FLIP ROBO

MICRO CREDIT DEFAULTER PROJECT

Submitted by: **Haindavi Chakravarthi**

Internship: **23**

ACKNOWLEDGEMENT

- The successful completion of any work would be always be incomplete unless we mention the valuable cooperation and assistance of those people who were a source of constant guidance and encouragement, they served as bacon light and crowned our efforts with success. First of all, I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity. I wish to express our sincere thanks to the above people, without whom I would not have been able to complete this project. I thank that I got the chance to do this project because this project has given me a lot many thoughts whether in handling the imbalanced dataset and also focussing on Visualization, it helped me to regain my knowledge levels and also helped to smoothly handle the projects, finally this project has given a good idea of handling the projects.

INTRODUCTION

- A Microfinance Institution (MFI) is an organization that offers financial services to low-income populations. MFS becomes very useful when targeting especially the unbanked poor families living in remote areas with not much sources of income. The Microfinance services (MFS) provided by MFI are Group Loans, Agricultural Loans, Individual Business Loans and so on.
- Many microfinance institutions (MFI), experts and donors are supporting the idea of using mobile financial services (MFS) which they feel are more convenient and efficient, and cost saving, than the traditional high-touch model used since long for the purpose of delivering microfinance services. Though, the MFI industry is primarily focusing on low-income families and 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.
- They understand the importance of communication and how it affects a person’s life, thus, focusing on providing their services and products to low-income families and poor customers that can help them in the need of hour.
- They are collaborating with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be defaulter if he deviates from the path of paying back the loaned amount within the time duration of 5 days. For the loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).
- Here with the historical data of the customers on repaying the loan amount, we will predict the defaulters through machine learning model.

ANALYTICAL PROBLEM FRAMING

- Here our dataset has 209593 rows and 37 columns, using this dataset we will be building the model followed by training the data and then finally the model is tested by using 67% of the training data and 33% of the testing data.
- Here the data has no null values and even then, we face skewness and outliers over here for certain variables which we treat for the better accuracy of our model.
- Also, here we face the problem imbalanced data where the count of defaulters is much less than the non-defaulters which should be treated.
- The following are the features of our dataset with its definition.

label	Flag indicating whether the user paid back the credit amount within 5 days of issuing the loan{1:success, 0:failure}
msisdn	mobile number of user
aon	age on cellular network in days
daily_decr30	Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)
daily_decr90	Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah)
rental30	Average main account balance over last 30 days
rental90	Average main account balance over last 90 days
last_rech_date_ma	Number of days till last recharge of main account
last_rech_date_da	Number of days till last recharge of data account
last_rech_amt_ma	Amount of last recharge of main account (in Indonesian Rupiah)
cnt_ma_rech30	Number of times main account got recharged in last 30 days
fr_ma_rech30	Frequency of main account recharged in last 30 days
sumamnt_ma_rech30	Total amount of recharge in main account over last 30 days (in Indonesian Rupiah)
medianamnt_ma_rech30	Median of amount of recharges done in main account over last 30 days at user level (in Indonesian Rupiah)
medianmarechprebal30	Median of main account balance just before recharge in last 30 days at user level (in Indonesian Rupiah)
cnt_ma_rech90	Number of times main account got recharged in last 90 days
fr_ma_rech90	Frequency of main account recharged in last 90 days
sumamnt_ma_rech90	Total amount of recharge in main account over last 90 days (in Indonesian Rupiah)
medianamnt_ma_rech90	Median of amount of recharges done in main account over last 90 days at user level (in Indonesian Rupiah)
medianmarechprebal90	Median of main account balance just before recharge in last 90 days at user level (in Indonesian Rupiah)
cnt_da_rech30	Number of times data account got recharged in last 30 days
fr_da_rech30	Frequency of data account recharged in last 30 days
cnt_da_rech90	Number of times data account got recharged in last 90 days

fr_da_rech90	Frequency of data account recharged in last 90 days
cnt_loans30	Number of loans taken by user in last 30 days
amnt_loans30	Total amount of loans taken by user in last 30 days
maxamnt_loans30	maximum amount of loan taken by the user in last 30 days
medianamnt_loans30	Median of amounts of loan taken by the user in last 30 days
cnt_loans90	Number of loans taken by user in last 90 days
amnt_loans90	Total amount of loans taken by user in last 90 days
maxamnt_loans90	maximum amount of loan taken by the user in last 90 days
medianamnt_loans90	Median of amounts of loan taken by the user in last 90 days
payback30	Average payback time in days over last 30 days
payback90	Average payback time in days over last 90 days
pcircle	telecom circle
pdate	date

- Here we will start our model building and we undergo through various steps for better model accuracy.

Importing the necessary libraries:

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

Data Collection:

```
data = pd.read_csv("Micro credit project.csv")
```

```
data.head()
```

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	medianami
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	...	6.0	
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	...	12.0	
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	...	6.0	
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	...	6.0	
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	...	6.0	

5 rows × 37 columns

Observation : Here we can see that the data has unnecessary column like "unnamed0" which is not at all necessary and also there may be few features which are irrelevant and so there are pre-processing steps needed to be done for the beter model building.

- Here we have imported our necessary libraries and collected the data and also the data has few unrequired columns which have to be removed during pre-processing of the data.

➢ Now we will have a look at the **datatypes of the features** present in the dataset.

```
data.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
```

➢ Here we have the features with **"Float and int"** datatypes except the feature **"pdate"** which is **"object"** datatype and from this feature multiple new features are to be extracted and then further they are changed to the appropriate datatypes.

➢ Here we will find out the **Information of the data** whether the data has any **null values** or not.

```
data.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
```

➢ Here we have come to know that the data has no null values and all the features of the dataset has the data in it.

➢ Now we will have a look at the **statistical analysis of the data:**

```
data.describe()
```

	Unnamed: 0	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_r
count	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.00000	209593.000000	20
mean	104797.000000	0.875177	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.84780	3712.202921	
std	60504.431823	0.330519	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.89223	53374.833430	
min	1.000000	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.00000	-29.000000	
25%	52399.000000	1.000000	246.000000	42.440000	42.692000	280.420000	300.260000	1.00000	0.000000	
50%	104797.000000	1.000000	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.00000	0.000000	
75%	157195.000000	1.000000	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.00000	0.000000	
max	209593.000000	1.000000	999860.755200	265926.000000	320630.000000	198926.110000	200148.110000	998650.37770	999171.809400	5

8 rows × 34 columns

```
data.describe(include = "O")
```

	msisdn	pcircle	pdate
count	209593	209593	209593
unique	186243	1	82
top	04581I85330	UPW	04-07-2016
freq	7	209593	3150

➢ Here the data has **few columns** in which the **difference between mean and the standard deviation is more and, in few columns, it is less** and is appropriate that few columns has mean value higher than standard deviation value and also there are few columns in which standard deviation is higher than the mean value and also we can see that statistical analysis of the object datatype columns also in which the unique values of the data are mentioned and also we get more information regarding the frequent values present in the data of the columns.

➢ Now, we will **drop the unrequired columns** present in the data.

```
data = data.drop(columns = ["Unnamed: 0","msisdn"])
```

```
data.shape
```

```
(209593, 35)
```

➢ Here the features **"Unnamed: 0" and "msisdn"** have been **removed** which are not at all useful for the model building and which don't rely to the dataset.

➢ Here now we will know the **value counts of our column "label"**.

```
label_column_count = pd.DataFrame(data["label"].value_counts())
```

```
label_column_count
```

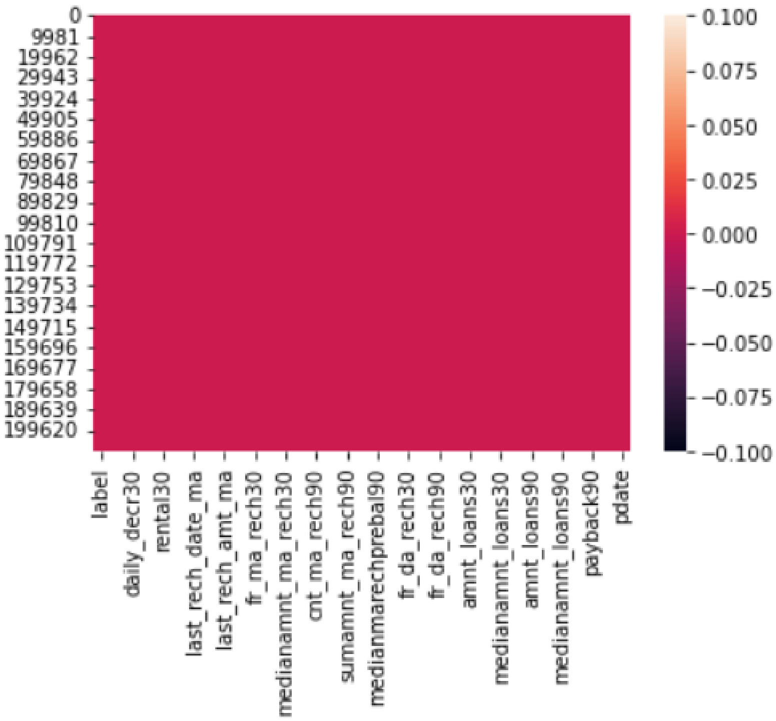
	label
1	183431
0	26162

➢ Here we can see that the label column has imbalanced data in it and we have to balance the data and then proceed with our model building.

