FLIP ROBO

MICRO CREDIT LOAN DEFAULTER

Submitted by-:
NITISH KUMAR SHARMA

ACKNOWLEDGEMENT

It is great pleasure for me to undertake this project. I feel overwhelmed doing this project entitled – “Micro Credit Defaulter“.

Some of the resources that helped me to complete this project are as follows:

- Internet/web
- Stack overflow
- Analytics Vidhya
- Articles published in Medium.com
- Wikipedia

Contents

INTRODUCTION

BUSINESS PROBLEM FRAMING

This project includes the real time problem for Microfinance Institution (MFI) offering financial services to low income population in terms of providing mobile balance loan.

MFI provides 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).

CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM

Microcredit is an extremely small loan given to those who lack a steady source of income, collateral, or any credit history. It is also more common in underdeveloped countries, as it is aimed to support people of a lower socioeconomic background. Individuals who receive a microcredit loan may be illiterate; thus, they are unable to apply for conventional loans due to the paperwork involved.

The telecom company in collaboration with a Microfinance Institute (MFI) provides loans of amount 5 and 10 (Indonesian Rupiah) for a very short period and the payback amount is 6 and 12 (Indonesian Rupiah) respectively which corresponds to a high interest rate of 20% in a very short period (usually 5 days). While the return is high, there is considerable risk of default involved, because the loan is being provided to low income population.

Therefore, it is necessary to classify all the defaulters to minimize business risk and avoid losses.

REVIEW OF LITERATURE

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.

The Client has provided sample data from its database 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.

We have used different machine learning models to predict the above. Since we have categorical target data so classification model algorithms has been used.

We will start our project with the sample dataset which contains loan default status (Defaulter / Non-Defaulter) along with associated features. We will observe all the features with following goals in mind:

- Relevance of the features
- Distribution of the features
- Data Cleaning of the features
- Visualization of the features
- Visualization of the features as per loan default status for data analysis

After having gone through all the features and cleaning the dataset, we will move on to machine learning classification modelling:

- Pre-processing of the dataset for models
- Testing multiple algorithms with multiple evaluation metrics
- Select evaluation metric as per our specific business application
- Doing hyper-parameter tuning using GridSearchCV for the best model
- Finally saving the best model
- Loading and predicting with the loaded model

MOTIVATION FOR THE PROBLEM UNDERTAKEN

This project deals with finance domain which is a hot market. Here a company is investing in billions to get profitable returns. So I am very much excited and thrilled to deal with real time data provided by the client so that through my analysis and observation I could generate profits by solving their business problem.

Analyse users behaviour pattern through data visualization and come with solutions to prevent loss of time & money of the company.

It's a good motive initiated by MFI in collaboration with one of the client of telecom industry for providing loan services in terms of mobile balance to low income families and poor customers which will help them in the need of hour. They understand the importance of communication and how it affects a person's life.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL / ANALYTICAL MODELING OF THE PROBLEM

The dataset is a csv file with 37 attributes (36 features and 1 target). The target variable contains 1 or 0 which means non defaulter and defaulter respectively. The other key attributes are aon (age on cellular network), msisdn (mobile number of user), pcircle, pdate, account balances, median recharge balance for 30 and 90 days. The similar attributes like number of loans taken, maximum amount of loan taken, frequency of data account recharged, average payback time etc for 30 and 90 days.

To know the overview/stats of the dataset , we will be using df.describe() function which gives informations like count, min, max, mean, standard deviation values of features . By plotting of heat map we can correlate features if they are highly inter correlated or not. So we can drop columns if problem of multicollinearity arises (if vif >5) when we observe through variance inflation factor.

From an initial statistical overview of the dataset, we infer that our target variable is of binary classification . Age on cellular network (aon) is having negative value which is not correct at all. Most features have standard deviation value greater than mean which shows that the data is messed up.

The features having the data with a time span of 90 days gives more information about the user as compared to the features with a time span of 30 days.

DATA SOURCES AND THEIR FORMATS

The data that I received was in CSV format (Comma Separated Values). After reading the data by using function df=pd.read_csv (‘ ---file path --- ’) in jupyter notebook there were 20953 rows and 37 columns.

Dataset/Attributes Description :

1	Variable	Definition
2	label	Flag indicating whether the user paid back the credit amount within 5 days of issuing the loan{1:success, 0:failure}
3	msisdn	mobile number of user
4	aon	age on cellular network in days
5	daily_decr30	Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)
6	daily_decr90	Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah)
7	rental30	Average main account balance over last 30 days
8	rental90	Average main account balance over last 90 days
9	last_rech_date_ma	Number of days till last recharge of main account
10	last_rech_date_da	Number of days till last recharge of data account
11	last_rech_amt_ma	Amount of last recharge of main account (in Indonesian Rupiah)
12	cnt_ma_rech30	Number of times main account got recharged in last 30 days
13	fr_ma_rech30	Frequency of main account recharged in last 30 days
14	sumamnt_ma_rech30	Total amount of recharge in main account over last 30 days (in Indonesian Rupiah)
15	medianamnt_ma_rech30	Median of amount of recharges done in main account over last 30 days at user level (in Indonesian Rupiah)
16	medianmarechprebal30	Median of main account balance just before recharge in last 30 days at user level (in Indonesian Rupiah)
17	cnt_ma_rech90	Number of times main account got recharged in last 90 days
18	fr_ma_rech90	Frequency of main account recharged in last 90 days
19	sumamnt_ma_rech90	Total amount of recharge in main account over last 90 days (in Indonasian Rupiah)
20	medianamnt_ma_rech90	Median of amount of recharges done in main account over last 90 days at user level (in Indonasian Rupiah)
21	medianmarechprebal90	Median of main account balance just before recharge in last 90 days at user level (in Indonasian Rupiah)

22	cnt_da_rech30	Number of times data account got recharged in last 30 days
23	fr_da_rech30	Frequency of data account recharged in last 30 days
24	cnt_da_rech90	Number of times data account got recharged in last 90 days
25	fr_da_rech90	Frequency of data account recharged in last 90 days
26	cnt_loans30	Number of loans taken by user in last 30 days
27	amnt_loans30	Total amount of loans taken by user in last 30 days
28	maxamnt_loans30	maximum amount of loan taken by the user in last 30 days
29	medianamnt_loans30	Median of amounts of loan taken by the user in last 30 days
30	cnt_loans90	Number of loans taken by user in last 90 days
31	amnt_loans90	Total amount of loans taken by user in last 90 days
32	maxamnt_loans90	maximum amount of loan taken by the user in last 90 days
33	medianamnt_loans90	Median of amounts of loan taken by the user in last 90 days
34	payback30	Average payback time in days over last 30 days
35	payback90	Average payback time in days over last 90 days
36	pcircle	telecom circle
37	pdate	date

Dataset datatypes are as follow :

```
df.info() # to know datatype of each columns
```

```
<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
```

