FLIP ROBO

“A project report on Micro credit – Loan Defaulter Model”

SUBMITTED BY

HIMAJA IJJADA

ACKNOWLEDGMENT

I express my sincere gratitude to FlipRobo Technologies for giving me the opportunity to work on the Micro credit loan defaulter project using machine learning algorithms. I would also like to thank FlipRobo Technologies for providing me with the requisite datasets to work with. And I would like to express my gratitude to Mr. Mohd Kashif (SME FlipRobo) and Ms. Sapna Verma (SME FlipRobo) for being of a great help in completion of the project.

Most of the concepts used to predict the Micro-Credit loandefaultersarelearned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/
- https://imbalanced-learn.org/stable/

CONTENTS

INTRODUCTION

Business Problem Framing

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

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

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

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

Conceptual Background of the Domain Problem

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

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

The sample data is provided to us from our client database. It is hereby given to you for this exercise. In order to improve the selection of customers for the credit, the client wants some

predictions that could help them in further investment and improvement in selection of customers.

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

Review of Literature

An attempt has been made in this report to review the available literature in the area of microfinance. Approaches to microfinance, issues related to measuring social impact versus profitability of MFIs, issue of sustainability, variables impacting sustainability, effect of regulations of profitability and impact assessment of MFIs have been summarized in the below report. We hope that the below report of literature will provide a platform for further research and help the industry to combine theory and practice to take microfinance forward and contribute to alleviating the poor from poverty.

Motivation for the Problem Undertaken

I have to model the micro credit defaulters with the available independent variables. This model will then be used by the management to understand how the customer is considered as defaulter or non-defaulter based on the independent variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand whether the customer will be paying back the loaned amount within 5 days of insurance of loan. The **relationship between predicting defaulter and the economy** is an important motivating factor for predicting micro credit defaulter model.

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modeling of the Problem

In this particular problem I had label as my target column and it was having two classes Label ‘1’ indicates that the loan has been payed i.e. Non- defaulter, while, Label ‘0’ indicates that the loan has not been payed i.e. defaulter. So clearly it is a binary classification problem and I have to use all classification algorithms while building the model. There was no null values in the dataset. Also, I observed some unnecessary entries in some of the columns like in some columns I found more than 90% zero values so I decided to drop those columns. If I keep those columns as it is, it will create high skewness in the model. To get better insight on the features I have used plotting like distribution plot, bar plot and count plot. With these plotting I was able to understand the relation between the features in better manner. Also, I found outliers and skewness in the dataset so I removed outliers using percentile method and I removed skewness using yeo-Johnson method. I have used all the classification algorithms while building model then tuned the best model and saved the best model. At last I have predicted the label using saved model.

Data Sources and their formats

The data was collected for my internship company – FlipRobo technologies in excel format. 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.

```
#importing dataset
df=pd.read_csv("Micro_Credit_Data.csv") #Reading csv file
df.head()
```

	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
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7

Dataset Description

Also, my dataset was having 209593 rows and 36 columns including target. In this particular da tasets I have object, float and integer types of data. The information of features is as follows.

Features Information:

1. label : Flag indicating whether the user paid back the credit amount within 5 days of issuing t he loan{1:success, 0:failure}
2. msisdn : mobile number of user
3. aon : age on cellular network in days
4. daily_decr30 : Daily amount spent from main account, averaged over last 30 days (in Indone sian Rupiah)
5. daily_decr90 : Daily amount spent from main account, averaged over last 90 days (in Indone sian Rupiah)
6. rental30 : Average main account balance over last 30 days
7. rental90 : Average main account balance over last 90 days
8. last_rech_date_ma : Number of days till last recharge of main account
9. last_rech_date_da: Number of days till last recharge of data account
10. last_rech_amt_ma : Amount of last recharge of main account (in Indonesian Rupiah)
11. cnt_ma_rech30 : Number of times main account got recharged in last 30 days
12. fr_ma_rech30 : Frequency of main account recharged in last 30 days
13. sumamnt_ma_rech30 : Total amount of recharge in main account over last 30 days (in Indo nesian Rupiah)
14. medianamnt_ma_rech30 : Median of amount of recharges done in main account over last 3 0 days at user level (in Indonesian Rupiah)
15. medianmarechprebal30 : Median of main account balance just before recharge in last 30 d ays at user level (in Indonesian Rupiah)
16. cnt_ma_rech90 : Number of times main account got recharged in last 90 days
17. fr_ma_rech90 : Frequency of main account recharged in last 90 days
18. sumamnt_ma_rech90 : Total amount of recharge in main account over last 90 days (in Indo nesian Rupiah)
19. medianamnt_ma_rech90 : Median of amount of recharges done in main account over last 9 0 days at user level (in Indonesian Rupiah)
20. medianmarechprebal90 : Median of main account balance just before recharge in last 90 d ays at user level (in Indonesian Rupiah)
21. cnt_da_rech30 : Number of times data account got recharged in last 30 days
22. fr_da_rech30: Frequency of data account recharged in last 30 days
23. cnt_da_rech90 : Number of times data account got recharged in last 90 days
24. fr_da_rech90 : Frequency of data account recharged in last 90 days
25. cnt_loans30 : Number of loans taken by user in last 30 days
26. amnt_loans30: Total amount of loans taken by user in last 30 days

27. maxamnt_loans30 : maximum amount of loan taken by the user in last 30 days
28. medianamnt_loans30 : Median of amounts of loan taken by the user in last 30 days
29. cnt_loans90 : Number of loans taken by user in last 90 days
30. amnt_loans90 : Total amount of loans taken by user in last 90 days
31. maxamnt_loans90 : maximum amount of loan taken by the user in last 90 days
32. medianamnt_loans90 : Median of amounts of loan taken by the user in last 90 days
33. payback30 : Average payback time in days over last 30 days
34. payback90 : Average payback time in days over last 90 days
35. pcircle : telecom circle
36. pdate : date

Data Preprocessing Done

Checking the shape of dataset

```
#Checking the shape of dataset
df.shape
```

```
(209593, 37)
```

The dataset has 209593 rows and 37 columns

Checking the names of all the columns

```
#Checking the names of all the columns
df.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')
```

Above is the list of the column names in the dataset

Checking the data types of the columns

```
#Checking the data types of the columns
df.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
cnt_loans30               int64
amnt_loans30              int64
maxamnt_loans30         float64
medianamnt_loans30      float64
cnt_loans90             float64
amnt_loans90              int64
maxamnt_loans90           int64
medianamnt_loans90      float64
payback30               float64
payback90               float64
pcircle                  object
pdate                    object
dtype: object
```

We can observe that the data has 3 types of data - integer, float and object.

Checking the info of the dataset

```
#Checking the info of the dataset
df.info()
```

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

Observations

- There are no null values in the dataset
- We can see the datatypes of each column
- The column 'pdate' should be a datetime datatype so we need to change it from object to datetime datatype.

Checking unique values of each column

```
#Checking unique values of each column
df.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
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
```

Observations

- 'Unnamed:0' is an index column in raw dataset which has all the entries as unique numbers so we can drop this column.
- The column 'pcircle' has only one entry in the column, which has no contribution with our model training. So we can drop this column also.
- The column 'msisdn' is a column with phone numbers of users, which has no contribution to the model building. So this column can be dropped.

Dropping Unnamed: 0, msisdn and pcircle columns

```
#Droping Unnamed: 0, msisdn and pcircle columns
df = df.drop(["Unnamed: 0"],axis=1)
df = df.drop(["pcircle"],axis=1)
df = df.drop(["msisdn"],axis=1)
```

We have dropped the above columns from our dataset

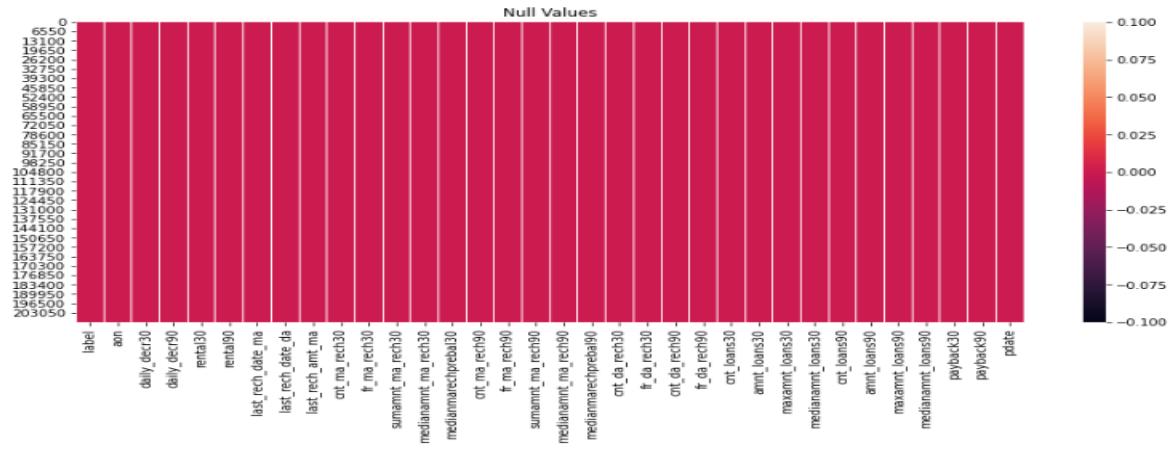
Checking null values in the dataset

```
#Checking null values in the dataset
df.isnull().sum()
```

```
label                    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
pdate                    0
dtype: int64
```

Visualizing null values

```
#Visualizing null values
plt.figure(figsize=[15,6])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



Observations

There are no null values in our dataset.

Feature Engineering:

Converting object data type to datetime

```
#Converting object data type to datetime
df['pdate'] =  pd.to_datetime(df['pdate'])
```

We have converted the pdate column from object to datetime datatype

Extracting paid year,month and day from pdate

```
#Extracting paid year,month and day from pdate

#Extracting year
df["pyear"]=pd.to_datetime(df.pdate, format="%d/%m/%Y").dt.year

#Extracting month
df["pmonth"]=pd.to_datetime(df.pdate, format="%d/%m/%Y").dt.month

#Extracting day
df["pday"]=pd.to_datetime(df.pdate, format="%d/%m/%Y").dt.day
```

We have extracted the date month and year from the pdate column

Dropping pdate column after extraction

```
#Droping pdate column after extraction
df = df.drop(["pdate"],axis=1)
```

We have dropped the pdate column as it is redundant in the dataset

Checking the value counts of pyear column

```
#Checking the value counts of pyear column
df.pyear.value_counts()
```

```
2016    209593
Name: pyear, dtype: int64
```

Observations

The column pyear has only one entry in all the rows. So this can be dropped as it has no insights for model training

Dropping pyear column