➢ Here we are going to **plot the heatmap for the null values** of the data:

```
sns.heatmap(data.isnull())
```

```
<AxesSubplot:>
```



➢ Here we can see that the data has no null values in it and so heatmap is with unicoloured.

➢ Now, we will be **Pre-processing our data** for our better model accuracy.

```
pcircle = pd.DataFrame(data["pcircle"].value_counts())
```

```
pcircle
```

	pcircle
UPW	209593

```
data = data.drop(columns = ["pcircle"])
```

```
data.shape
```

```
(209593, 34)
```

➢ Here first of all we have seen the data present in the column "pcircle" and which we got to know that is same throughout the records and so we can delete the column as it is of no use for our prediction.

➢ Now we will start the **pre-processing of the column "pdate"**.

```
data["Pdate"] = pd.to_datetime(data.pdate,format = "%d-%m-%Y").dt.day
```

```
data["Pmonth"] = pd.to_datetime(data.pdate,format = "%d-%m-%Y").dt.month
```

```
data["Pyear"] = pd.to_datetime(data.pdate,format = "%d-%m-%Y").dt.year
```

➢ Here we can see that from the column **"pdate"**, multiple columns are extracted with the help of **"pd.to_datetime".**

```
data.head(7)
```

last_rech_amt_ma	cnt_ma_rech30	...	cnt_loans90	amnt_loans90	maxamnt_loans90	medianamnt_loans90	payback30	payback90	pdate	Pdate	Pmonth	Pyear
1539	2	...	2.0	12	6	0.0	29.000000	29.000000	20-07-2016	20	7	2016
5787	1	...	1.0	12	12	0.0	0.000000	0.000000	10-08-2016	10	8	2016
1539	1	...	1.0	6	6	0.0	0.000000	0.000000	19-08-2016	19	8	2016
947	0	...	2.0	12	6	0.0	0.000000	0.000000	06-06-2016	6	6	2016
2309	7	...	7.0	42	6	0.0	2.333333	2.333333	22-06-2016	22	6	2016
1539	4	...	3.0	18	6	0.0	11.000000	8.333333	02-07-2016	2	7	2016
5787	1	...	1.0	6	6	0.0	0.000000	0.000000	05-07-2016	5	7	2016

```
data.drop(["pdate"],axis = 1,inplace = True)
```

➢ Here can see that the additional columns are added at the end of the dataframe and nextly the parent column is dropped from the dataframe.

➢ Now here we will the **check the value counts** of the extracted column **"pyear".**

```
Pyear = pd.DataFrame(data["Pyear"].value_counts())
```

```
Pyear
```

	Pyear
2016	209593

```
data.drop(["Pyear"],axis = 1,inplace = True)
```

```
data.head()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30	...	maxamnt_loans30
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2	...	6.0
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1	...	12.0
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1	...	6.0
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0	...	6.0
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7	...	6.0

5 rows × 35 columns

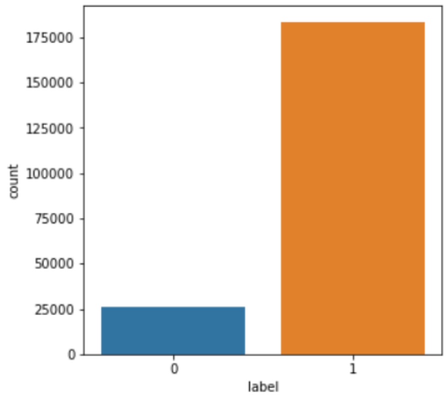
➢ Here the newly created column **"pyear"** has the entire data filled with only one type of data and is no use and so we can drop this column.

- Here, we will start with the **Visualization of the features** of the data.
- Here we will visualize few of the features with univariate and bivariate analysis.

Univariate Analysis:

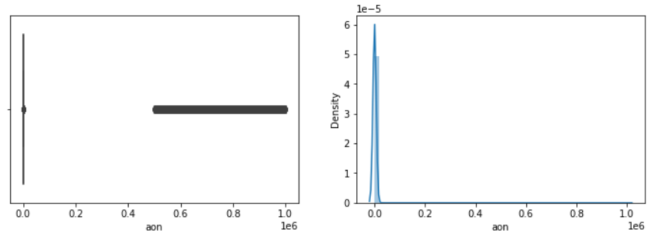
label:

```
plt.figure(figsize=(5,5))
sns.countplot(data.label);
```



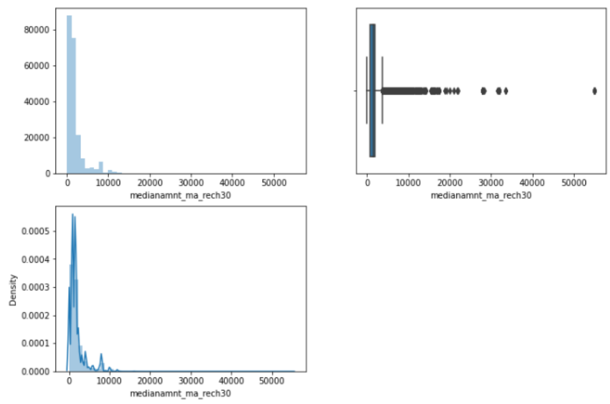
aon :

```
# aon

plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.boxplot(data['aon']);
plt.subplot(2,2,2)
sns.distplot(data['aon']);
```



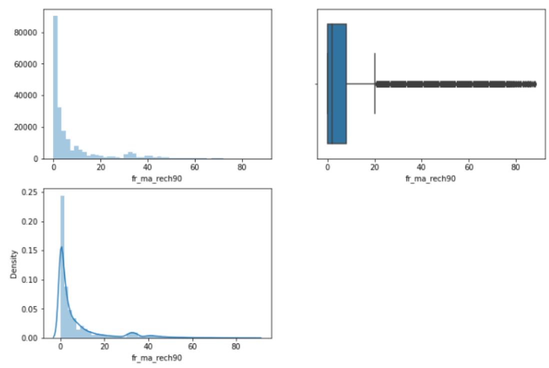
medianamnt_ma_rech30:

```
# medianamnt_ma_rech30

plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['medianamnt_ma_rech30'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['medianamnt_ma_rech30']);
plt.subplot(2,2,3)
sns.distplot(data['medianamnt_ma_rech30']);
```



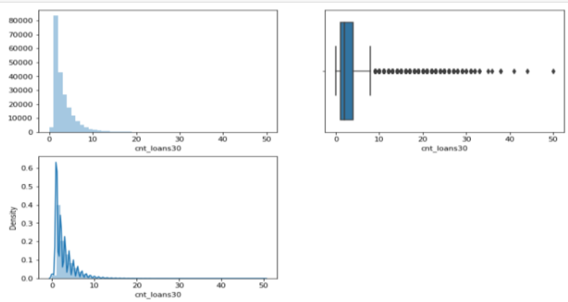
fr_ma_rech90 :

```
# fr_ma_rech90

plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['fr_ma_rech90'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['fr_ma_rech90']);
plt.subplot(2,2,3)
sns.distplot(data['fr_ma_rech90']);
```



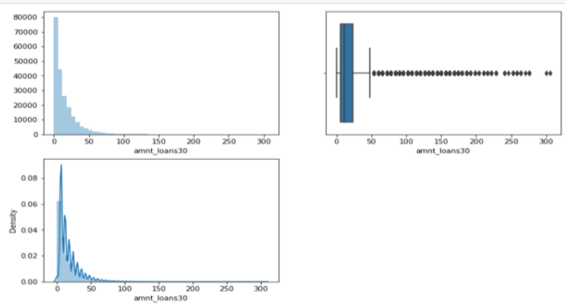
cnt_loans30 :

```
# cnt_loans30

plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['cnt_loans30'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['cnt_loans30']);
plt.subplot(2,2,3)
sns.distplot(data['cnt_loans30']);
```



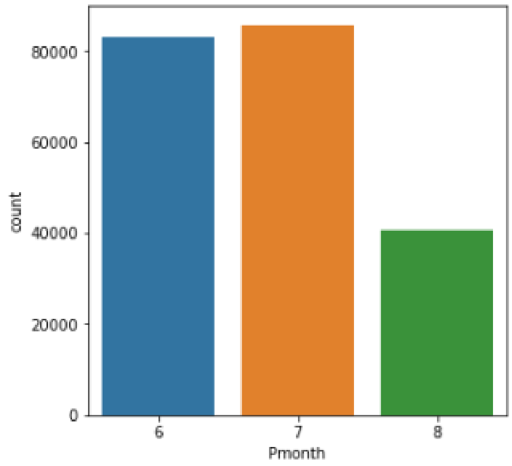
amnt_loans30 :

