FLIP ROBO

Micro-Credit Defaulter

Submitted by:

HEMANT PATAR

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to Datatarained & FlipRobo who gave me the golden opportunity to do this internship project on the topic (Micro-Credit Defaulter), which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.

The sample data is provided to us from FlipRobo’s client database. Kaggle & Github are the websites which helped me in completing the project.

INTRODUCTION

- ## Business Problem

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

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. Here we need to build 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.

- ## Conceptual Background of the Domain Problem

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 are very useful in such areas, the implementation of MFS has been uneven with both significant challenges and successes. Here the client that is in Telecom Industry is 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).

- ## Review of Literature

Microfinance is a banking service provided to unemployed or low-income individuals or groups who otherwise would have no other access to **financial** services. **Microfinance** allows people to take on reasonable small business loans safely, and in a manner that is consistent with ethical **lending** practices.

- ## Motivation for the Problem Undertaken

With the help of this project deserved people will get loan more easily & quickly. Being a part of this project and reducing poverty is a proud feeling & motivation.

Analytical Problem Framing

- Data Sources and their formats

  The sample data is provided to us from FlipRobo's client database.

In [ ]:

```
#Load dataset
df = pd.read_csv('Micro Credit Data.csv')
df.drop('Unnamed: 0',axis='columns', inplace=True)
df
```

Out[8]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30
0	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2
1	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1
2	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1
3	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0
4	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7

Variable	Definition
label	Flag indicating whether the user paid back the credit amount within 5 days of issuing the loan{1:success, 0:failure}
msisdn	mobile number of user
aon	age on cellular network in days
daily_decr30	Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)
daily_decr90	Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah)
rental30	Average main account balance over last 30 days
rental90	Average main account balance over last 90 days
last_rech_date_ma	Number of days till last recharge of main account
last_rech_date_da	Number of days till last recharge of data account
last_rech_amt_ma	Amount of last recharge of main account (in Indonesian Rupiah)
cnt_ma_rech30	Number of times main account got recharged in last 30 days
fr_ma_rech30	Frequency of main account recharged in last 30 days
sumamnt_ma_rech30	Total amount of recharge in main account over last 30 days (in Indonesian Rupiah)
medianamnt_ma_rech30	Median of amount of recharges done in main account over last 30 days at user level (in Indonesian Rupiah)
medianmarechprebal30	Median of main account balance just before recharge in last 30 days at user level (in Indonesian Rupiah)
cnt_ma_rech90	Number of times main account got recharged in last 90 days
fr_ma_rech90	Frequency of main account recharged in last 90 days
sumamnt_ma_rech90	Total amount of recharge in main account over last 90 days (in Indonesian Rupiah)
medianamnt_ma_rech90	Median of amount of recharges done in main account over last 90 days at user level (in Indonesian Rupiah)
medianmarechprebal90	Median of main account balance just before recharge in last 90 days at user level (in Indonesian Rupiah)
cnt_da_rech30	Number of times data account got recharged in last 30 days
fr_da_rech30	Frequency of data account recharged in last 30 days
cnt_da_rech90	Number of times data account got recharged in last 90 days
fr_da_rech90	Frequency of data account recharged in last 90 days
cnt_loans30	Number of loans taken by user in last 30 days
amnt_loans30	Total amount of loans taken by user in last 30 days
maxamnt_loans30	maximum amount of loan taken by the user in last 30 days
medianamnt_loans30	Median of amounts of loan taken by the user in last 30 days
cnt_loans90	Number of loans taken by user in last 90 days
amnt_loans90	Total amount of loans taken by user in last 90 days
maxamnt_loans90	maximum amount of loan taken by the user in last 90 days
medianamnt_loans90	Median of amounts of loan taken by the user in last 90 days
payback30	Average payback time in days over last 30 days
payback90	Average payback time in days over last 90 days
pcircle	telecom circle
pdate	date

```
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   label                 209593 non-null  int64
 1   msisdn                209593 non-null  object
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64
 10  cnt_ma_rech30         209593 non-null  int64
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64
 16  fr_ma_rech90          209593 non-null  int64
 17  sumamnt_ma_rech90     209593 non-null  int64
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64
 23  fr_da_rech90          209593 non-null  int64
 24  cnt_loans30           209593 non-null  int64
 25  amnt_loans30          209593 non-null  int64
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64
 30  maxamnt_loans90       209593 non-null  int64
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
 34  pcircle               209593 non-null  object
 35  pdate                 209593 non-null  object
dtypes: float64(21), int64(12), object(3)
memory usage: 57.6+ MB
```

- Mathematical/ Analytical Modeling of the Problem

  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. In the provided **dataset**, our target variable "label" is a **categorical** with two categories: " defaulter " and " Non-defaulter ". Therefore we will be handling this modelling problem as classification.

- Hardware and Software Requirements and Tools Used

The version of the notebook server is: **6.0.3**
The server is running on this version of Python:

```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
```

Current Kernel Information:

```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
```

Windows edition

Windows 10 Pro

System

Processor:	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz 2.40 GHz
Installed memory (RAM):	4.00 GB (3.90 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	lenovo

Model training was done on Google colab as the dataset was huge.

