Micro-Credit Defaulter Model

Submitted by:

Vijay Ashley Rodrigues K.

ACKNOWLEDGMENT

- I would like to thank FlipRobo Technologies for providing me this opportunity and guidance throughout the project and all the steps that are implemented.
- I have primarily referred to various articles scattered across various websites for the purpose of getting an idea on Microfinance in general.
- I would like to thank the technical support team also for helping me out and reaching out to me on clearing all my doubts as early as possible
- I would like to thank my project SME Sajid Choudhary for providing the flexibility in time and also for giving us guidance in creating the project.
- The following are some of the articles I referred to in this project.

- https://www.gdrc.org/icm/model/model-fulldoc.html
- https://startuptalky.com/microfinance-models-india/
- https://link.springer.com/article/10.1057/s41264-020-00085-7
- http://www.gbgindonesia.com/en/finance/article/2016/indonesia_s_microfinance_sector_overview_key_component_for_sustainable_growth_11549.php
- https://idemest.com/reports/indonesia-telecoms-report-mobile-broadband-market-industry-analysis/

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.

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).

- Conceptual Background of the Domain Problem

    - A general of 14 fashions are defined below. They include, associations, financial institution guarantees, network banking, cooperatives, credit score unions, grameen, group, individual, intermediaries, NGOs, peer pressure, ROSCAs, small business, and village banking fashions.
    - In reality, the fashions are loosely associated with every other, and maximum right and sustainable microfinance establishments have functions of or greater fashions of their activities.
    - From my understanding, I believe the scenario in the project is related to Community Banking Model.
    - Community Banking version basically treats the entire network as one unit, and establishes semi-formal or formal establishments via which microfinance is dispensed. Such establishments are normally shaped via way of means of significant assist from NGOs and different organizations, who additionally educate the network individuals in numerous economic sports of the network bank.
    - These establishments may also have financial savings additives and different income-producing initiatives covered of their structure. In many cases, network banks also are a part of large network improvement programmes which use finance as an inducement for action.

- Motivation for the Problem Undertaken

    - Many of those fashions are indeed "formalized" variations of casual economic structures. Informal structures have historic precedents that predate current banking structures. They are nonetheless in lifestyles these days used broadly speaking through low-income families who do now no longer have get right of entry to formal banks.
    - GDRC has evolved a continuum of casual credit score providers that simply illustrates the hyperlink among such casual structures and the fashions illustrated.
    - Given how the current economic conditions work and how this microfinance in meant to support low-income groups as a poverty-reduction tool, but also representing $70 billion in outstanding loans with a global outreach of 200 million clients.

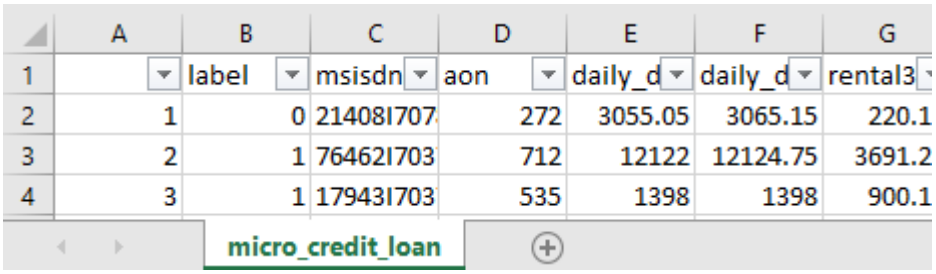
Analytical Problem Framing

- Mathematical/ Analytical Modelling of the Problem

  - The pattern statistics is supplied to us from our patron database. It is hereby given to you for this exercise. In order to enhance the choice of clients for the credit, the patron desires a few predictions that might assist them in similarly funding and development in choice of clients.
  - Build a version which may be used to expect in phrases of a chance for every mortgage transaction, whether or not the purchaser may be paying lower back the loaned quantity inside five days of coverage of mortgage. In this case, Label '1' shows that the mortgage has been paid i.e. Non- defaulter, while, Label '0' shows that the mortgage has now no longer been paid i.e. defaulter.