```
#Droping pyear column
df = df.drop(["pyear"],axis=1)
```

We have dropped the 'pyear' column from the dataset

Viewing the columns in the dataset

```
#Viewing the columns in the dataset
df.columns
```

```
Index(['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',
       'pmonth', 'pday'],
      dtype='object')
```

Dropping the people who haven't taken any loan

```
#droping the people who haven't taken any loan
df.drop(df[df['amnt_loans90']==0].index, inplace = True)
```

Observations

Dropping people who haven't taken any loans as we don't have any insights from this column for model training.

Checking the value counts of last_rech_date_da column

```
#Checking the value counts of last_rech_date_da column
df.last_rech_date_da.value_counts()
```

```
 0.000000         200900
 7.000000            163
 8.000000            160
 11.000000           148
 13.000000           148
 12.000000           147
 17.000000           123
 10.000000           122
 16.000000           121
 9.000000            120
 14.000000           114
 18.000000           113
 25.000000           107
 1.000000            105
 19.000000           105
 20.000000           104
 24.000000           102
 15.000000           102
 34.000000           100
```

Observations

We have 97% zeros in this column.

Checking the value counts of cnt_da_rech30 column

```
#Checking the value counts of cnt_da_rech30 column
df.cnt_da_rech30.value_counts()
```

```
0.000000        203448
1.000000          2329
2.000000           372
3.000000           156
4.000000            86
5.000000            34
6.000000            27
7.000000            20
9.000000            12
8.000000             9
10.000000            5
11.000000            4
13.000000            3
16.000000            2
12.000000            2
13710.643665         1
25255.945856         1
42113.095246         1
34432.654245         1
```

Observations

We have 98% zeros in this column.

Checking the value counts of fr_da_rech30 column

```
#Checking the value counts of fr_da_rech30 column
df.fr_da_rech30.value_counts()
```

```
0.000000         205985
3.000000             64
1.000000             55
2.000000             51
7.000000             47
4.000000             45
6.000000             38
5.000000             36
8.000000             34
10.000000            19
9.000000             19
11.000000            18
12.000000            16
14.000000            15
13.000000            15
15.000000            13
18.000000            11
16.000000             9
19.000000             8
```

Observations

We have 99% zeros in this column

Checking the value counts of cnt_da_rech90 column

```
#Checking the value counts of cnt_da_rech90 column
df.cnt_da_rech90.value_counts()
```

```
0     202210
1       4149
2        553
3        227
4        150
5         74
6         49
7         36
8         30
9         18
14        10
11         7
12         7
13         6
10         6
16         4
18         3
15         2
17         1
19         1
20         1
22         1
25         1
27         1
28         1
30         1
38         1
Name: cnt_da_rech90, dtype: int64
```

Observations

We have 97% zeros in this column.

Checking the value counts of fr_da_rech90 column

```
#Checking the value counts of fr_da_rech90 column
df.fr_da_rech90.value_counts()
```

```
0     206685
3         78
1         66
2         66
4         56
7         55
6         52
5         45
8         44
10        33
9         32
14        32
11        30
12        27
19        22
28        19
13        17
15        17
18        16
29        15
17        14
16        14
22        11
25        11
21        10
20         9
24         8
27         8
32         8
26         7
30         7
31         6
34         5
37         5
23         5
57         3
35         2
41         2
33         1
39         1
42         1
43         1
46         1
55         1
56         1
64         1
Name: fr_da_rech90, dtype: int64
```

Observations

We have 99% zeros in this column.

Checking the value counts of medianamnt_loans30 column

```
#Checking the value counts of medianamnt_loans30 column
df.medianamnt_loans30.value_counts()
```

```
0.0    193402
1.0      7149
0.5      6538
2.0       420
1.5        38
3.0         3
Name: medianamnt_loans30, dtype: int64
```

Observations

We have 93% zeros in this column.

Checking the value counts of medianamnt_loans90 column

```
#Checking the value counts of medianamnt_loans90 column
df.medianamnt_loans90.value_counts()
```

```
0.0    195381
1.0      6172
0.5      5668
2.0       307
1.5        19
3.0         3
Name: medianamnt_loans90, dtype: int64
```

Observations

We have 94% zeros in this column.

In all the above columns we can observe there are more than 90% of the entries as zeroes which leads to skewness in the dataset. So we have to drop these columns.

Dropping columns with more than 90% zeros

```
#Droping columns with more than 90% zeros
df.drop(columns = ['last_rech_date_da','cnt_da_rech30','fr_da_rech30','cnt_da_rech90','fr_da_rech90','m
```

We have dropped those columns with more than 90% of entries as zeroes to avoid skewness in the dataset

Now let us check the statistical summary of the dataset

Data Inputs- Logic- Output Relationships

Checking description of data set

```
#Checking description of data set
df.describe()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rec
count	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000	2075
mean	0.873948	8095.156177	5352.424286	6044.967417	2674.303807	3451.557712	3744.288249	20
std	0.331908	75605.569198	9208.694592	10902.815812	4272.953526	5714.928132	53813.277038	23
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	
25%	1.000000	246.000000	41.760000	41.979500	278.130000	299.830000	1.000000	7
50%	1.000000	527.000000	1414.400000	1443.355000	1074.880000	1318.480000	3.000000	15
75%	1.000000	982.000000	7200.000000	7723.997500	3330.570000	4163.570000	7.000000	23
max	1.000000	999860.755168	265926.000000	320630.000000	198926.110000	200148.110000	998650.377733	550

Observations

We can observe there are negative values in the following columns

- aon
- daily_decr30
- daily_decr90
- rental30
- rental90
- last_rech_date_ma
- medianmarechprebal30
- medianmarechprebal90

The entries in the above columns cannot be negative as they are age, account balance and number of days. So we shall change them to positive.

Converting all negative values to positive values in above columns

```
#Converting all negative values to positive values in above columns
df['aon']=abs(df['aon'])
df['daily_decr30']=abs(df['daily_decr30'])
df['daily_decr90']=abs(df['daily_decr90'])
df['rental30']=abs(df['rental30'])
df['rental90']=abs(df['rental90'])
df['last_rech_date_ma']=abs(df['last_rech_date_ma'])
df['medianmarechprebal30']=abs(df['medianmarechprebal30'])
df['medianmarechprebal90']=abs(df['medianmarechpreba190'])
```

We have converted the negative values in the entries to positive

Checking if there are other entries in maxamnt_loans30 column except 0,6,12

```
#checking if there are other entries in maxamnt_loans30 column except 0,6,12
df.loc[(df['maxamnt_loans30'] != 6.0) & (df['maxamnt_loans30'] != 12.0) & (df['maxamnt_loans30']!=0.0),
```

```
118       61907.697372
125       22099.413732
146       98745.934048
369       58925.364061
374       78232.464324
499       28574.830964
520       71532.935430
645       83207.495583
887       34044.361979
1091      43221.389235
1142       6041.731394
1201      27793.098567
1930      55723.858041
2030       6721.146132
2520      45683.595975
3073      56383.080492
3378      81942.467056
3751      86416.878765
3754      45073.630216
```

In the problem statement it is specified that we can have only 0,6,12 as maximum amount of loan taken by the user in last 30 days. So converting all the above values to zero

Converting the values into zero which are not 0, 6 and 12

```
#converting the values into zero which are not 0,6 and 12
df.loc[(df['maxamnt_loans30'] != 6.0) & (df['maxamnt_loans30'] != 12.0) & (df['maxamnt_loans30']!=0.0),
```

We have converted those values into zeroes which are not 0, 6 and 12.

Checking the value count of maxamnt_loans30 column again

```
#Now let's check the valuecount of maxamnt_loans30 column agian
df.maxamnt_loans30.value_counts()
```

```
6.0     179193
12.0     26109
0.0       2248
Name: maxamnt_loans30, dtype: int64
```

Observations

Here we have a class imbalance in the above column, i.e., the entries are not evenly distributed. But this is not the target column so we need not fix this

Checking description of data set again

```
#Checking description of data set again
df.describe(include='integer')
```

	label	last_rech_amt_ma	cnt_ma_rech30	cnt_ma_rech90	fr_ma_rech90	sumamnt_ma_rech90	cnt_loans30
count	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000	207550.000000
mean	0.873948	2057.044751	3.991477	6.324110	7.707916	12380.958497	2.786138
std	0.331908	2363.829442	4.264318	7.203957	12.594178	16849.059437	2.552263
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	770.000000	1.000000	2.000000	0.000000	2317.000000	1.000000
50%	1.000000	1539.000000	3.000000	4.000000	2.000000	7218.000000	2.000000
75%	1.000000	2309.000000	5.000000	9.000000	8.000000	16000.000000	4.000000
max	1.000000	55000.000000	203.000000	336.000000	88.000000	953036.000000	50.000000

Assumptions

- The count of all the columns are same which says there are no null values in the dataset
- We need not consider the statistical summary of the target as the entries in it are classes rather than values
- The following columns have Low standard deviation, which means data are clustered around the mean
    - cnt_loans30
    - amnt_loans30
    - maxamnt_loans90
    - pmonth
    - pday
- The other columns have High standard deviation, which indicates data in those columns is more spread out.
- The following columns have mean almost equal to median, which says the distribution of curve is normal
    - maxamnt_loans90
    - pmonth
    - pday
- The other columns have mean greater than the median (50th percentile) , which says the distribution is skewed to right
- The 'pmonth' column has no much difference between the 75% (3rd quantile) and the max values, which shows there are less chance for presence of outliers
- All the other columns have huge difference between the 75% (3rd quantile) and the max values, which shows the chance for presence of outliers

Checking unique values of target column

```
#Checking unique values of target column
df['label'].unique()
```

There are only two unique values in target column. Therefore it is a binary classification problem

Checking for empty observations

```
#Checking for empty observations
df.loc[df['label'] == " "]
```

label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rech30

We can observe that there are no empty observations in the target column.

Hardware and Software Requirements and Tools Used

Hardware Used:

- Processor AMD Ryzen 9 5900HX(8 Cores 16 Logical Processors)
- Physical Memory: 16.0GB (3200MHz)
- GPU: Nvidia RTX 3060 (192 bits), 6GB DDR6 VRAM, 3840 CUDA cores.