- Data Preprocessing Done

```
In [ ]:
#frequency of object features
for col in df.columns:
    if df[col].dtype=="object":
        print(df[col].value_counts())
        print()
```

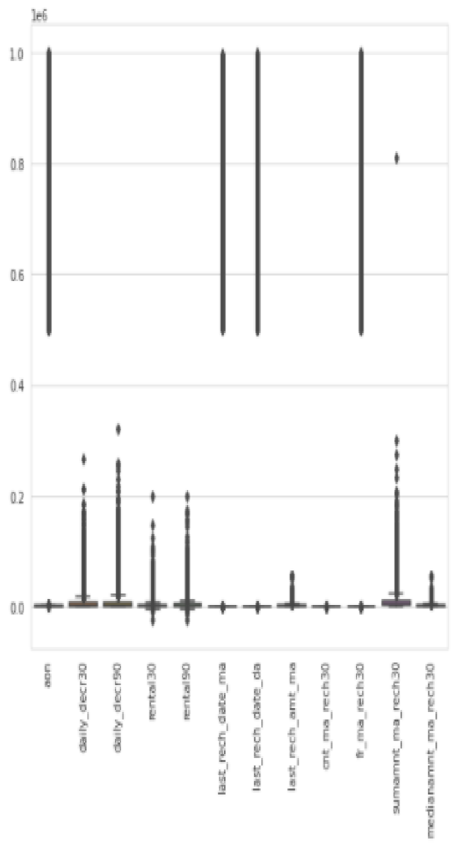
```
47819I90840    7
04581I85330    7
43096I88688    6
87592I84456    6
71742I90843    6
              ..
38920I90586    1
91722I89230    1
19075I70780    1
48565I70371    1
18593I88680    1
Name: msisdn, Length: 186243, dtype: int64

UPW    209593
Name: pcircle, dtype: int64

2016-07-04    3150
2016-07-05    3127
2016-07-07    3116
2016-06-20    3099
```

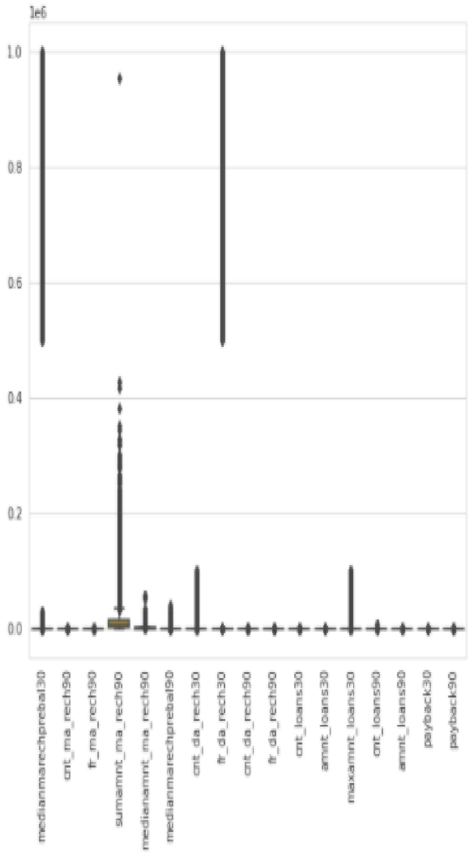
```
In [ ]:
#we can drop some features for further processing
df.drop(['pdate','pcircle','msisdn'],axis='columns', inplace=True)
df
```

In [ ]:

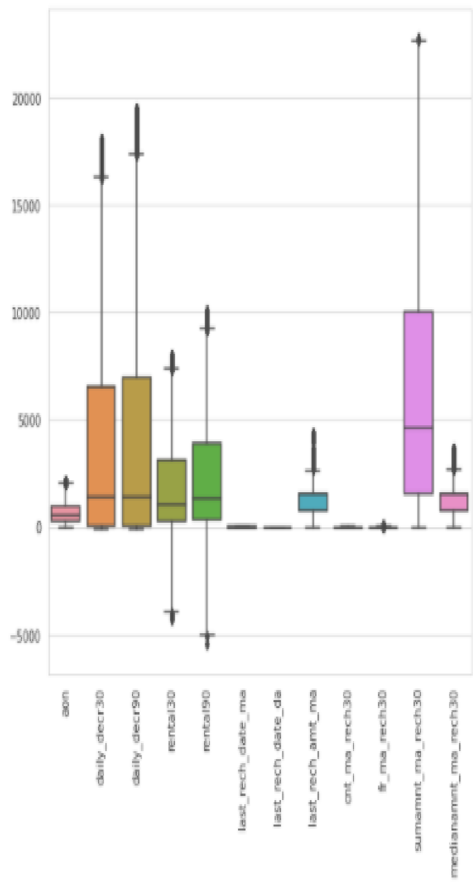
```python
# Outlier points
lis=['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90',
       'last_rech_date_ma', 'last_rech_date_da', 'last_rech_amt_ma',
       'cnt_ma_rech30', 'fr_ma_rech30', 'sumamnt_ma_rech30',
       'medianamnt_ma_rech30', 'medianmarechprebal30', 'cnt_ma_rech90',
       'fr_ma_rech90', 'sumamnt_ma_rech90', 'medianamnt_ma_rech90',
       'medianmarechprebal90', 'cnt_da_rech30', 'fr_da_rech30',
       'cnt_da_rech90', 'fr_da_rech90', 'cnt_loans30', 'amnt_loans30',
       'maxamnt_loans30', 'cnt_loans90', 'amnt_loans90',
         'payback30', 'payback90']
q3 = df[lis].quantile(0.75)
q1 = df[lis].quantile(0.25)
iqr = q3 - q1
print('IQR for numerical attributes')
print(iqr)
```