Unique values of each columns are as follows :

```
# to count number of unique values in each columns
df.nunique()
```

```
Unnamed: 0              209593
label                        2
msisdn                  186243
aon                       4507
daily_decr30            147026
daily_decr90            158670
rental30                132148
rental90                141033
last_rech_date_ma         1186
last_rech_date_da         1174
last_rech_amt_ma            70
cnt_ma_rech30               71
fr_ma_rech30              1083
sumamnt_ma_rech30        15141
medianamnt_ma_rech30       510
medianmarechprebal30     30428
cnt_ma_rech90              110
fr_ma_rech90                89
sumamnt_ma_rech90        31771
medianamnt_ma_rech90       608
medianmarechprebal90     29785
cnt_da_rech30             1066
fr_da_rech30              1072
cnt_da_rech90               27
fr_da_rech90                46
cnt_loans30                 40
amnt_loans30                48
maxamnt_loans30           1050
medianamnt_loans30           6
cnt_loans90               1110
amnt_loans90                69
maxamnt_loans90              3
medianamnt_loans90           6
payback30                 1363
payback90                 2381
pcircle                      1
pdate                       82
dtype: int64
```

Categorical & Continous features in the dataset are as follows :

```
# to list out categorical features from dataset
cat_features=[i for i in df.columns if df.dtypes[i]=='object']
cat_features
```

```
['msisdn', 'pcircle', 'pdate']
```

```
# to list out continous features from dataset
con_features=[i for i in df.columns if df.dtypes[i]=='int64' or df.dtypes[i]=='float64']
con_features
```

```
['Unnamed: 0',
 'label',
 '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',
 'medianamnt_loans30',
 'cnt_loans90',
 'amnt_loans90',
 'maxamnt_loans90',
 'medianamnt_loans90',
 'payback30',
 'payback90']
```

DATA PREPROCESSING DONE

There were no null values in the dataset.

```
df.isnull().sum()  # to check null values
```

```
Unnamed: 0              0
label                   0
msisdn                  0
aon                     0
daily_decr30            0
daily_decr90            0
rental30                0
rental90                0
last_rech_date_ma       0
last_rech_date_da       0
last_rech_amt_ma        0
cnt_ma_rech30           0
fr_ma_rech30            0
sumamnt_ma_rech30       0
medianamnt_ma_rech30    0
medianmarechprebal30    0
cnt_ma_rech90           0
fr_ma_rech90            0
sumamnt_ma_rech90       0
medianamnt_ma_rech90    0
medianmarechprebal90    0
cnt_da_rech30           0
fr_da_rech30            0
cnt_da_rech90           0
fr_da_rech90            0
cnt_loans30             0
amnt_loans30            0
maxamnt_loans30         0
medianamnt_loans30      0
cnt_loans90             0
amnt_loans90            0
maxamnt_loans90         0
medianamnt_loans90      0
payback30               0
payback90               0
pcircle                 0
pdate                   0
dtype: int64
```

```
# to know all types of unique values
df['Unnamed: 0'].unique()
```

```
array([      1,       2,       3, ..., 209591, 209592, 209593], dtype=int64)
```

```
# dropping column because its just a serial number from first to last
df=df.drop(columns='Unnamed: 0')
```

- Dropping column ‘ Unnamed: 0’ from the dataset as it was containing values from 1 to the last count as shown above. It was a list of serial numbers.

```
# to know all types of unique values
df['pcircle'].unique()
```

```
array(['UPW'], dtype=object)
```

```
# dropping column because it has one value throughout the column
df=df.drop(columns='pcircle')
```

- Dropping ‘pcircle’ column as it was having one value throughout.

```
# dropping column as it has phone numbers of customers which is not useful for prediction
df=df.drop(columns='msisdn')
```

- Dropping ‘msisdn’ column because it was having contact numbers of users.

```
# to change format
df['pdate']=pd.to_datetime(df['pdate'])
df['pdate']
```

```
0         2016-07-20
1         2016-08-10
2         2016-08-19
3         2016-06-06
4         2016-06-22
             ...
209588    2016-06-17
209589    2016-06-12
209590    2016-07-29
209591    2016-07-25
209592    2016-07-07
Name: pdate, Length: 209593, dtype: datetime64[ns]
```

Year is same throughout so we will not add it while splitting.

```
# splitting Date into Month and Day
df['Month']=df['pdate'].dt.month
df['Day']=df['pdate'].dt.day
```

```
# dropping 'pdate' column
df.drop('pdate',axis=1,inplace=True)
```

- Formatted the ‘pdate’ column and splitting into month and day only because year is same throughout.

```
# storing dataframe value into variable
rental30=df['rental30']
```

```
rental=[]  # empty list
for i in rental30:  # running loop
    if(i<=0):
        rental.append('no balance')  # adding the result according to the condition
    elif (i>0 and i<=12655):
        rental.append('low balance')
    elif (i>=12655 and i<118766):
        rental.append('average balance')
    elif (i>118766):
        rental.append('high balance')
```

```
# storing the value back to dataframe after categorizing
df['rental30']=rental
```

```
df['rental30'].unique()
```

```
array(['low balance', 'no balance', 'average balance', 'high balance'],
      dtype=object)
```

- Categorizing ‘rental30’ column into (no balance, low balance, average balance, high balance) for better visualization purposes.

```
# visualizing aon (age on cellular network in days) by categorizing it

category=[(df['aon'] <2),df['aon'].between(2,5),(df['aon'] > 5)]
value= ['New Users','Average Users','Old Users']
df['Users_Category']=np.select(category,value)

# mapping the balance groups with precentage value with respect to label

UsersCategorypercent = pd.crosstab(df['label'],df['Users_Category']).apply(lambda x: x/x.sum()*100)
UsersCategorypercent = UsersCategorypercent.transpose()
UsersCategorypercent

UsersCategorypercent.plot(kind='bar',figsize=(12,5))
plt.title('Age on cellular network in days vs Loan Repayment Percentage within 5 days',fontsize=13)
plt.ylabel('Loan Repayment Percentage within 5 days',fontsize=13)
plt.xlabel('Users Category',fontsize=12)
plt.xticks(rotation = 'horizontal',fontsize=10);
```

- Categorizing ‘aon’ column into (New Users, Average Users, Old Users) for better visualization purposes.

```python
# check the number of loans taken by user in last 90 days
category=[(df['cnt_loans90'] <=0),df['cnt_loans90'].between(0,2),(df['cnt_loans90'] > 2)]
value= ['No Loans Taken', 'Average number of loans Taken','Too much loans taken']
df['Loans_Frequency']=np.select(category,value)
```

```python
df['Loans_Frequency'].value_counts()
```

