FLIP ROBO

Micro-Credit Defaulter Model

Prepared by: SME Name:

ARCHANA KUMARI SHWETANK MISHRA

Internship-29

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project “Micro-Credit Defaulter Model” and also want to thank my SME “Shwetank Mishra” for providing the dataset and directions to complete this project. This project would not have been accomplished without their help and insights.

I would also like to thank my academic “Data Trained Education” and their team who has helped me to learn Machine Learning and how to work on it.

Working on this project was an incredible experience as I learnt more from this Project during completion and also, I have to do some research from various learning website.

INTRODUCTION

1. Business Problem Framing

A Microfinance Institution (MFI) is an organization that offers financial services to low-income populations. 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.

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.

2. Conceptual Background of the Domain Problem

Telecom Industry is 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 us 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.

3. 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. 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.

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

4. Motivation for the Problem Undertaken

MFI industry is primarily focusing on low-income families. The Microfinance services (MFS) provided by MFI are Group Loans, Agricultural Loans, Individual Business Loans and so on. 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.

Analytical Problem Framing

1. Mathematical/ Analytical Modeling of the Problem

1) Descriptive Statistics
2) Analysed correlation
3) Detected Outliers and removed
4) Detected Skewness and removed
5) Handled Oversampling using SMOTE
6) Scaled data using Standard Scaler
7) Removed Multicollinearity

2. **Data Sources and their formats**

The data is provided by Flip Robo Technologies in the csv file: **Data file.csv**. Target and Features variables of this dataset are:

Target:

- **label:** Flag indicating whether the user paid back the credit amount within 5 days of issuing the loan {1: success, 0: failure}

Features:

- **msisdn:** mobile number of users
- **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 Indonasian Rupiah)
- **medianamnt_ma_rech90:** Median of amount of recharges done in main account over last 90 days at user level (in Indonasian Rupiah)
- **medianmarechprebal90:** Median of main account balance just before recharge in last 90 days at user level (in Indonasian 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

3. Data Pre-processing Done:

a) Checked Total Numbers of Rows and Column

```
defaulter.shape
```

```
(209593, 37)
```

b) Checked All Column Name

```
defaulter.columns
```

```
Index(['Unnamed: 0', 'label', 'msisdn', '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', 'pcircle', 'pdate'],
      dtype='object')
```

c) Checked Data Type of All Data

```
defaulter.dtypes
```

```
Unnamed: 0                int64
label                     int64
msisdn                   object
aon                     float64
daily_decr30            float64
daily_decr90            float64
rental30                float64
rental90                float64
last_rech_date_ma       float64
last_rech_date_da       float64
last_rech_amt_ma          int64
cnt_ma_rech30             int64
fr_ma_rech30            float64
sumamnt_ma_rech30       float64
medianamnt_ma_rech30    float64
medianmarechprebal30    float64
cnt_ma_rech90             int64
fr_ma_rech90              int64
sumamnt_ma_rech90         int64
medianamnt_ma_rech90    float64
medianmarechprebal90    float64
cnt_da_rech30           float64
fr_da_rech30            float64
cnt_da_rech90             int64
fr_da_rech90              int64
```

d) Checked for Null Values

```
defaulter.isnull().sum()
```

```
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
```

There is no null value in the dataset.

e) Information about Data

```
defaulter.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
```

f) Checked total number of unique values

```
defaulter.nunique()
```

```
Unnamed: 0              209593
label                        2
msisdn                  186243
aon                       4507
daily_decr30            147025
daily_decr90            158669
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
```

g) Data cleaning

- Column 'Unnamed: 0' contains serial no, so dropped this column.

Dropping column 'Unnamed: 0'

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

- Column 'pcircle' is containing only one type of data which is not important for prediction. So, dropped this column.

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

- converted data type from object to datetime

```
defaulter['pdate']=pd.to_datetime(defaulter['pdate'])
```

- Extracting year from pdate

```
#mapping year values from column 'pdate' to 'year' column in main dataframe
defaulter['year']=defaulter['pdate'].apply(lambda y:y.year)
```

- Extracting month from pdate

```
#mapping year values from column 'pdate' to 'month' column in main dataframe
defaulter['month']=defaulter['pdate'].apply(lambda m:m.month)
```

- Extracting day from pdate

```
#mapping Day values from column 'pdate' to 'day' column in main dataframe
defaulter['day']=defaulter['pdate'].apply(lambda d:d.day)
```

- Dropping column "Date_of_Journey" after extracting Day and Month

```
defaulter.drop(columns=['pdate'],inplace=True)
```

- We can see there is only one type of value that is 2016, thus it is irrelevant for prediction. So, will drop this column.

```
defaulter.drop(columns=['year'],inplace=True)
```

- Column 'msisdn' is not relevant for prediction. So, will drop this column too.

```
defaulter.drop(columns=['msisdn'],inplace=True)
```

h) Data Visualization

i. Univariate Analysis
   - Using Countplot
   - Using Histplot

ii. Bivariate Analysis
   (For comparison between each feature with target)
   - Using Barplot

iii. Multivariate Analysis
   (For comparison between all feature with target)
   - Using Heatmap

4. Data Inputs- Logic- Output Relationships

i) Checking Correlation

```
defaulter.corr()
```

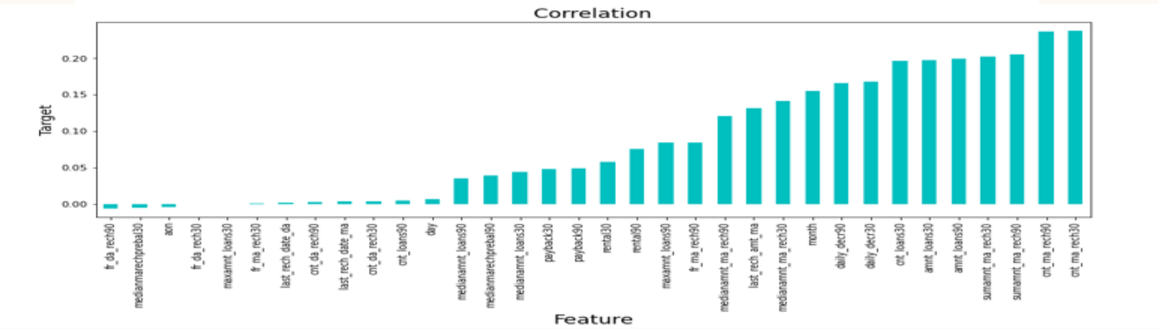
	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma
label	1.000000	-0.003785	0.168298	0.166150	0.058085	0.075521	0.003728	0.001711	0.131804
aon	-0.003785	1.000000	0.001104	0.000374	-0.000960	-0.000790	0.001692	-0.001693	0.004256
daily_decr30	0.168298	0.001104	1.000000	0.977704	0.442066	0.458977	0.000487	-0.001636	0.275837
daily_decr90	0.166150	0.000374	0.977704	1.000000	0.434685	0.471730	0.000908	-0.001886	0.264131
rental30	0.058085	-0.000960	0.442066	0.434685	1.000000	0.955237	-0.001095	0.003261	0.127271
rental90	0.075521	-0.000790	0.458977	0.471730	0.955237	1.000000	-0.001688	0.002794	0.121416
last_rech_date_ma	0.003728	0.001692	0.000487	0.000908	-0.001095	-0.001688	1.000000	0.001790	-0.000147
last_rech_date_da	0.001711	-0.001693	-0.001636	-0.001886	0.003261	0.002794	0.001790	1.000000	-0.000149
last_rech_amt_ma	0.131804	0.004256	0.275837	0.264131	0.127271	0.121416	-0.000147	-0.000149	1.000000
cnt_ma_rech30	0.237331	-0.003148	0.451385	0.426707	0.233343	0.230260	0.004311	0.001549	-0.002662
fr_ma_rech30	0.001330	-0.001163	-0.000577	-0.000343	-0.001219	-0.000503	-0.001629	0.001158	0.002876
sumamnt_ma_rech30	0.202828	0.000707	0.636536	0.603886	0.272649	0.259709	0.002105	0.000046	0.440821
medianamnt_ma_rech30	0.141490	0.004306	0.295356	0.282960	0.129853	0.120242	-0.001358	0.001037	0.794646
medianmarechprebal30	-0.004829	0.003930	-0.001153	-0.000746	-0.001415	-0.001237	0.004071	0.002849	-0.002342
cnt_ma_rech90	0.236392	-0.002725	0.587338	0.593069	0.312118	0.345293	0.004263	0.001272	0.016707
fr_ma_rech90	0.084385	0.004401	-0.078299	-0.079530	-0.033530	-0.036524	0.001414	0.000798	0.106267

```
defaulter.corr()["label"].sort_values()
```

```
fr_da_rech90            -0.005418
medianmarechprebal30    -0.004829
aon                     -0.003785
fr_da_rech30            -0.000027
maxamnt_loans30          0.000248
fr_ma_rech30             0.001330
last_rech_date_da        0.001711
cnt_da_rech90            0.002999
last_rech_date_ma        0.003728
cnt_da_rech30            0.003827
cnt_loans90              0.004733
day                      0.006825
medianamnt_loans90       0.035747
medianmarechprebal90     0.039300
medianamnt_loans30       0.044589
payback30                0.048336
payback90                0.049183
rental30                 0.058085
rental90                 0.075521
maxamnt_loans90          0.084144
fr_ma_rech90             0.084385
medianamnt_ma_rech90     0.120855
last_rech_amt_ma         0.131804
medianamnt_ma_rech30     0.141490
month                    0.154949
daily_decr90             0.166150
daily_decr30             0.168298
cnt_loans30              0.196283
amnt_loans30             0.197272
amnt_loans90             0.199788
sumamnt_ma_rech30        0.202828
sumamnt_ma_rech90        0.205793
cnt_ma_rech90            0.236392
cnt_ma_rech30            0.237331
label                    1.000000
```