```
# amnt_loans30

plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['amnt_loans30'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['amnt_loans30']);
plt.subplot(2,2,3)
sns.distplot(data['amnt_loans30']);
```



Pmonth :

```
plt.figure(figsize=(5,5))
sns.countplot(data.Pmonth);
```



Documentation of the above visualized columns:

- **Label:** Here we can see that the column has an attribute(non-defaulter) with very high count than the other attribute (defaulter).
- **Aon:** Here we can see that the column has many numbers of outliers present and also there are dense in nature and the distribution peak is very narrow.
- **Medianamnt_ma_rech30:** Here we can see that the column has a large number of outliers which are dense in nature and the distribution curve has multiple peaks and also is with skewness.
- **fr_ma_rech90:** Here we can see that the column has many outliers which are very dense in nature to the quartile and distribution curve has much skewness and the peak is also narrow in nature.
- **cnt_loans30:** Here we can see that the column has a large number of outliers which are dense in nature and the distribution curve has multiple peaks and also is with skewness.
- **amnt_loans30:** Here we can see that the column has a large number of outliers which are dense in nature and the distribution curve has multiple peaks and also is with skewness.
- **Pmonth:** Here we can see that the column has the highest count for the attribute 7th month, followed by the 6th month and the least count is for 8th month.

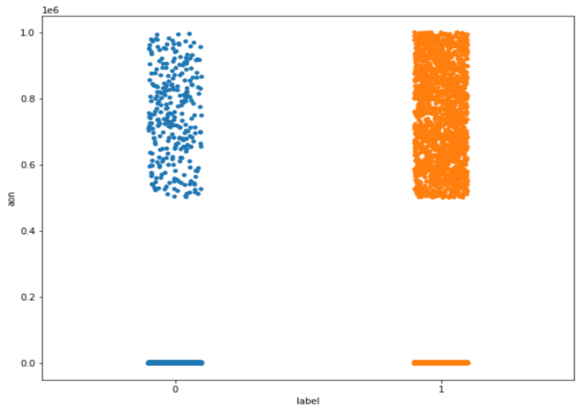
- These are few of the features for which we have seen the univariant analysis**.**

Bivariant Analysis:

aon with label:

```
plt.figure(figsize = (10,8))
sns.stripplot(x = 'label',y ='aon', data = data)
```

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

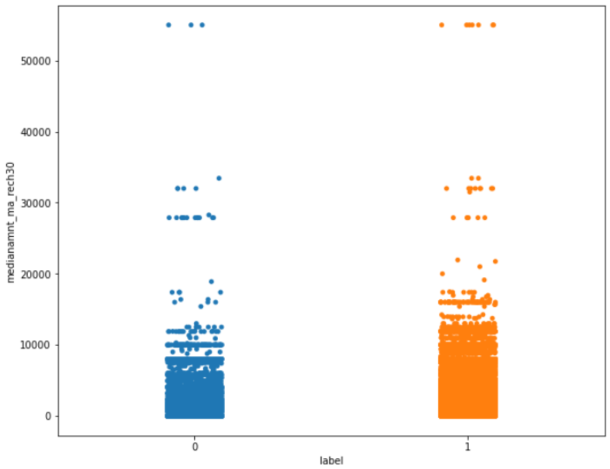


- ✓ **Observation:** Here we can see that the label 1 attribute has high and dense customers than the label 0 attribute.

medianamnt_ma_rech30 with label:

```
plt.figure(figsize = (10,8))
sns.stripplot(x = 'label',y ='medianamnt_ma_rech30', data = data)
```

<AxesSubplot:xlabel='label', ylabel='medianamnt_ma_rech30'>

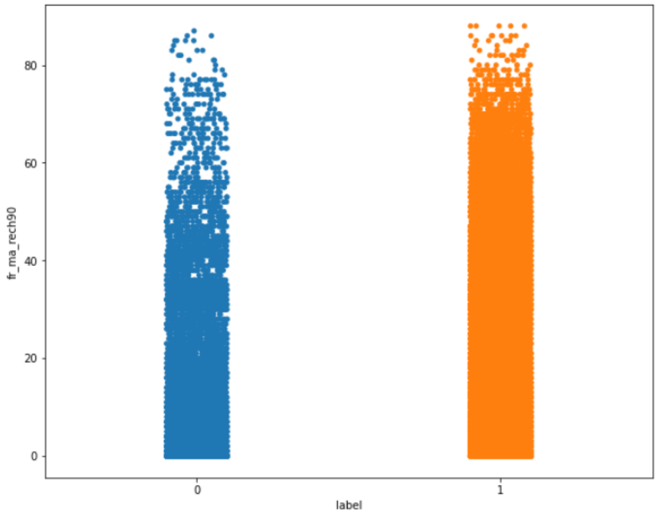


- ✓ **Observation:** Here we can see that there is high density of customers is for both the label attributes at their starting points is same but as the amount for the last recharge increases there, we can see the decrease in the density of the customers and at the highest last recharge amount point the density for both of the label attributes is almost same but when compared to label 0 ie., defaulters the label 1 ie., non-defaulters attribute has the high density.

fr_ma_rech90 with label:

```
plt.figure(figsize = (10,8))
sns.stripplot(x = 'label',y ='fr_ma_rech90', data = data)
```

```
<AxesSubplot:xlabel='label', ylabel='fr_ma_rech90'>
```

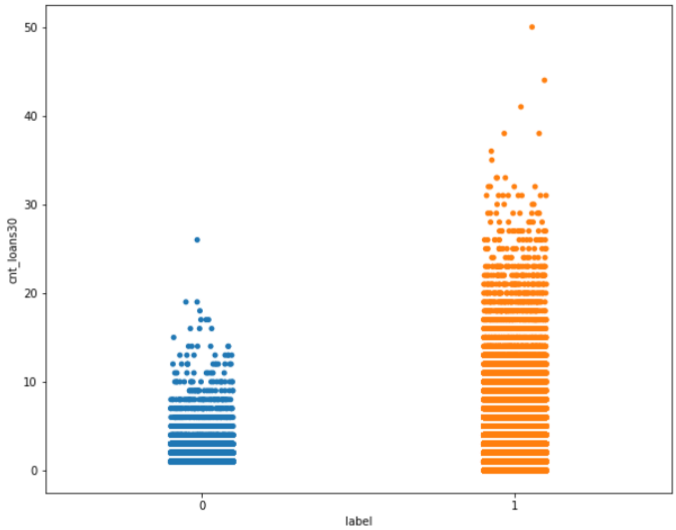


- ✓ **Observation:** here we can see that the high density is present at the starting stage of the frequency of main account recharged but as there is increase in the day count the density decreased in label 0 attribute but there is density remained in the label 1 attribute and at the final point the density becomes less in label 1 attribute and it becomes negligible in label 0 attribute.

cnt_loans30 with label :

```
plt.figure(figsize = (10,8))
sns.stripplot(x = 'label',y ='cnt_loans30', data = data)
```

```
<AxesSubplot:xlabel='label', ylabel='cnt_loans30'>
```

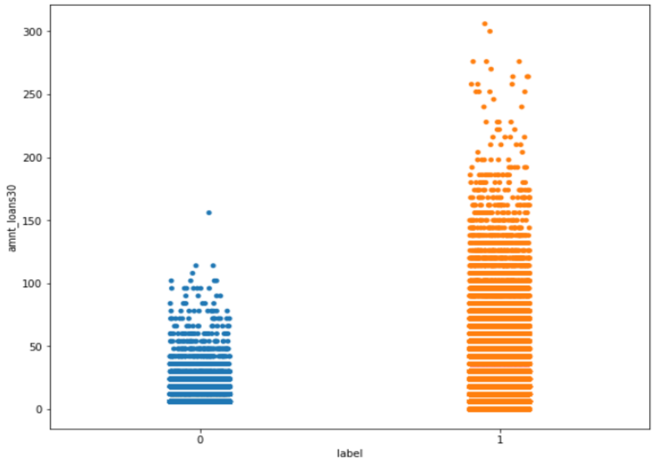


- ✓ **Observation:** Here we can see that the customers who are non-defaulters are more in number for the number of loans taken in last 30days.

amnt_loans30 with label :