Software Used:

- Windows 10 Operating System
- Anaconda Package and Environment Manager: Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. The distribution includes data- science packages suitable for Windows and provides a host of tools and environment for conducting Data Analytical and Scientific works. Anaconda provides all the necessary Python packages and libraries for Machine learning projects.
- Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows data scientists to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources, along with explanatory text in a single document.
- Python3: It is open source, interpreted, high level language and provides great approach for object-oriented programming. It is one of the best languages used for Data Analytics and Data science projects/application. Python provides numerous libraries to deal with mathematics, statistics and scientific function.
- Python Libraries used:
    - **Pandas:** For carrying out Data Analysis, Data Manipulation, and Data Cleaning etc.
    - **Numpy:** For performing a variety of operations on the datasets.
    - **matplotlib.pyplot, Seaborn**: For visualizing Data and various relationships between Feature and Label Columns
    - **Scipy:** For performing operations on the datasets
    - **Statsmodels:** For performing statistical analysis

sklearn for Modelling Machine learning algorithms, Data Encoding, Evaluation metrics, Data Transformation, Data Scaling, Component analysis, Feature selection etc.

MODEL/S DEVELOPMENT AND EVALUATION

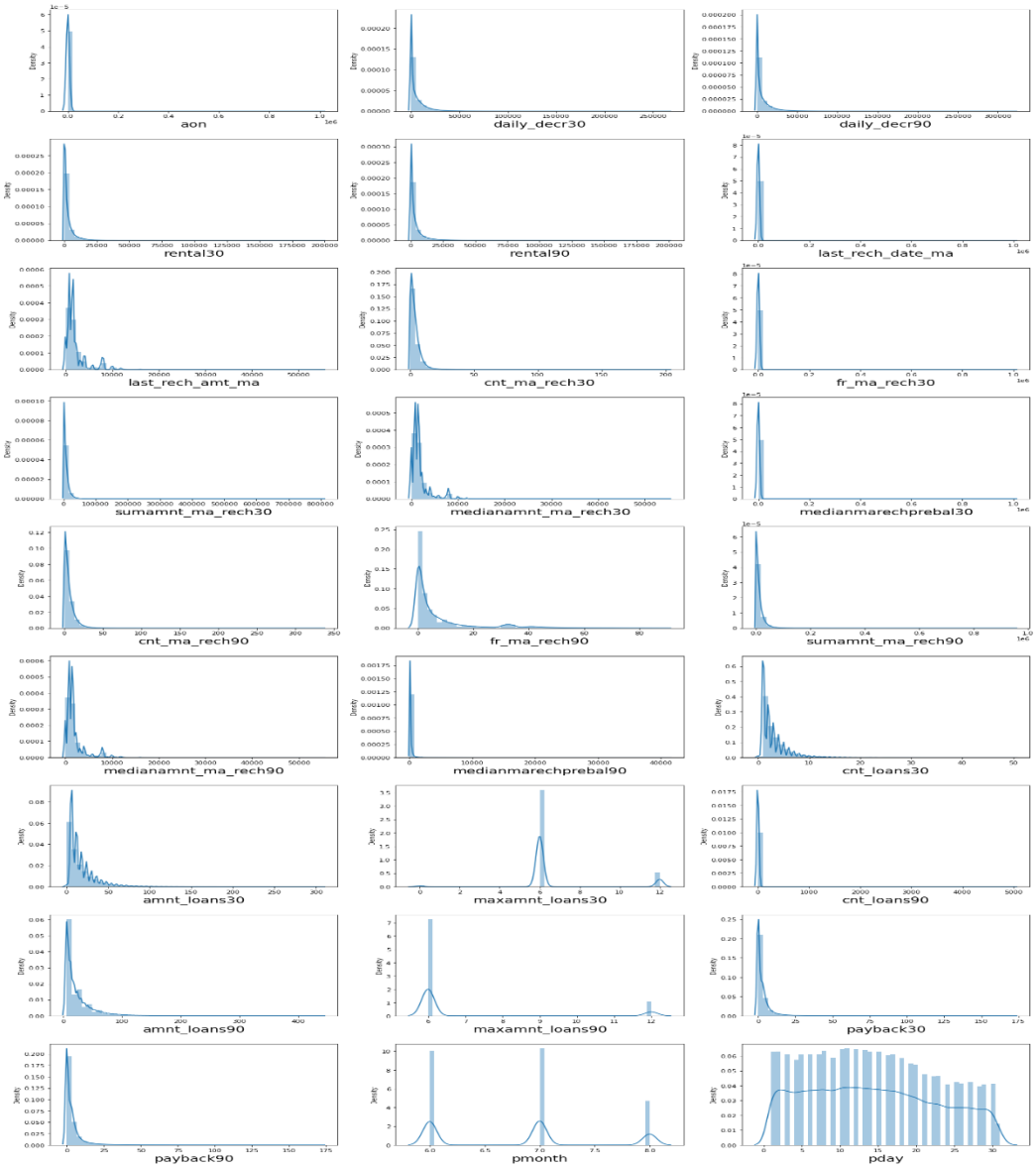
Visualizations

Univariate Analysis:

Let us separate the columns with numerical data fro performing the univariate analysis

```
col = ['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90', 'last_rech_date_ma', 'last_rech_a
```

```
#Distribution plot for all numerical columns except label
plt.figure(figsize = (20,40))
plotnumber = 1
for column in df[col]:
    if plotnumber <=30:
        ax = plt.subplot(10,3,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize = 20)
    plotnumber+=1
plt.tight_layout()
```



Observations

- We can clearly see that there is skewness in most of the columns so we need to treat them.

- The following columns are skewed to the right
    - 'daily_decr30'
    - 'daily_decr90'
    - 'rental30'
    - 'rental90'
    - 'last_rech_amt_ma'
    - 'sumamnt_ma_rech30'
    - 'medianamnt_ma_rech30'
    - 'payback30'
    - 'payback90'
    - 'amnt_loans90'
    - 'medianamnt_ma_rech90'
    - 'fr_ma_rech90'
    - 'cnt_ma_rech30'
    - 'cnt_ma_rech90'
    - 'sumamnt_ma_rech90'

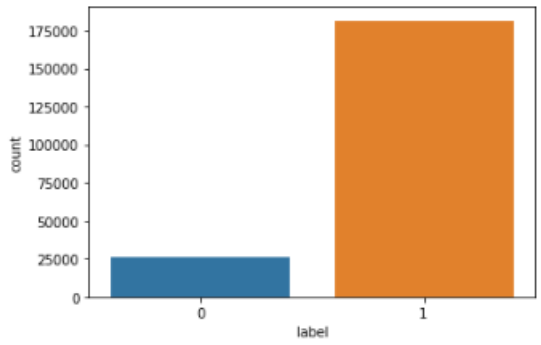
- The following columns are normally distributed
    - 'aon'
    - 'last_rech_date_ma'
    - 'fr_ma_rech30'
    - 'cnt_loans30'
    - 'amnt_loans30'
    - 'medianmarechprebal90'
    - 'cnt_loans90'
    - 'medianmarechprebal30'

- Though some of the columns have integers as entries the data is considered categorical, so we need not take them into consideration.

Count plot for target column

```
#count plot for target column
sns.countplot(df['label'])
```

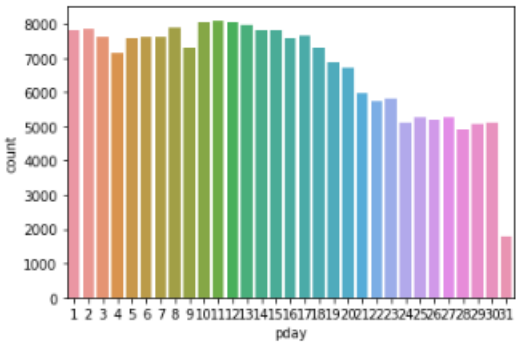
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



Observations: **There is a data imbalance issue so we have to treat this by using oversampling or undersampling techniques.**

```
# Plotting countplot for the pday column
sns.countplot(df['pday'])
```

```
<AxesSubplot:xlabel='pday', ylabel='count'>
```

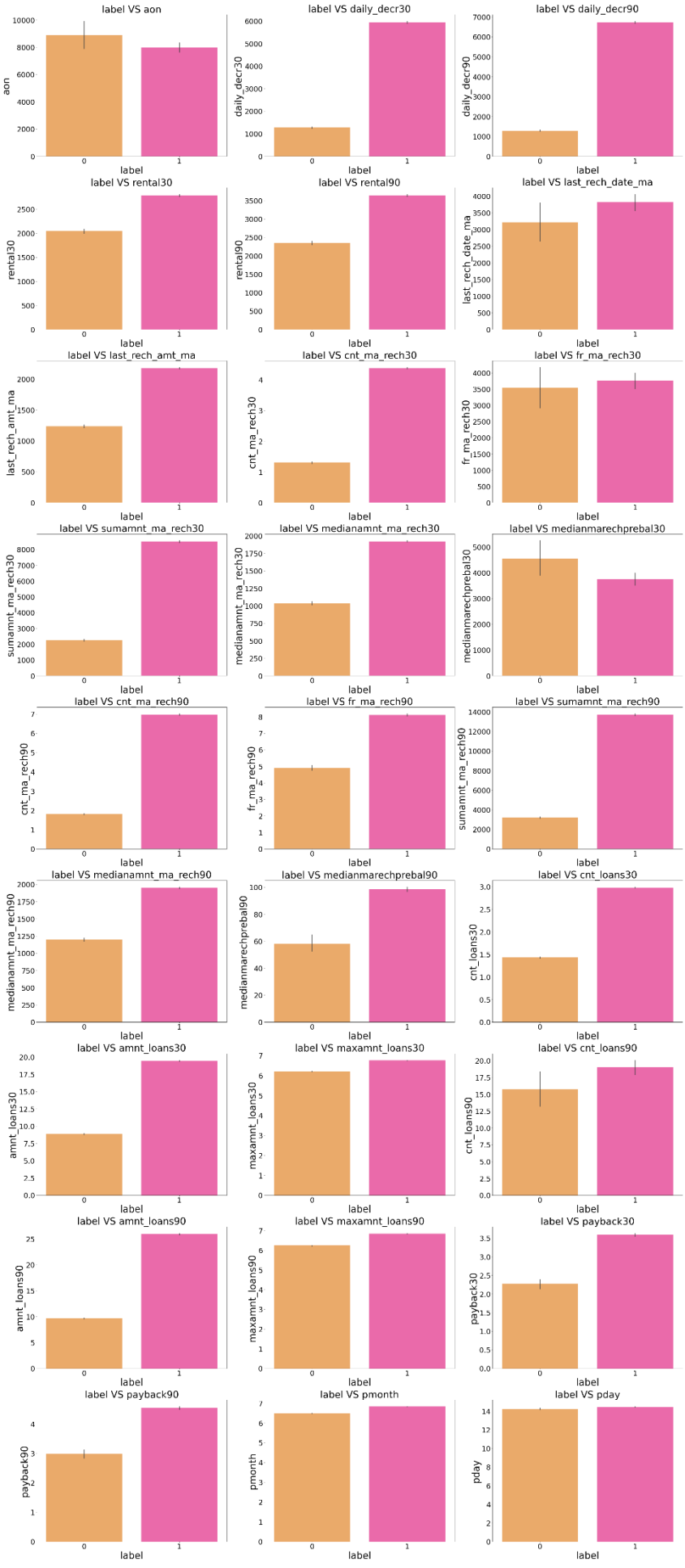


Observations

- All the days till third week of the month have similar no of entries
- The last week of every month has lesser entrie sthan the first 3 weeks.