- Data Sources and their formats

  - The dataset is provided by FlipRobo and is available for academic purpose only and not for any kind of commercial activities.
  - The dataset contains the customer purchasing preferences of Indian E-Tailers with 209593 records (rows) and 35 features (columns).
  - The file is in .csv format and has a tab name "micro_credit_loan" as shown below.
  - We have another excel file named "Data_Description" in the .xlsx format that gives us description of the information of individual columns
  - The dataset is already in numerical format and consists both numerical and categorical features



	A	B	C	D	E	F	G
1		label	msisdn	aon	daily_d	daily_d	rental3
2	1	0	21408I707	272	3055.05	3065.15	220.1
3	2	1	76462I703	712	12122	12124.75	3691.2
4	3	1	17943I703	535	1398	1398	900.1

micro_credit_loan

- Data Pre-processing Done

1. Acquire the dataset

   - We have received this dataset from FlipRobo Technologies which is related to a Micro Finance bank supporting Telecom sectors in Indonesia.
   - The sample data is provided to us from our client database. It is hereby given 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.

2. Import all the crucial libraries

   - For this project I have used the following major libraries like Pandas-profiling, Pandas, Matplotlib and Seaborn that are used for EDA or any other pre-processing done in this scenario.

In [16]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
A
from sklearn.tree import ExtraTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


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

3. Import the dataset

- The dataset is in CSV format and it is imported using Pandas library in Jupyter Notebook.

```
In [125]: pd. set_option("display.max_columns", None)
          df = pd.read_csv("D:/micro_credit_loan.csv")
          df.head()
```

Out[125]:

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90

- The statement **pd. set_option("display.max_columns", None)** simply allows us to physically see all the feature columns.
- By default, Jupyter Notebook doesn't display all the rows and columns at the same time and only selected portion from starting and ending of dataset are displayed.

4. Identifying and handling the missing values

- The column "Unnamed: 0" has no significance fo any kind
- It's just a serial number which is same as the index number and hence this column is dropped in the beginning itself
- column "msisdn" is the mobile number of the person and it doesn't really add any value in the analysis as well hence it's dropped.

```
In [126]: df.drop(columns=["Unnamed: 0", "msisdn"], inplace=True)
```

- The dataset appears to have a total of 209593 records (rows) and 35 features (columns) including 1 target column "label"

```
In [5]: df.shape
Out[5]: (209593, 35)
```

- I have not considered year to be split because this dataset by default is for the year 2016 and has no other years included.

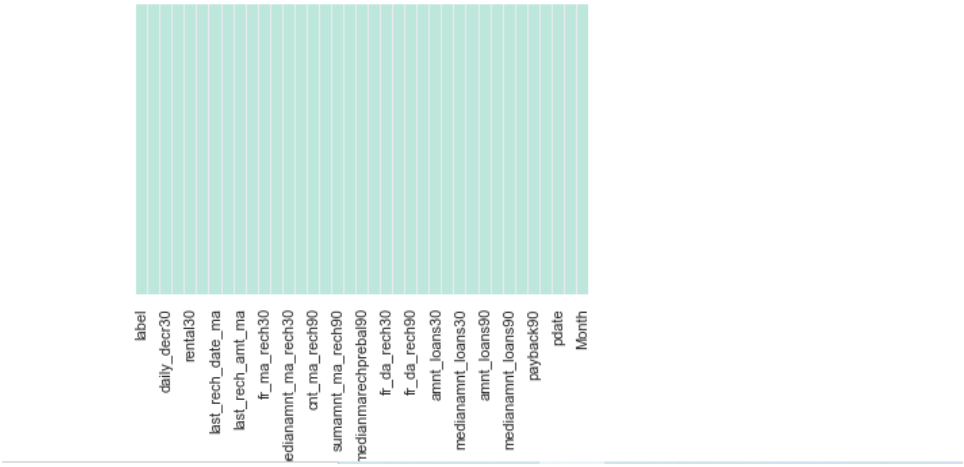
```
In [127]: df["Day"] = pd.DatetimeIndex(df['pdate']).day
          df["Month"] = pd.DatetimeIndex(df['pdate']).month
```