```
plt.figure(figsize = (10,8))
sns.stripplot(x = 'label',y ='amnt_loans30', data = data)
```

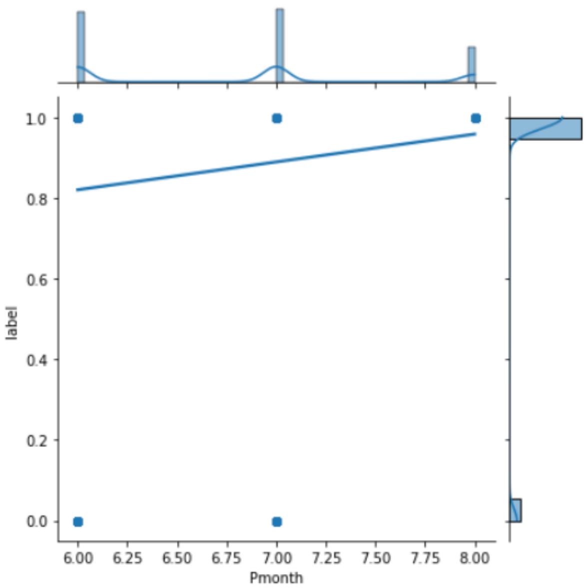
<AxesSubplot:xlabel='label', ylabel='amnt_loans30'>



- ✓ **Observation:** Here we can see that the customers who are non-defaulters are more in number for the amount of loans taken in last 30days.

Pmonth with label:

```
sns.jointplot(data=data, x='Pmonth', y='label', kind='reg');
```



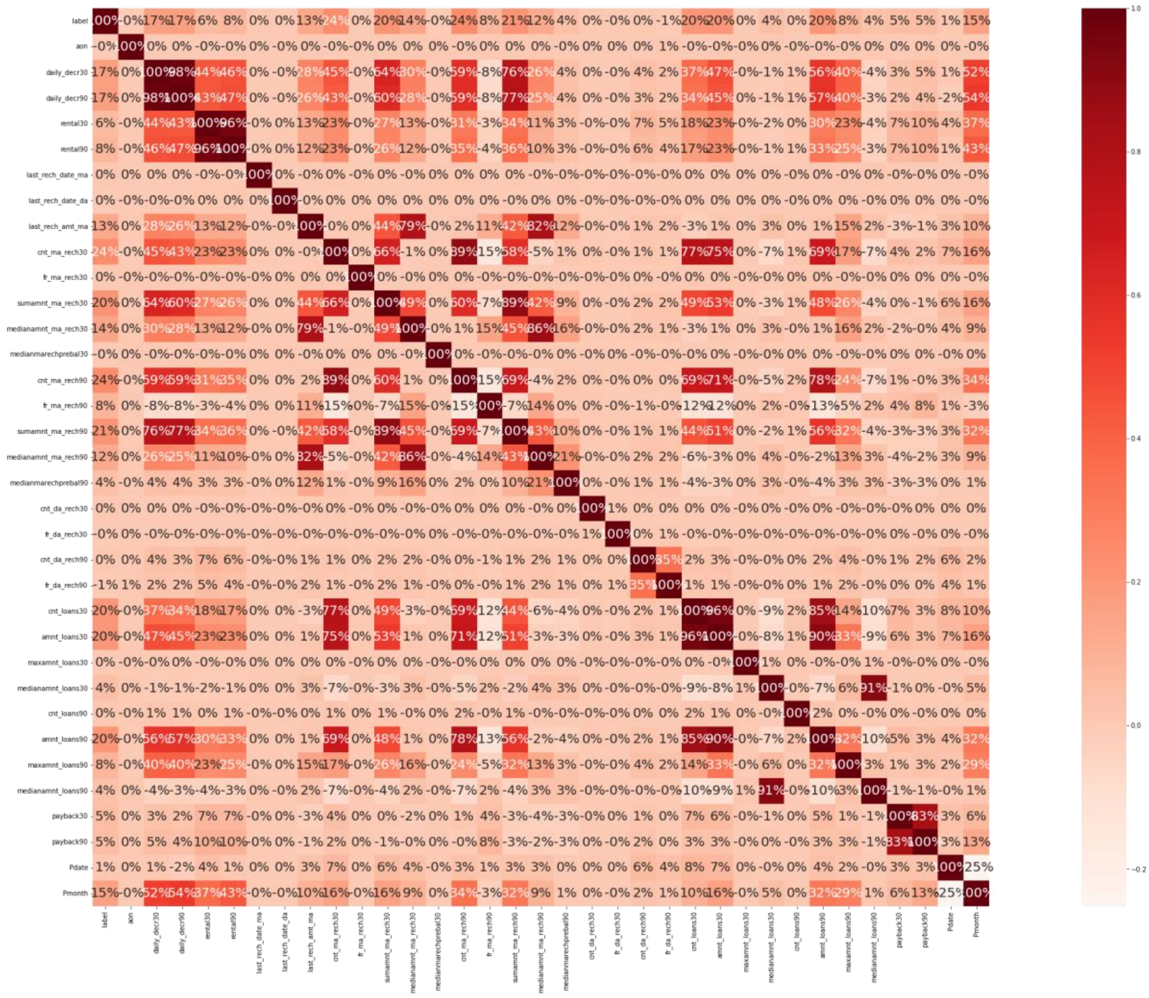
- ✓ **Observation:** Here we can see that both of the label attributes have same density in both of the 6th and 7th months but the label 1 attribute also has customers in 8th month which are absent in label 0 attribute.

➢ These are few of the features for which we have seen the bivariate analysis.

➢ Here we will find the **Correlation** between the features and the label column.

```
corr_data = data.corr()
corr_data['label'].sort_values(ascending = False)
```