- Correlation is checked for relation between the dependent and independent variables.
- Also Checked through heatmap and BarPlot (Visualization)
- Checking Correlation through Barplot :

```
plt.figure(figsize=(15,5))
defaulter.corr()['label'].sort_values(ascending=True).drop(['label']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```



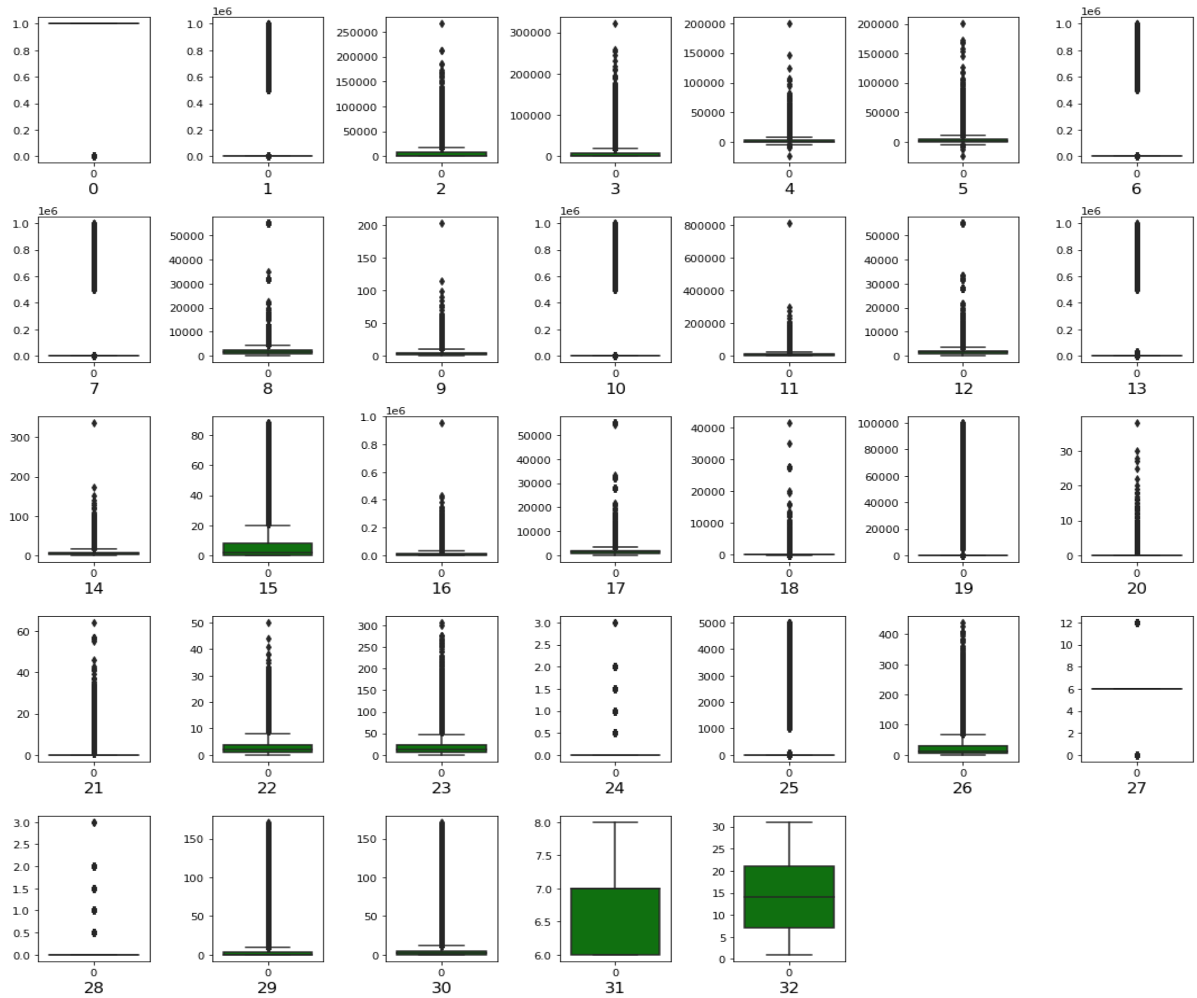
CHECKING OUTLIERS

- *Outliers are removed only from continuous features and not from target*

```
collist=defaulter.columns.values
ncol=33
nrows=7
plt.figure(figsize=(15,15))
for column in range(0,len(collist)):
    plt.subplot(5,7,column+1)
    sns.boxplot(data=defaulter[collist[column]],color='green',orient='v')
    plt.xlabel(column,fontsize=15)
    plt.tight_layout()
```

Observation:

- **Outliers present in columns:** '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', 'cnt_da_rech90', 'fr_da_rech90', 'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30', 'cnt_loans90', 'amnt_loans90','maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90',
- **Outliers not present in columns:** 'month', 'day'



REMOVING OUTLIERS

- Checking two methods and compare between them which is give less percentage loss and then using that method for further process.

1. Zscore method using Scipy
2. IQR (Inter Quantile Range) method

1. Zscore method using Scipy

```
#We will not remove outliers from Target column 'label'
variable = defaulter[['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',  'cnt_da_rech90',
                      'fr_da_rech90', 'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30', 'cnt_loans90',
                      'amnt_loans90','maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90']]

z1=np.abs(zscore(variable))

# Creating new dataframe
credit_defaulter = defaulter[(z1<3).all(axis=1)]
credit_defaulter.head()
```

	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	sumamr
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2	21.0	
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1	0.0	
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1	0.0	
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0	0.0	
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7	2.0	

Comparing shape of old and new DataFrame after outliers removal

```
print("Old DataFrame data in Rows and Column:",defaulter.shape)
print("New DataFrame data in Rows and Column:",credit_defaulter.shape)
print("Total Dropped rows:",defaulter.shape[0]-credit_defaulter.shape[0])
```

```
Old DataFrame data in Rows and Column: (209593, 33)
New DataFrame data in Rows and Column: (163026, 33)
Total Dropped rows: 46567
```

2. IQR (Inter Quantile Range) method

```
#1st quantile
Q1=variable.quantile(0.25)

# 3rd quantile
Q3=variable.quantile(0.75)

#IQR
IQR=Q3 - Q1
micro_defaulter=defaulter[~((defaulter < (Q1 - 1.5 * IQR)) |(defaulter > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Comparing shape of old and new DataFrame after outliers removal

```
print("Old DataFrame data in Rows and Column:",defaulter.shape)
print("\nNew DataFrame data in Rows and Column:",micro_defaulter.shape)
print("\nTotal Dropped rows:",defaulter.shape[0]-micro_defaulter.shape[0])
```

```
Old DataFrame data in Rows and Column: (209593, 33)

New DataFrame data in Rows and Column: (71330, 33)

Total Dropped rows: 138263
```

Comparing Data Loss Using both Method after Outlier Removal

Percentage Data Loss using Zscore

```
loss_percent1=(209593-163026)/209593*100
print(loss_percent1,"%")
```

```
22.217822160091224 %
```

Percentage Data Loss using IQR

```
loss_perc1 = (209593-57328)/209593*100
print(loss_perc1,"%")
```

```
72.64794148659544 %
```

We can check by using IQR method there is large data loss in comparison to Zscore method. So, we will consider Zscore method.