Bivariate Analysis

```
#barplot for numerical columns
plt.figure(figsize=(40,100))
for i in range(len(col)):
    plt.subplot(10,3,i+1)
    sns.barplot(x=df['label'], y=df[col[i]], palette="spring_r")
    plt.title(f"label VS {col[i]}",fontsize=40)
    plt.xticks(fontsize=30)
    plt.yticks(fontsize=30)
    plt.xlabel('label',fontsize = 40)
    plt.ylabel(col[i],fontsize = 40)
    plt.tight_layout()
```


label VS aon
label VS daily_decr30
label VS daily_decr90
label VS rental30
label VS rental90
label VS last_rech_date_ma
label VS last_rech_amt_ma
label VS cnt_ma_rech30
label VS fr_ma_rech30
label VS sumamnt_ma_rech30
label VS medianamnt_ma_rech30
label VS medianmarechprebal30
label VS cnt_ma_rech90
label VS fr_ma_rech90
label VS sumamnt_ma_rech90
label VS medianamnt_ma_rech90
label VS medianmarechprebal90
label VS cnt_loans30
label VS amnt_loans30
label VS maxamnt_loans30
label VS cnt_loans90
label VS amnt_loans90
label VS maxamnt_loans90
label VS payback30
label VS payback90
label VS pmonth
label VS pday
label

OBSERVATIONS:

1. Customers with high value of Age on cellular network in days (aon) are maximum defaulters (who have not paid there loan amount-0).

2. Customers with high value of Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)(daily_decr30) are maximum Non-defaulters(who have paid there loan amount-1).

3. Customers with high value of Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah) (daily_decr90) are maximum Non-defaulters (who have paid there loan amount-1).

4. Customers with high value of Average main account balance over last 30 days (rental30) are maximum Non-defaulters (who have paid there loan amount-1).

5. Customers with high value of Average main account balance over last 90 days (rental90) are maximum Non-defaulters (who have paid there loan amount-1).

6. Customers with high Number of days till last recharge of main account (last_rech_date_ma) are maximum Non-defaulters (who have paid there loan amount-1).

7. Customers with high value of Amount of last recharge of main account (in Indonesian Rupiah) (last_rech_amt_ma) are maximum Non-defaulters (who have paid there loan amount-1).

8. Customers with high value of Number of times main account got recharged in last 30 days(cnt_ma_rech30) are maximum Non-defaulters(who have paid there loan amount-1).

9. Customers with high value of Frequency of main account recharged in last 30 days (fr_ma_rech30) are maximum Non-defaulters (who have paid there loan amount-1) and also the count is high for defaulters comparatively Non-defaulters are more in number.

10. Customers with high value of Total amount of recharge in main account over last 30 days (in Indonesian Rupiah) (sumamnt_ma_rech30) are maximum Non-defaulters (who have paid there loan amount-1).

11. Customers with high value of Median of amount of recharges done in main account over last 30 days at user level (in Indonesian Rupiah) (medianamnt_ma_rech30) are maximum Non-defaulters (who have paid there loan amount-1).

12. Customers with high value of Median of main account balance just before recharge in last 30 days at user level (in Indonesian Rupiah) (medianmarechprebal30) are maximum defaulters (who have not paid there loan amount-0).

13. Customers with high value of Number of times main account got recharged in last 90 days(cnt_ma_rech90) are maximum Non-defaulters(who have paid there loan amount-1).

14. Customers with high value of Frequency of main account recharged in last 90 days (fr_ma_rech90) are maximum Non-defaulters (who have paid there loan amount-1).

15. Customers with high value of Total amount of recharge in main account over last 90 days (in Indonesian Rupiah) (sumamnt_ma_rech90) are maximum Non-defaulters (who have paid there loan amount-1).

16. Customers with high value of Median of amount of recharges done in main account over last 90 days at user level (in Indonesian Rupiah) (medianamnt_ma_rech90) are maximum Non-defaulters (who have paid there loan amount-1).

17. Customers with high value of Median of main account balance just before recharge in last 90 days at user level (in Indonesian Rupiah) (medianmarechprebal90) are maximum Non-defaulters (who have paid there loan amount-1).

18. Customers with high value of Number of loans taken by user in last 30 days (cnt_loans30) are maximum Non-defaulters (who have paid there loan amount-1).

19. Customers with high value of Total amount of loans taken by user in last 30 days (amnt_loans30) are maximum Non-defaulters (who have paid there loan amount-1).

20. Customers with high value of maximum amount of loan taken by the user in last 30 days (maxamnt_loans30) are maximum Non-defaulters (who have paid there loan amount-1).

21. Customers with high value of Number of loans taken by user in last 90 days (cnt_loans90) are maximum Non-defaulters (who have paid there loan amount-1).

22. Customers with high value of Total amount of loans taken by user in last 90 days (amnt_loans90) are maximum Non-defaulters (who have paid there loan amount-1).

23. Customers with high value of maximum amount of loan taken by the user in last 90 days (maxamnt_loans90) are maximum Non-defaulters (who have paid there loan amount-1).

24. Customers with high value of Average payback time in days over last 30 days (payback30) are maximum Non-defaulters (who have paid there loan amount-1).

25. Customers with high value of Average payback time in days over last 90 days (payback90) are maximum Non-defaulters (who have paid there loan amount-1).

26. In between 6th and 7th month maximum customers both defaulters and Non-defaulters have paid there loan amount.

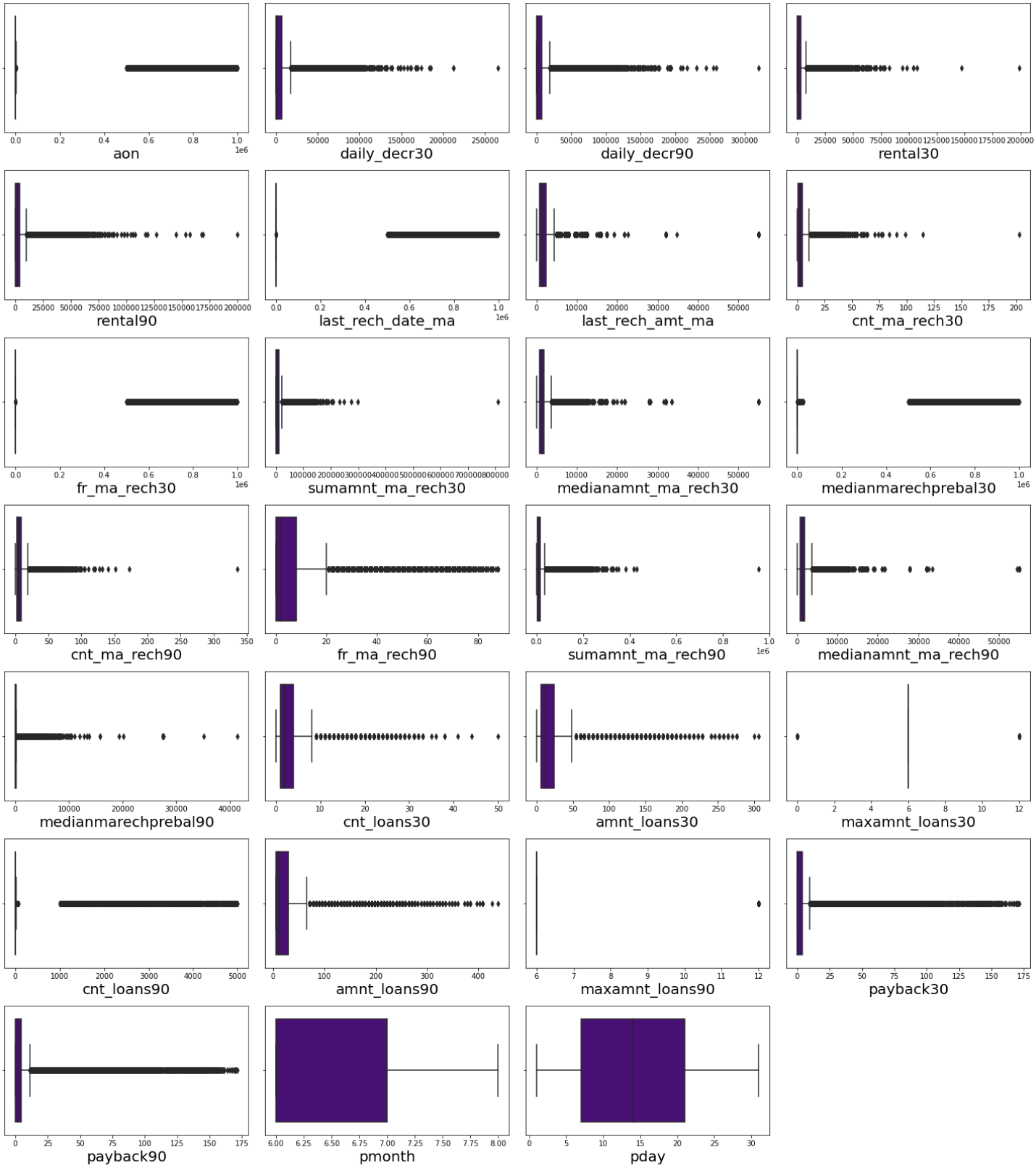
27. Below 14th of each month all the customers have paid there loan amount.

Identification of possible problem-solving approaches (methods)

Checking for outliers:

Identifying the outliers using boxplot

```
# Identifying the outliers using boxplot

plt.figure(figsize=(20,25),facecolor='white')
plotnumber=1
for column in col:
    if plotnumber<=30:
        ax=plt.subplot(8,4,plotnumber)
        sns.boxplot(df[column],color='indigo')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Removing Outliers

i) Zscore method

```
#Features having outliers
features=df[['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90', 'last_rech_date_ma', 'last_
```

Above are the list of columns with outliers in the dataset.

Removing the outliers using zscore method and viewing the dataset

```
from scipy.stats import zscore
z=np.abs(zscore(features))
df_new=df[(z<3).all(axis=1)]
df_new.head()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rec
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	1539	2	
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	5787	1	
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	1539	1	
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	947	0	
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	2309	7	

Checking shape of new dataset

```
#Checking shape of new dataset
df_new.shape
```

```
(170071, 28)
```

The new dataset has 170071 rows and 28 columns.

