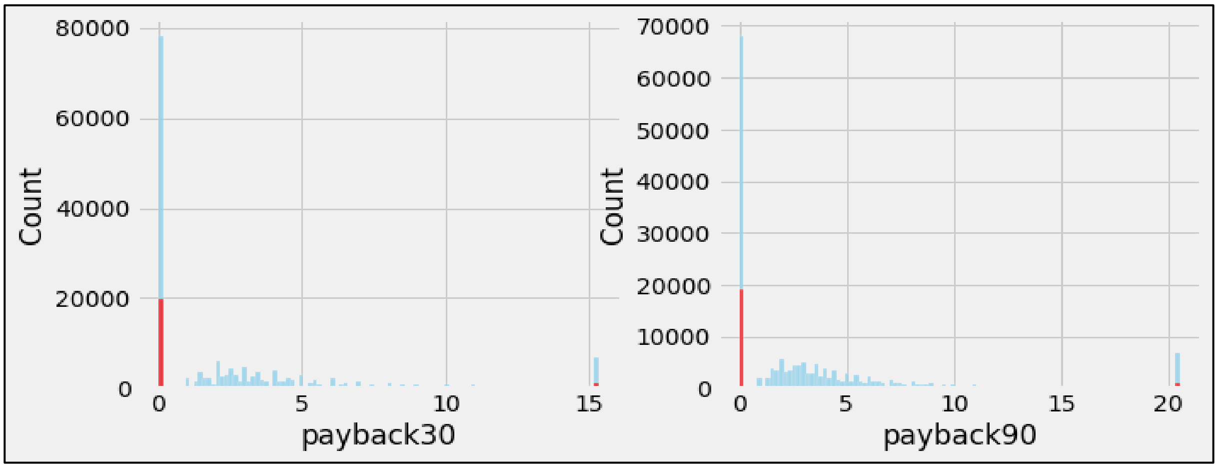


Fig 20

It is seen that when Average payback time in days over last 30 & 90 days is when zero then only customers will not pay back the loan amount

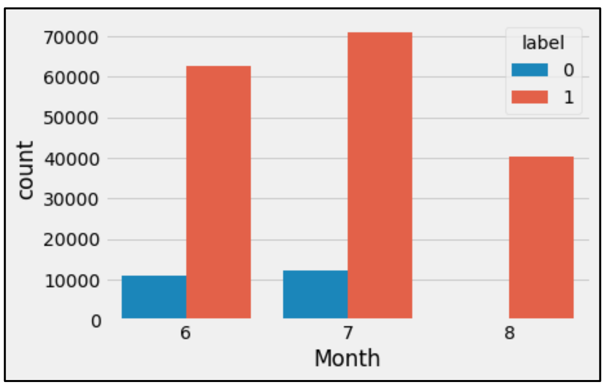
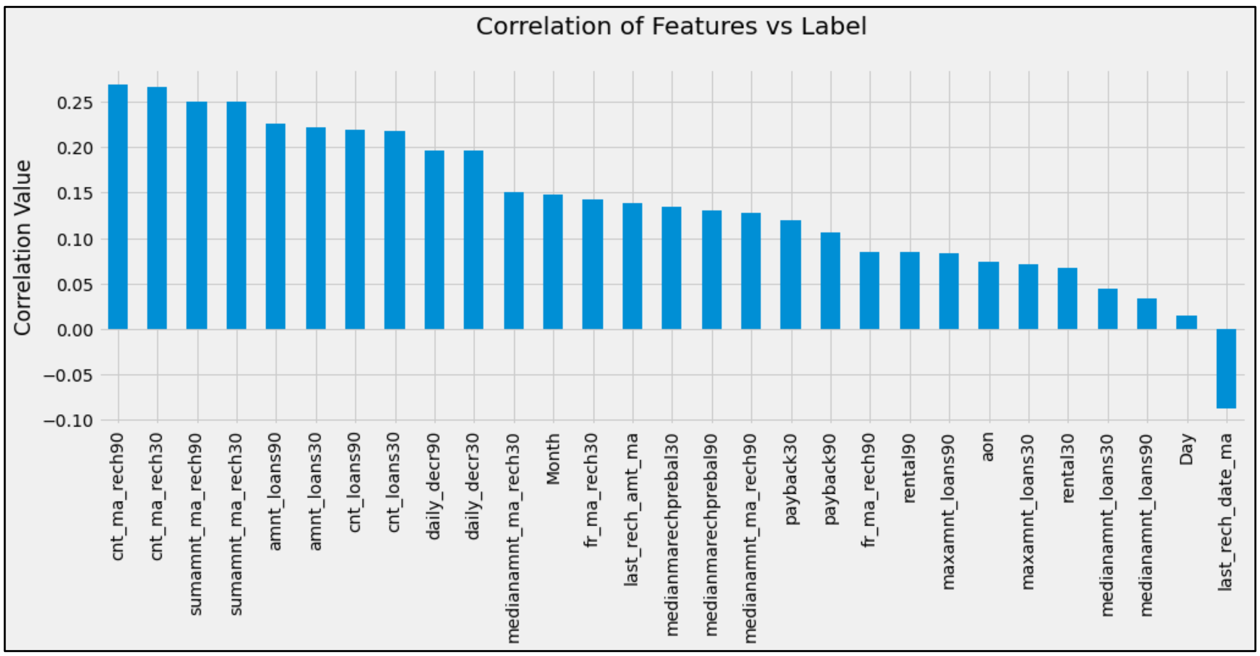