- The dataset appears to have all the information intact and as 209593 non-null out of 209593 rows.
- We have a total of 21 float64 features, 14 int64 type features and 2 object type features.

```
In [8]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 37 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   label                   209593 non-null  int64
 1   aon                     209593 non-null  float64
 2   daily_decr30            209593 non-null  float64
 3   daily_decr90            209593 non-null  float64
 4   rental30                209593 non-null  float64
 5   rental90                209593 non-null  float64
 6   last_rech_date_ma       209593 non-null  float64
 7   last_rech_date_da       209593 non-null  float64
 8   last_rech_amt_ma        209593 non-null  int64
 9   cnt_ma_rech30           209593 non-null  int64
 10  fr_ma_rech30            209593 non-null  float64
 11  sumamnt_ma_rech30       209593 non-null  float64
 12  medianamnt_ma_rech30    209593 non-null  float64
 13  medianmarechpreba130    209593 non-null  float64
 14  cnt_ma_rech90           209593 non-null  int64
 15  fr_ma_rech90            209593 non-null  int64
 16  sumamnt_ma_rech90       209593 non-null  int64
 17  medianamnt_ma_rech90    209593 non-null  float64
 18  medianmarechpreba190    209593 non-null  float64
 19  cnt_da_rech30           209593 non-null  float64
 20  fr_da_rech30            209593 non-null  float64
 21  cnt_da_rech90           209593 non-null  int64
 22  fr_da_rech90            209593 non-null  int64
 23  cnt_loans30             209593 non-null  int64
 24  amnt_loans30            209593 non-null  int64
 25  maxamnt_loans30         209593 non-null  float64
 26  medianamnt_loans30      209593 non-null  float64
 27  cnt_loans90             209593 non-null  float64
 28  amnt_loans90            209593 non-null  int64
 29  maxamnt_loans90         209593 non-null  int64
 30  medianamnt_loans90      209593 non-null  float64
 31  payback30               209593 non-null  float64
 32  payback90               209593 non-null  float64
 33  pcircle                 209593 non-null  object
 34  pdate                   209593 non-null  object
 35  Day                     209593 non-null  int64
 36  Month                   209593 non-null  int64
dtypes: float64(21), int64(14), object(2)
```

- We don't have any null / missing values in any columns

```
In [137]: sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap="icefire")
Out[137]: <AxesSubplot:>
```



- But if we observe, the std values are higher than the mean values for almost all the features and it's not correct.
- We can assume that some human error may have happened and due to this we surely will find a lot of outliers and skewness in dataset.
- Since it's a Bank dataset, there could be a possibility where certain number is actually genuine but all the features with similar issue is likely impossible just by coincidence and the data may not be accurate as seen.
- For Eg: if you consider column "aon" we have values from min increasing gradually with some gap which is similar as it approaches higher quartiles, but the max value for the same is suddenly jumped to 999860.75 which is not accurate.
- It's same for majority of features if not all and we have to reduce the outliers.

```
In [133]: df.describe()
Out[133]:
```

	label	aon	daily_decr30	daily_decr90	rental30
count	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000
mean	0.875177	8112.343445	5381.402289	6082.515068	2692.581910
std	0.330519	75696.082531	9220.623400	10918.812767	4308.586781
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000
25%	1.000000	246.000000	42.440000	42.692000	280.420000
50%	1.000000	527.000000	1469.175667	1500.000000	1083.570000
75%	1.000000	982.000000	7244.000000	7802.790000	3356.940000
max	1.000000	999860.755168	265926.000000	320630.000000	198926.110000

- By splitting the date column to "Day' and "Month" the results were effective. Hence this "pdate" column is dropped as it will be a redundant column.

```
In [70]: df.drop(columns="pdate", inplace=True)
```