CHECKING SKEWNESS

```
credit_defaulter.skew()
```

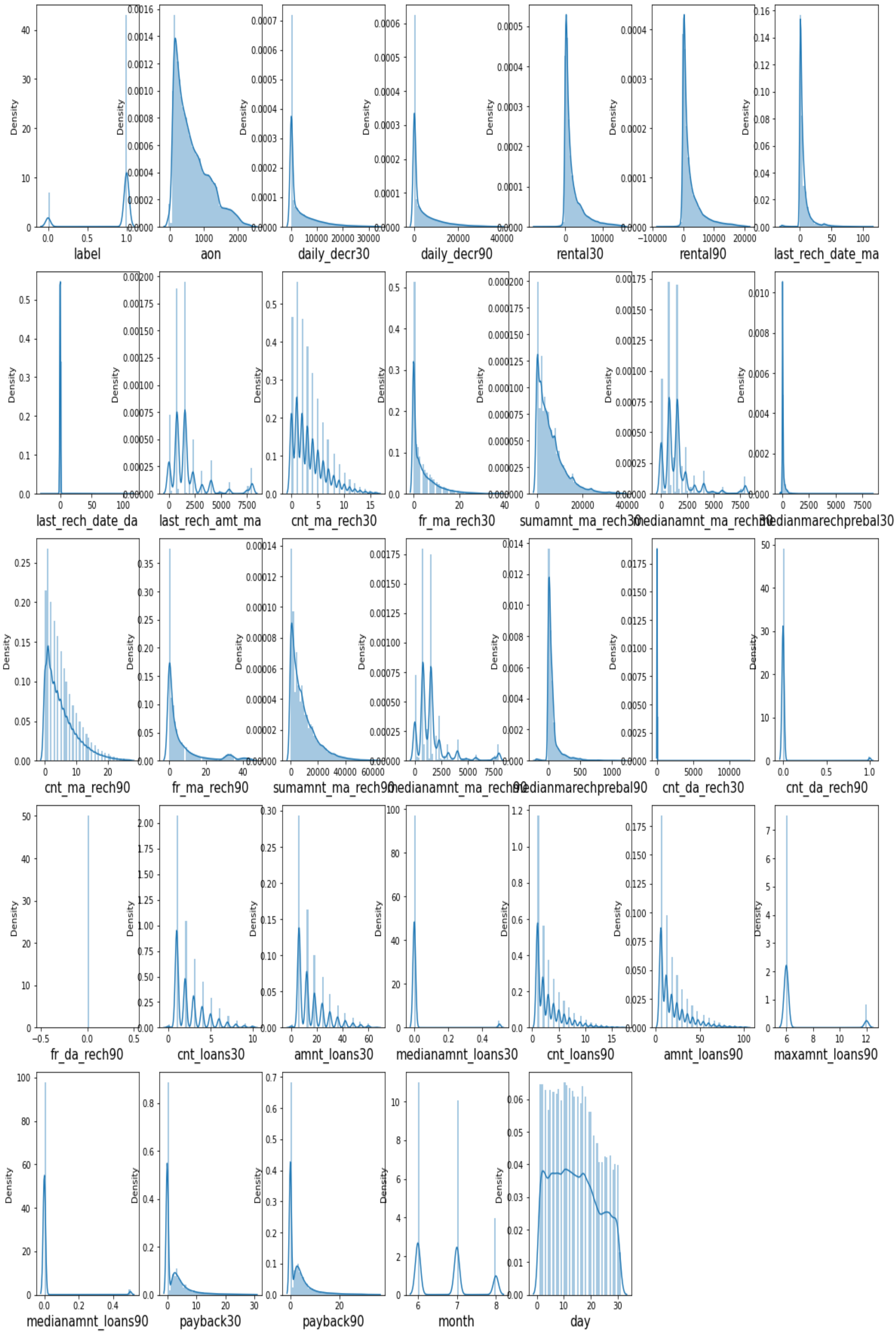
label	-2.090282
aon	0.958194
daily_decr30	1.963119
daily_decr90	2.077247
rental30	2.195563
rental90	2.244957
last_rech_date_ma	3.098605
last_rech_date_da	10.390692
last_rech_amt_ma	2.125025
cnt_ma_rech30	1.174958
fr_ma_rech30	2.005026
sumamnt_ma_rech30	1.635182
medianamnt_ma_rech30	2.325820
medianmarechprebal30	10.514000
cnt_ma_rech90	1.320723
fr_ma_rech90	1.984255
sumamnt_ma_rech90	1.706347
medianamnt_ma_rech90	2.372843
medianmarechprebal90	3.688959
cnt_da_rech30	51.006049
cnt_da_rech90	6.932690
fr_da_rech90	0.000000
cnt_loans30	1.465618
amnt_loans30	1.441398
medianamnt_loans30	5.355036
cnt_loans90	1.708099
amnt_loans90	1.694063
maxamnt_loans90	2.680114
medianamnt_loans90	6.100818
payback30	2.608750
payback90	2.528904
month	0.476131
day	0.190279

Observation:

- Skewness threshold taken is +/-0.50
- **Columns which are having skewness:** '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', 'cnt_da_rech90', 'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30', 'cnt_loans90', 'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90'
- **Columns which are not having skewness:** 'fr_da_rech90', 'month', 'day'
- Only the 'label' column data is negatively skewed and it is also our target column
- Column 'cnt_da_rech30' is highly positively skewed
- All the columns are not normallly distributed.
- We will not remove skewness from Target Column 'label'.

Checking skweness through Data Visualization

```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in credit_defaulter:
    if plotnumber<=33:
        ax = plt.subplot(5,7,plotnumber)
        sns.distplot(credit_defaulter[column])
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.show()
```



Density
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
cnt_da_rech90
fr_da_rech90
cnt_loans30
amnt_loans30
medianamnt_loans30
cnt_loans90
amnt_loans90
maxamnt_loans90
medianamnt_loans90
payback30
payback90
month
day

REMOVING SKEWNESS

Using yeo-johnson method

```
from sklearn.preprocessing import PowerTransformer
```

```
collist1=['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', 'cnt_da_rech90', 'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30',
         'cnt_loans90', 'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90']
pt = PowerTransformer(method='yeo-johnson')
credit_defaulter[collist1] = pt.fit_transform(credit_defaulter[collist1])
```

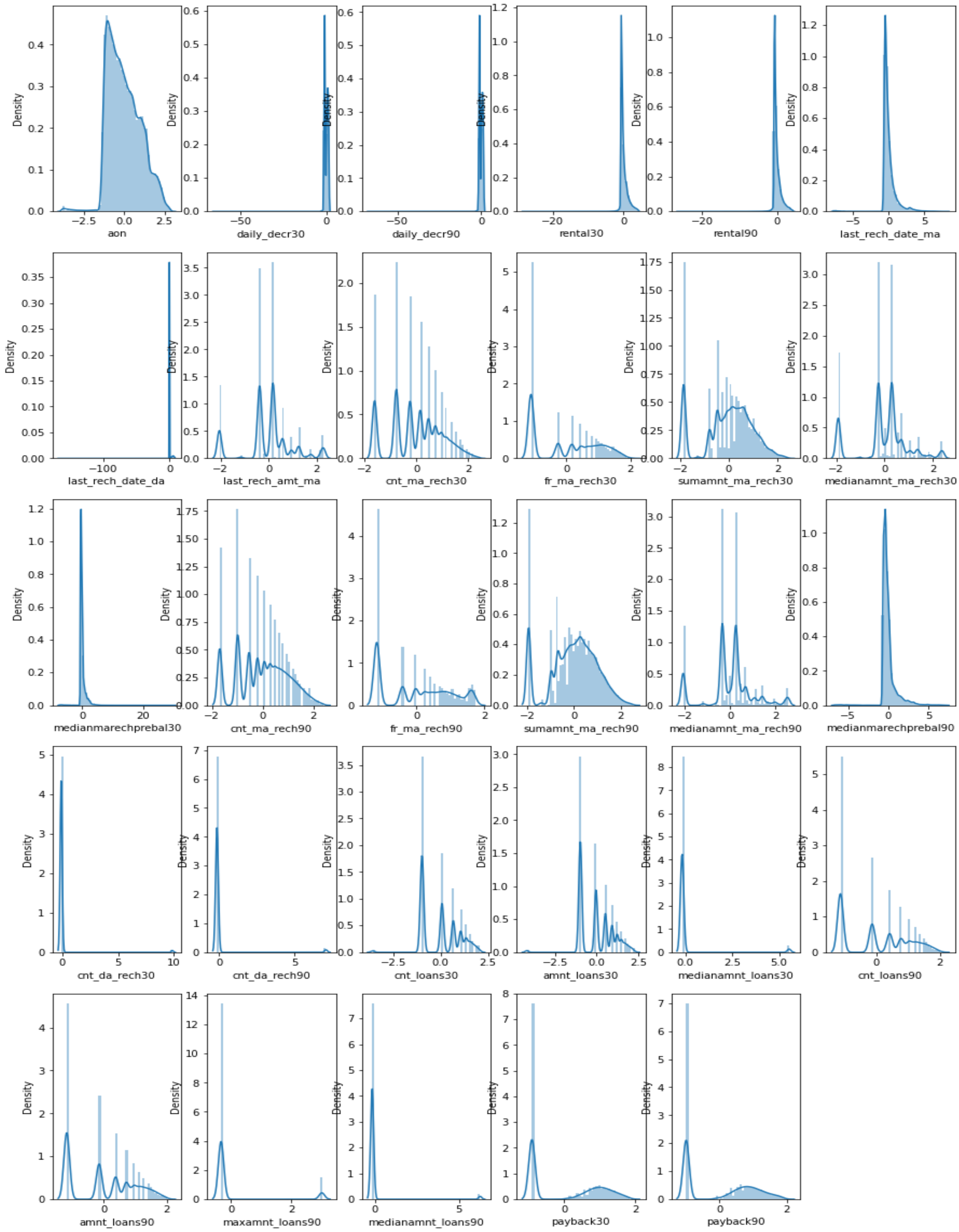
CHECKING SKEWNESS AFTER REMOVAL

```
credit_defaulter.skew()
```