Fig 21

Fig 21 represents a count plot showing the month when customers have taken the loan. This will tell us that customers who have taken loans in the month of august they are paying back the credit amount within 5 day.

Correlation:

- I have plotted a heat map for checking correlation between different features as well as to check the correlation of features with label.
- Looking at the heat map we can say that there is no any strong relation between features with the label.
- Features like daily_decr30 & daily_decr90, rental30 & rental90 are strongly related to each other.
- Columns maxamnt_loans90 and maxamnt_loans30 have maximum correlation between them.
- The below figure shows the correlation of different features with the label



Looking at above plot we can conclude that all features are having very less correlation with label, which is below 0.30.

- cnt_ma_rech90 has maximum correlation with label.
- Column Day is having least relation with the label.
- And last_rech_date_ma is in negative relation with label.

Outlier Removing

As we have removed unrealistic data already from our data set I am not removing outliers now because we don't want to lose our data.

Skewness

```
#lets check the skewness
df.skew()
```

```
label                  -2.365223
aon                     0.830852
daily_decr30            1.578630
daily_decr90            1.671968
rental30                1.744314
rental90                1.786469
last_rech_date_ma       1.996298
last_rech_amt_ma        1.953839
cnt_ma_rech30           1.097175
fr_ma_rech30            1.384312
sumamnt_ma_rech30       1.326655
medianamnt_ma_rech30    2.077778
medianmarechprebal30    2.166324
cnt_ma_rech90           1.239206
fr_ma_rech90            1.868194
sumamnt_ma_rech90       1.444872
medianamnt_ma_rech90    2.123488
medianmarechprebal90    2.149255
cnt_loans30             1.218336
amnt_loans30            1.273042
maxamnt_loans30         1.348610
medianamnt_loans30      3.444119
cnt_loans90             1.460614
amnt_loans90            1.495790
maxamnt_loans90         1.616369
medianamnt_loans90      3.795964
payback30               1.833825
payback90               1.989670
Month                   0.281771
Day                     0.196770
```

We can see many features are having skewed data, we will remove skewness using suitable method.

5. Model Development and Evaluation

After doing data processing, cleaning and visualizing we are now with cleaned and better data for our model building.

As we can observe our target variable contains categorical data (two class 1&0), I can conclude that this is a **classification** problem.

Separate the data into features and label that is x & y respectively:

```
x = df.drop(columns = 'label')
y = df['label']
```

Reducing skewness

As we are having lot of skewed data, I am applying cube root for positively skewed data and squaring the negatively skewed data to reduce the skewnes from our features.

```
#treat the skewness
for index in x.skew().index:
    if x.skew().loc[index]>0.5:
        x[index]=np.cbrt(x[index])
    if x.skew().loc[index]<-0.5:
        x[index]=np.square(x[index])
```

Standard Scaler

StandardScaler removes the mean and scales each feature/variable to unit variance. To bring every feature data to common scale I am applying StandardScaler to Numerical features. In our cleaned dataset we are left with only numerical data.

```
#Lets bring all numerical features to common scale by applying
standard scaler
scaler = StandardScaler()
X = scaler.fit_transform(x)
X = pd.DataFrame(X,columns=x.columns)
```

Class Imbalance Problem

Data is said to suffer the Class Imbalance Problem when the target variable's class distributions are highly imbalanced. We will check for imbalance problem by checking count of label.

```
#check value count for target variable
y.value_counts()

   1    173688
   0     25624
```

Looking at the count of our label we can say the data is with class imbalance problem. To overcome this problem of imbalance I am oversampling the data suing SMOTE. Oversampling involves randomly selecting examples from the minority class, with replacement, and adding them to the training dataset.