```
label                   1.000000
cnt_ma_rech30           0.237331
cnt_ma_rech90           0.236392
sumamnt_ma_rech90       0.205793
sumamnt_ma_rech30       0.202828
amnt_loans90            0.199788
amnt_loans30            0.197272
cnt_loans30             0.196283
daily_decr30            0.168298
daily_decr90            0.166150
Pmonth                  0.154949
medianamnt_ma_rech30    0.141490
last_rech_amt_ma        0.131804
medianamnt_ma_rech90    0.120855
fr_ma_rech90            0.084385
maxamnt_loans90         0.084144
rental90                0.075521
rental30                0.058085
payback90               0.049183
payback30               0.048336
medianamnt_loans30      0.044589
medianmarechprebal90    0.039300
medianamnt_loans90      0.035747
Pdate                   0.006825
cnt_loans90             0.004733
cnt_da_rech30           0.003827
last_rech_date_ma       0.003728
cnt_da_rech90           0.002999
last_rech_date_da       0.001711
fr_ma_rech30            0.001330
maxamnt_loans30         0.000248
fr_da_rech30           -0.000027
aon                    -0.003785
medianmarechprebal30   -0.004829
fr_da_rech90           -0.005418
Name: label, dtype: float64
```



✓ **Observation:** Here we can see that the maximum correlation is present between: a) medianamnt_loans90 and medianamnt_loans30, b) rental_90 and rental_30, c) daily_decr90 and daily_decr30, d) amnt_loans90 and amnt_loan s90, e) cnt-loans30 and amnt_loans30 etc...

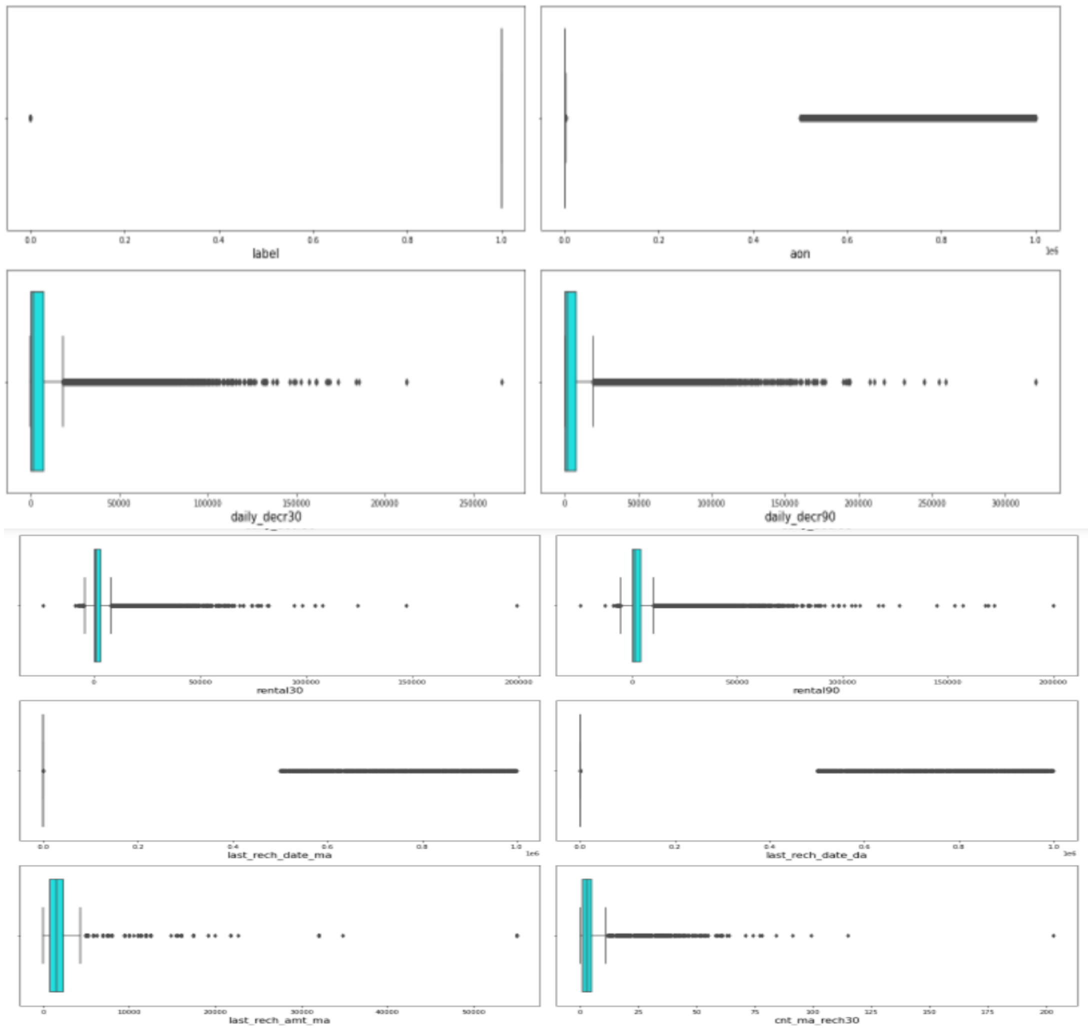
we have a lot features which with high correlation above 80%.

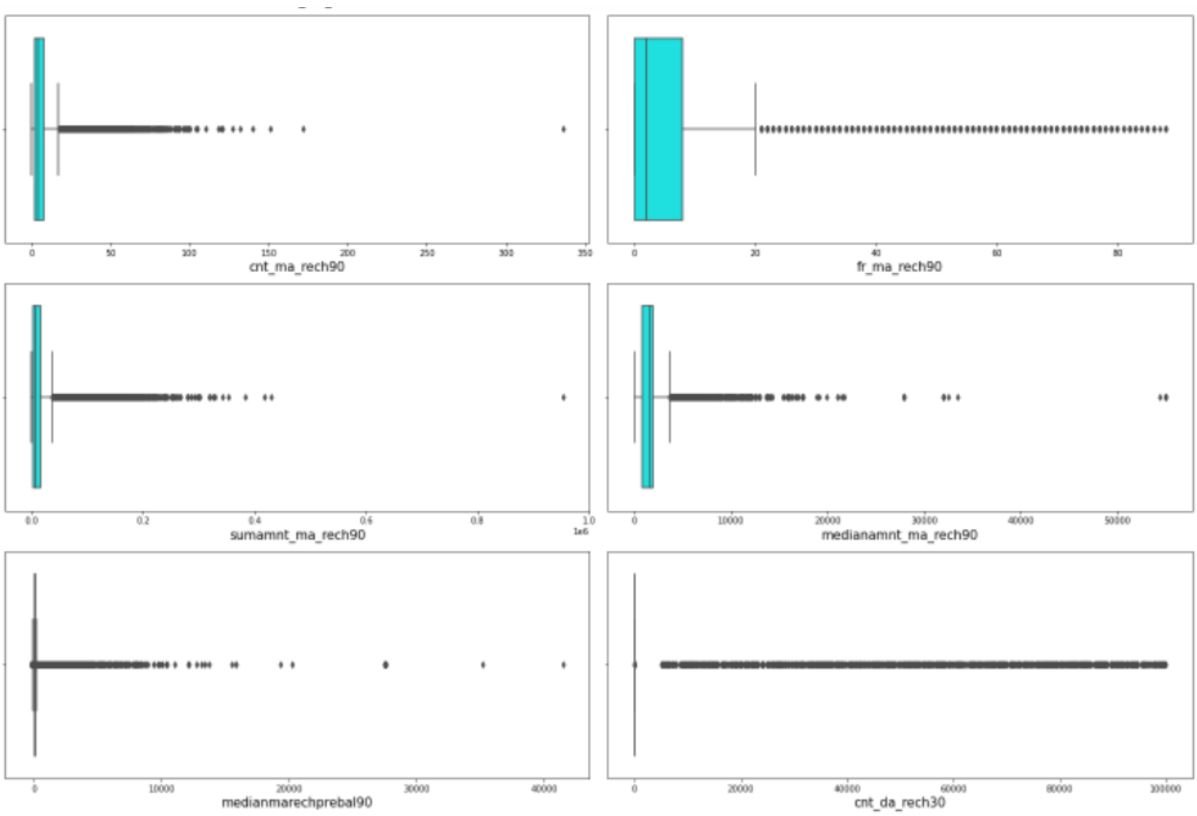
➢ Detection of the Outliers:

Here we will find the outliers present in the different features of our data. Here we will see the outliers of few of the features.

Detection of the Outliers:

```
plt.figure(figsize = (20,80))
pltnum = 1
for i in data:
    if pltnum<=36:
        plt.subplot(18,2,pltnum)
        sns.boxplot(data[i], color = 'cyan', orient = 'h')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```





- ✓ **Observation:** Here we can see that most of the columns have outliers present in them and also with dense and also with number of outliers present and so we have to treat them for better accuracy in our model building.
- ➢ Now we will move for **treating the outliers** present in our data to some extent, for our model better accuracy.
- ✓ Here we use Z-Score method for treating the outliers present in the columns of the data.

```
from scipy.stats import zscore
```

```
z = np.abs(zscore(data))
z.shape
```

```
(209593, 35)
```

```
threshold = 5.5
print(np.where(z>5.5))
```

```
(array([    30,     53,     65, ..., 209531, 209533, 209576], dtype=int64), array([6, 6, 1, ..., 7, 6, 1], dtype=int64))
```

```
data_new = data[(z<5.5).all(axis = 1)]
print(data.shape)
print(data_new.shape)
```

```
(209593, 35)
(192459, 35)
```

- ✓ Here we can see that we have taken 5.5 as threshold as we cannot afford of losing more than 8% of the data and we can see that the number of records have been decreased to extent which means we have succeeded in treating the outliers to some extent.