- Correlation of the features with the dependent variable "label"
- We can observe that "cnt_ma_rech30", "cnt_ma_rech90", "sumamnt_ma_rech90", "sumamnt_ma_rech30", "cnt_loans90", "amnt_loans90", "amnt_loans30", "cnt_loans30", "daily_decr30", "daily_decr90", "Month", "medianamnt_ma_rech30", "last_rech_amt_ma", "payback30", "medianamnt_ma_rech90" and "payback90" seem to have a strong correlation with the output variable.

```
In [76]: corr_matrix = df.corr()
         corr_matrix["label"].sort_values(ascending=False)
```

```
Out[76]: label                    1.000000
         cnt_ma_rech30            0.241199
         cnt_ma_rech90            0.241018
         sumamnt_ma_rech90        0.212883
         sumamnt_ma_rech30        0.210898
         cnt_loans90              0.202929
         amnt_loans90             0.202883
         amnt_loans30             0.200838
         cnt_loans30              0.199318
         daily_decr30             0.170049
         daily_decr90             0.168126
         Month                    0.149214
         medianamnt_ma_rech30     0.146445
         last_rech_amt_ma         0.136284
         payback30                0.126034
         medianamnt_ma_rech90     0.125716
         payback90                0.124257
         fr_ma_rech90             0.088636
         rental90                 0.082368
         aon                      0.082206
         maxamnt_loans90          0.081955
         rental30                 0.062313
         medianamnt_loans30       0.046152
         medianmarechprebal90     0.045540
         medianamnt_loans90       0.037678
         Day                      0.008700
         cnt_da_rech90            0.005204
         last_rech_date_ma        0.004112
         cnt_da_rech30            0.003602
         maxamnt_loans30          0.002501
         last_rech_date_da        0.001226
         fr_ma_rech30             0.000778
         fr_da_rech30             0.000090
         fr_da_rech90            -0.004226
         medianmarechprebal30    -0.006196
         pcircle                       NaN
         Name: label, dtype: float64
```

5. Encoding the categorical data

- I have used LabelEncoder as the data is categorical and is not ordinal in nature.
- The column "pcircle" is converted into its corresponding numerical value.

```
In [71]: from sklearn.preprocessing import LabelEncoder

         encoder = LabelEncoder()

         # Encode the training dataset

         df.pcircle = encoder.fit_transform(df.pcircle)
```

```
In [72]: df.head(2)
```

Out[72]:

ans30	maxamnt_loans30	medianamnt_loans30	cnt_loans90	amnt_loans90	maxamnt_loans90	medianamnt_loans90	payback30	payback90	pcircle	Day	Month
12	6.0	0.0	2.0	12	6	0.0	29.0	29.0	0	20	7
12	12.0	0.0	1.0	12	12	0.0	0.0	0.0	0	10	8

6. Splitting the dataset

- Splitting up of dataset between x (features) and y (target column)

```
In [78]: x = df.drop(columns = ["label"], axis=1)
         y = df["label"]
```

- Split the dataset into train and test data set.
- I have chosen 200 random state and 30% of data is divided in text dataset.

```
In [7]: x_train, x_test, y_train, y_test = train_test_split(x_over, y_over, test_size=0.30, random_state = 200)
```

- We can see majority of features have skewness and since we could not remove outliers, let's try to reduce skewness to some extent if possible.

```
In [79]: x.skew()

Out[79]: aon                      0.950030
         daily_decr30             3.423044
         daily_decr90             3.720929
         rental30                 1.992012
         rental90                 2.025094
         last_rech_date_ma       14.800614
         last_rech_date_da       14.896702
         last_rech_amt_ma         3.094107
         cnt_ma_rech30            2.770319
         fr_ma_rech30            14.783584
         sumamnt_ma_rech30        3.932491
         medianamnt_ma_rech30     2.979389
         medianmarechprebal30    14.819168
         cnt_ma_rech90            2.877944
         fr_ma_rech90             2.272848
         sumamnt_ma_rech90        3.857468
         medianamnt_ma_rech90     3.220217
         medianmarechprebal90    45.407089
         cnt_da_rech30           17.821791
         fr_da_rech30            14.703786
         cnt_da_rech90           23.999877
         fr_da_rech90            29.598337
         cnt_loans30              2.679032
         amnt_loans30             2.863712
         maxamnt_loans30         54.211825
         medianamnt_loans30       4.565602
         cnt_loans90              2.917161
         amnt_loans90             3.035988
         maxamnt_loans90          1.763249
         medianamnt_loans90       4.905571
         payback30                2.730572
         payback90                2.626818
         pcircle                  0.000000
         Day                      0.192239
         Month                    0.389931
         dtype: float64
```