Oversampling

```
#lets do oversampling using SMOTE

import imblearn
from imblearn.over_sampling import SMOTE
SM = SMOTE()
x_over,y_over = SM.fit_resample(X,y)
```

Using SMOTE I have oversampled our data as x_over & y_over. We will again check for count of y_over to ensure the imbalance.

```
#lets check the target variable now
y_over.value_counts()

0    173688
1    173688
```

Great we have successfully removed the problem of imbalance as the count for both the classes are same now.

Finding the best random_state:

Random state ensures that the splits that you generate are reproducible. Scikit-learn use random permutations to generate the splits. For finding the best random_state for our model I am making use of LogisticRegression model. First we will find best random_state for LogisticRegression and use same for different models.

```
#Lets find the best random state using LogisticRegression
from sklearn.linear_model import LogisticRegression
max_accu = 0
max_rs = 0
for i in range(1,200):
    x_train,x_test,y_train,y_test =
train_test_split(x_over,y_over,test_size = 0.25, random_state = i)
    LR = LogisticRegression()
    LR.fit(x_train,y_train)
    pred = LR.predict(x_test)
    acc = accuracy_score(y_test,pred)
    if acc > max_accu:
        max_accu = acc
        max_rs = i
print("Best accuracy is",max_accu,"on Random State",max_rs)
```

Code to select best random_state

By running above code we will get best random_state for LogisticRegression model, and then we will split our data into train and test sets with this random_state.

```
#lets split our data into train and test parts with best
random_state
x_train,x_test,y_train,y_test = train_test_split(x_over, y_over,
test_size = 0.25, random_state = max_rs)
```

Machine Learning model with Evaluation metrics

I am defining a function which will take classification algorithms as a parameter and will perform the train-test split, train the model on training data and will make prediction test features. Accuracy score will be measured by comparing predictions with actual test values.

As we are dealing with binary target variable the roc_auc_score will be an important factor to take into consideration so I am calculating roc_auc_score as well.

Getting Confusion matrix and classification report as evaluation metrics and cross validation score to test the ability of a machine learning model to predict new data. And at last calculation of difference between accuracy score and cross-validation scores for checking suitability of our model.

```
def BuiltModel(model):
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    accuracy = accuracy_score(y_test,pred)*100

    print(f"Accuracy Score:", accuracy)
    print(f"roc_auc_score: {roc_auc_score(y_test,pred)*100}")
    print("---------------------------------------------------")

    #confusion matrix & classification report
    print(f"Confusion Matrix : \n {confusion_matrix(y_test,pred)}\n")
    print(f"CLASSIFICATION REPORT : \n
{classification_report(y_test,pred)}")

    #cross validation score
    scores = cross_val_score(model, x_over, y_over, cv = 5,scoring =
"accuracy" ).mean()*100
    print("\nCross validation score :", scores)

    #result of accuracy minus cv score
    result = accuracy - scores
    print("\nAccuracy Score - Cross Validation Score :", result)
```

Code for function

After defining the function I am calling different functions as model and will check the respective results and compare all of them and select a best suitable model for hyper parameter tuning.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
BuiltModel(lr)
```

```
Accuracy Score: 77.90676290745253
roc_auc_score: 77.8969091737288
---------------------------------------------------
Confusion Matrix :
 [[34964  8742]
 [10480 32818]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.77      0.80      0.78     43706
           1       0.79      0.76      0.77     43298

    accuracy                           0.78     87004
   macro avg       0.78      0.78      0.78     87004
weighted avg       0.78      0.78      0.78     87004