Checking shape of old dataset

```
#Checking shape of old dataset
df.shape
```

```
(207550, 28)
```

In Z-score method the data loss is more than 10% so let us have a look into IQR method to remove outliers.

ii) IQR method:

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

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

# IQR
IQR=Q3 - Q1

df_1=df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

We have removed the skewness of the dataset using IQR method.

Checking shape of new dataset

```
#Checking shape of new dataset
df_1.shape
```

```
(78654, 28)
```

In the new dataset we have 78654 rows and 28 columns.

Checking shape of old dataset

```
#Checking shape of old dataset
df.shape
```

```
(207550, 28)
```

The dataset previously had 207550 rows and 28 columns.

Checking data loss in IQR method of the dataset

```
#Checking dataloss in IQR method of the dataset
Dataloss = (((207550-78654)/207550)*100)
Dataloss
```

```
62.103589496506864
```

In IQR method the data loss is more than 50%, which is not acceptable. So let us have a look into percentile method to remove outliers.

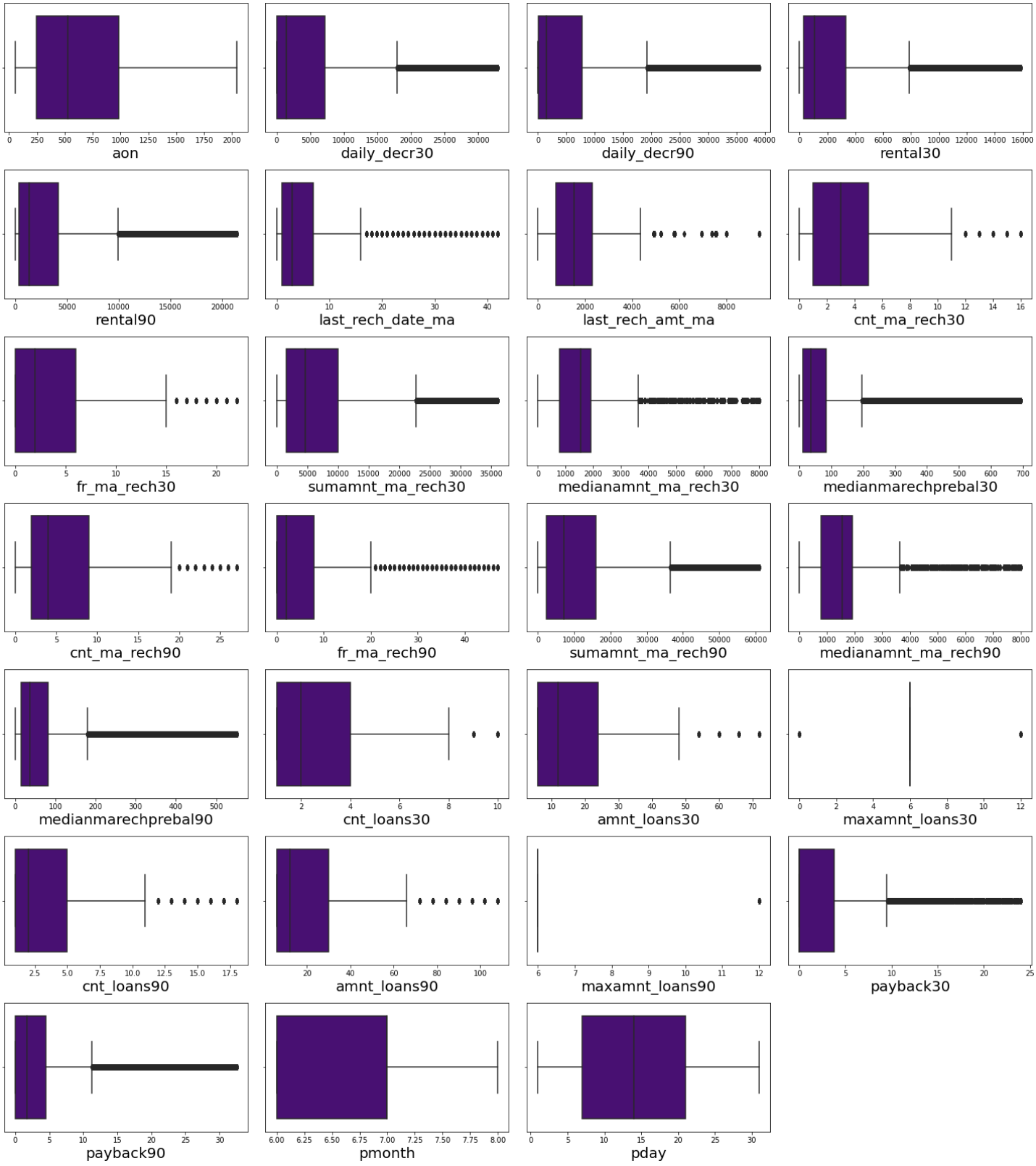
iii) Percentile Method:

```
#Removing outliers using percentile method
for colu in features:
    if df[colu].dtypes != 'object':
        percentile = df[colu].quantile([0.01,0.98]).values
        df[colu][df[colu]<=percentile[0]]=percentile[0]
        df[colu][df[colu]>=percentile[1]]=percentile[1]
```

We have successfully removed outliers in the dataset using percentile method.

Checking how the outliers are impacted

```
# Checking how the outliers are impacted
plt.figure(figsize=(20,25),facecolor='white')
plotnumber=1
for column in col:
    if plotnumber<=30:
        ax=plt.subplot(8,4,plotnumber)
        sns.boxplot(df[column],color='indigo')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Observations

We can observe that the Outliers has been significantly reduced in all the columns.

Checking for skewness:

```
#Checking for skewness in the dataset
df.skew()
```

```
label                 -2.253346
aon                    0.934791
daily_decr30           1.978547
daily_decr90           2.098290
rental30               2.117210
rental90               2.205817
last_rech_date_ma      2.565623
last_rech_amt_ma       2.016661
cnt_ma_rech30          1.410702
fr_ma_rech30           1.703431
sumamnt_ma_rech30      1.749207
medianamnt_ma_rech30   2.122065
medianmarechprebal30   2.799234
cnt_ma_rech90          1.566573
fr_ma_rech90           1.987801
sumamnt_ma_rech90      1.863681
medianamnt_ma_rech90   2.143777
medianmarechprebal90   2.631175
cnt_loans30            1.597669
amnt_loans30           1.752260
maxamnt_loans30        1.634976
cnt_loans90            2.000454
amnt_loans90           1.910837
maxamnt_loans90        2.224471
payback30              2.635055
payback90              2.826565
pmonth                 0.358219
pday                   0.184762
dtype: float64
```

Observations

There is skewness in almost all columns except pmonth,pday and as label is my target i should not remove skewness from this column.

Removing skewness

```
#Creating a list of skewed features
fea=['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90', 'last_rech_date_ma', 'last_rech_amt
```

Creating a list as fea with all the columns having skewness.

Removing skewness using yeo-johnson method:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box_cox' or 'yeo-johnson'
'''
```

```
"\nparameters:\nmethod = 'box_cox' or 'yeo-johnson'\n"
```

Using yeo_johnson method for removing the skewness

```
df[fea] = scaler.fit_transform(df[fea].values)
```

The skewness has been removed from the dataset.

```
#Checking skewness again
df[fea].skew()
```

```
aon                   -0.059261
daily_decr30          -0.137650
daily_decr90          -0.127335
rental30              -0.062522
rental90              -0.062946
last_rech_date_ma      0.043916
last_rech_amt_ma      -0.106643
cnt_ma_rech30         -0.010536
fr_ma_rech30           0.131926
sumamnt_ma_rech30     -0.369147
medianamnt_ma_rech30  -0.237104
medianmarechprebal30  -0.046085
cnt_ma_rech90         -0.012334
fr_ma_rech90           0.141522
sumamnt_ma_rech90     -0.266852
medianamnt_ma_rech90  -0.101431
medianmarechprebal90  -0.029793
cnt_loans30            0.219771
amnt_loans30           0.146844
maxamnt_loans30        0.381337
cnt_loans90            0.201541
amnt_loans90           0.129122
maxamnt_loans90        0.000000
payback30              0.278283
payback90              0.193524
dtype: float64
```

Skewness in all the columns has been reduced.