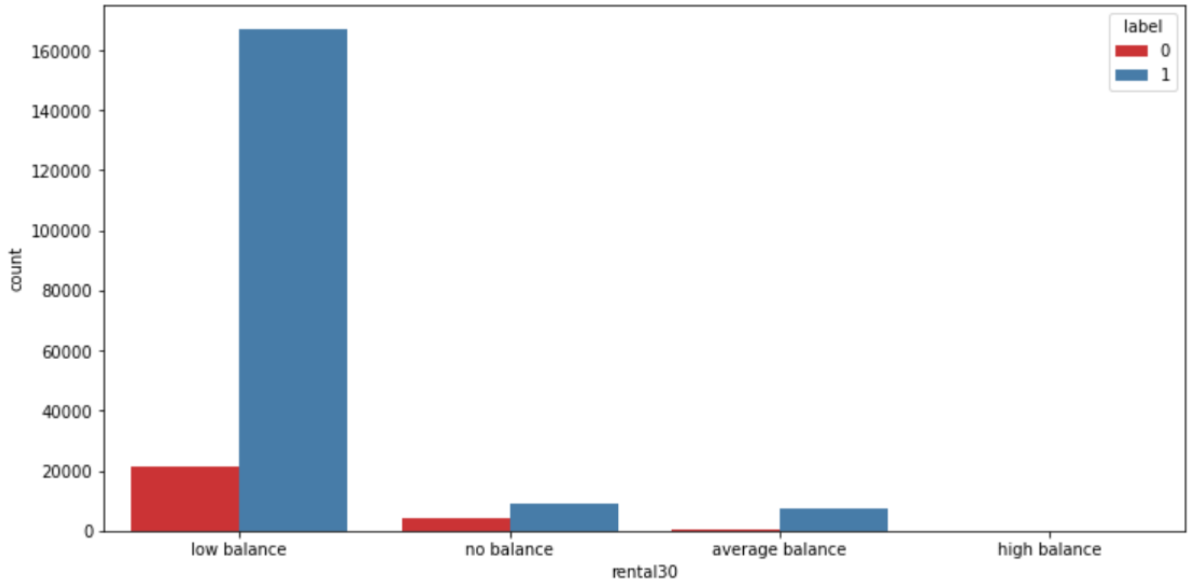
```
Average number of loans Taken    111148
Too much loans taken              96409
No Loans Taken                     2036
Name: Loans_Frequency, dtype: int64
```

```python
# Mapping the balance groups with precentage value with respect to label
LoansFrequencypercent = pd.crosstab(df['label'],df['Loans_Frequency']).apply(lambda x: x/x.sum()*100)
LoansFrequencypercent = LoansFrequencypercent.transpose()
LoansFrequencypercent
```

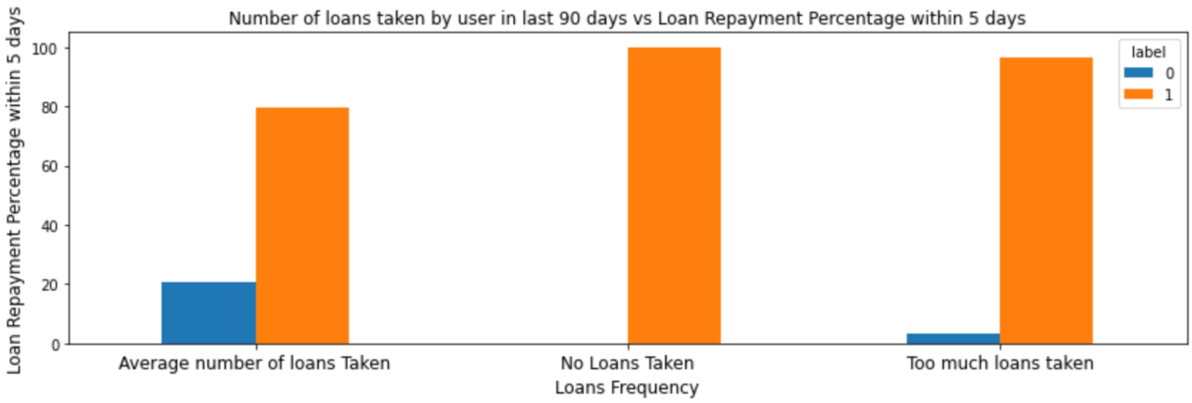
label	0	1
Loans_Frequency		
Average number of loans Taken	20.649944	79.350056
No Loans Taken	0.000000	100.000000
Too much loans taken	3.329565	96.670435

- Categorizing 'cnt_loans90' column into (Average number of loans taken, Too much loans taken, No loans Taken) for better visualization purposes.

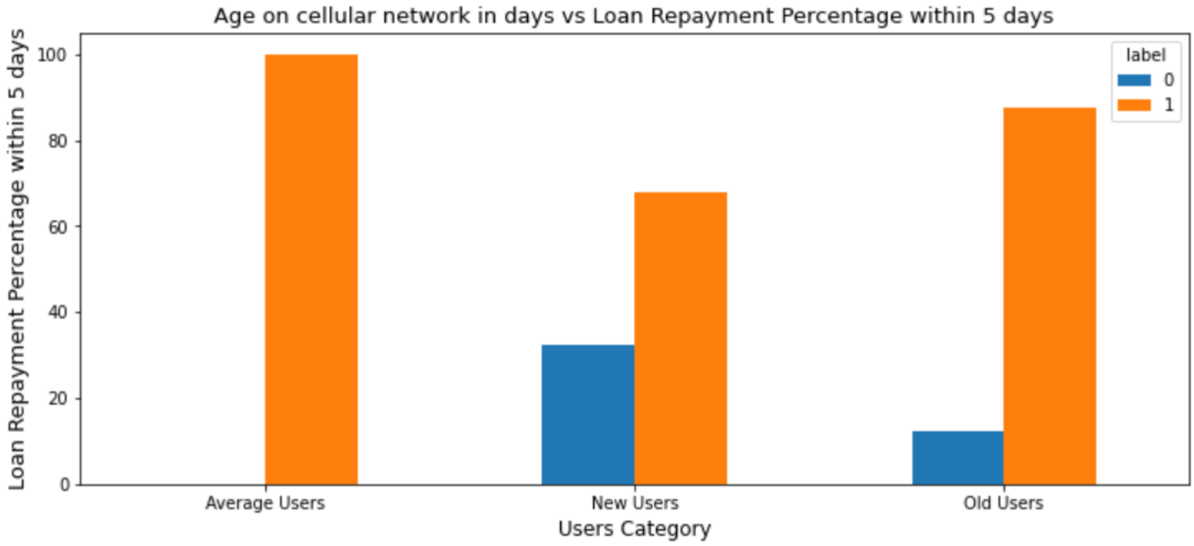
DATA INPUTS- LOGIC- OUTPUT RELATIONSHIPS



- Lots of users maintaining low balance is in defaulter list.



- Around 98% Users taking too much loans are non-defaulters as they repay the loan within stipulated time.
- Users taking average number of loans which is approx 20% are defaulters.



- New Users category has lots of defaulters around 35%.