Cross validation score : 77.58379029312898

Accuracy Score - Cross Validation Score : 0.32297261432354674
```

DecisionTreeClassifier Model

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
BuiltModel(dt)
```

```
Accuracy Score: 91.68773849478184
roc_auc_score: 91.68466664042954
---------------------------------------------------
Confusion Matrix :
 [[40358  3348]
 [ 3884 39414]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.91      0.92      0.92     43706
           1       0.92      0.91      0.92     43298

    accuracy                           0.92     87004
   macro avg       0.92      0.92      0.92     87004
weighted avg       0.92      0.92      0.92     87004


Cross validation score : 91.28289248484498

Accuracy Score - Cross Validation Score : 0.40484600993686115
```

RandomForestClassifier Model

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
BuiltModel(rf)
```

```
Accuracy Score: 95.6197416210749
roc_auc_score: 95.6179689936958
---------------------------------------------------
Confusion Matrix :
 [[41956  1750]
 [ 2061 41237]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.95      0.96      0.96     43706
           1       0.96      0.95      0.96     43298

    accuracy                           0.96     87004
   macro avg       0.96      0.96      0.96     87004
weighted avg       0.96      0.96      0.96     87004


Cross validation score : 95.25598496768313

Accuracy Score - Cross Validation Score : 0.36375665339176066
```

XGBClassifier Model

```
from xgboost import XGBClassifier
xgb = XGBClassifier(verbosity = 0)
BuiltModel(xgb)
```

```
Accuracy Score: 95.33239851041331
roc_auc_score: 95.33727086240458
---------------------------------------------------
Confusion Matrix :
 [[41214  2492]
 [ 1569 41729]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.96      0.94      0.95     43706
           1       0.94      0.96      0.95     43298

    accuracy                           0.95     87004
   macro avg       0.95      0.95      0.95     87004
weighted avg       0.95      0.95      0.95     87004


Cross validation score : 93.85463556020021

Accuracy Score - Cross Validation Score : 1.4777629502131049
```

SGDClassifier Model

```python
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
BuiltModel(sgd)
```

```
Accuracy Score: 77.48954071077192
roc_auc_score: 77.47792604165791
---------------------------------------------------
Confusion Matrix :
 [[34945  8761]
 [10824 32474]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.76      0.80      0.78     43706
           1       0.79      0.75      0.77     43298

    accuracy                           0.77     87004
   macro avg       0.78      0.77      0.77     87004
weighted avg       0.78      0.77      0.77     87004


Cross validation score : 77.24903517288351

Accuracy Score - Cross Validation Score : 0.24050553788841
```

ExtraTreeClassifier Model

```python
from sklearn.ensemble import ExtraTreesClassifier
ext = ExtraTreesClassifier()
BuiltModel(ext)
```

```
Accuracy Score: 96.50131028458462
roc_auc_score: 96.49484076682288
---------------------------------------------------
Confusion Matrix :
 [[42777   929]
 [ 2115 41183]]

CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.95      0.98      0.97     43706
           1       0.98      0.95      0.96     43298

    accuracy                           0.97     87004
   macro avg       0.97      0.96      0.97     87004
weighted avg       0.97      0.97      0.97     87004