Correlation

```
cor=df.corr()
cor
```

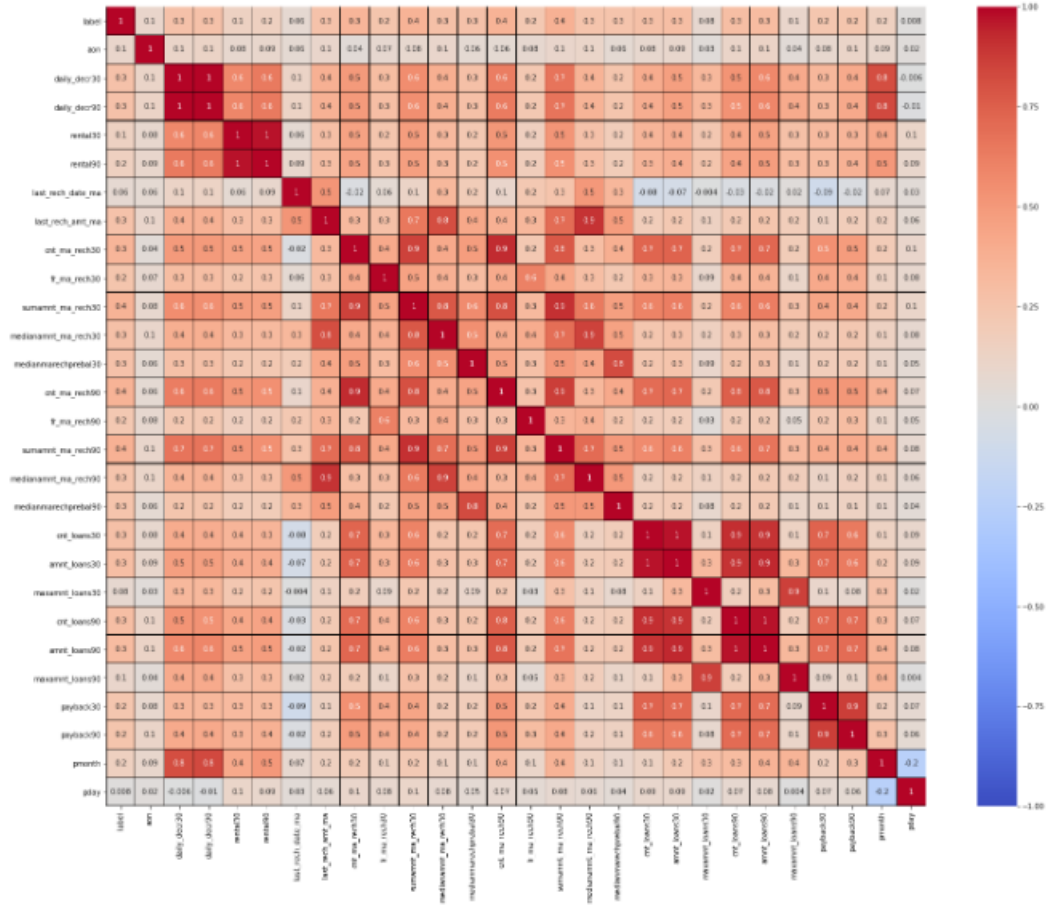
	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_amt_n
label	1.000000	0.097208	0.266444	0.268775	0.142205	0.155661	0.064305	0.2620
aon	0.097208	1.000000	0.117096	0.117840	0.084111	0.092045	0.062278	0.0976
daily_decr30	0.266444	0.117096	1.000000	0.998435	0.585638	0.641529	0.123072	0.3750
daily_decr90	0.268775	0.117840	0.998435	1.000000	0.586036	0.646414	0.126662	0.3745
rental30	0.142205	0.084111	0.585638	0.586036	1.000000	0.974866	0.055640	0.2849
rental90	0.155661	0.092045	0.641529	0.646414	0.974866	1.000000	0.086931	0.2947
last_rech_date_ma	0.064305	0.062278	0.123072	0.126662	0.055640	0.086931	1.000000	0.5181
last_rech_amt_ma	0.262083	0.097625	0.375006	0.374533	0.284950	0.294793	0.518141	1.0000
cnt_ma_rech30	0.347162	0.038139	0.498882	0.494780	0.457289	0.450131	-0.016311	0.3247
fr_ma_rech30	0.243564	0.070579	0.297504	0.296841	0.248717	0.251783	0.062349	0.2747
sumamnt_ma_rech30	0.361711	0.081316	0.556595	0.551778	0.483215	0.473789	0.120058	0.6561
medianamnt_ma_rech30	0.279565	0.097596	0.388856	0.386170	0.321685	0.314687	0.330315	0.8151
medianmarechprebal30	0.266752	0.057144	0.274683	0.273615	0.224043	0.221703	0.193080	0.4289
cnt_ma_rech90	0.363878	0.059595	0.629258	0.632905	0.513308	0.541062	0.097029	0.3750
fr_ma_rech90	0.221068	0.075794	0.219986	0.221059	0.162761	0.176180	0.220031	0.3377
sumamnt_ma_rech90	0.370102	0.098734	0.664160	0.667208	0.520826	0.546615	0.250383	0.7020
medianamnt_ma_rech90	0.252702	0.102863	0.364479	0.364683	0.273283	0.282589	0.507557	0.9022
medianmarechprebal90	0.257523	0.057758	0.243450	0.244228	0.180335	0.189007	0.338033	0.4908
cnt_loans30	0.276082	0.081993	0.398639	0.394111	0.351445	0.344863	-0.076541	0.1846
amnt_loans30	0.291492	0.087453	0.469091	0.465628	0.391547	0.391728	-0.067983	0.2210
maxamnt_loans30	0.075952	0.028230	0.329454	0.329547	0.229190	0.243832	-0.004410	0.1476
cnt_loans90	0.293548	0.116259	0.530528	0.532191	0.426924	0.447225	-0.027212	0.2169
amnt_loans90	0.309039	0.118862	0.585352	0.587372	0.457818	0.482003	-0.022283	0.2464
maxamnt_loans90	0.101247	0.038690	0.406822	0.409956	0.279351	0.306454	0.016073	0.1856
payback30	0.236554	0.080812	0.336835	0.334597	0.313625	0.311970	-0.094731	0.1471
payback90	0.244596	0.111218	0.413587	0.414701	0.349548	0.368359	-0.020555	0.1843
pmonth	0.151680	0.088821	0.819261	0.832069	0.420241	0.505436	0.070518	0.1522
pday	0.008241	0.016277	-0.005543	-0.012356	0.104843	0.088389	0.032959	0.0601

Observations

Above are the correlations of all of the features. To get better visualization on the correlation of features,let us plot it using a heat map.

Visualizing the correlation matrix by plotting heat map.

```
# Visualizing the correlation matrix by plotting heat map.
plt.figure(figsize=(25,20))
sns.heatmap(df.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g', annot = True, linecolor="black",annot_k
plt.yticks(rotation=0);
```

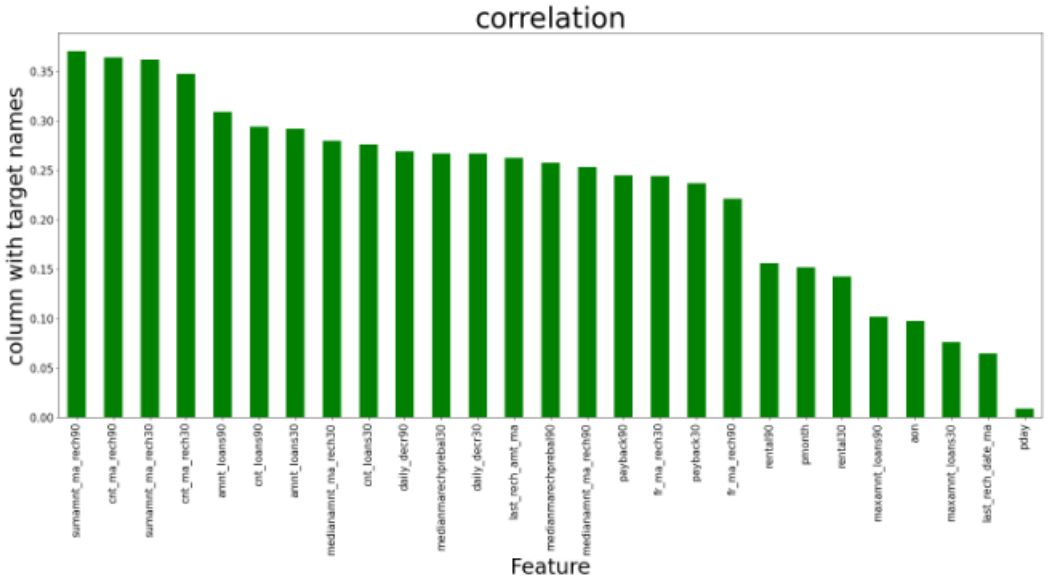


Observations

- We can observe that there are some fetures which are positively correlated with other features which says there is multicollinearity in the dataset. We need to terat the multicollinearity before building a model.

Let's visualize the correlation of all the features with target to get better insights.

```
plt.figure(figsize=(25,10))
df.corr()['label'].sort_values(ascending=False).drop(['label']).plot(kind='bar',color='g')
plt.xlabel('Feature',fontsize=30)
plt.ylabel('column with target names',fontsize=30)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('correlation',fontsize=40)
plt.show()
```



Observations

- All the features are positively correlated with the target. We don't have any negatively correlated features with the target.
- Even though pday feature is least correlated with the target, we can keep it and proceed.
- 'sumnamt_ma_rech90', 'cnt_ma_rech90', 'sumnamt_ma_rech30' are most correlated with the target.

Separating Features and Target:

```
#Seperating the target and the features
x = df.drop("label",axis=1)
y = df["label"]
```

We have separated the target and independent columns.

Scaling

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
```

We have scaled the data using MinMax scaler

Viewing the data after scaling

```
X.head()
```

	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rech3
0	0.348970	0.670388	0.661447	0.330465	0.335513	0.378383	0.498865	0.368297	0.99087
1	0.625204	0.851665	0.834288	0.706042	0.668524	0.865622	0.831756	0.229617	0.00000
2	0.537141	0.578404	0.572448	0.491889	0.470866	0.463708	0.498865	0.229617	0.00000
3	0.318099	0.196647	0.198199	0.299475	0.290305	0.995905	0.411921	0.000000	0.00000
4	0.718606	0.353656	0.353543	0.518649	0.495734	0.526388	0.584318	0.717633	0.44816

This is the data of independent variables after scaling.

Balancing

Checking the value count of target column

```
#Checking the value count of target column
y.value_counts()
```

```
1    181388
0     26162
Name: label, dtype: int64
```

Balancing the target variable using oversampling

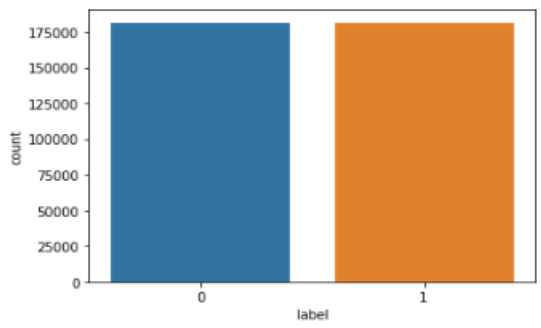
```
# Balancing the target variable using oversampling

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

```
# Checking the value counts again
y.value_counts()
```

```
0    181388
1    181388
Name: label, dtype: int64
```

```
# Visualizing the target data after oversampling
sns.countplot(y)
```

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



Now the data looks balanced.

Finding Best Random State and Accuracy:

```
#importing necessary libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

```
#finding the random state and accuracy

from sklearn.tree import DecisionTreeClassifier
maxAccu=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30, random_state =i)
    mod = DecisionTreeClassifier()
    mod.fit(X_train, y_train)
    pred = mod.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  0.9149338895371808  on Random_state  94
```

The above is the best accuracy and random state.

Creating the train and test split

```
#Creating the train and test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.30,random_state=maxRS)
```

We have created the train and test split.

Classification Algorithms:

```
#importing necessary libraries.
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.naive_bayes import GaussianNB as NB
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score, accuracy
from sklearn.model_selection import cross_val_score
```

We have imported all the necessary libraries for the classification algorithms to be evaluated

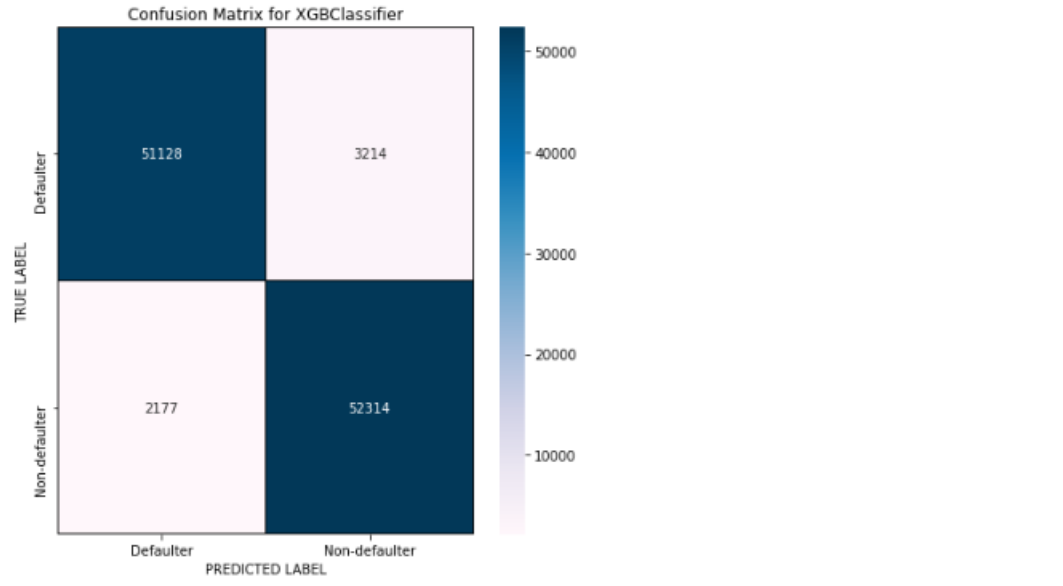
i) XGB Classifier:

```
XGB=XGBClassifier(verbosity=0)
XGB.fit(X_train,y_train)
predxg=XGB.predict(X_test)
Accuracy_Score = accuracy_score(y_test, predxg)*100
print('Accuracy Score:',Accuracy_Score)
print('Confusion Matrix:',confusion_matrix(y_test, predxg))
print(classification_report(y_test,predxg))

#cross validation score
scores = cross_val_score(XGB, X, y, cv = 5).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = Accuracy_Score - scores
print("\Accuracy_Score - Cross Validation Score :", diff)
```

```
Accuracy Score: 95.04653919307563
Confusion Matrix: [[51128  3214]
 [ 2177 52314]]
              precision    recall  f1-score   support

           0       0.96      0.94      0.95     54342
           1       0.94      0.96      0.95     54491

    accuracy                           0.95    108833
   macro avg       0.95      0.95      0.95    108833
weighted avg       0.95      0.95      0.95    108833


Cross validation score : 93.63413021526213
\Accuracy_Score - Cross Validation Score : 1.4124089778134987
```

XGBClassifier is giving 95% accuracy.

```
cm = confusion_matrix(y_test, predxg)

x_axis_labels = ["Defaulter","Non-defaulter"]
y_axis_labels = ["Defaulter","Non-defaulter"]

f, ax = plt.subplots(figsize =(7,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="PuBu", xtick
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for XGBClassifier')
plt.show()
```



We can see the true values and predicted values in XGB Classifier model using confusion matrix.

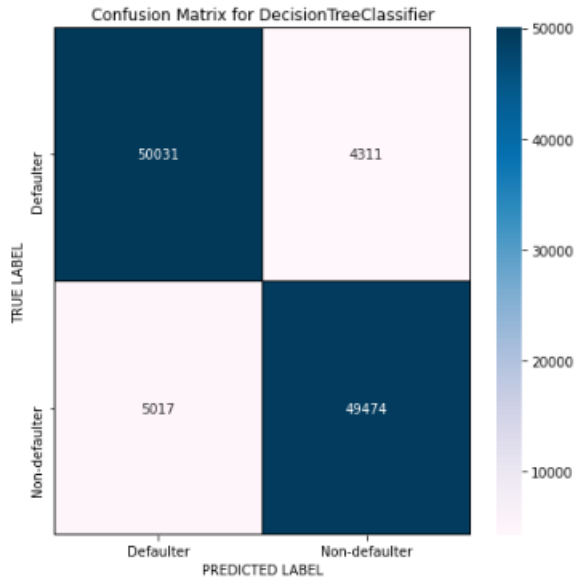
ii) DecisionTreeClassifier:

```
DTC=DecisionTreeClassifier()
DTC.fit(X_train,y_train)
preddt=DTC.predict(X_test)
Accuracy_Score = accuracy_score(y_test, preddt)*100
print('Accuracy Score:',Accuracy_Score)
print('Confusion Matrix:',confusion_matrix(y_test, preddt))
print(classification_report(y_test,preddt))

#cross validation score
scores = cross_val_score(DTC, X, y, cv = 5).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = Accuracy_Score - scores
print("\Accuracy_Score - Cross Validation Score :", diff)
```

```
Accuracy Score: 91.42907022686134
Confusion Matrix: [[50031  4311]
 [ 5017 49474]]
              precision    recall  f1-score   support

           0       0.91      0.92      0.91     54342
           1       0.92      0.91      0.91     54491

    accuracy                           0.91    108833
   macro avg       0.91      0.91      0.91    108833
weighted avg       0.91      0.91      0.91    108833


Cross validation score : 90.96193596149101
\Accuracy_Score - Cross Validation Score : 0.4671342653703334
```

DecisionTreeClassifier is giving 91.42% accuracy.

```
cm = confusion_matrix(y_test, preddt)

x_axis_labels = ["Defaulter","Non-defaulter"]
y_axis_labels = ["Defaulter","Non-defaulter"]

f, ax = plt.subplots(figsize =(7,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="PuBu", xtick
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for DecisionTreeClassifier')
plt.show()
```



We can see the true values and predicted values in DecisionTreeClassifier model using confusion matrix.

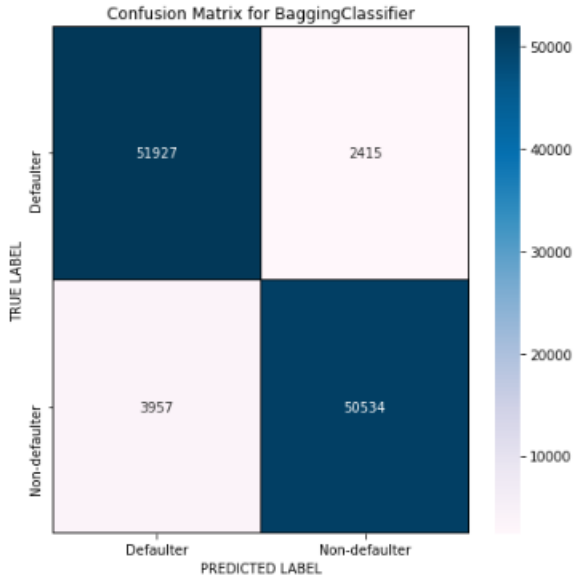
iii) BaggingClassifier:

```
BC=BaggingClassifier()
BC.fit(X_train,y_train)
predbc=BC.predict(X_test)
Accuracy_Score = accuracy_score(y_test, predbc)*100
print('Accuracy Score:',Accuracy_Score)
print('Confusion Matrix:',confusion_matrix(y_test, predbc))
print(classification_report(y_test,predbc))

#cross validation score
scores = cross_val_score(BC, X, y, cv = 5).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = Accuracy_Score - scores
print("\Accuracy_Score - Cross Validation Score :", diff)
```

```
Accuracy Score: 94.14515817812612
Confusion Matrix: [[51927  2415]
 [ 3957 50534]]
              precision    recall  f1-score   support

           0       0.93      0.96      0.94     54342
           1       0.95      0.93      0.94     54491

    accuracy                           0.94    108833
   macro avg       0.94      0.94      0.94    108833
weighted avg       0.94      0.94      0.94    108833


Cross validation score : 93.72093744692671
\Accuracy_Score - Cross Validation Score : 0.42422073119941217
```

BaggingClassifier is giving 94.14% accuracy.

```
cm = confusion_matrix(y_test, predbc)

x_axis_labels = ["Defaulter","Non-defaulter"]
y_axis_labels = ["Defaulter","Non-defaulter"]

f, ax = plt.subplots(figsize =(7,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="PuBu", xtick
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for BaggingClassifier')
plt.show()
```



We can see the true values and predicted values in BaggingClassifier model using confusion matrix.

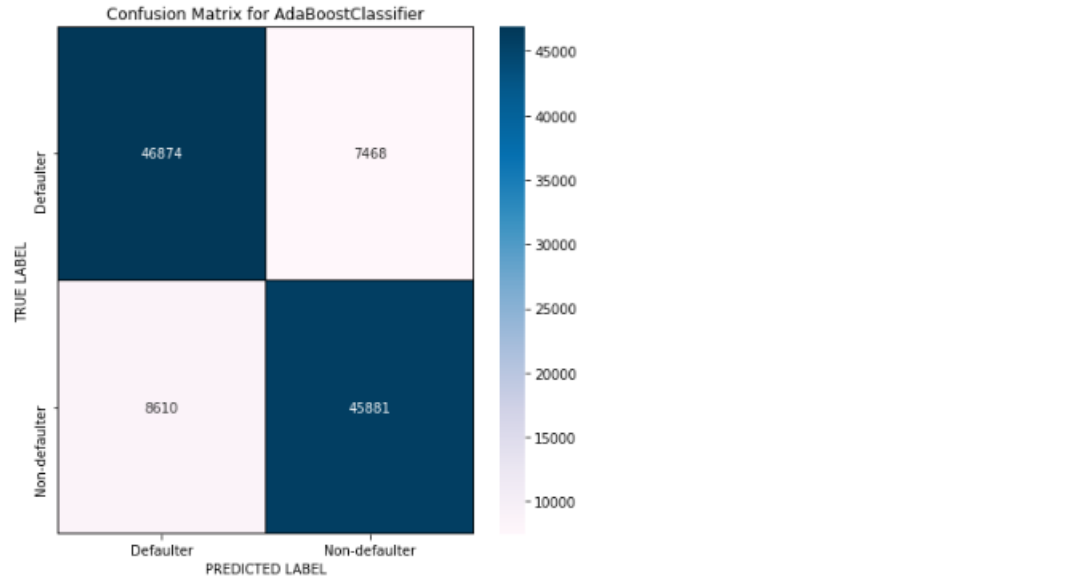
iv) AdaBoostClassifier:

```
ABC=AdaBoostClassifier()
ABC.fit(X_train,y_train)
predab=ABC.predict(X_test)
Accuracy_Score = accuracy_score(y_test, predab)*100
print('Accuracy Score:',Accuracy_Score)
print('Confusion Matrix:',confusion_matrix(y_test, predab))
print(classification_report(y_test,predab))

#cross validation score
scores = cross_val_score(ABC, X, y, cv = 5).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = Accuracy_Score - scores
print("\Accuracy_Score - Cross Validation Score :", diff)
```

```
Accuracy Score: 85.22690727996104
Confusion Matrix: [[46874  7468]
 [ 8610 45881]]
              precision    recall  f1-score   support

           0       0.84      0.86      0.85     54342
           1       0.86      0.84      0.85     54491

    accuracy                           0.85    108833
   macro avg       0.85      0.85      0.85    108833
weighted avg       0.85      0.85      0.85    108833


Cross validation score : 84.9761406253136
\Accuracy_Score - Cross Validation Score : 0.2507666546474354
```

AdaBoost Classifier is giving 85% accuracy.