➢ This is our new data now:

```
data_new
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30	...	maxamnt_loa
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2	...	
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1	...	
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1	...	
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0	...	
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7	...	
...	...	...	...	...	...	...	...	...	...	...	...	
209588	1	404.0	151.872333	151.872333	1089.19	1089.19	1.0	0.0	4048	3	...	
209589	1	1075.0	36.936000	36.936000	1728.36	1728.36	4.0	0.0	773	4	...	
209590	1	1013.0	11843.111670	11904.350000	5861.83	8893.20	3.0	0.0	1539	5	...	
209591	1	1732.0	12488.228330	12574.370000	411.83	984.58	2.0	38.0	773	5	...	
209592	1	1581.0	4489.362000	4534.820000	483.92	631.20	13.0	0.0	7526	2	...	

192459 rows × 35 columns

- ✓ Now here we can see that our dataset has a smaller number of records compared to before which indicates that we have treated the outliers also to some extent.

- ➢ Now we will have a look at the **Data loss%.**

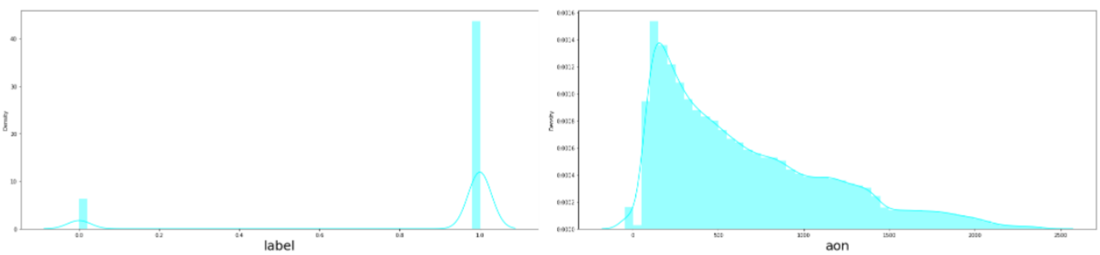
```
data_loss = (209593-192459)/209593*100
```

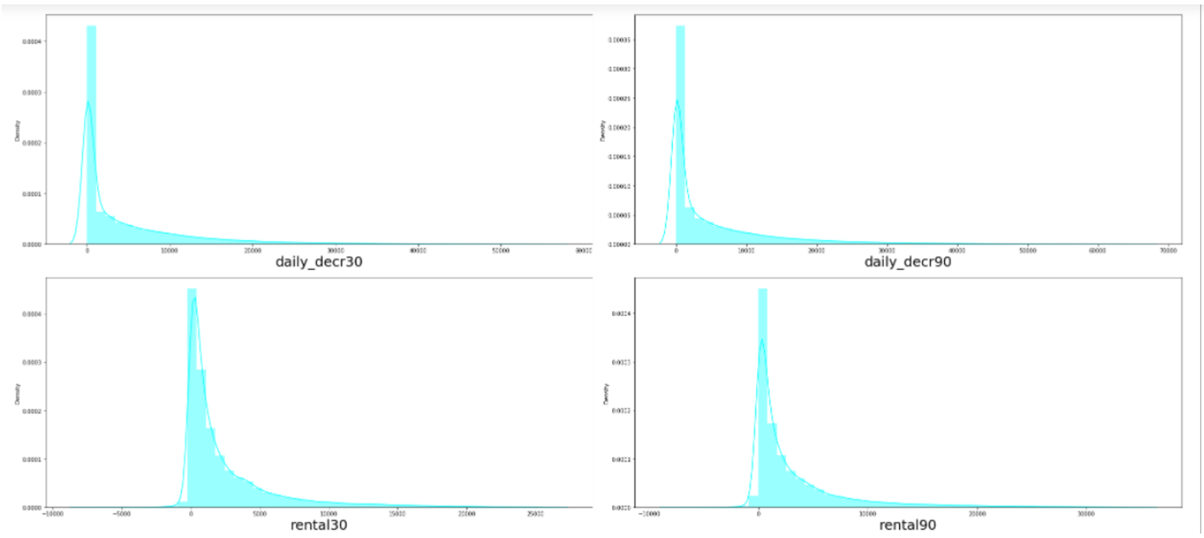
```
data_loss
```

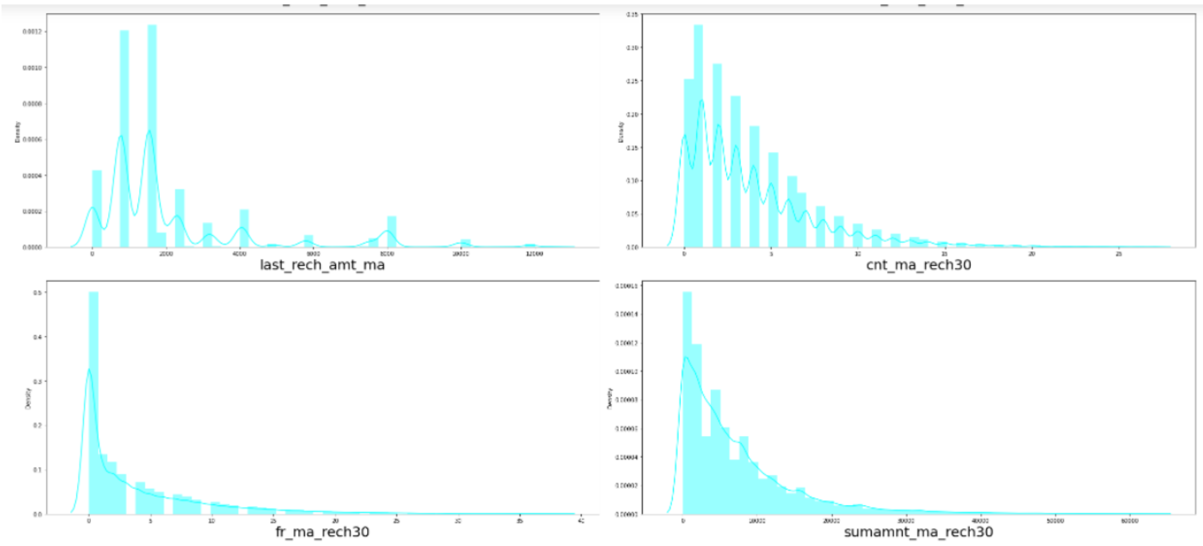
```
8.174891337019844
```

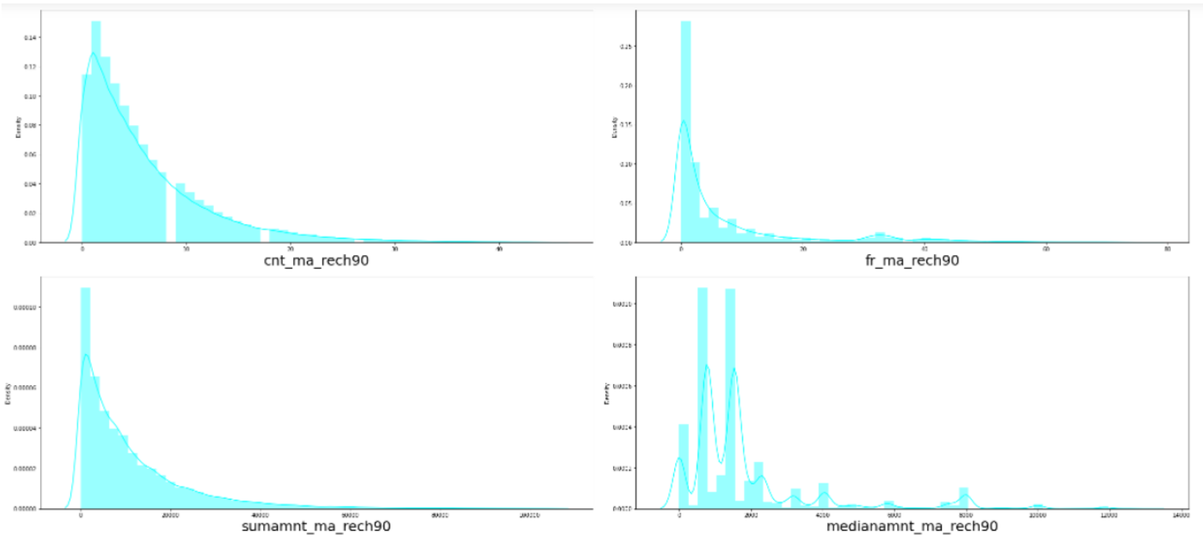
➢ Checking the skewness of the data:

```
plt.figure(figsize = (30,120))
pltnum = 1
for i in data_new:
    if pltnum<=36:
        plt.subplot(18,2,pltnum)
        sns.distplot(data_new[i], color = 'cyan')
        plt.xlabel(i, fontsize = 25)
    pltnum+=1
plt.tight_layout()
```




daily_decr30
daily_decr90
rental30
rental90


last_rech_amt_ma
cnt_ma_rech30
fr_ma_rech30
sumamnt_ma_rech30


cnt_ma_rech90
fr_ma_rech90
sumamnt_ma_rech90
medianamnt_ma_rech90

- Here we can see that the columns have skewness and we will have a look at the skew values of the features with the label column.

```
data_new.skew().sort_values()

label                   -2.248861
Pdate                    0.206862
Pmonth                   0.371847
aon                      0.947905
cnt_ma_rech30            1.730082
maxamnt_loans90          1.747901
cnt_ma_rech90            1.891819
cnt_loans30              1.945914
amnt_loans30             1.967762
fr_ma_rech30             2.010125
sumamnt_ma_rech30        2.176749
last_rech_amt_ma         2.221092
fr_ma_rech90             2.228096
amnt_loans90             2.244584
sumamnt_ma_rech90        2.268428
daily_decr30             2.372679
medianamnt_ma_rech30     2.451058
medianamnt_ma_rech90     2.465278
daily_decr90             2.514251
rental30                 2.561126
rental90                 2.689101
last_rech_date_ma        3.097659
payback90                3.601847
payback30                3.903939
medianamnt_loans30       4.075996
medianamnt_loans90       4.453088
medianmarechprebal90     5.252733
cnt_da_rech90            7.427612
last_rech_date_da        9.974328
medianmarechprebal30    10.838790
cnt_da_rech30           35.421656
maxamnt_loans30         37.890780
cnt_loans90             54.802217
fr_da_rech90            68.727574
fr_da_rech30            88.162720
dtype: float64
```

  - ✓ Here most of the features have skewness and except the label column all the other feature columns are positively skewed, in that few columns are with medium positive skewness and there are also few columns with very high positive skewness.

- Before moving to the removal of the skewness present in the data we will first split the data and then scale the data which has features and then we will remove skewness.

Splitting of the data:

```
x = data_new.drop(columns = 'label')
y = data_new['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	...	maxamnt_l
0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2	21.0	...	
1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1	0.0	...	
2	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1	0.0	...	
3	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0	0.0	...	
4	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7	2.0	...	