7. <u>Feature scaling</u>

- Finding variance inflation factor in each scaled column
- This gives us relationship between feature vs feature and we can drop if necessary to avoid multicollinearity.
- From the below observation, we can find many variables that have VIF and I have not considered dropping these columns as data is dynamic and every value seemed important.
- Also without dropping the features I got even better accuracy in the final model.

In [86]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor( x_scaled, i) for i in range(x_scaled.shape[1])]
vif["Features"] = x.columns
vif
```

Out[86]:

	vif	Features
0	1.024803	aon
1	32.208290	daily_decr30
2	35.408004	daily_decr90
3	12.031288	rental30
4	13.301839	rental90
5	1.000192	last_rech_date_ma
6	1.000189	last_rech_date_da
7	3.588004	last_rech_amt_ma
8	15.454352	cnt_ma_rech30
9	1.000184	fr_ma_rech30
10	12.944923	sumamnt_ma_rech30
11	5.188186	medianamnt_ma_rech30
12	1.000143	medianmarechprebal30
13	16.611628	cnt_ma_rech90
14	1.088977	fr_ma_rech90
15	15.269484	sumamnt_ma_rech90
16	5.758212	medianamnt_ma_rech90
17	1.060004	medianmarechprebal90
18	1.000240	cnt_da_rech30
19	1.000200	fr_da_rech30
20	1.189758	cnt_da_rech90
21	1.181037	fr_da_rech90
22	405.857508	cnt_loans30
23	484.422927	amnt_loans30
24	1.000195	maxamnt_loans30
25	6.255777	medianamnt_loans30
26	801.422284	cnt_loans90
27	920.127833	amnt_loans90
28	2.034355	maxamnt_loans90
29	6.294564	medianamnt_loans90
30	2.721532	payback30
31	2.798337	payback90
32	NaN	pcircle
33	1.143824	Day
34	2.028358	Month

- Storing the list of features having skewness in a variable "feat_skew". It's would be easier to pass it through at one go.

In [81]:
```
feat_skew = ['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']
```

- Let’s us now Scale the data for further processing.
- we have used StandardScaler for further scaling up of data

```
In [82]: from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()
         x_scaled = scaler.fit_transform(x)
         x_scaled
```

```
Out[82]: array([[-0.75903326, -0.23252738, -0.25895965, ...,  0.        ,
                  0.6637883 ,  0.31378569],
                [ 0.12340608,  0.82789936,  0.64375803, ...,  0.        ,
                 -0.5216368 ,  1.67606495],
                [-0.2315752 , -0.42632796, -0.425078  , ...,  0.        ,
                  0.54524579,  1.67606495],
                ...,
                [ 0.72707482,  0.79528193,  0.62179691, ...,  0.        ,
                  1.73067089,  0.31378569],
                [ 2.16906092,  0.87073166,  0.68855912, ...,  0.        ,
                  1.25650085,  0.31378569],
                [ 1.86622378, -0.06477717, -0.11251863, ...,  0.        ,
                 -0.87726433,  0.31378569]])
```

- Power Transformer (yeo-Johnson)
- To reduce the skewness to some extent, I have used a Power Transformer technique
- Since we have both positive and negative values in skewness, I have used "Yeo-Johnson" technique
- The data is further standardized.