```
cm = confusion_matrix(y_test, predab)

x_axis_labels = ["Defaulter","Non-defaulter"]
y_axis_labels = ["Defaulter","Non-defaulter"]

f, ax = plt.subplots(figsize =(7,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="PuBu", xtick
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for AdaBoostClassifier')
plt.show()
```

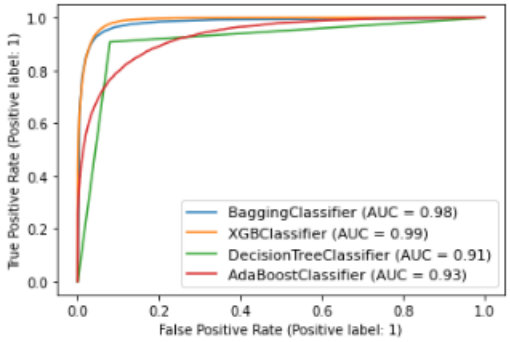


We can see the true values and predicted values in AdaboostClassifier model using confusion matrix.

By looking into the difference of model accuracy and cross validation score i found BaggingClassifier as the best model with 95.16% accuracy and the difference between model accuracy and cross validation score is 0.44.

ROC-AUC curve:

```
# Plotting ROC for all the models used here
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn.metrics import plot_roc_curve
disp = plot_roc_curve(BC,X_test,y_test)
plot_roc_curve(XGB, X_test, y_test, ax=disp.ax_)
plot_roc_curve(DTC, X_test, y_test, ax=disp.ax_)
plot_roc_curve(ABC, X_test, y_test, ax=disp.ax_)

plt.legend(prop={'size':11}, loc='lower right')
plt.show()
```



Above is the ROC curves for all the models that we have predicted. The AUC values can also be seen in the plot.

AUC values for XGBClassifier and BaggingClassifier are higher compared to other models. We have least difference in model accuracy and cross validation score for BaggingClassifier so BaggingClassifier can be considered the best model to proceed with hyper parameter tuning.

Testing of Identified Approaches (Algorithms)

Hyper Parameter tuning:

Importing necessary libraries

```
#importing necessary libraries
from sklearn.model_selection import GridSearchCV
```

```
#Defining the parameters
parameter = {'bootstrap':['True','False'],
             'n_jobs': [-2,-1,1,2],
             'n_estimators':[10,20,30,40],
             'warm_start':['True','False']}
```

We have given the list of parameters for BaggingClassifier model.

Defining grid search CV for bagging classifier

```
# Defining grid search cv for bagging classifier
GCV=GridSearchCV(BaggingClassifier(),parameter,cv=5)
```

Running grid search CV for BaggingClassifier

Training the model

```
# Training the model
GCV.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=BaggingClassifier(),
             param_grid={'bootstrap': ['True', 'False'],
                         'n_estimators': [10, 20, 30, 40],
                         'n_jobs': [-2, -1, 1, 2],
                         'warm_start': ['True', 'False']})
```

We have trained the model with the above defined GCV.

Viewing the list of best parameters

```
#Viewing the list of best parameters
GCV.best_params_
```

```
{'bootstrap': 'True', 'n_estimators': 40, 'n_jobs': -2, 'warm_start': 'False'}
```

We have got the best parameters for BaggingClassifier

Run and Evaluate selected model

```
Final_mod=BaggingClassifier(bootstrap='True', n_jobs=-1,warm_start='True', n_estimators=40)
Final_mod.fit(X_train,y_train)
pred=Final_mod.predict(X_test)
acc=accuracy_score(y_test, pred)

print('Accuracy Score:',(accuracy_score(y_test,pred)*100))
print('Confusion matrix:',confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

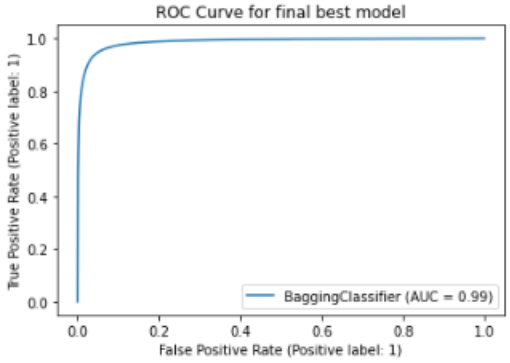
```
Accuracy Score: 94.86369024101145
Confusion matrix: [[51831  2511]
 [ 3079 51412]]
              precision    recall  f1-score   support

           0       0.94      0.95      0.95     54342
           1       0.95      0.94      0.95     54491

    accuracy                           0.95    108833
   macro avg       0.95      0.95      0.95    108833
weighted avg       0.95      0.95      0.95    108833
```

The accuracy of our final model has improved from 94.14% to 94.86%

AUC ROC CURVE for final model

Plotting ROC curve for final best model

```
#Ploting ROC curve for final best model
plot_roc_curve(Final_mod, X_test, y_test)
plt.title('ROC Curve for final best model')
plt.show()
```



After hyper parameter tuning we can notice the improvement in roc curve and AUC too. This is near ideal.

Key Metrics for success in solving problem under consideration

Saving the model

```
#Saving the model as .pkl file
import joblib
joblib.dump(Final_mod,"MicroCreditLoan.pkl")
```

```
['MicroCreditLoan.pkl']
```

We have saved the final model as MicroCredit.pkl

Predictions

Loading the saved model

```
# Loading the saved model
model=joblib.load("MicroCreditLoan.pkl")

#Prediction
prediction = model.predict(X_test)
prediction
```

```
array([0, 1, 1, ..., 1, 0, 0], dtype=int64)
```

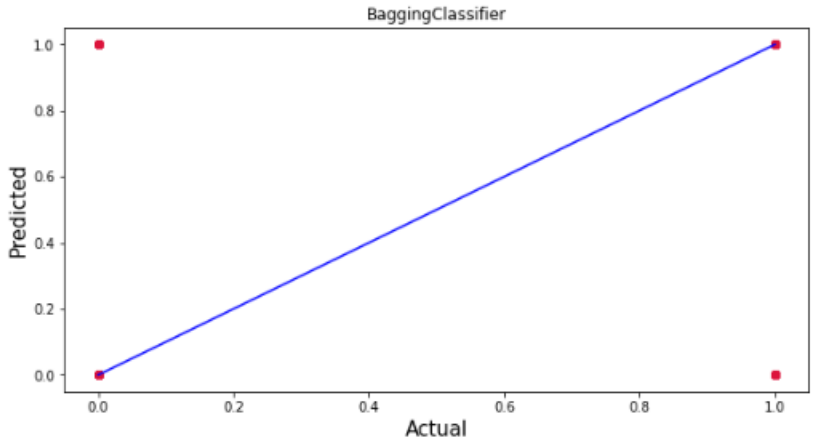
```
#Creating a dataframe for the actual values vs predicted values
pd.DataFrame([model.predict(X_test)[:],y_test[:]],index=["Predicted","Actual"])
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Predicted	0	1	1	0	1	0	1	0	1	0	1	1	1	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1	0	1	1
Actual	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	1

Above are the predicted values vs the actual values. They look similar with minimal exceptions

Plotting Actual vs Predicted values

```
# Plotting Actual vs Predicted values
plt.figure(figsize=(10,5))
plt.scatter(y_test, prediction, c='crimson')
p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
plt.title("BaggingClassifier")
plt.show()
```



Observations

- We have plotted the Actual vs Predicted values to get better insights.
- Here we can observe that the Blue line is the actual values and red dots are the predicted values.
- This says there are minimal to no exceptions, most of the predicted values are in sync with the actual values.

Interpretation of the Results

- The dataset was very challenging to handle it had 37 features with 30days and 90days information of customers.
- Firstly, the datasets were not having any null values.
- But there was huge number of zero entries in maximum columns so we have to be careful while going through the statistical analysis of the datasets.
- And proper plotting for proper type of features will help us to get better insight on the data. I found maximum numerical columns in the dataset so I have chosen bar plot to see the relation between target and features.
- I have noticed that most of the columns are skewed, so we have chosen proper methods to deal with the outliers and skewness. If we ignore the outliers and skewness we may end up with a model which has lesser accuracy at predicting the output.
- Then scaling dataset has a good impact like it will help the model not to get biased. Since we have not removed outliers and skewness completely from the dataset so we have to choose Normalization.
- We have to use multiple models while building model using dataset as to get the best model out of it.
- And we have to use multiple metrics like F1_score, precision, recall and accuracy_score which will help us to decide the best model.
- I found BaggingClassifier as the best model with 94.14% accuracy_score. Also I have improved the accuracy of the best model by running hyper parameter tuning.
- At last I have predicted whether the loan is paid back using saved model. The predictions looked similar to the actual values.

CONCLUSION

Key Findings and Conclusions of the Study

In this project report, we have used machine learning algorithms to predict the micro credit defaulters. We have mentioned the step by step procedure to analyze the dataset and finding the correlation between the features. Thus we can select the features which are correlated to each other and are independent in nature. These feature set were then given as an input to four algorithms and a hyper parameter tuning was done to the best model and the accuracy has been improved. Hence we calculated the performance of each model using different performance metrics and compared them based on these metrics. Then we have also saved the best model and predicted the label. It was good the predicted and actual values were almost same.

Learning Outcomes of the Study in respect of Data Science

I found that the dataset was quite interesting to handle as it contains all types of data in it. Improvement in computing technology has made it possible to examine social information that cannot previously be captured, processed and analyzed. New analytical techniques of machine learning can be used in property research.

The power of visualization has helped us in understanding the data by graphical representation it has made me to understand what data is trying to say. Data cleaning is one of the most important steps to remove unrealistic values and zero values. This study is an exploratory attempt to use four machine learning algorithms in estimating micro credit defaulter, and then compare their results.

To conclude, the application of machine learning in micro credit is still at an early stage. We hope this study has moved a small step ahead in providing some methodological and empirical contributions to crediting institutes, and presenting an alternative approach to the valuation of defaulters. Future direction of research may consider incorporating additional micro credit transaction data from a larger economical background with more features.

Limitations of this work and Scope for Future Work

- First drawback is the shape of the dataset it is such a huge dataset that makes it hard to handle.
- Followed by more number of outliers and skewness these two will reduce our model accuracy.
- Also, we have tried best to deal with outliers, skewness and zero values. So it looks quite good that we have achieved an accuracy of 94.86% even after dealing all these drawbacks.
- Also, this study will not cover all Classification algorithms instead, it is focused on the chosen algorithm, starting from the basic ensembling techniques to the advanced ones.