STATE THE SET OF ASSUMPTIONS (IF ANY) RELATED TO THE PROBLEM UNDER CONSIDERATION

From the above statistical summary of the above part of the dataset, the important thing is that some features even have negative values like the age on cellular network, main account last recharge date, data account last recharge date. Negative values in these features make no sense thus these values should be removed.

```
df.describe().T # to get high understanding of dataset or to get overview/stats of the dataset
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	209593.0	104797.000000	60504.431823	1.000000	52399.000	104797.000000	157195.00	209593.000000
label	209593.0	0.875177	0.330519	0.000000	1.000	1.000000	1.00	1.000000
aon	209593.0	8112.343445	75696.082531	-48.000000	246.000	527.000000	982.00	999860.755168
daily_decr30	209593.0	5381.402289	9220.623400	-93.012667	42.440	1469.175667	7244.00	265926.000000
daily_decr90	209593.0	6082.515068	10918.812767	-93.012667	42.692	1500.000000	7802.79	320630.000000
rental30	209593.0	2692.581910	4308.586781	-23737.140000	280.420	1083.570000	3356.94	198926.110000
rental90	209593.0	3483.406534	5770.461279	-24720.580000	300.260	1334.000000	4201.79	200148.110000
last_rech_date_ma	209593.0	3755.847800	53905.892230	-29.000000	1.000	3.000000	7.00	998650.377733
last_rech_date_da	209593.0	3712.202921	53374.833430	-29.000000	0.000	0.000000	0.00	999171.809410
last_rech_amt_ma	209593.0	2064.452797	2370.786034	0.000000	770.000	1539.000000	2309.00	55000.000000
cnt_ma_rech30	209593.0	3.978057	4.256090	0.000000	1.000	3.000000	5.00	203.000000
fr_ma_rech30	209593.0	3737.355121	53643.625172	0.000000	0.000	2.000000	6.00	999606.368132
sumamnt_ma_rech30	209593.0	7704.501157	10139.621714	0.000000	1540.000	4628.000000	10010.00	810096.000000
medianamnt_ma_rech30	209593.0	1812.817952	2070.864620	0.000000	770.000	1539.000000	1924.00	55000.000000
medianmarechprebal30	209593.0	3851.927942	54006.374433	-200.000000	11.000	33.900000	83.00	999479.419319
cnt_ma_rech90	209593.0	6.315430	7.193470	0.000000	2.000	4.000000	8.00	336.000000
fr_ma_rech90	209593.0	7.716780	12.590251	0.000000	0.000	2.000000	8.00	88.000000
sumamnt_ma_rech90	209593.0	12396.218352	16857.793882	0.000000	2317.000	7226.000000	16000.00	953036.000000
medianamnt_ma_rech90	209593.0	1864.595821	2081.680664	0.000000	773.000	1539.000000	1924.00	55000.000000
medianmarechprebal90	209593.0	92.025541	369.215658	-200.000000	14.600	36.000000	79.31	41456.500000
cnt_da_rech30	209593.0	262.578110	4183.897978	0.000000	0.000	0.000000	0.00	99914.441420
fr_da_rech30	209593.0	3749.494447	53885.414979	0.000000	0.000	0.000000	0.00	999809.240107
cnt_da_rech90	209593.0	0.041495	0.397556	0.000000	0.000	0.000000	0.00	38.000000
fr_da_rech90	209593.0	0.045712	0.951386	0.000000	0.000	0.000000	0.00	64.000000
cnt_loans30	209593.0	2.758981	2.554502	0.000000	1.000	2.000000	4.00	50.000000
amnt_loans30	209593.0	17.952021	17.379741	0.000000	6.000	12.000000	24.00	306.000000
maxamnt_loans30	209593.0	274.658747	4245.264648	0.000000	6.000	6.000000	6.00	99864.560864
medianamnt_loans30	209593.0	0.054029	0.218039	0.000000	0.000	0.000000	0.00	3.000000
cnt_loans90	209593.0	18.520919	224.797423	0.000000	1.000	2.000000	5.00	4997.517944
amnt_loans90	209593.0	23.645398	26.469861	0.000000	6.000	12.000000	30.00	438.000000
maxamnt_loans90	209593.0	6.703134	2.103864	0.000000	6.000	6.000000	6.00	12.000000
medianamnt_loans90	209593.0	0.046077	0.200692	0.000000	0.000	0.000000	0.00	3.000000
payback30	209593.0	3.398826	8.813729	0.000000	0.000	0.000000	3.75	171.500000
payback90	209593.0	4.321485	10.308108	0.000000	0.000	1.666667	4.50	171.500000

Our dataset consists of some features giving information about the user for the time span of 30 days and 90 days. According to me, with data of large number of days for a particular user then we could interpret User's behaviour more precisely because many users have the tendency of repeating the same things. Thus the features having the data with a time span of 90 days gives more information about the user as compared to the features with a time span of 30 days.

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

Anaconda Navigator 1.10.0

Jupyter Notebook 6.1.4 , Python 3

Libraries

```
import pandas as pd # for handling dataset
import numpy as np  # for mathematical computation

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, plot_roc_curve, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier

from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from scipy.stats import skew

# for visualization
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style
import seaborn as sns
import pickle

import warnings
warnings.filterwarnings('ignore')
```

```
# converting objects into integers
lab_enc = LabelEncoder()
list1 = ['rental30','Loans_Frequency','Users_Category']
for val in list1:
    df[val] = lab_enc.fit_transform(df[val].astype(str))
```

Libraries and Packages used:

- Pickle for saving & loading machine learning model.
- GridSearchCV for Hyper-parameter tuning.
- Cross validation score to cross check if the model is overfitting or not.
- Label Encoder to convert objects into integers.
- PCA as a dimensionality reduction tool.
- Seaborn and matplotlib for visualization.

from sklearn.model_selection import train_test_split

Split arrays or matrices into random train and test subsets.
Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

from sklearn.linear_model import LogisticRegression

It is used to obtain odds ratio in the presence of more than one explanatory variable. The procedure is quite similar to multiple linear regression, with the exception that the response variable is binomial. The result is the impact of each variable on the odds ratio of the observed event of interest.

from sklearn.tree import DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

from sklearn.ensemble import RandomForestClassifier

It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

from sklearn.neighbors import KNeighborsClassifier

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

The k-neighbors classification in KNeighborsClassifier is the most commonly used technique. The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

from sklearn.ensemble import GradientBoostingClassifier

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

from sklearn.svm import SVC

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.

from sklearn.linear_model import SGDClassifier

This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning via the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

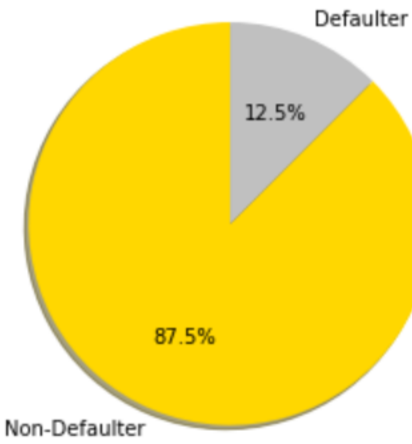
MODEL/s DEVELOPMENT AND EVALUATION

IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS)

```
# to visualize column 'label' in pie chart

labels = ['Non-Defaulter', 'Defaulter']

fig, ax = plt.subplots(figsize=(4,4))
colors = ['gold', 'silver']
ax.pie(df['label'].value_counts(),  labels=labels,
       autopct='%1.1f%%',shadow=True, startangle=90, colors=colors)
ax.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



From the above we can observe that the data was highly imbalanced so we have used SMOTETomek to balance the dataset.