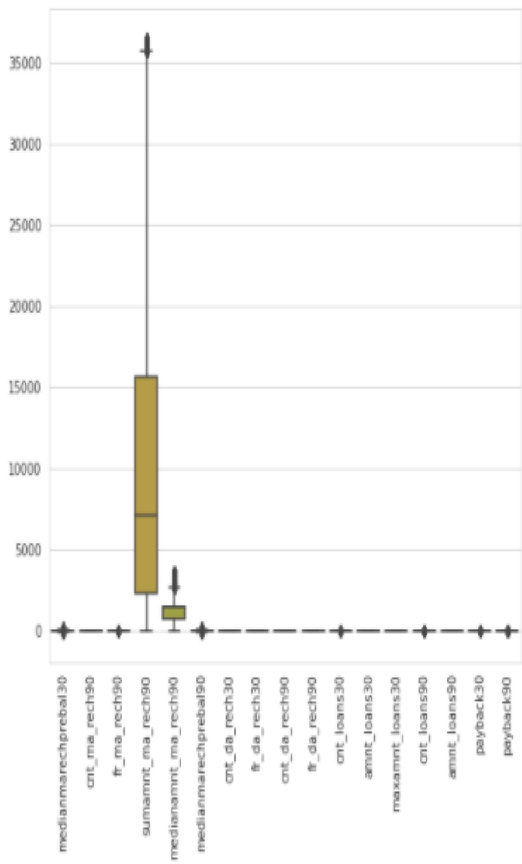
```
IQR for numerical attributes
aon                       736.000
daily_decr30             7201.560
daily_decr90             7760.098
rental30                 3076.520
rental90                 3901.530
last_rech_date_ma           6.000
last_rech_date_da           0.000
last_rech_amt_ma         1539.000
cnt_ma_rech30               4.000
fr_ma_rech30                6.000
sumamnt_ma_rech30        8470.000
medianamnt_ma_rech30     1154.000
medianmarechprebal30       72.000
cnt_ma_rech90               6.000
fr_ma_rech90                8.000
sumamnt_ma_rech90       13683.000
medianamnt_ma_rech90     1151.000
medianmarechprebal90       64.710
cnt_da_rech30               0.000
fr_da_rech30                0.000
cnt_da_rech90               0.000
fr_da_rech90                0.000
cnt_loans30                 3.000
amnt_loans30               18.000
maxamnt_loans30             0.000
cnt_loans90                 4.000
amnt_loans90               24.000
payback30                   3.750
payback90                   4.500
dtype: float64
```



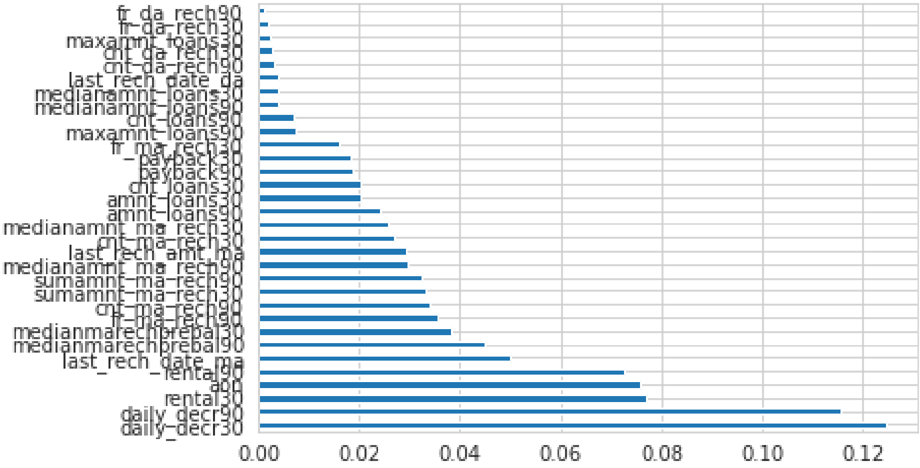


After processing the outliers.





Feature importance



In [ ]:

```
#we can drop less important featues for further process
df.drop(['last_rech_date_da','cnt_da_rech30','fr_da_rech30','cnt_da_rech90','fr_da_rech90','maxamnt_loans30' ],axis='
df
```

In [ ]:

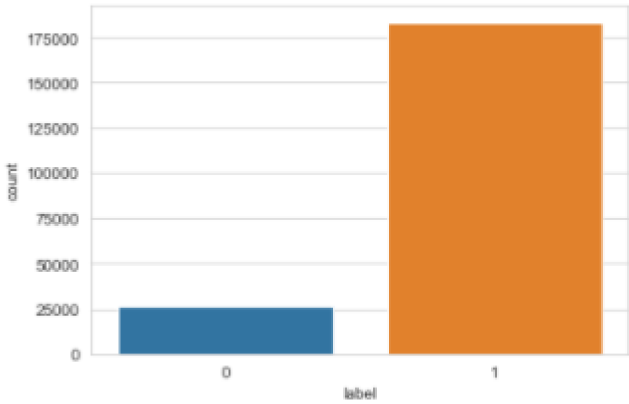
```
#Normalization of continuous variables
for i in ['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90',
          'last_rech_date_ma', 'last_rech_amt_ma', 'cnt_ma_rech30',
          'fr_ma_rech30', 'sumamnt_ma_rech30', 'medianamnt_ma_rech30',
          'medianmarechprebal30', 'cnt_ma_rech90', 'fr_ma_rech90',
          'sumamnt_ma_rech90', 'medianamnt_ma_rech90', 'medianmarechprebal90',
          'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30', 'cnt_loans90',
          'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30',
          'payback90']:

    df[i] = (df[i] - df[i].min())/(df[i].max()-df[i].min())
```

```
In [ ]: df.describe().transpose()
```