5 rows × 34 columns

```
y.head()
```

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

```
from sklearn.preprocessing import MinMaxScaler
```

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

```
x
```

	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	...	m
0	0.128617	0.055251	0.047018	0.234328	0.187155	0.218310	0.201389	0.12312	0.074074	0.552632	...	
1	0.305466	0.216785	0.183925	0.335913	0.266956	0.345070	0.201389	0.46296	0.037037	0.000000	...	
2	0.234325	0.025730	0.021825	0.254229	0.202040	0.225352	0.201389	0.12312	0.037037	0.000000	...	
3	0.116158	0.001202	0.001019	0.232551	0.184812	0.492958	0.201389	0.07576	0.000000	0.000000	...	
4	0.399920	0.003507	0.002975	0.260046	0.206663	0.232394	0.201389	0.18472	0.259259	0.052632	...	
...	...	...	...	...	...	...	...	...	...	...	...	
192454	0.181672	0.003529	0.002993	0.259762	0.206437	0.211268	0.201389	0.32384	0.111111	0.052632	...	
192455	0.451367	0.001481	0.001257	0.278468	0.221303	0.232394	0.201389	0.06184	0.148148	0.026316	...	
192456	0.426447	0.211817	0.180595	0.399437	0.387943	0.225352	0.201389	0.12312	0.185185	0.210526	...	
192457	0.715434	0.223310	0.190720	0.239938	0.204004	0.218310	0.465278	0.06184	0.185185	0.105263	...	
192458	0.654743	0.080805	0.069228	0.242048	0.195785	0.295775	0.201389	0.60208	0.074074	0.026316	...	

192459 rows × 34 columns

- ✓ Here in the above we have splitted the data into x and y variables where all the features are assigned to the variable "x" and the label column is assigned to the variable "y" and we transform the features present in the variable x to be scaled and then we will get a scaled and transformed data of x, which is used for removing the skewness present in the features.

Removing the skewness of the data:

✓ Here we remove the skewness of the data through **"power_transform"** method.

```
from sklearn.preprocessing import power_transform
```

```
transform_data = power_transform(x, method = 'yeo-johnson')
x = pd.DataFrame(transform_data, 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	...	maxa
0	-0.772547	0.159357	0.095064	-0.882238	-0.891112	-0.400277	-0.124951	0.111881	-0.374333	1.857935	...	
1	0.390238	1.451604	1.382771	0.947386	0.707262	1.608371	-0.124951	1.734547	-0.885060	-1.012938	...	
2	-0.008536	-0.366210	-0.400540	-0.363228	-0.482142	-0.256674	-0.124951	0.111881	-0.885060	-1.012938	...	
3	-0.879643	-0.935841	-0.921750	-0.933980	-0.961697	2.937812	-0.124951	-0.456250	-1.510166	-1.012938	...	
4	0.815744	-0.876137	-0.867678	-0.230492	-0.367920	-0.117609	-0.124951	0.651870	1.131382	-0.090680	...	

5 rows × 34 columns

✓ Here we can see that we have imported the "powertransform" library and passed the features present in the variable x into the powertransform and then we have removed the skewness of the data and converted into dataframe.

✓ Now, here we have a look at the skewness of the features whether they are changed or not.

```
x.skew()
```

```
aon                      0.140676
daily_decr30             0.569044
daily_decr90             0.609128
rental30                 0.067193
rental90                 0.332709
last_rech_date_ma       -0.776433
last_rech_date_da      -27.554947
last_rech_amt_ma         0.195634
cnt_ma_rech30            0.174297
fr_ma_rech30             0.444399
sumamnt_ma_rech30        0.253799
medianamnt_ma_rech30     0.143526
medianmarechprebal30    -0.262342
cnt_ma_rech90            0.226483
fr_ma_rech90             0.654612
sumamnt_ma_rech90        0.302047
medianamnt_ma_rech90     0.142348
medianmarechprebal90    -0.501943
cnt_da_rech30           27.955768
fr_da_rech30            74.134435
cnt_da_rech90            6.645226
fr_da_rech90            57.088248
cnt_loans30              0.277226
amnt_loans30             0.281108
maxamnt_loans30          4.807431
medianamnt_loans30       3.513169
cnt_loans90              0.552003
amnt_loans90             0.379588
maxamnt_loans90         -0.324459
medianamnt_loans90       3.850225
payback30                0.585642
payback90                0.544606
Pdate                   -0.012758
Pmonth                   0.039828
dtype: float64
```

```
x = np.cbrt(x)
```

```
x.skew()
```

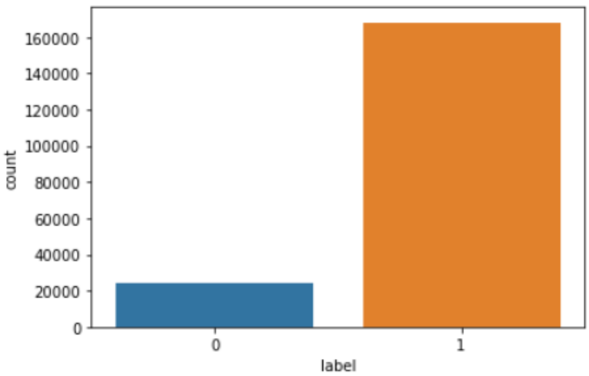
```
aon                      0.070230
daily_decr30             0.340121
daily_decr90             0.374143
rental30                 0.363987
rental90                 0.347998
last_rech_date_ma        0.647721
last_rech_date_da        6.373282
last_rech_amt_ma        -0.187723
cnt_ma_rech30           -0.026788
fr_ma_rech30             0.274985
sumamnt_ma_rech30        0.108371
medianamnt_ma_rech30    -0.083915
medianmarechprebal30     0.531069
cnt_ma_rech90            0.122436
fr_ma_rech90             0.384721
sumamnt_ma_rech90        0.135732
medianamnt_ma_rech90    -0.100244
medianmarechprebal90     0.510589
cnt_da_rech30           10.540065
fr_da_rech30            74.134435
cnt_da_rech90            6.645226
fr_da_rech90            57.088248
cnt_loans30              0.306502
amnt_loans30             0.309343
maxamnt_loans30          2.234694
medianamnt_loans30       3.513169
cnt_loans90              0.246665
amnt_loans90             0.168387
maxamnt_loans90          2.154257
medianamnt_loans90       3.850225
payback30                0.224000
payback90                0.152812
Pdate                   -0.023090
Pmonth                  -0.321926
dtype: float64
```

✓ Here we can see that most of the features have change in their skewness but still we have skewness present and so we use "cuberoot" from NumPy and then we pass "x" into it and assign to the variable "x" again where we can see that most of the columns have a lot of changes in their skewness except few and so we can proceed with our model building.

➢ Now we will balance the data as the data is imbalanced through the **“oversampling”** method where we import **SMOTE** to handle the imbalanced data.

```
sns.countplot(y)
```

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



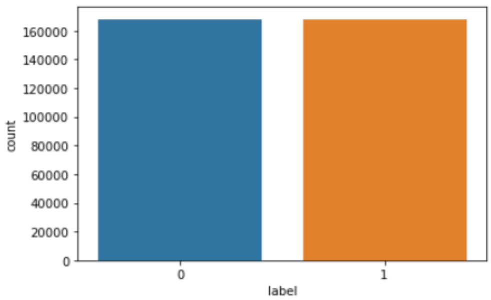
✓ The data is imbalanced as we can see and so we will import “SMOTE” and handle the imbalanced data.

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE()
x_over, y_over = sm.fit_resample(x,y)
```

```
sns.countplot(y_over)
```

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



✓ Here we can see that we have balanced the imbalanced data and also we can see here that both of the attributes present in the label column which were actually imbalanced are now in equal proportions and so by this we can say that our model is set for training and testing of the data.

Checking the Random state:

- ✓ Here we use train_test_split method for finding good random state number.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
from sklearn.model_selection import train_test_split
```

```
rs = 0
for i in range(0,200):
    x_train,x_test, y_train,y_test = train_test_split(x_over,y_over,test_size = 0.33, random_state = i)
    lg = LogisticRegression()
    lg.fit(x_train,y_train)
    ts_pred = lg.predict(x_test)
    tr_pred = lg.predict(x_train)
    ts_score = accuracy_score(y_test,ts_pred)
    tr_score = accuracy_score(y_train, tr_pred)
    if round(ts_score*100,1) == round(tr_score*100,1):
        if i>rs:
            rs = i
print('the best random state for the data set is', rs)
```

```
the best random state for the data set is 194
```

Observation : Here we have used train_test_split and passed x_over, y_over which are the variables after balancing the data and we used ". fit" method to train the data and predicted the test data and accuracy score for which we got the random state as 194.

```
x_train,x_test, y_train,y_test = train_test_split(x_over,y_over,test_size = 0.33, random_state = rs)
```

- ✓ Here we have used train_test_split and passed x_over, y_over which are the variables after balancing the data and we used ". fit" method to train the data and predicted the test data and accuracy score for which we got the random state as 194.

- ➢ Now we should proceed with the model testing with the **testsize 33%** and we present classification report and accuracy score for accuracy score.

✓ Models:

Logistic Regression:

```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
logreg_pred = logreg.predict(x_test)
logreg_score = accuracy_score(y_test,logreg_pred)
logreg_score
```

```
0.7765972785437506
```

```
print(classification_report(y_test, logreg_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.78     55610
           1       0.80      0.74      0.77     55360

    accuracy                           0.78    110970
   macro avg       0.78      0.78      0.78    110970
weighted avg       0.78      0.78      0.78    110970
```

```
print(roc_auc_score(y_test, logreg_pred))
```

```
0.776521495567292
```

- ✓ Here we can see that the model tested with 77% accuracy and the roc_auc_score is 77%.

Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier
```

```
randf = RandomForestClassifier()
randf.fit(x_train,y_train)
randf_pred = randf.predict(x_test)
randf_score = accuracy_score(y_test,randf_pred)
randf_score
```

```
0.9504370550599262
```

```
print(classification_report(y_test, randf_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     55610
           1       0.95      0.95      0.95     55360

    accuracy                           0.95    110970
   macro avg       0.95      0.95      0.95    110970
weighted avg       0.95      0.95      0.95    110970
```

```
print(roc_auc_score(y_test, randf_pred))
```

```
0.9504329868001036
```

✓ Here we can see that the model tested with 95% accuracy and the roc_auc_score is 95%.