label	-2.090282
aon	0.311720
daily_decr30	-1.974969
daily_decr90	-2.102745
rental30	0.203565
rental90	0.226037
last_rech_date_ma	0.134369
last_rech_date_da	-59.569519
last_rech_amt_ma	-0.185569
cnt_ma_rech30	-0.026529
fr_ma_rech30	0.137574
sumamnt_ma_rech30	-0.457447
medianamnt_ma_rech30	-0.313082
medianmarechprebal30	2.005466
cnt_ma_rech90	-0.029965
fr_ma_rech90	0.142800
sumamnt_ma_rech90	-0.365256
medianamnt_ma_rech90	-0.175668
medianmarechprebal90	0.994015
cnt_da_rech30	9.764166
cnt_da_rech90	6.932690
fr_da_rech90	0.000000
cnt_loans30	0.086332
amnt_loans30	-0.003124
medianamnt_loans30	5.355036
cnt_loans90	0.189622
amnt_loans90	0.123345
maxamnt_loans90	2.680114
medianamnt_loans90	6.100818
payback30	0.307386
payback90	0.209380
month	0.476131
day	0.190279

checking skewness after removal through data visualization using distplot

```
collist=['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', 'cnt_da_rech90', 'cnt_loans30', 'amnt_loans30', 'medianamnt_loans30',
         'cnt_loans90', 'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90']
plt.figure(figsize=(15,25), facecolor='white')
plotnumber = 1

for column in credit_defaulter[collist]:
    if plotnumber<=29:
        ax = plt.subplot(5,6,plotnumber)
        sns.distplot(credit_defaulter[column])
        plt.xlabel(column,fontsize=10)
    plotnumber+=1
plt.show()
```


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
cnt_da_rech90
cnt_loans30
amnt_loans30
medianamnt_loans30
cnt_loans90
amnt_loans90
maxamnt_loans90
medianamnt_loans90
payback30
payback90
Density

Spliting data into Target and Features:

```
x=credit_defaulter.drop("label",axis=1)
y=credit_defaulter["label"]
```

```
x.head()
```

	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
0	-0.712297	0.582660	0.555598	-0.624890	-0.616794	-0.364567	-0.099221	0.181981	-0.246169	1.704555
1	0.262696	1.309589	1.255917	0.693145	0.436130	1.435815	-0.099221	1.760082	-0.777359	-1.105248
2	-0.095609	0.239380	0.220925	-0.321585	-0.384763	-0.242427	-0.099221	0.181981	-0.777359	-1.105248
3	-0.795794	-0.987441	-0.983084	-0.656214	-0.659128	3.159851	-0.099221	-0.221974	-1.621580	-1.105248
4	0.696476	-0.523008	-0.524342	-0.240988	-0.319129	-0.125792	-0.099221	0.582742	1.163872	0.169776

```
y.head()
```

```
0    0
1    1
2    1
3    1
4    1
Name: label, dtype: int64
```

```
x.shape, y.shape
```

```
((163026, 32), (163026,))
```

```
y.value_counts()
```

```
1    140409
0     22617
Name: label, dtype: int64
```

Oversampling using the SMOTE

```
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
```

```
SM = SMOTE()
x, y = SM.fit_resample(x,y)
y.value_counts()
```

```
0    140409
1    140409
Name: label, dtype: int64
```

Scaling data using Standard Scaler

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

```
x.head()
```

	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_
0	-0.608904	0.849890	0.827763	-0.570103	-0.558061	-0.404528	-0.095756	0.428880	0.131975	1.998770	
1	0.377844	1.578415	1.532290	0.772326	0.543217	1.229821	-0.095756	1.839070	-0.388517	-0.831828	
2	0.015220	0.505856	0.491079	-0.261184	-0.315375	-0.293651	-0.095756	0.428880	-0.388517	-0.831828	
3	-0.693407	-0.723659	-0.720164	-0.602007	-0.602340	2.794863	-0.095756	0.067906	-1.215735	-0.831828	
4	0.816853	-0.258206	-0.258666	-0.179096	-0.246726	-0.187772	-0.095756	0.787000	1.513619	0.452632	

Checking for Multicolinearity

VIF (Variance Inflation factor)

```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.032092	aon
1	602.152913	daily_decr30
2	641.169149	daily_decr90
3	19.676605	rental30
4	21.125132	rental90
5	2.278583	last_rech_date_ma
6	2.039850	last_rech_date_da
7	12.083240	last_rech_amt_ma
8	70.405989	cnt_ma_rech30
9	2.898250	fr_ma_rech30
10	131.818824	sumamnt_ma_rech30
11	29.916143	medianamnt_ma_rech30
12	3.438447	medianmarechprebal30
13	77.486055	cnt_ma_rech90
14	2.568925	fr_ma_rech90
15	119.661888	sumamnt_ma_rech90
16	31.674810	medianamnt_ma_rech90
17	3.449040	medianmarechprebal90
18	2.076854	cnt_da_rech30
19	3.487085	cnt_da_rech90
20	NaN	fr_da_rech90
21	198.354245	cnt_loans30
22	179.880360	amnt_loans30
23	3.362461	medianamnt_loans30
24	203.876076	cnt_loans90
25	199.735993	amnt_loans90
26	4.933642	maxamnt_loans90
27	3.374421	medianamnt_loans90
28	7.764739	payback30
29	7.678771	payback90
30	5.083654	month
31	1.170405	day

The VIF value is more than 10 in the columns: 'daily_decr30', 'daily_decr90', 'rental30', 'rental90', 'last_rech_amt_ma', 'cnt_ma_rech30', 'sumamnt_ma_rech30', 'medianamnt_ma_rech30', 'cnt_ma_rech90', 'sumamnt_ma_rech90', 'medianamnt_ma_rech90', 'cnt_loans30', 'amnt_loans30', 'cnt_loans90', 'amnt_loans90'. But column 'daily_decr90' is having highest VIF value. So, we will drop column 'daily_decr90' and also column 'fr_da_rech90' as it have no relation.

```
x.drop(['daily_decr90', 'fr_da_rech90'], axis =1, inplace=True)
```

Checking again Multicolinearity using VIF

```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.031949	aon
1	5.934306	daily_decr30
2	18.649651	rental30
3	20.061224	rental90
4	2.273022	last_rech_date_ma
5	2.039812	last_rech_date_da
6	12.077969	last_rech_amt_ma
7	70.354824	cnt_ma_rech30
8	2.896367	fr_ma_rech30
9	130.073285	sumamnt_ma_rech30
10	29.614100	medianamnt_ma_rech30
11	3.438164	medianmarechprebal30
12	77.470075	cnt_ma_rech90
13	2.566073	fr_ma_rech90
14	117.655263	sumamnt_ma_rech90
15	31.554868	medianamnt_ma_rech90
16	3.448209	medianmarechprebal90
17	2.076754	cnt_da_rech30
18	3.487085	cnt_da_rech90
19	198.144043	cnt_loans30
20	179.879871	amnt_loans30
21	3.362396	medianamnt_loans30
22	203.578171	cnt_loans90
23	199.729166	amnt_loans90
24	4.933129	maxamnt_loans90
25	3.374404	medianamnt_loans90
26	7.760005	payback30
27	7.672636	payback90
28	4.738079	month
29	1.170206	day

The VIF value is more than 10 in the columns: 'rental30', 'rental90', 'last_rech_amt_ma', 'cnt_ma_rech30', 'sumamnt_ma_rech30', 'medianamnt_ma_rech30', 'cnt_ma_rech90', 'sumamnt_ma_rech90', 'medianamnt_ma_rech90', 'cnt_loans30', 'amnt_loans30', 'cnt_loans90', 'amnt_loans90'. But column 'cnt_loans30' is having highest VIF value. So, we will drop column 'cnt_loans30' .

```
x.drop('cnt_loans30', axis =1, inplace=True)
```

Like this have checked multicollinearity total 10 times and removed those features which are having multicollinearity.