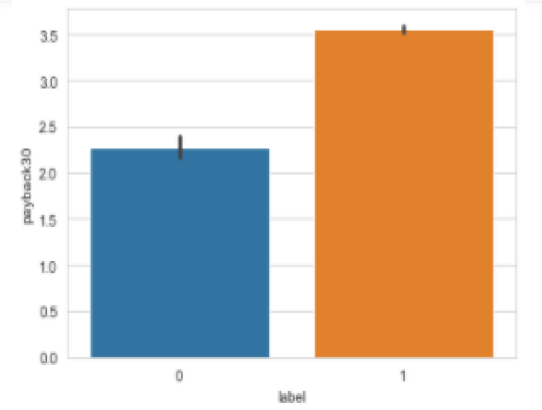
Out[12]:

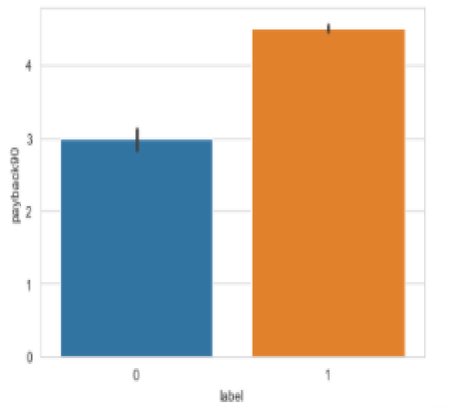
	count	mean	std	min	25%	50%	75%	max
label	209593.0	0.875177	0.330519	0.0	1.000000	1.000000	1.000000	1.0
aon	209593.0	0.324747	0.225261	0.0	0.137301	0.266167	0.468135	1.0
daily_decr30	209593.0	0.214313	0.263643	0.0	0.007466	0.082364	0.365838	1.0
daily_decr90	209593.0	0.214793	0.269805	0.0	0.006945	0.077857	0.362279	1.0
rental30	209593.0	0.506820	0.169295	0.0	0.373573	0.438048	0.605738	1.0
rental90	209593.0	0.505520	0.167879	0.0	0.372065	0.437330	0.603056	1.0
last_rech_date_ma	209593.0	0.501578	0.154866	0.0	0.375000	0.458333	0.625000	1.0
last_rech_amt_ma	209593.0	0.336278	0.227863	0.0	0.177419	0.354608	0.356452	1.0
cnt_ma_rech30	209593.0	0.319405	0.264746	0.0	0.090909	0.272727	0.454545	1.0
fr_ma_rech30	209593.0	0.207787	0.252945	0.0	0.000000	0.133333	0.333333	1.0
sumamnt_ma_rech30	209593.0	0.281789	0.254318	0.0	0.067797	0.203654	0.440238	1.0
medianamnt_ma_rech30	209593.0	0.348047	0.214863	0.0	0.212121	0.423967	0.425069	1.0
medianmarechprebal30	209593.0	0.485819	0.147903	0.0	0.373432	0.438676	0.560976	1.0
cnt_ma_rech90	209593.0	0.318655	0.270754	0.0	0.117647	0.235294	0.470588	1.0
fr_ma_rech90	209593.0	0.185963	0.221374	0.0	0.000000	0.100000	0.300000	1.0
sumamnt_ma_rech90	209593.0	0.278192	0.259373	0.0	0.063445	0.195674	0.429819	1.0
medianamnt_ma_rech90	209593.0	0.355862	0.203068	0.0	0.212948	0.423967	0.426171	1.0
medianmarechprebal90	209593.0	0.485305	0.150291	0.0	0.374448	0.441327	0.562582	1.0
cnt_loans30	209593.0	0.309081	0.223789	0.0	0.125000	0.250000	0.375000	1.0
amnt_loans30	209593.0	0.326416	0.232219	0.0	0.125000	0.250000	0.500000	1.0
medianamnt_loans30	209593.0	0.018010	0.072680	0.0	0.000000	0.000000	0.000000	1.0
cnt_loans90	209593.0	0.283796	0.228244	0.0	0.090909	0.181818	0.363636	1.0
amnt_loans90	209593.0	0.296968	0.234108	0.0	0.090909	0.181818	0.454545	1.0
maxamnt_loans90	209593.0	0.558595	0.175322	0.0	0.500000	0.500000	0.500000	1.0
medianamnt_loans90	209593.0	0.015359	0.066897	0.0	0.000000	0.000000	0.000000	1.0
payback30	209593.0	0.175204	0.239323	0.0	0.000000	0.000000	0.320000	1.0
payback90	209593.0	0.182883	0.232048	0.0	0.000000	0.111111	0.302222	1.0

- Data Inputs- Logic- Output Relationships Visualizations

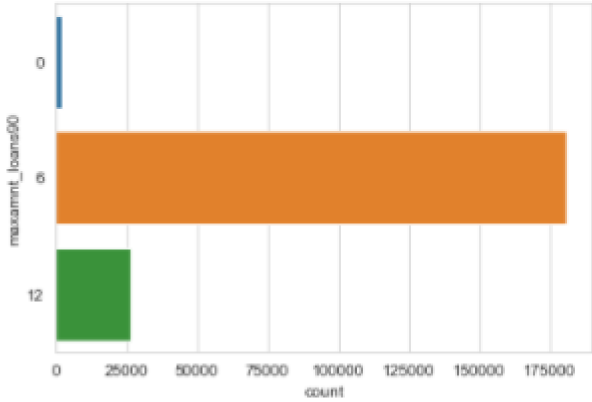


The users that didn't paid back the credit amount within 5 days is around 1/8th of the total people who took loan.