TESTING OF IDENTIFIED APPROACHES (ALGORITHMS)

Algorithms used:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- KNN Classifier
- Support Vector Machines
- Gradient Boosting Classifier
- Stochastic Gradient Descent

RUN AND EVALUATE SELECTED MODELS

1. Logistic Regression

```
log_reg = LogisticRegression()
log_reg.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred = log_reg.predict(x_test)
```

```
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

```
0.883013034599897
```

```
# Confusion Matrix
conf_mat =confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[  841,  5748],
       [  382, 45428]], dtype=int64)
```

```
print('\n---------------Classification Report------------------')
print (classification_report(y_test,y_pred,digits=2))
```

```
---------------Classification Report------------------
              precision    recall  f1-score   support

           0       0.69      0.13      0.22      6589
           1       0.89      0.99      0.94     45810

    accuracy                           0.88     52399
   macro avg       0.79      0.56      0.58     52399
weighted avg       0.86      0.88      0.85     52399
```

Accuracy score : 88.30%

2. Decision Tree Classifier

```
dt_clf = DecisionTreeClassifier()
dt_clf.fit(x_train,y_train)
pred=dt_clf.predict(x_train)
dt_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n===================Train Result======================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{dt_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#*********************** Test Score *************************

pred = dt_clf.predict(x_test)
dt_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n===================Test Result======================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{dt_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 100% Test Result : 88.67%

3. Random Forest Classifier

```
rand_clf = RandomForestClassifier(random_state=101)
rand_clf.fit(x_train,y_train)
pred=dt_clf.predict(x_train)
rand_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n===================Train Result======================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{rand_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#*********************** Test Score *************************

pred = rand_clf.predict(x_test)
rand_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n===================Test Result======================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{rand_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 100% Test Result : 92.12%

4. KNN Classifier

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
pred=knn.predict(x_train)
knn_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{knn_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#*********************** Test Score *************************

pred = knn.predict(x_test)
knn_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{knn_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 91.90% Test Result : 88.95%

5. Support-Vector Machines

```
svc = SVC(kernel = 'rbf',C=1)
svc.fit(x_train,y_train)
pred=svc.predict(x_train)
svc_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result=====================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{svc_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#*********************** Test Score *************************

pred = svc.predict(x_test)
svc_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result=====================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{svc_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 89.47% Test Result : 89.07%

6. Gradient Boosting Classifier

```
gbdt_clf = GradientBoostingClassifier()
gbdt_clf.fit(x_train,y_train)
pred=gbdt_clf.predict(x_train)
gbdt_clf_report = pd.DataFrame(classification_report(y_train,pred, output_dict=True))

print("\n==================Train Result======================")

print(f"Accuracy Score: {accuracy_score(y_train,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{gbdt_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_train,pred)}\n")

#*********************** Test Score *************************

pred = gbdt_clf.predict(x_test)
gbdt_clf_report = pd.DataFrame(classification_report(y_test,pred, output_dict=True))
print("\n==================Test Result======================")
print(f"Accuracy Score: {accuracy_score(y_test,pred) * 100:.2f}%")
print("_______________________________________________")
print(f"CLASSIFICATION REPORT:\n{gbdt_clf_report}")
print("_______________________________________________")
print(f"Confusion Matrix:\n {confusion_matrix(y_test,pred)}\n")
```