At last we have features:

	VIF values	Features
0	1.027206	aon
1	5.622551	daily_decr30
2	1.232097	rental30
3	1.885739	last_rech_date_ma
4	2.039491	last_rech_date_da
5	6.482339	last_rech_amt_ma
6	3.807674	cnt_ma_rech30
7	2.818386	fr_ma_rech30
8	5.907733	medianamnt_ma_rech30
9	3.408846	medianmarechprebal30
10	2.420967	fr_ma_rech90
11	3.405983	medianmarechprebal90
12	2.075243	cnt_da_rech30
13	3.485922	cnt_da_rech90
14	3.878720	amnt_loans30
15	3.355606	medianamnt_loans30
16	1.307836	maxamnt_loans90
17	3.340617	medianamnt_loans90
18	7.276607	payback30
19	6.165986	payback90
20	4.258115	month
21	1.157511	day

Variance Threshold Method

It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features.

```
var_threshold = VarianceThreshold(threshold=0)
var_threshold.fit(x)
```

```
VarianceThreshold(threshold=0)
```

```
var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True])
```

```
x.columns[var_threshold.get_support()]
```

```
Index(['aon', 'daily_decr30', 'rental30', 'last_rech_date_ma',
       'last_rech_date_da', 'last_rech_amt_ma', 'cnt_ma_rech30',
       'fr_ma_rech30', 'medianamnt_ma_rech30', 'medianmarechprebal30',
       'fr_ma_rech90', 'medianmarechprebal90', 'cnt_da_rech30',
       'cnt_da_rech90', 'amnt_loans30', 'medianamnt_loans30',
       'maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90',
       'month', 'day'],
      dtype='object')
```

```
# taking out all the constant columns
cons_columns = [column for column in x.columns
                if column not in x.columns[var_threshold.get_support()]]
print(len(cons_columns))
```

```
0
```

So we can see that, with the help of variance threshold method, we got to know all the features here are important. So now we will use SelectKBest method.

SelectKBest method ¶

```
best_fit = SelectKBest(score_func = f_classif, k ='all')
fit = best_fit.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = best_fit.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(22,'Score'))
```

```
                 Feature          Score
6          cnt_ma_rech30  101518.482698
14          amnt_loans30   64799.773254
19             payback90   54842.280291
18             payback30   51868.418683
7           fr_ma_rech30   50934.493102
8   medianamnt_ma_rech30   50267.063247
1           daily_decr30   43749.556320
5       last_rech_amt_ma   43733.613390
10          fr_ma_rech90   42525.152222
20                 month   15745.367836
9    medianmarechprebal30   14265.503557
11   medianmarechprebal90   13757.045628
0                    aon    6637.508139
16       maxamnt_loans90    6390.067000
3      last_rech_date_ma    3025.002050
2               rental30    2131.291877
15    medianamnt_loans30    1279.784280
17    medianamnt_loans90     692.126631
13         cnt_da_rech90     519.675407
4      last_rech_date_da     345.718360
12         cnt_da_rech30     208.889821
21                   day     173.980807
```

Selecting the best features based on above scores, we can see that the column "day" has most lowest features for the prediction, so we will drop this column.

```
x = x.drop([ "day"],axis=1)
x.shape
```

```
(280818, 21)
```

```
x.head()
```

	aon	daily_decr30	rental30	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rech30	medianamnt_ma_rech30	medianm
0	-0.608424	0.849432	-0.572503	-0.404719	-0.094347	0.429433	0.131498	2.000581	0.556187	
1	0.378413	1.578140	0.773042	1.230774	-0.094347	1.841492	-0.388959	-0.831266	1.962568	
2	0.015756	0.505312	-0.262867	-0.293765	-0.094347	0.429433	-0.388959	-0.831266	0.556187	
3	-0.692935	-0.724511	-0.604481	2.796914	-0.094347	0.067980	-1.216122	-0.831266	-1.424439	
4	0.817462	-0.258942	-0.180588	-0.187812	-0.094347	0.788027	1.513050	0.453761	0.913262	

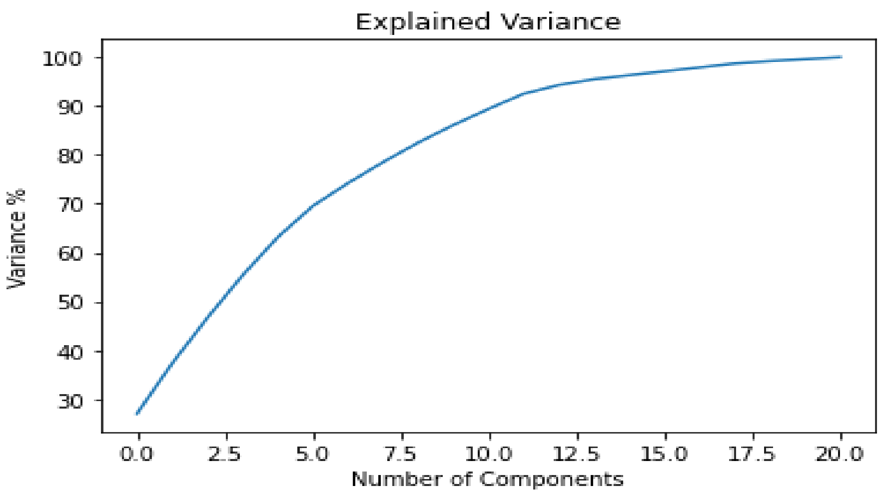
Principle Component Analysis

```
from sklearn.decomposition import PCA
```

```
covar_matrix = PCA(n_components = len(x.columns))
covar_matrix.fit(x)
```

```
PCA(n_components=21)
```

```
variance = covar_matrix.explained_variance_ratio_
var=np.cumsum(np.round(covar_matrix.explained_variance_ratio_, decimals=3)*100)

plt.figure()
plt.plot(var)
plt.xlabel('Number of Components')
plt.ylabel('Variance %')
plt.title('Explained Variance')
plt.show()
```



```
pca=PCA(n_components=10)
xpca=pca.fit_transform(x)
x=xpca
```

```
x
```

```
array([[-3.03021578,  0.67864881, -1.34135419, ...,  0.09074626,
        -1.59390659, -0.99388932],
       [-1.72843604, -0.32132601,  1.15163148, ...,  0.10972497,
         2.72819807, -0.61144553],
       [ 0.49410687, -0.14254497,  0.64332631, ..., -0.27282335,
        -0.43782886,  0.05534136],
       ...,
       [ 3.20861114,  0.1855121 , -0.6341109 , ..., -0.30334716,
         0.16127073, -0.40588813],
       [-1.04040334,  0.0928366 ,  0.1566486 , ..., -1.45914986,
        -0.91365485,  1.71327319],
       [ 3.39744158,  0.20530322, -0.56518249, ...,  0.02226151,
        -0.07529797, -0.25202274]])
```

```
x=pd.DataFrame(data=x)
```

5. State the set of assumptions (if any) related to the problem under consideration

- By observing Target Variable "label" it is already assumed that it is a Classification Problem and to understand it have to use classification model. And also, data was imbalanced so have to use oversampling method.
- Also, it was observed that there is one column "Unnamed 0" which is irrelevant column as it contains serial no so have to drop this column.
- Have to extract day, month and year from "pdate" column.

6. Hardware and Software Requirements and Tools Used

- **Hardware used:**
  - **Processor**: 11th Gen Intel(R) Core(TM) i3-1125G4 @ 2.00GHz 2.00 GHz
  - **System Type**: 64-bit OS
- **Software used:**
  - **Anaconda** for 64-bit OS
  - **Jupyter** notebook

- **<u>Tools, Libraries and Packages used:</u>**

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from scipy.stats import zscore
from sklearn.preprocessing  import power_transform, StandardScaler, LabelEncoder
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score,plot_roc_curve,  accuracy_score, classification_report
from sklearn.metrics import confusion_matrix, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

import pickle
```

```
from sklearn.preprocessing import PowerTransformer
```

```
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
```

```
from sklearn.decomposition import PCA
```

```
from xgboost import XGBClassifier
```

```
from sklearn.metrics import plot_roc_curve
```

<u>Model/s Development and Evaluation</u>

1. <u>Identification of possible problem-solving approaches (methods)</u>

   In this project, we want to predict the micro-credit defaulter and for this we have used these approaches:

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Information about Data

- Checked total number of unique values
- Extracted Hidden Features
- Dropped irrelevant Features
- Checked defaulter and non-defaulter through visualization.
- Checked correlation of features with target
- Detected Outliers and removed
- Checked skewness and removed
- Scaled data using Standard Scaler
- Handled Imbalanced Data though oversampling method using SMOTE
- Checked Multicollinearity
- Used Feature Selection Method:
  Variance threshold method and SelectKBest method
- Performed PCA (Principle Component Analysis)

2. Testing of Identified Approaches (Algorithms)

1. Logistic Regression
2. Random Forest Regressor
3. KNN Regressor
4. Gradient Boosting Regressor
5. Decision Tree Regressor
6. XGBoost Classifier

3. Run and evaluate selected models

Creating Model

We are using Classifier Models for prediction.

Finding the best random state among all the models

```
maxAccu=0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test= train_test_split(x,y,test_size= .30, random_state= i)
    DTC = DecisionTreeClassifier()
    DTC.fit(x_train, y_train)
    pred = DTC.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.791147354177053  on Random_state  48
```

Creating train-test-split

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .30, random_state = maxRS)
```