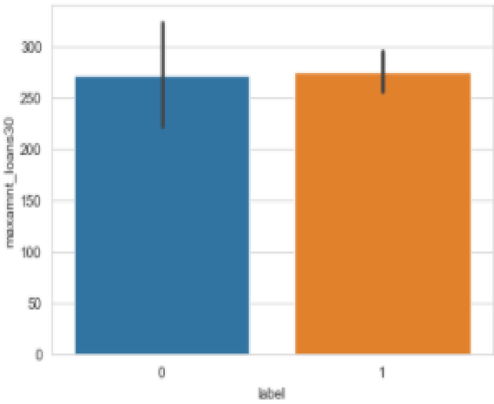


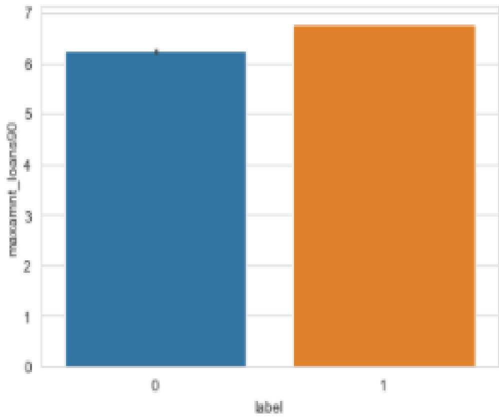


Average loan payback time is 3-4 days.

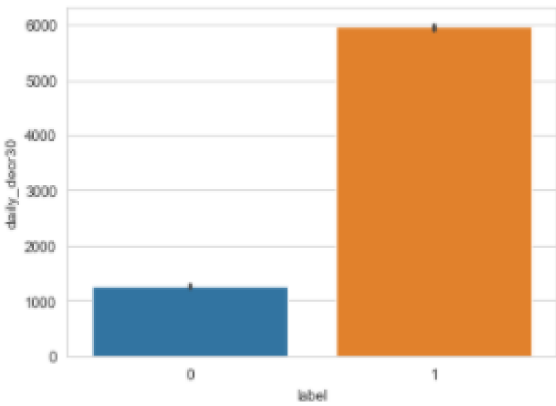


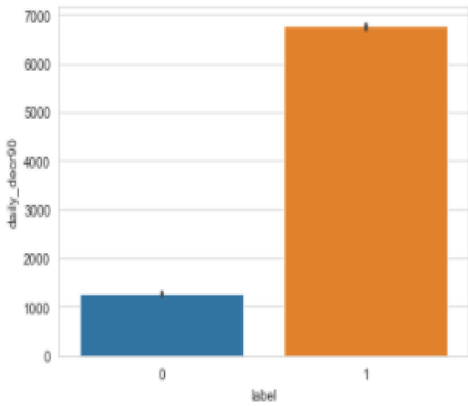
There are only two options: 5 & 10 Rs., for which the user needs to pay back 6 & 12 Rs.



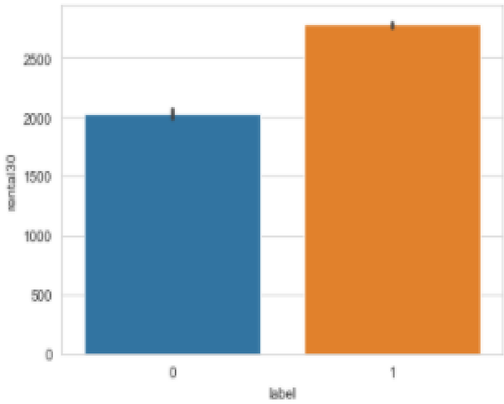


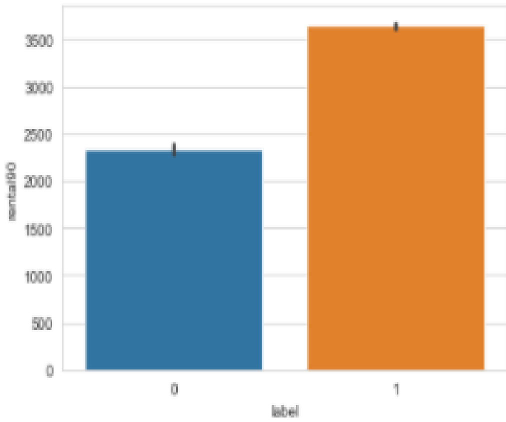
We also see outliers present in maximum amount loan taken in 30 days. And 50%



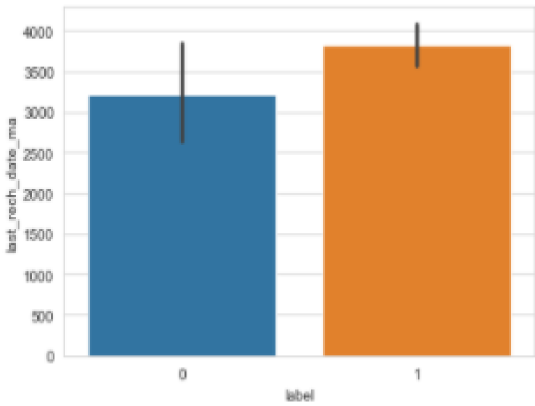


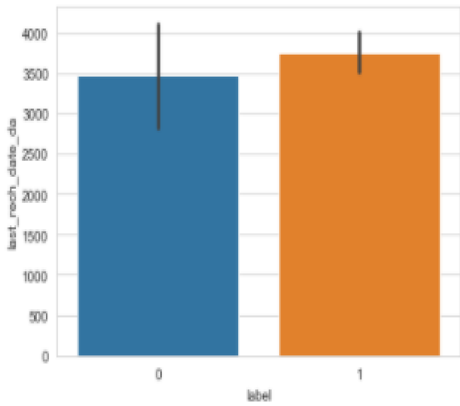
#non defaulters spent 6 times higher daily amount from main account within 30 days
#non defaulters spent 7 times higher daily amount from main account within 90 days