Cross validation score : 96.61883674843732

Accuracy Score - Cross Validation Score : -0.11752646385269827
```

LGBMClassifier Model

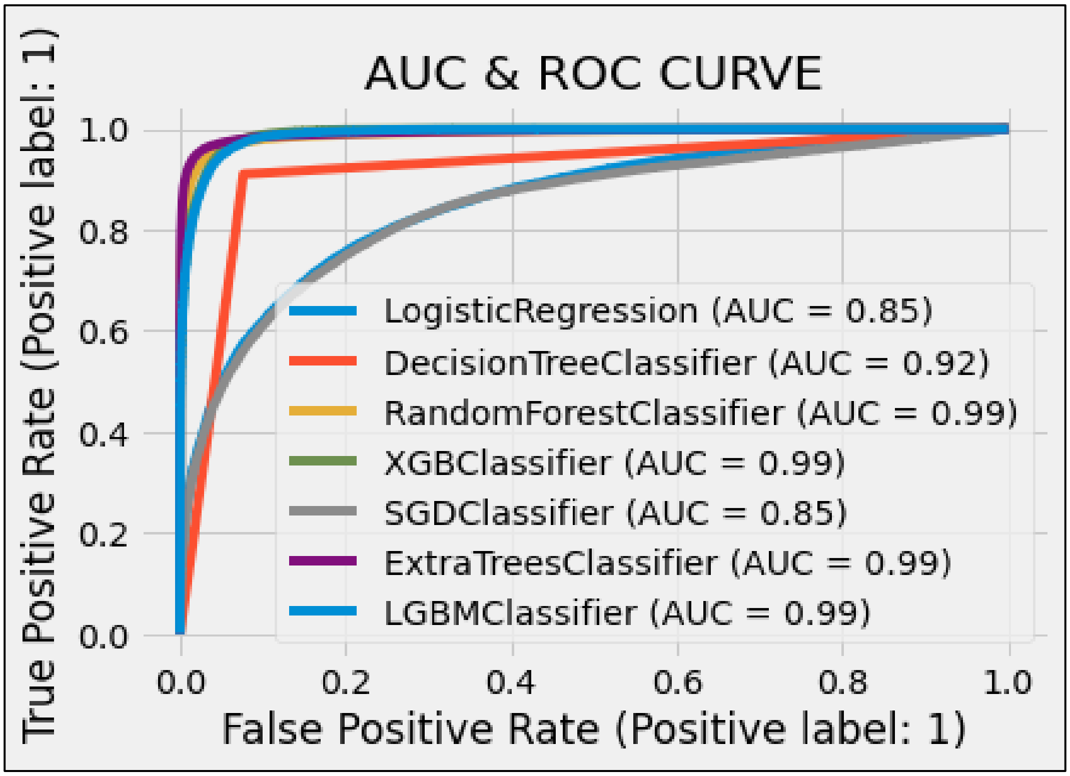
```
from lightgbm import LGBMClassifier
lgbm = LGBMClassifier()
BuiltModel(lgbm)
```

```
Accuracy Score: 94.95540434922532
roc_auc_score: 94.95845736015167
--------------------------------------------------
Confusion Matrix :
 [[41218  2488]
 [ 1901 41397]]

CLASSIFICATION REPORT :
              precision    recall  f1-score   support

           0       0.96      0.94      0.95     43706
           1       0.94      0.96      0.95     43298

    accuracy                           0.95     87004
   macro avg       0.95      0.95      0.95     87004
weighted avg       0.95      0.95      0.95     87004


Cross validation score : 93.72963612195421

Accuracy Score - Cross Validation Score : 1.2257682272711037
```

Important observations

Algorithm	Accuracy score	Roc auc score	Cross-val score	Accuracy score – CV score
LogisticRegression	77.90	77.89	77.58	0.32
Decision tree	91.68	91.68	91.28	0.40
Random forest	95.61	95.61	95.25	0.36
Xgb classifier	95.33	95.33	93.85	1.44
SGD Classifier	77.48	77.46	77.24	0.24
Extra tree	96.50	96.49	96.61	-0.117
LGBMClassifier	94.95	94.95	93.72	1.22

AUC & ROC Curve



Looking at observations we came to know that ExtraTreeClassifier is giving highest accuracy score as well as least difference in accuracy and cv-score.

By observing AUC & ROC curve we can say that the model performance that means AUC of xgbclassifier, random forest, extra tree and LGBMClassifier is almost same. And among these four algorithms LGBMClassifier is giving better performance after hyperparameter tuning. So I have selected LGBMClassifier for our final model.

Hyperparameter Tuning:

Below you can see the code of the hyperparamter tuning for the parameters boosting_type, n_estimators, max_depth, and importance_type.

```
#lets selects different parameters for tuning
params ={
       'boosting_type': ['gbdt','dart'],
       'n_estimators':[100,200,500,700],
       'max_depth': [-1,1,2,3],
       'importance_type': ['split','gain']

        }

#train the model with given parameters using RandomizedSearchCV
RCV =  RandomizedSearchCV(LGBMClassifier(), params, cv = 3)
RCV.fit(x_train,y_train)

RCV.best_params_        #printing the best parameters
```

After running above code we will get best parameters for our final model.