1. Logistic Regression

```
lr=LogisticRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)

print("accuracy_score: ", accuracy_score(y_test, pred_lr))
print("confusion_matrix: \n", confusion_matrix(y_test, pred_lr))
print("classification_report: \n", classification_report(y_test,pred_lr))
```

```
accuracy_score:  0.7617334947653301
confusion_matrix:
 [[32507  9498]
 [10575 31666]]
classification_report:
               precision    recall  f1-score   support

           0       0.75      0.77      0.76     42005
           1       0.77      0.75      0.76     42241

    accuracy                           0.76     84246
   macro avg       0.76      0.76      0.76     84246
weighted avg       0.76      0.76      0.76     84246
```

2. Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(x_train,y_train)
pred_rfc = rfc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_rfc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_rfc))
print("classification_report: \n",classification_report(y_test,pred_rfc))
```

```
accuracy_score:  0.853108752937825
confusion_matrix:
 [[35710  6295]
 [ 6080 36161]]
classification_report:
               precision    recall  f1-score   support

           0       0.85      0.85      0.85     42005
           1       0.85      0.86      0.85     42241

    accuracy                           0.85     84246
   macro avg       0.85      0.85      0.85     84246
weighted avg       0.85      0.85      0.85     84246
```

3. Decision Tree Classifier

```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc = dtc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_dtc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_dtc))
print("classification_report: \n",classification_report(y_test,pred_dtc))
```

```
accuracy_score:  0.791147354177053
confusion_matrix:
 [[33801  8204]
 [ 9391 32850]]
classification_report:
               precision    recall  f1-score   support

           0       0.78      0.80      0.79     42005
           1       0.80      0.78      0.79     42241

    accuracy                           0.79     84246
   macro avg       0.79      0.79      0.79     84246
weighted avg       0.79      0.79      0.79     84246
```

4. KNN Classifier

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_knn))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_knn))
print("classification_report: \n",classification_report(y_test,pred_knn))
```

```
accuracy_score:  0.8251192934976141
confusion_matrix:
 [[36575  5430]
 [ 9303 32938]]
classification_report:
               precision    recall  f1-score   support

           0       0.80      0.87      0.83     42005
           1       0.86      0.78      0.82     42241

    accuracy                           0.83     84246
   macro avg       0.83      0.83      0.82     84246
weighted avg       0.83      0.83      0.82     84246
```

5. Gradient Boosting Classifier

```
gb = GradientBoostingClassifier(n_estimators =100,learning_rate=0.1, max_depth=4)
gb.fit(x_train,y_train)
pred_gb = gb.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_gb))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_gb))
print("classification_report: \n",classification_report(y_test,pred_gb))
```

```
accuracy_score:  0.7868978942620422
confusion_matrix:
 [[32539  9466]
 [ 8487 33754]]
classification_report:
               precision    recall  f1-score   support

           0       0.79      0.77      0.78     42005
           1       0.78      0.80      0.79     42241

    accuracy                           0.79     84246
   macro avg       0.79      0.79      0.79     84246
weighted avg       0.79      0.79      0.79     84246
```

6. XGBoost Classifier

```
from xgboost import XGBClassifier
```

```
XGBC= XGBClassifier()
XGBC.fit(x_train,y_train)
pred_XGBC = XGBC.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_XGBC))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_XGBC))
print("classification_report: \n",classification_report(y_test,pred_XGBC))
```

```
accuracy_score:  0.8045841939083161
confusion_matrix:
 [[32989  9016]
 [ 7447 34794]]
classification_report:
               precision    recall  f1-score   support

           0       0.82      0.79      0.80     42005
           1       0.79      0.82      0.81     42241

    accuracy                           0.80     84246
   macro avg       0.81      0.80      0.80     84246
weighted avg       0.80      0.80      0.80     84246
```

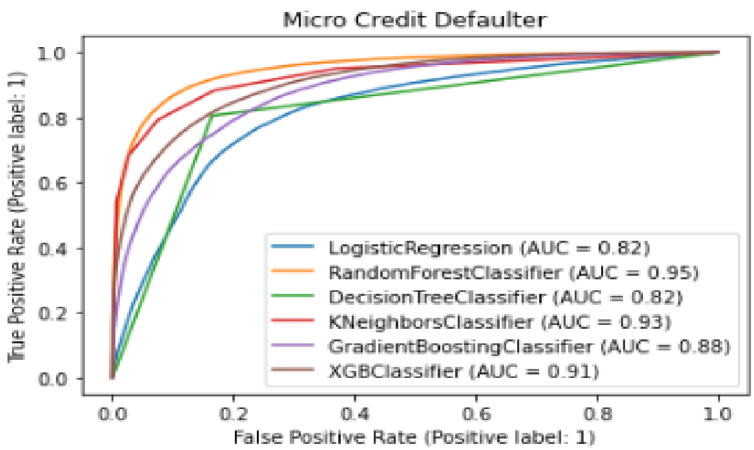
Cross Validation Score for all the model

```
#CV Score for Logistic Regression
print('CV score for Logistic Regression: ',cross_val_score(lr,x,y,cv=5).mean())

#CV Score for Random Forest Classifier
print('CV score for Random forest Classifier: ',cross_val_score(rfc,x,y,cv=5).mean())

#CV Score for Decision Tree Classifier
print('CV score for Decision Tree Classifier: ',cross_val_score(dtc,x,y,cv=5).mean())

#CV Score for KNN Classifier
print('CV score for KNN Classifier: ',cross_val_score(knn,x,y,cv=5).mean())

#CV Score for Gradient Boosting Classifier
print('CV score for Gradient Boosting Classifier: ',cross_val_score(gb,x,y,cv=5).mean())

#CV Score for XGB Classifier
print('CV score for XGB Classifier: ',cross_val_score(XGBC,x,y,cv=5).mean())
```

```
CV score for Logistic Regression:  0.7609020864662165
CV score for Random forest Classifier:  0.858741241237119
CV score for Decision Tree Classifier:  0.7942973754792039
CV score for KNN Classifier:  0.8331837565630222
CV score for Gradient Boosting Classifier:  0.7846648106909782
CV score for XGB Classifier:  0.8040439138861641
```

ROC & AUC Curve for all model

```
#Lets plot roc curve and check auc and performance of all algorithms
from sklearn.metrics import plot_roc_curve
disp = plot_roc_curve(lr, x_test, y_test)
plot_roc_curve(rfc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(dtc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
plot_roc_curve(gb, x_test, y_test, ax = disp.ax_)
plot_roc_curve(XGBC, x_test, y_test, ax = disp.ax_)
plt.title("Micro Credit Defaulter")
plt.legend(prop={"size" :10} ,loc = 'lower right')
plt.show()
```



Hyper parameter tuning for best model using GridsearchCV

The Random Forest Classifier with GridsearchCV

```
rfc=RandomForestClassifier(random_state=33,n_estimators=320)
```

```
parameters = {
    'n_estimators'      : [100, 320,330,340],
    'max_depth'         : [8, 9, 10, 11, 12],
    'random_state'      : [0]
}
```

```
clf = GridSearchCV(RandomForestClassifier(n_estimators=320), parameters, cv=5)
```

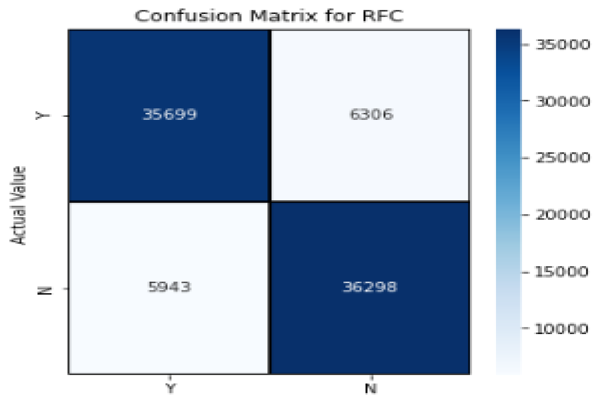
```
clf.fit(x_train, y_train)
```

```
RandomForestClassifier(n_estimators=320)
```

```
micro_credit_defaulter =RandomForestClassifier(n_estimators=320, random_state=33)
micro_credit_defaulter.fit(x_train, y_train)
pred = micro_credit_defaulter.predict(x_test)
# calculating the scores
print("accuracy score: ",accuracy_score(y_test,pred))
print("confusion_matrix: \n",confusion_matrix(y_test,pred))
print("classification_report: \n",classification_report(y_test,pred))
```

```
accuracy score:  0.8546043729079126
confusion_matrix:
 [[35699  6306]
 [ 5943 36298]]
classification_report:
               precision    recall  f1-score   support

           0       0.86      0.85      0.85     42005
           1       0.85      0.86      0.86     42241

    accuracy                           0.85     84246
   macro avg       0.85      0.85      0.85     84246
weighted avg       0.85      0.85      0.85     84246
```