Extra Trees Classifier:

```
from sklearn.ensemble import ExtraTreesClassifier
```

```
extr = ExtraTreesClassifier()
extr.fit(x_train,y_train)
extr_pred = extr.predict(x_test)
extr_score = accuracy_score(y_test,extr_pred)
extr_score
```

```
0.9581868973596468
```

```
print(classification_report(y_test, extr_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     55610
           1       0.97      0.95      0.96     55360

    accuracy                           0.96    110970
   macro avg       0.96      0.96      0.96    110970
weighted avg       0.96      0.96      0.96    110970
```

```
print(roc_auc_score(y_test, extr_pred))
```

```
0.9581573289751188
```

✓ Here we can see that the model tested with 95.8% accuracy and the roc_auc_score is 95.8%.

KNN Classifier:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
knn_pred = knn.predict(x_test)
knn_score = accuracy_score(y_test, knn_pred)
knn_score
```

```
0.8712805262683608
```

```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     55610
           1       0.97      0.77      0.86     55360

    accuracy                           0.87    110970
   macro avg       0.89      0.87      0.87    110970
weighted avg       0.89      0.87      0.87    110970
```

```
print(roc_auc_score(y_test, knn_pred))
```

```
0.8710463635449399
```

- ✓ Here we can see that the model tested with 87% accuracy and the roc_auc_score is 87%.

Checking for Cross Validation Score:

```
from sklearn.model_selection import cross_val_score
```

```
cv1 = cross_val_score(logreg, x_over,y_over,cv = 5)
cv1 = cv1.mean()
cv1
```

```
0.7765350743114267
```

```
cv2 = cross_val_score(randf, x_over,y_over,cv = 5)
cv2 = cv2.mean()
cv2
```

```
0.9489670709693794
```

```
cv3 = cross_val_score(extr, x_over,y_over,cv = 5)
cv3 = cv3.mean()
cv3
```

```
0.9643116623993808
```

```
cv4 = cross_val_score(knn, x_over,y_over,cv = 5)
cv4 = cv4.mean()
cv4
```

```
0.88203300546599
```

- ✓ Here we can see that out of all the models used for prediction, Extra Trees Classifier model is with high accuracy score and also high cross validation score which is 96.4%.

- ✓ Here we can see that our best model is "Extra Trees Classifier model" with an accuracy of 96% which is highest than the rest of the models and so we choose this model for **"Hyper parameter tuning".**

```python
from sklearn.model_selection import GridSearchCV
```

```python
params ={'n_estimators':[0,50],
        'criterion':['gini','entropy'],
        'max_depth':[2,4,6],
        'min_samples_split':[2,3,4],
        'bootstrap':[True,False]}
```

```python
final = GridSearchCV(ExtraTreesClassifier(),params,cv=5, n_jobs =-1)
final.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6], 'min_samples_split': [2, 3, 4],
                         'n_estimators': [0, 50]})
```

```python
final.best_params_
```

```
{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 2,
 'n_estimators': 50}
```

```python
final_rf = ExtraTreesClassifier(bootstrap = False, criterion= 'gini', max_depth = 32, min_samples_split = 3, n_estimators =600)
final_rf.fit(x_train,y_train)
final_pred = final_rf.predict(x_test)
final_score = accuracy_score(y_test,final_pred)
final_score
```

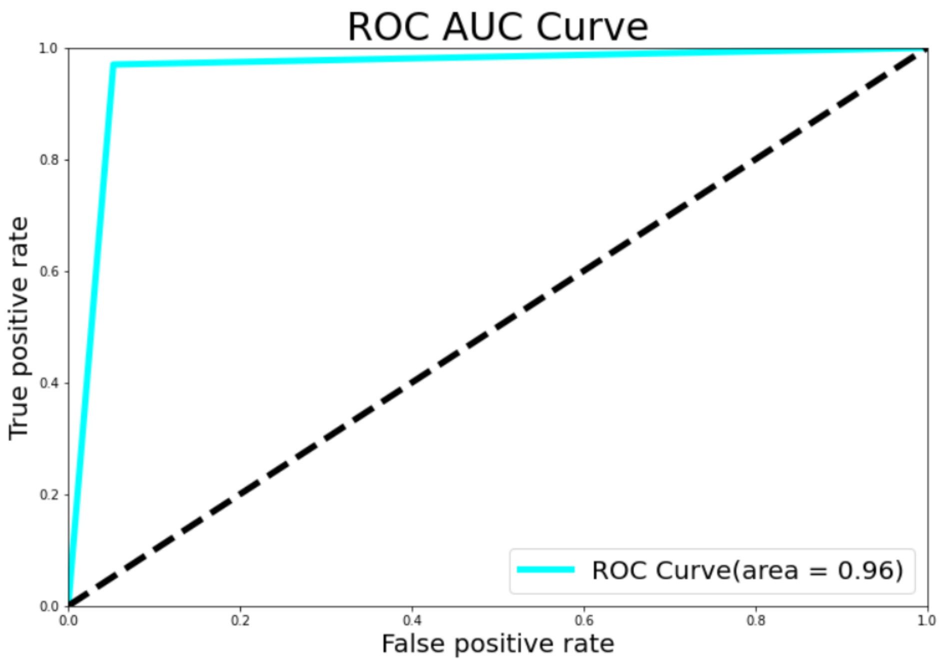
```
0.943840677660629
```

- ✓ Here we can see that we have imported **"GridSearchCV"** and we have used selected parameters and here we use Cross validation "5", and we train the model and select the best parameters and also, we have predicted the final accuracy score which is 94.3%

- ➢ Here we will present the **ROC and AUC Curve** and predict the area under the ROC Curve which is 0.96 ie., 96%

```python
from sklearn.metrics import roc_curve, auc
```

```python
fpr,tpr, thresholds  = roc_curve(extr_pred, y_test)
roc_auc = auc(fpr,tpr)

plt.figure(figsize = (12,8))
plt.plot(fpr, tpr, lw=5, color = 'cyan',label = 'ROC Curve(area = %0.2f)'%roc_auc)
plt.plot([0,1],[0,1],lw =5, color ='black', linestyle = '--')
plt.xlim(0.0,1.0)
plt.ylim(0.0,1.0)
plt.xlabel('False positive rate', fontsize = 20)
plt.ylabel('True positive rate', fontsize  =20)
plt.title('ROC AUC Curve', fontsize = 30)
plt.legend(loc ='lower right', fontsize = 20)
plt.show()
```



✓ Here we have presented the ROC AUC Curve which we got 0.96

➢ Now we will **"Save the model".**

```
import joblib
joblib.dump(final,'loan.pkl')
```

```
['loan.pkl']
```

✓ Finally, our model is saved and the file is **"loan.pkl".**

➢ **The key factors which helped us to finalize the model was "F1 Score", "Cross Validation Score" and "AUC-ROC Curve" and finally got to understand that "Extra Trees" is the best model used for predicting the Micro Credit defaulters.**

CONCLUSION

- ✓ We have successfully built a model using multiple models and found that the **Extra Trees Classifier model** is the best model for predicting the values.
- ✓ These are the keys which are used for model prediction of our dataset:
  - i. Average precision is 0.96
  - ii. F1 Score is 0.96 and
  - iii. ROC – AUC Score is also 0.958
- ✓ We can see from the boxplot that we have a lot of outliers present in the data and I had to proceed with the outliers as I cannot afford to lose more data as the data is expensive.
- ✓ Also, I couldn't handle skewness completely inspite of using few transforming methods and so I had to proceed with skewness to build the model.
- ✓ Looking at the heat map for correlation, I could see there were few independent variables which were correlated with each other, yet I have not removed any variable based on their correlation because multi-collinearity will not affect prediction.

Limitations of this work and Scope for Future Work:

- ✓ Due to the presence of lot of outliers and also skewness for few variables, we are unsure whether the model is going to perform well to a completely new dataset.
- ✓ There was Class Imbalance which had to be handled and for which we had to rebalance the data and by which we get completely new data and this may have certain affect on the model on the model building.

- ➢ Other than these above limitations, I couldn't find more scope for improvement.

Thank You