```
]{'n_estimators': 700,
  'max_depth': -1,
  'importance_type': 'gain',
  'boosting_type': 'dart'}
```

Final Model

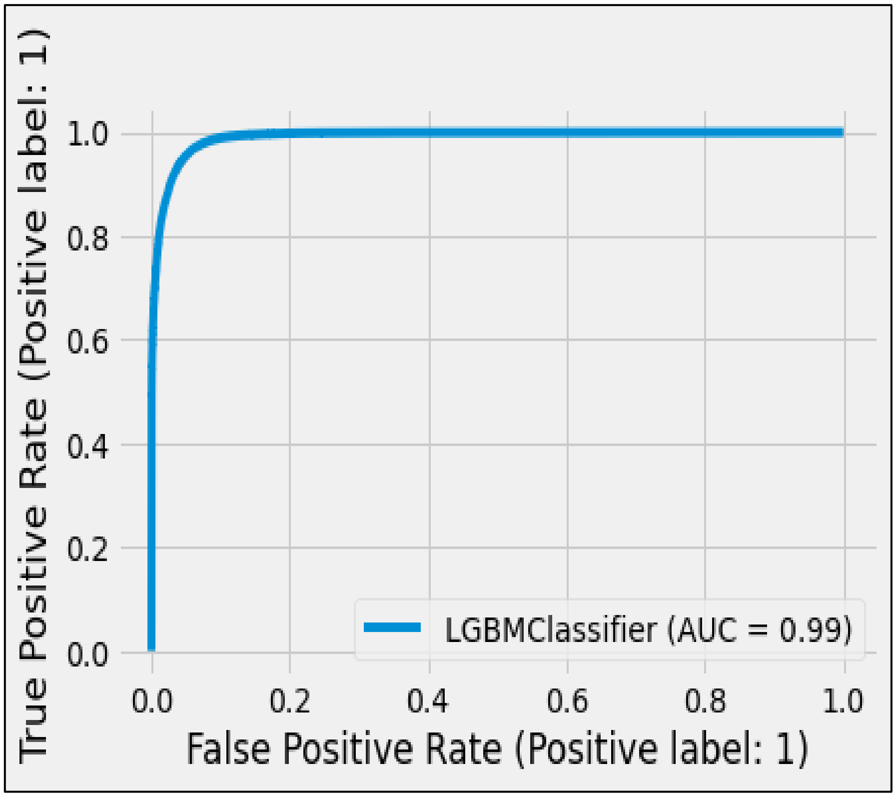
Now we will train our final model with LGBMClassifier using best parameters.

```
model = LGBMClassifier(importance_type = 'gain', max_depth = -
1,boosting_type = 'dart',n_estimators = 700)
model.fit(x_train,y_train)
pred = model.predict(x_test)

print(f"Accuracy Score: {accuracy_score(y_test,pred)*100}%")
print("--------------------------------------------------------")

print(f"roc_auc_score: {roc_auc_score(y_test,pred)*100}%")
print("--------------------------------------------------------")

print(f"Confusion Matrix : \n {confusion_matrix(y_test,pred)}\n")
print("-----------------------------------------------------------------
------------")
print(f"CLASSIFICATION REPORT : \n
{classification_report(y_test,pred)}")
```

```
Accuracy Score: 95.34389223483977%
--------------------------------------------------------
roc_auc_score: 95.34870015921962%
--------------------------------------------------------
Confusion Matrix :
 [[41225  2481]
 [ 1570 41728]]

---------------------------------------------------------------
CLASSIFICATION REPORT :
               precision    recall  f1-score   support

           0       0.96      0.94      0.95     43706
           1       0.94      0.96      0.95     43298

    accuracy                           0.95     87004
   macro avg       0.95      0.95      0.95     87004
weighted avg       0.95      0.95      0.95     87004
```

Great we have improved our model accuracy from 94.95% to 95.34%.

AUC ROC Curve for final model



Above plot shows roc curve and area under the curve for our final model.

6. Conclusion

We started with loading the dataset, and checked for missing values; luckily we don't have any missing values. But when we observed the data by describing and visualizing it we came to know that lot of the data is unrealistic. I dropped negative data and for the columns which are having too large values I used percentile method to replace those large values.

I used matplotlib and seaborn to visualize the data. And while data processing I created new columns and have dropped unwanted columns. After removing skewness and scaling the data I have build Machine learning model and tested with different algorithms and selected a best model with LGBMClassifier algorithm.

As the data set is with many unrealistic values, there is still room for improvement, like doing a more extensive feature engineering, by comparing and plotting the features against each other and identifying and removing the noisy features. Another thing that can improve the overall result is to do more extensive hyperparameter tuning on several machine learning models.