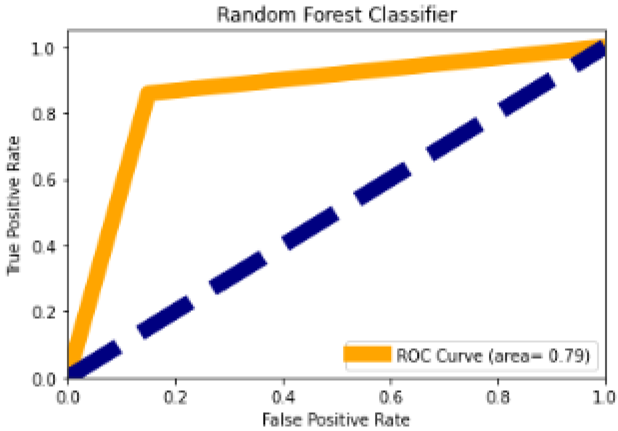
```
cm = confusion_matrix(y_test, pred)

x_axis_labels = ["Y","N"]
y_axis_labels = ["Y","N"]

f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black",
plt.xlabel("Predicted Value")
plt.ylabel("Actual Value ")
plt.title('Confusion Matrix for RFC')
plt.show()
```



ROC-AUC Curve

```
fpr,tpr,thresholds = roc_curve(y_test,pred,pos_label=True)
```

```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label = "ROC Curve (area= %0.2f)" % acc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Random Forest Classifier")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So, the model has the area under curve as 0.79.

Saving the Classification Model

```
filename='micro_credit_defaulter.pickle'
pickle.dump(rfc,open(filename,'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
```

Checking predicted and original values

```
a =np.array(y_test)
predicted=np.array(micro_credit_defaulter.predict(x_test))
Micro_Credit_Defaulter_Model=pd.DataFrame({'Orginal':a,'Predicted':predicted}, index=range(len(a)))
Micro_Credit_Defaulter_Model
```

	Orginal	Predicted
0	0	1
1	0	0
2	1	1
3	0	1
4	1	1
5	0	0
6	1	1
7	1	1

Saving the model in CSV format

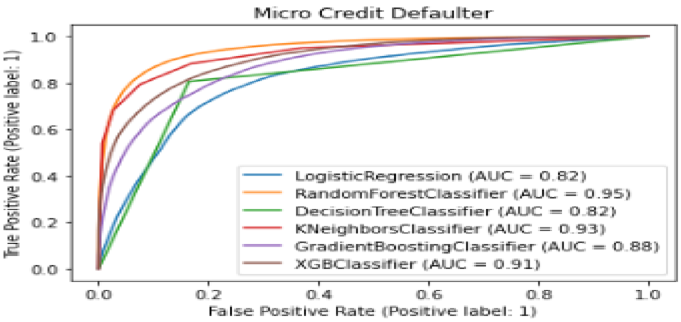
```
model =Micro_Credit_Defaulter_Model.to_csv('Micro_Credit_Defaulter_Model.csv')
model
```

4. Key Metrics for success in solving problem under consideration

- Accuracy Score, CV score, Precision Score, recall, AUC-ROC Curve Metrics are used for success.

ROC & AUC Curve for all model

```
#Lets plot roc curve and check auc and performance of all algorithms
from sklearn.metrics import plot_roc_curve
disp = plot_roc_curve(lr, x_test, y_test)
plot_roc_curve(rfc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(dtc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
plot_roc_curve(gb, x_test, y_test, ax = disp.ax_)
plot_roc_curve(XGBC, x_test, y_test, ax = disp.ax_)
plt.title("Micro Credit Defaulter")
plt.legend(prop={"size" :10} ,loc = 'lower right')
plt.show()
```

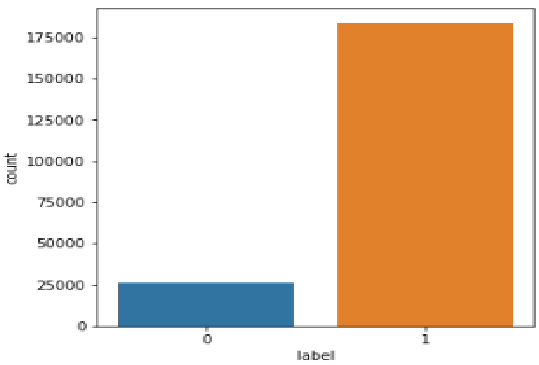


5. Visualization

- Univariate Analysis
  - Using Countplot

```
plt.figure(figsize=(5,5))
sns.countplot(x='label', data=defaulter)
```

<AxesSubplot:xlabel='label', ylabel='count'>

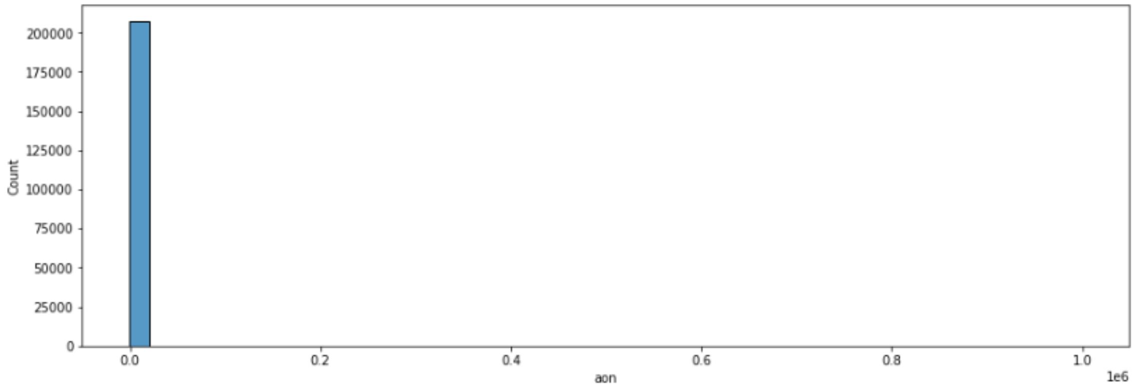


- Non-Defaulter are most and Defaulter are least
- We can see more than 175000 is Non-Defaulter and 25000 is Defaulter

➢ Using Histplot

```
plt.figure(figsize=(15,5))
sns.histplot(x='aon', data=defaulter, bins=50)
```

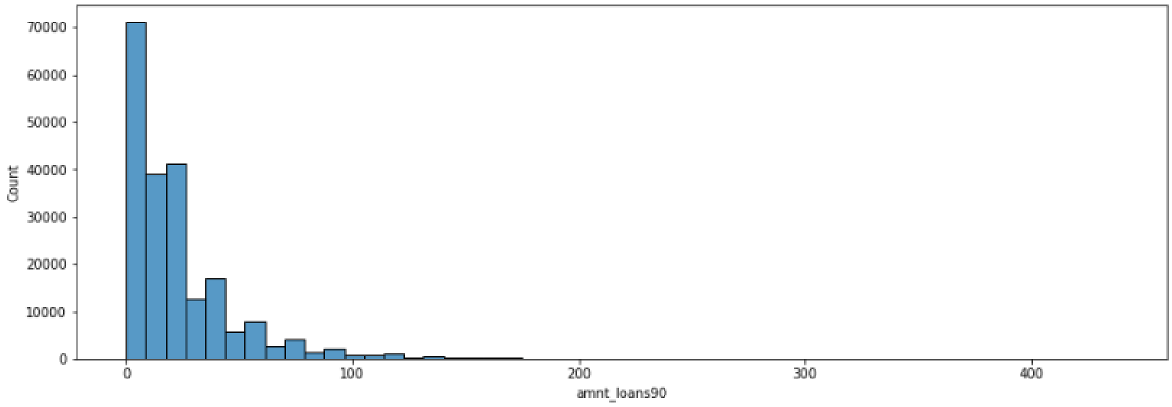
<AxesSubplot:xlabel='aon', ylabel='Count'>



More than 200000 is age on cellular network in days

```
plt.figure(figsize=(15,5))
sns.histplot(x='amnt_loans90', data=defaulter, bins=50)
```

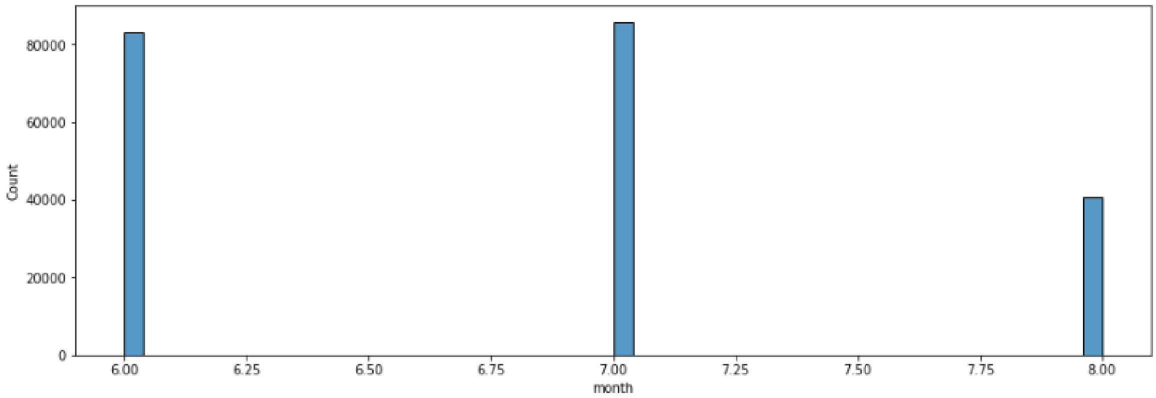
<AxesSubplot:xlabel='amnt_loans90', ylabel='Count'>



Mostly 71000 is Total amount of loans taken by user in last 90 days