Training Result : 91.98% Test Result : 91.85%

7. Stochastic Gradient Descent

```
sgd=SGDClassifier(loss='modified_huber',shuffle=True,random_state=101)
sgd.fit(x_train,y_train)
y_pred = sgd.predict(x_test)
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Test Result : 87.66%

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

Precision: It is the ratio between the True Positives and all the Positives. It can be seen as a measure of quality, higher precision means that an algorithm returns more relevant results than irrelevant ones

Recall : It is the measure of our model correctly identifying True Positives. It is used as a measure of quantity and high recall means that an algorithm returns most of the relevant results.

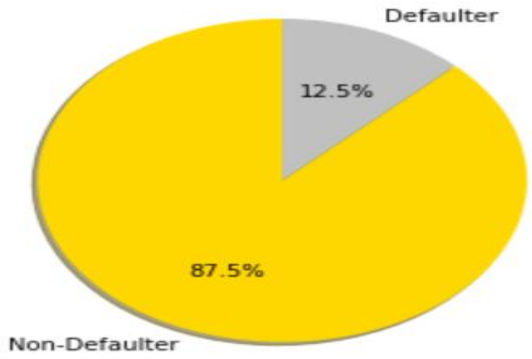
Accuracy score : It is the ratio of the total number of correct predictions and the total number of predictions. It is used when the True Positives and True negatives are more important. Accuracy can be used when the class distribution is similar

F1-score : It is used when the False Negatives and False Positives are crucial. Hence F1-score is a better metric when there are imbalanced classes.

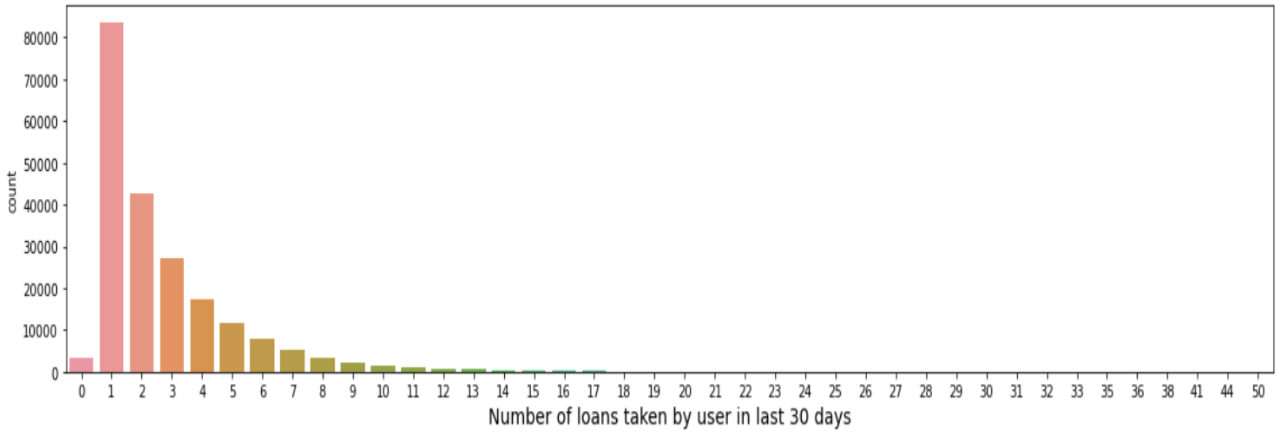
Cross_val_score : To run cross-validation on multiple metrics and also to return train scores, fit times and score times. Get predictions from each split of cross-validation for diagnostic purposes. Make a scorer from a performance metric or loss function.

roc _auc _score : ROC curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0

VISUALIZATION



- Defaulter users not paying back the loan amount within 5 days of issuing is 12.5%.



Maximum times users taking loan in last 30 days:

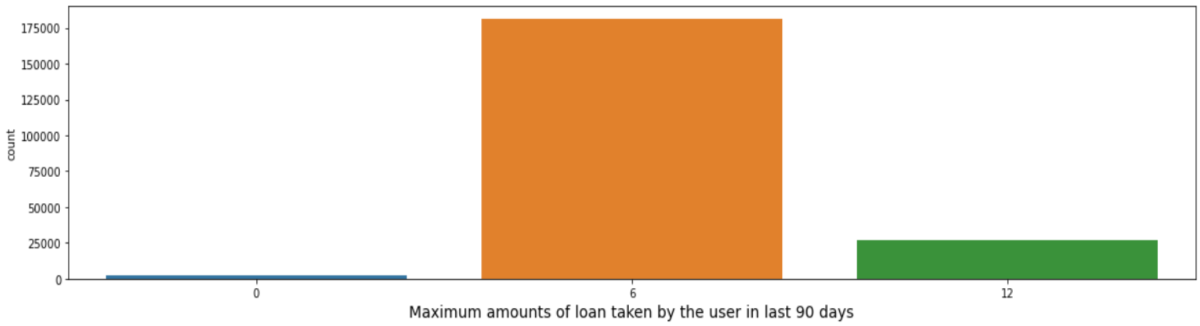
Only 1 time - Approx 40%

Only 2 times - 20%

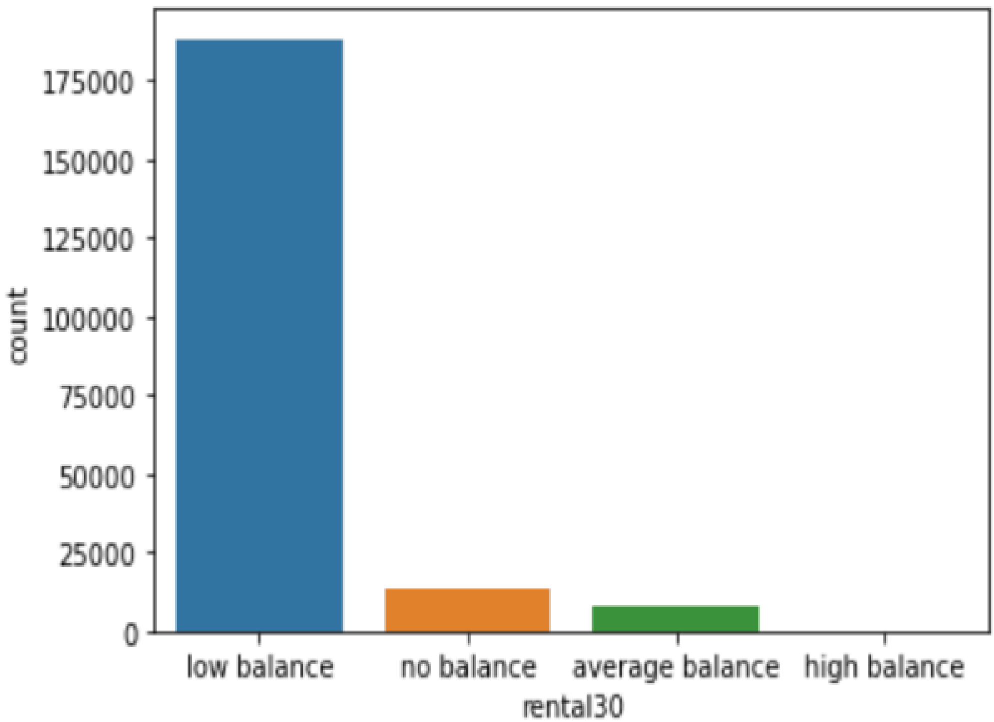
Only 3 times - Approx 13%

Only 4 times- 8%

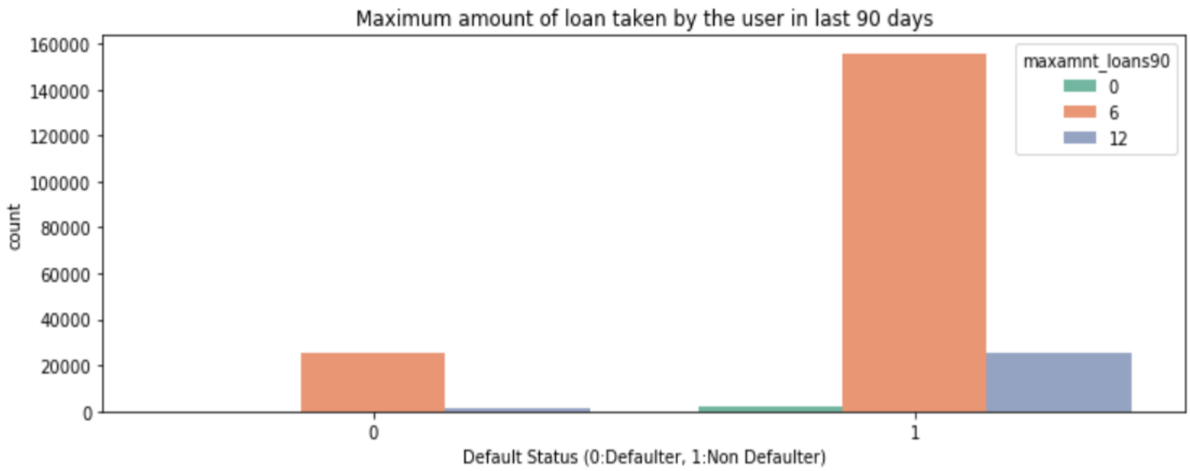
Only 5 times- 5%



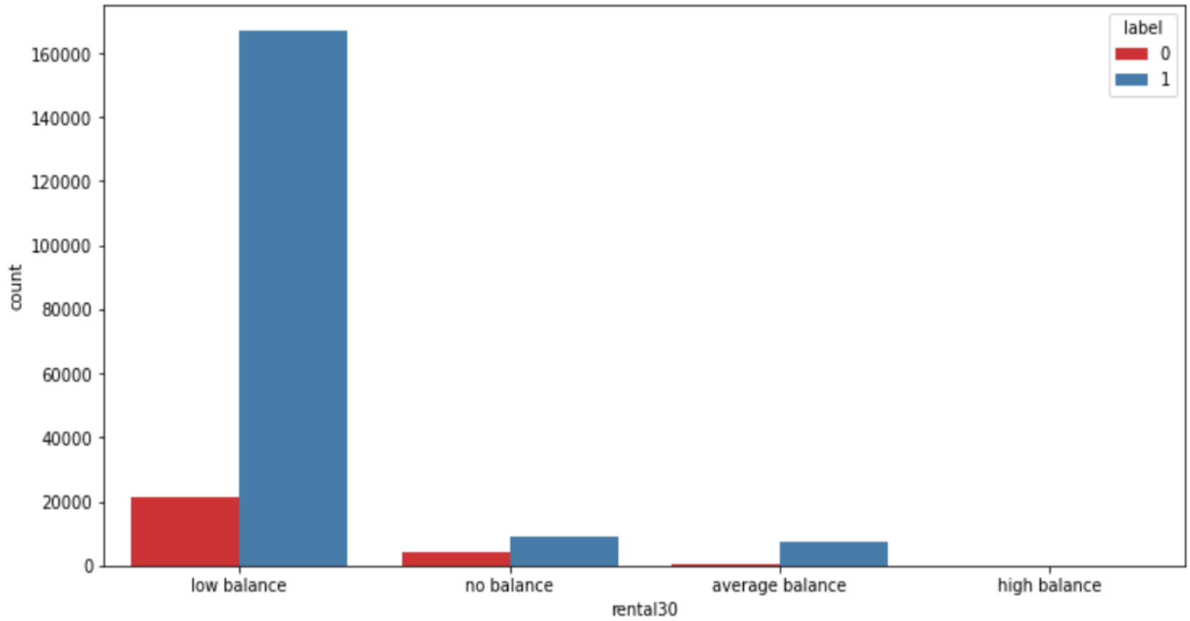
- Users have taken loan mostly where the payback loan amount is 6 (in Indonesian Rupiah) than the payback loan amount 12 (in Indonesian Rupiah).



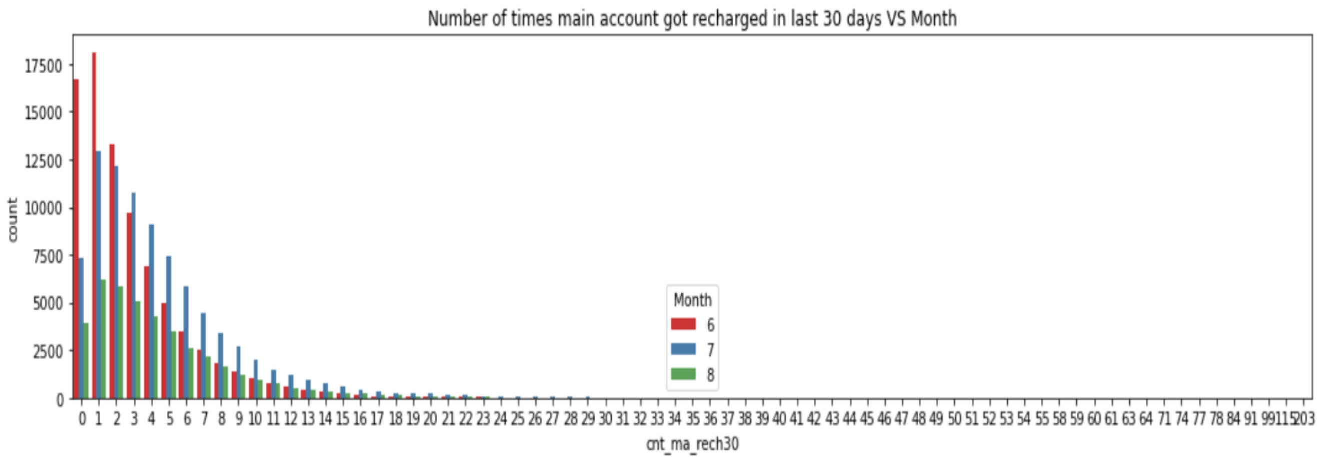
- Approx 90% users has maintained low balance, 3% users have maintained average balance and 6% users have no balance.



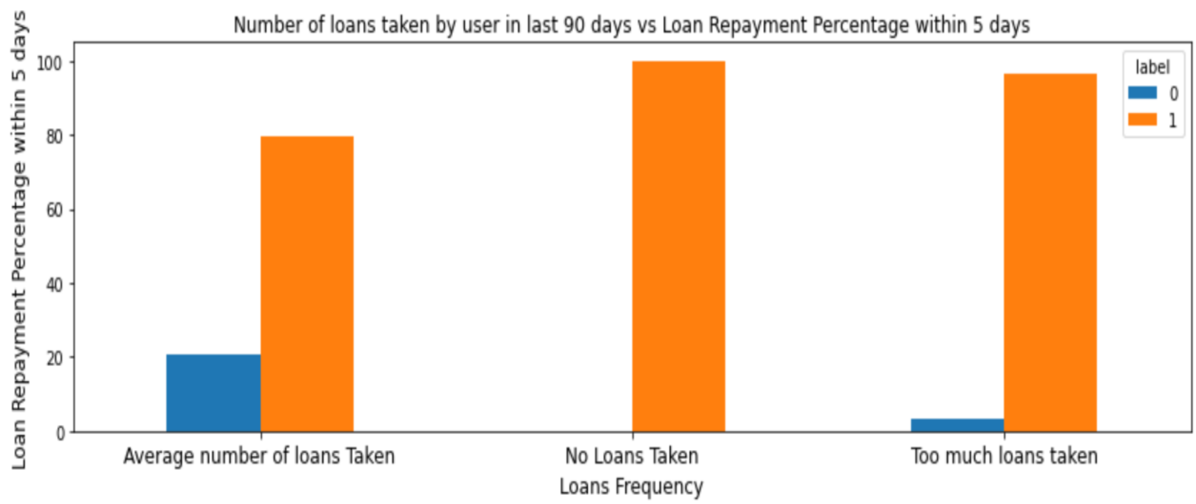