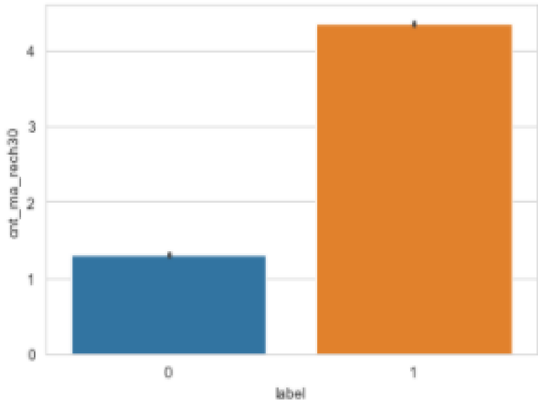


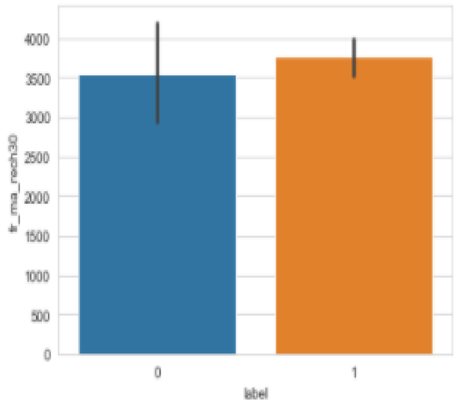
Average main account balance is high for non defaulters





Number of days till last recharge of main account & data account is higher for non defaulters. Outliers are present.





Number of times main account got recharged is higher for non defaulters in last 30 days. Frequency of main account recharged in last 30 days is slight higher for non defaulter and significant amount of outliers are present.

Model/s Development and Evaluation

- Testing of Identified Approaches (Algorithms)

Model
Random Forest
Gradient Boosting Trees
Logistic Regression
Linear SVC
Stochastic Gradient Decent
KNN
Decision Tree
Naive Bayes

- Run and Evaluate selected models

In [ ]:
```
#Lets divide the dataset into input and output
df_x=df.drop(columns=["label"])
y=df[["label"]]
```

In [ ]:
```
#preprocessing standardisation of dataset.
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
scaled_df_x=scaler.fit_transform(df_x)
X=scaled_df_x
```

In [ ]:
```
#split train and test dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

In [ ]:
```
#check shape of train dataset
X_train.shape
```

Out[11]:
```
(146715, 26)
```

In [ ]:
```

# Logistic Regression
import datetime
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log = fit_ml_algo(LogisticRegression(
log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))
```

```
Accuracy: 87.72
Accuracy CV 10-Fold: 87.89
Running Time: 0:00:20.492116
```

In [ ]:
```
print (metrics.classification_report(y_train, train_pred_log) )
```

```
              precision    recall  f1-score   support

         0.0       0.61      0.07      0.13     18260
         1.0       0.88      0.99      0.93    128455

    accuracy                           0.88    146715
   macro avg       0.75      0.53      0.53    146715
weighted avg       0.85      0.88      0.83    146715
```

In [ ]:
```
print (metrics.classification_report(y_test, test_pred_log) )
```

```
              precision    recall  f1-score   support

         0.0       0.60      0.07      0.12      7902
         1.0       0.88      0.99      0.93     54976

    accuracy                           0.88     62878
   macro avg       0.74      0.53      0.53     62878
weighted avg       0.85      0.88      0.83     62878
```

In [ ]:

```python
# k-Nearest Neighbors
start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn = fit_ml_algo(KNeighborsClassifier(n_neighbors = 3,
                                                                                                  n_jobs = -1),
                                                                                                  X_train,
                                                                                                  y_train,
                                                                                                  X_test,
                                                                                                  10)
knn_time = (time.time() - start_time)
print("Accuracy: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))
```

```
Accuracy: 87.12
Accuracy CV 10-Fold: 86.83
Running Time: 0:32:55.087251
```

In [ ]:

```python
print (metrics.classification_report(y_train, train_pred_knn) )
```

```
              precision    recall  f1-score   support

         0.0       0.46      0.35      0.40     18260
         1.0       0.91      0.94      0.93    128455

    accuracy                           0.87    146715
   macro avg       0.69      0.64      0.66    146715
weighted avg       0.85      0.87      0.86    146715
```

In [ ]:

```python

print (metrics.classification_report(y_test, test_pred_knn) )
```

```
              precision    recall  f1-score   support

         0.0       0.48      0.35      0.41      7902
         1.0       0.91      0.95      0.93     54976

    accuracy                           0.87     62878
   macro avg       0.70      0.65      0.67     62878
weighted avg       0.86      0.87      0.86     62878
```


In [ ]:

```python
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian, probs_gau = fit_ml_algo(GaussianNB(),
                                                                                                 X_train,
                                                                                                 y_train,
                                                                                                 X_test,
                                                                                                 10)
gaussian_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))
```

```
Accuracy: 68.94
Accuracy CV 10-Fold: 68.88
Running Time: 0:00:02.190648
```

In [ ]:

```python
print (metrics.classification_report(y_train, train_pred_gaussian) )
```

```
              precision    recall  f1-score   support

         0.0       0.26      0.82      0.40     18260
         1.0       0.96      0.67      0.79    128455

    accuracy                           0.69    146715
   macro avg       0.61      0.75      0.59    146715
weighted avg       0.88      0.69      0.74    146715
```

In [ ]:

```python
print (metrics.classification_report(y_test, test_pred_gaussian) )
```

```
              precision    recall  f1-score   support

         0.0       0.26      0.82      0.40      7902
         1.0       0.96      0.67      0.79     54976

    accuracy                           0.69     62878
   macro avg       0.61      0.74      0.59     62878
weighted avg       0.87      0.69      0.74     62878
```

In [ ]:

```
# Linear SVC
start_time = time.time()
train_pred_svc, test_pred_svc, acc_linear_svc, acc_cv_linear_svc, probs_svc = fit_ml_algo(LinearSVC(),
                                                                                            X_train,
                                                                                            y_train,
                                                                                            X_test,
                                                                                            10)
linear_svc_time = (time.time() - start_time)
print("Accuracy: %s" % acc_linear_svc)
print("Accuracy CV 10-Fold: %s" % acc_cv_linear_svc)
print("Running Time: %s" % datetime.timedelta(seconds=linear_svc_time))
```

```
Accuracy: 87.5
Accuracy CV 10-Fold: 87.64
Running Time: 0:05:10.191326
```

In [ ]:

```

print (metrics.classification_report(y_train, train_pred_svc) )
```

```
              precision    recall  f1-score   support

         0.0       0.70      0.01      0.02     18260
         1.0       0.88      1.00      0.93    128455

    accuracy                           0.88    146715
   macro avg       0.79      0.51      0.48    146715
weighted avg       0.86      0.88      0.82    146715
```

In [ ]:

```
print (metrics.classification_report(y_test, test_pred_svc) )
```

```
              precision    recall  f1-score   support

         0.0       0.66      0.01      0.02      7902
         1.0       0.88      1.00      0.93     54976

    accuracy                           0.88     62878
   macro avg       0.77      0.51      0.48     62878
weighted avg       0.85      0.88      0.82     62878
```

In [ ]:

```
# Stochastic Gradient Descent
start_time = time.time()
train_pred_sgd, test_pred_sgd, acc_sgd, acc_cv_sgd, probs_sgd = fit_ml_algo(SGDClassifier(n_jobs = -1),
                                                                              X_train,
                                                                              y_train,
                                                                              X_test,
                                                                              10)
sgd_time = (time.time() - start_time)
print("Accuracy: %s" % acc_sgd)
print("Accuracy CV 10-Fold: %s" % acc_cv_sgd)
print("Running Time: %s" % datetime.timedelta(seconds=sgd_time))
```

```
Accuracy: 87.43
Accuracy CV 10-Fold: 87.55
Running Time: 0:00:06.221039
```

In [ ]:

```
print (metrics.classification_report(y_train, train_pred_sgd) )
```

```
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00     18260
         1.0       0.88      1.00      0.93    128455

    accuracy                           0.88    146715
   macro avg       0.44      0.50      0.47    146715
weighted avg       0.77      0.88      0.82    146715
```

In [ ]:

```
print (metrics.classification_report(y_test, test_pred_sgd) )
```

```
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00      7902
         1.0       0.87      1.00      0.93     54976

    accuracy                           0.87     62878
   macro avg       0.44      0.50      0.47     62878
weighted avg       0.76      0.87      0.82     62878
```

In [ ]:

```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, test_pred_dt, acc_dt, acc_cv_dt, probs_dt = fit_ml_algo(DecisionTreeClassifier(),
                                                                        X_train,
                                                                        y_train,
                                                                        X_test,
                                                                        10)
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))
```

```
Accuracy: 86.2
Accuracy CV 10-Fold: 86.34
Running Time: 0:00:30.899525
```

In [ ]:

```
print (metrics.classification_report(y_train, train_pred_dt) )
```

```
              precision    recall  f1-score   support

         0.0       0.45      0.49      0.47     18260
         1.0       0.93      0.92      0.92    128455

    accuracy                           0.86    146715
   macro avg       0.69      0.70      0.70    146715
weighted avg       0.87      0.86      0.87    146715
```

In [ ]:

```
print (metrics.classification_report(y_test, test_pred_dt) )
```

```
              precision    recall  f1-score   support

         0.0       0.45      0.49      0.47      7902
         1.0       0.93      0.91      0.92     54976

    accuracy                           0.86     62878
   macro avg       0.69      0.70      0.70     62878
weighted avg       0.87      0.86      0.86     62878
```

In [ ]:

```
# Random Forest Classifier
start_time = time.time()
rfc = RandomForestClassifier(n_estimators=10,
                             min_samples_leaf=4,
                             min_samples_split=18,
                             criterion='entropy',
                             max_features=8)
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf = fit_ml_algo(rfc,
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
rf_time = (time.time() - start_time)
print("Accuracy: %s" % acc_rf)
print("Accuracy CV 10-Fold: %s" % acc_cv_rf)
print("Running Time: %s" % datetime.timedelta(seconds=rf_time))
```

```
Accuracy: 91.18
Accuracy CV 10-Fold: 91.16
Running Time: 0:01:07.114951
```

In [ ]:

```
print (metrics.classification_report(y_train, train_pred_rf) )
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.43      0.55     18260
         1.0       0.92      0.98      0.95    128455

    accuracy                           0.91    146715
   macro avg       0.84      0.71      0.75    146715
weighted avg       0.90      0.91      0.90    146715
```

In [ ]:

```
print (metrics.classification_report(y_test, test_pred_rf) )
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.44      0.55      7902
         1.0       0.92      0.98      0.95     54976

    accuracy                           0.91     62878
   macro avg       0.84      0.71      0.75     62878
weighted avg       0.90      0.91      0.90     62878
```

In [ ]:

```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt = fit_ml_algo(GradientBoostingClassifier(),
                                                                             X_train,
                                                                             y_train,
                                                                             X_test,
                                                                             10)
gbt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))
```

```
Accuracy: 90.88
Accuracy CV 10-Fold: 90.87
Running Time: 0:10:00.507756
```

In [ ]:

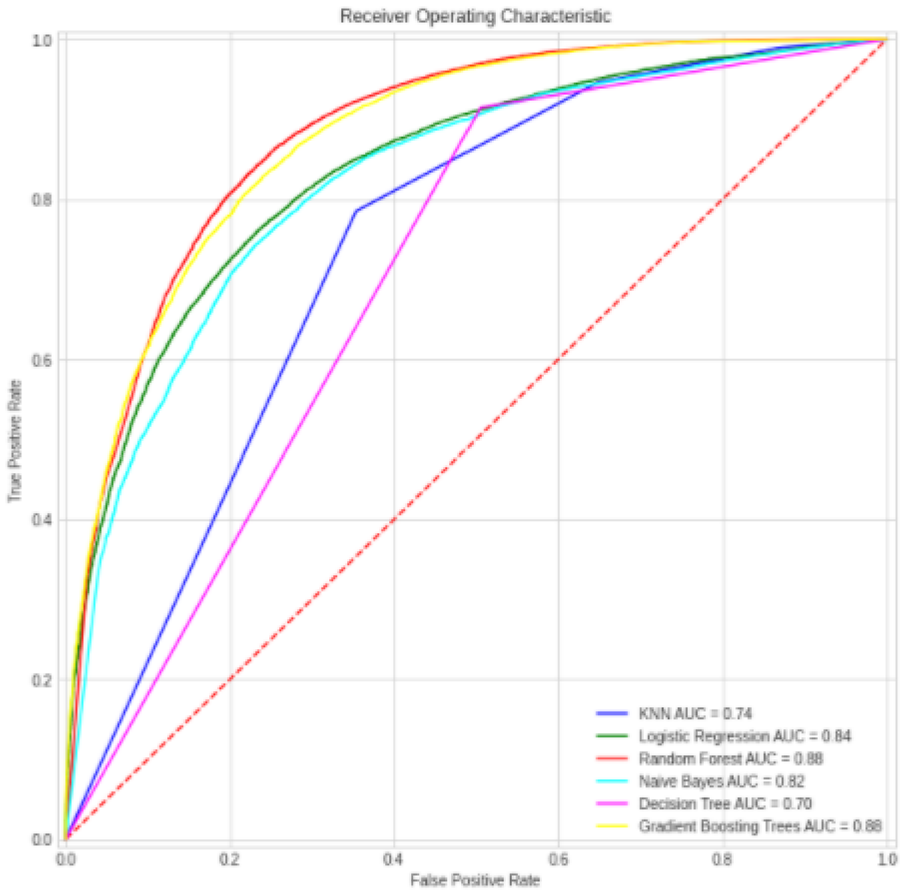
```
print (metrics.classification_report(y_train, train_pred_gbt) )
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.35      0.49     18260
         1.0       0.91      0.99      0.95    128455

    accuracy                           0.91    146715
   macro avg       0.86      0.67      0.72    146715
weighted avg       0.90      0.91      0.89    146715
```

In [ ]:

```
print (metrics.classification_report(y_test, test_pred_gbt) )
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.35      0.49      7902
         1.0       0.91      0.99      0.95     54976

    accuracy                           0.91     62878
   macro avg       0.86      0.67      0.72     62878
weighted avg       0.90      0.91      0.89     62878
```



After applying all the above classification algos on the dataset we see that Gradient Boosting trees & Random Forest both fits the best for our objective.

- ## Final model

we will use Random Forest as our final model

```
In [ ]:
# we will use Random Forest as our final model
from imblearn.over_sampling import SMOTE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
X_train, y_train = SMOTE().fit_sample(X_train, y_train)

rfc = RandomForestClassifier(n_estimators=10,
                             min_samples_leaf=4,
                             min_samples_split=18,
                             criterion='entropy',
                             max_features=8)
rfc.fit(X_train,y_train)
y_pred=rfc.predict(X_test)

```

- ## Final result with confusion matrix.

```
Confusion matrix
 [[ 4815  3087]
 [ 4011 50965]]
f1 score is :  0.9348974575338445
classification report
              precision    recall  f1-score   support

         0.0       0.55      0.61      0.58      7902
         1.0       0.94      0.93      0.93     54976

    accuracy                           0.89     62878
   macro avg       0.74      0.77      0.76     62878
weighted avg       0.89      0.89      0.89     62878

AUC ROC Score:  0.7681901491576527
```

CONCLUSION

- ## Conclusions of the Study

The last four days I spend quite a lot of my free time on a current data-science project. A Micro-Credit Defaulter prediction problem at FlipRobo. And yes, it was less sleep than usual but the learning's were worth it.

- ## Learning Outcomes of the Study in respect of Data Science

Here I learned about the micro credit industry, visualization, data cleaning, handling outliers and using various algorithms on huge dataset. This was the first time I worked on such huge dataset. It took a lot of time to hyper tune all the algorithms to find out the best one to work with. Working with such huge dataset that took a lot of time to train the algorithms and tuning it for the best prams was worth knowing in this project.

- ## Limitations of this work and Scope for Future Work

Training the huge dataset was a challenge for me. Balancing the imbalance dataset. Overcoming the outliers. Hyper tuning the algorithms can bring out more satisfactory result.