```
plt.figure(figsize=(15,5))
sns.histplot(x='month', data=defaulter, bins=50)
```

<AxesSubplot:xlabel='month', ylabel='Count'>



Most credit is 85000 in 7th month

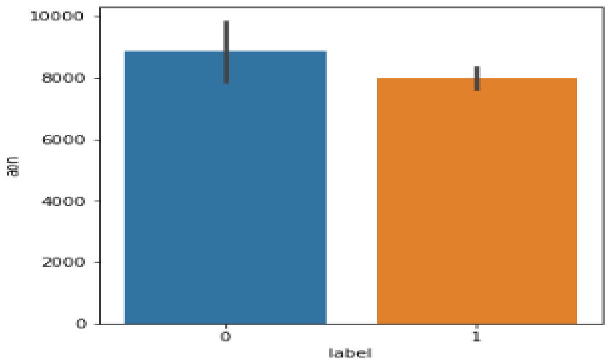
- Bivariate Analysis

  (For comparison between each feature with target)

  - Using Barplot

```
#BarPlot for comparision between "aon" column and "label" column
plt.figure(figsize=(5,5))
sns.barplot(y="aon",data=defaulter, x='label')
```

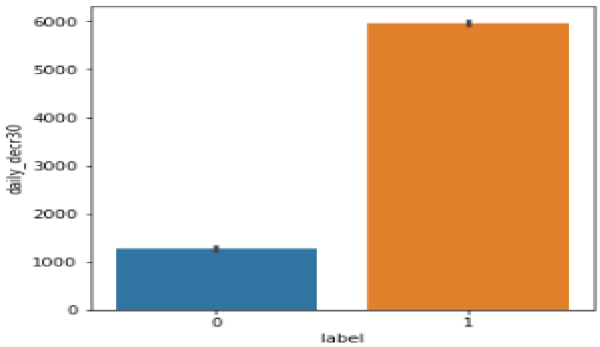
```
<AxesSubplot:xlabel='label', ylabel='aon'>
```



- Defaulter are most (More than 8500) compare to Non-Defaulter (More than 8000)

```
#BarPlot for comparision between "daily_decr30" column and "label" column
plt.figure(figsize=(5,5))
sns.barplot(y="daily_decr30",data=defaulter, x='label')
```

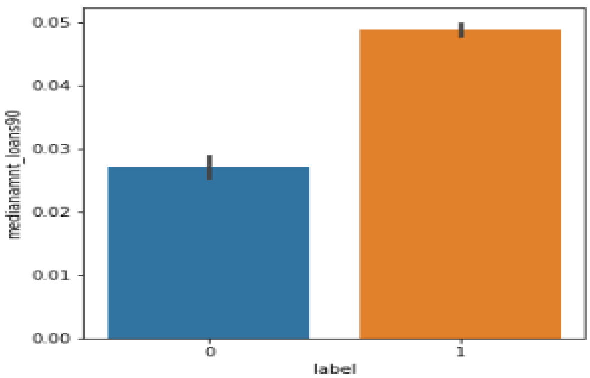
```
<AxesSubplot:xlabel='label', ylabel='daily_decr30'>
```



Non-Defaulter are most is 5900 compare to Defaulter is 1200

```
#BarPlot for comparision between "medianamnt_loans90" column and "label" column
plt.figure(figsize=(5,5))
sns.barplot(y="medianamnt_loans90",data=defaulter, x='label')
```

```
<AxesSubplot:xlabel='label', ylabel='medianamnt_loans90'>
```



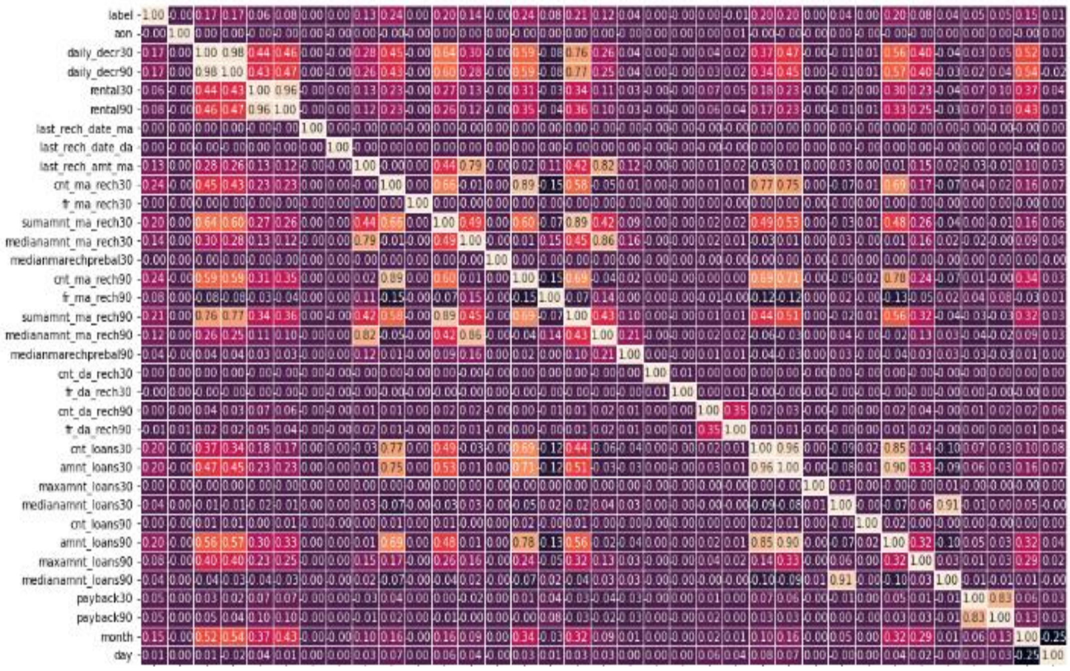
Non-Defaulter is more (Total No= 0.028) compare to Defaulter (Total No= 0.048)

- Multivariate Analysis

  (For comparison between all feature with target)

  - Using Heatmap

```
plt.figure(figsize=(15,10))
sns.heatmap(defaulter.transpose(),2) ,annot=True,annot_kws= {"size": 10}, linewidth=0.5, linecolor='white', fmt='.2f')
```



6. Interpretation of the Results

- Through Visualization it is interpretated that Data is Imbalanced; Data is skewed due to presence of outliers in Dataset.
- Through Pre-processing it is interpretated that hidden features need to extracted, outliers & skewness was present in dataset, data was improper scaled, multicollinearity was present, PCA is required.
- By creating/building model we get best model: Random Forest Classifier.

CONCLUSION

1. Key Findings and Conclusions of the Study

Here we have predicted defaulter Micro-Credit which will help in further investment and in selection of customers, we have done prediction on basis of Data using EDA, Data Visualization, Data Pre-processing, Checked Correlation, removed irrelevant features, Removed Outliers, Removed Skewness and used Oversampling method using SMOTE for imbalanced dataset and at last train our data by splitting our data through train-test split process.

Built our model using multiple models and finally selected best model (Random Forest Classifier) which was giving best accuracy. And at last compared our predicted and Actual Micro-Credit Defaulter. Thus, our project is completed.

2. Learning Outcomes of the Study in respect of Data Science

- This project has demonstrated the importance of sampling effectively, modelling and predicting data with an imbalanced dataset.
- Through different powerful tools of visualization, we were able to analyse and interpret different hidden insights about the data.
- Through data cleaning we were able to remove unnecessary columns and outliers from our dataset due to which our model would have suffered from overfitting or underfitting.

The few challenges while working on this project were: -

- Improper scaling: scaled it to a single scale using Standard Scaler
- Too many features: 37 features were present in the dataset, after removing multicollinearity we were able to reduce our 9 features. Some were removed due to no relation with target variable.
- Hidden features: Extracted day and month from “pdate” column
- Imbalanced data: Handled through SMOTE Oversampling Method

- Skewed data due to outliers: Removed using power transformer 'yeo-johnson' method and outliers was removed through zscore.

3. Limitations of this work and Scope for Future Work

While we couldn’t reach out goal of 100% accuracy in detecting defaulter but created a system that made data get very close to that goal. This project allows multiple algorithms to be integrated together to combine modules and their results to increase the accuracy of the final result. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others which will make modules easy to add as done in the code.