- In the period of 90 days, there is less case of defaulters where the users payback loan amount are 6 & 12(in Indonesian Rupiah).



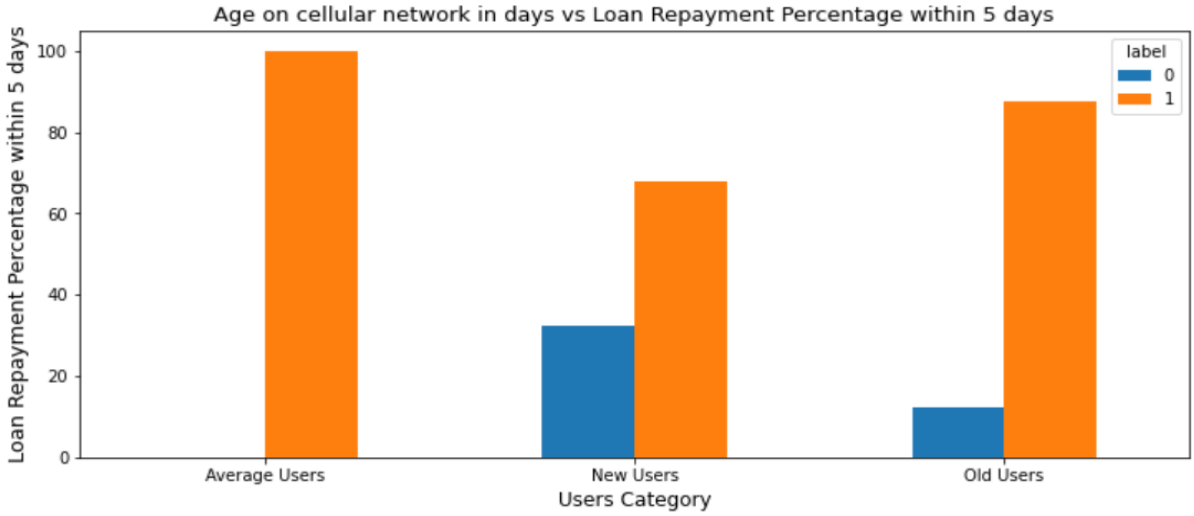
- Lots of users those who are maintaining low balance is in defaulter list.



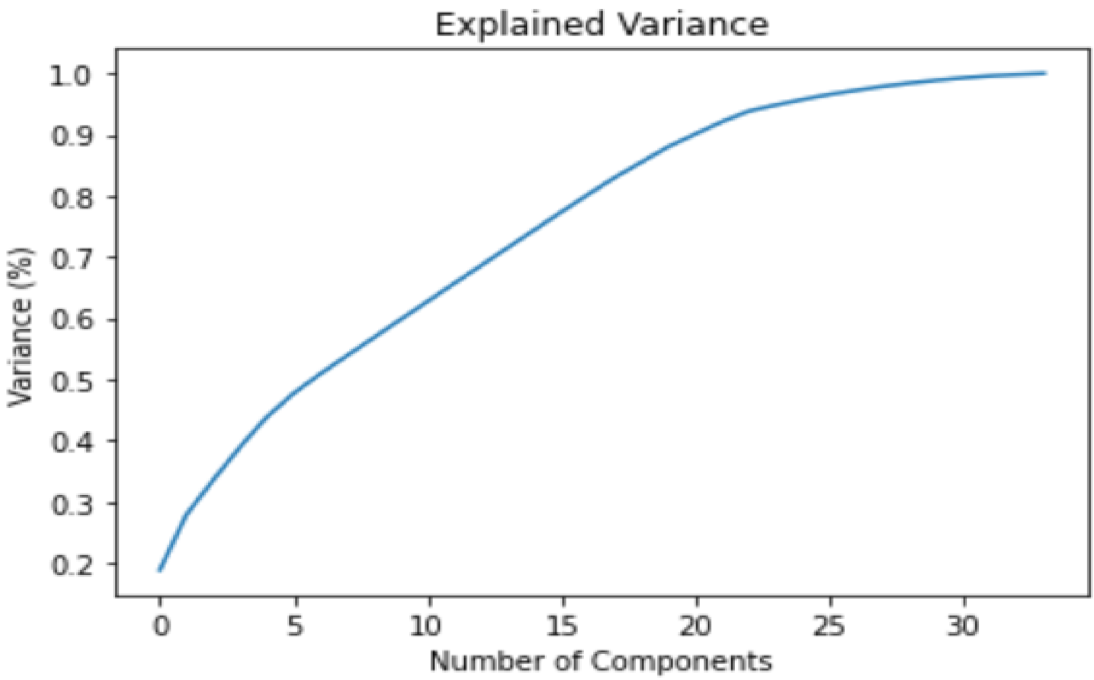
- According to the plot, maximum times users recharge their account in the month of July followed by then June.



- Around 98% Users taking too much loans are non-defaulters as they repay the loan within stipulated time.
- Users taking average number of loans which is approx 20% are defaulters.



- New Users category has lots of defaulters around 35%.



- We can see that 95% of the variance is being explained by 25 components by using Principal Component Analysis (PCA).

INTERPRETATION OF RESULTS

- The analysis showed that loans frequency, cnt_loans30 & 90, amnt_loans30 and cnt_ma_rech30 & 90 were important determinants of target (defaulter/non-defaulter) variable.
- Around 35% of new users falls into defaulter category. This shows the flickering mindset of new users of not paying back the loan amount within stipulated time(within 5 days).
- Users taking loans in last 30 days has more defaulters than taking loans in last 90 days.
- Mostly users maintaining low balance is in defaulter list.
- Users consider to take loan amount of Rupiah 5 mostly where the payback loan amount is Rupiah 6 than the loan amount of Rupiah 10 where the payback loan amount is Rupiah 12. This depicts that they use mobile balance precisely.
- Users taking average number of loans are more in defaulter count than users taking too much of loans.
- Users maintaining high balance pay their loans on the stipulated time. That's why non-defaulter ones of them is 99%.
- The % of number of times of loan taken in last 30 days by users keeps on decreasing.
- Users taking average number of loans which is approx 20% are defaulters.

CONCLUSION

KEY FINDINGS AND CONCLUSIONS OF THE STUDY

From the whole project evaluation these are the inferences that I could draw from the visualization of data.

- The attributes for 90 days for all features would have given more insights of users behaviour pattern.
- New users are mostly in the defaulter list.
- Users who take small loans are in defaulter list.
- Users preferring to take loan maximum one time most in a month which tells how precisely they use their mobile balance to communicate.
- Users having high balance and are also defaulters are very less in numbers.
- Users who take more number of loans are in non-defaulters category.
- Users who do very high amount of recharge always pay their loans on time.
- 34% of the Users who do less amount of recharge are defaulters.
- Users in three months not recharging their mobile even once are in defaulter list.
- Random Forest Classifier is our best model as compared to others models with high accuracy of 92.12% and roc_auc score of 99%.

LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE

Through Visualization it was clear that the target variable was a imbalanced dataset. Also the features that depicted a lot of story telling when visualization was done for all of them. Visualization gives meaning to a data which helps drawing inference from it.

Lots of outliers was there so used 1.5 IQR method to overcome those and also kept in mind to not lose 7-8% of data.

Random Forest Classifier was the best model to be deployed in production because its accuracy and CV value was least among all models/.

The challenge that I faced while working on this project was that while doing hyper parameter tuning on the best model it took approx 20 hours to complete as the dataset was large. This gives the idea when we will deal with millions and trillions of data we have to do proper data cleaning and processing because we cannot run again and again our algorithms as it will take days to complete. Hence time & money loss will be there.

LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK

Time limitation was there.The future scope of project is that we can train the machine identify & restrict the frauds in micro credit business.

If category of gender (Male/Female) would have been there in the features then it might be very good for analysis part and further improvement of results by targeting those points.

PCA (Principal Component Analysis) was used to reduce the dimensionality of large dataset to retrieve better results.

While providing loan in terms of mobile balance to users also informing them about the outcome through sms, voice call in their regional language when they fail to repay within stipulated time. This could further aid the telecom business income generation by restricting frauds.