```
In [83]: from sklearn.preprocessing import PowerTransformer

         scaler = PowerTransformer(method="yeo-johnson")

         x[feat_skew] = scaler.fit_transform(x[feat_skew].values)

         x[feat_skew]
```

Out[83]:

	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma
0	0.439640	0.411151	-0.651986	-0.645390	-0.023647
1	1.155676	1.097639	0.579587	0.330056	0.190154
2	0.107438	0.088636	-0.371183	-0.432930	0.001507
3	-1.038036	-1.032125	-0.680742	-0.683747	0.296844
4	-0.613436	-0.613646	-0.296141	-0.372423	0.022370
...	...	...	...	...	...
209588	-0.611273	-0.611526	-0.299761	-0.375340	-0.056022

- Data Inputs- Logic- Output Relationships

  - The dataset has predefined values in numerical format as majority of the dataset has both continuous and discrete data.
  - But the output variable “label” is in categorical format and hence this dataset is a classification problem.
  - The outliers are removed using **z-score** and only few features are selected to keep the threshold within 7 to 8% of data loss.
  - The following columns 'aon', 'rental30', 'rental90', 'maxamnt_loans30', 'cnt_loans90', 'payback30' and 'payback90' are considered for outlier removal.

In [74]:
```
from scipy.stats import zscore

z_score = zscore(df[['aon', 'rental30', 'rental90', 'maxamnt_loans30',
                     'cnt_loans90', 'payback30', 'payback90']])

abs_zscore = np.abs(z_score)

filtering_entry = (abs_zscore < 3).all(axis=1)

df = df[filtering_entry]
```

Out[74]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90
count	195842.000000	195842.000000	195842.000000	195842.000000	195842.000000	195842.000000
mean	0.874006	650.467525	5043.225193	5664.047640	2200.694444	2793.347325
std	0.331844	498.619115	8550.306012	10035.942886	2933.701720	3793.791613
min	0.000000	-48.000000	-46.215000	-46.215000	-8898.940000	-12912.400000
25%	1.000000	239.000000	39.600000	39.780000	256.410000	285.840000
50%	1.000000	515.000000	1243.587667	1269.595000	1004.940000	1232.010000
75%	1.000000	958.000000	6826.745167	7320.012500	3001.405000	3795.800000
max	1.000000	2440.000000	169237.902667	259525.000000	15617.510000	20794.550000

  - It is told that if the data loss is more than 7 to 8%, we may not get accurate results. In this case the data is reduced to approximately 6.5% after applying z-score.
  - If all columns were used for outlier removal there would have been approximately 22% of data loss which is very high. Also, by considering 6.5 %

data loss dataset, I have been able to achieve a better model in the later stage.

```
In [75]: # Percentage data loss:

         loss_percent = (209593-195842)/209593*100
         print(loss_percent)

         6.5608107140982765
```

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

  - There are no null values in the dataset.
  - There may be some customers with no loan history.
  - The dataset is imbalanced. Label '1' has approximately 87.5% records, while, label '0' has approximately 12.5% records.
  - There are multiple outliers in majority of the features which if removed in entirety we would lose data up to 22%. Keeping in mind that data is expensive and we cannot lose more than 7-8% of the data and models are built based on this scenario only.

- Hardware and Software Requirements and Tools Used

  Hardware / Software specifications

  Windows 10 64bit

  Anaconda 2021.05

  Python version – Python 3.9.5

  Jupyter Notebook 6.4 and Google Colab

  Pandas-profiling – package that performs simple EDAs for distribution of variables. This helped me give basic details about what the dataset consists of and its correlations with each other. The report is displayed using **.to_widgets()**

  Pandas – This is used in the data manipulation, processing and cleaning and also to get description, stats and almost everywhere in the project.

Matplotlib and **Seaborn** - Majority of the data visualizations are plotted using Seaborn and to some extent only Matplotlib is used here.

LabelEncoder - I have used this **Skipy** library to convert all the non-ordered categorical data into numerical data. In our case it's only one feature "pcircle".

train_test_split module from **sklearn.model_selection** to split the data into train and test and then used **StandardScaler** to bring the values to one level before imputing to model.

Warnings: I have used "ignore" to avoid the general errs that may occur and used "FutureWarning" to avoid errors that I got when running algorithms on Google Colab. To have a generic and efficient notebook file I used this as well.

SMOTE – I used this to check if there was an oversampling as the target column is imbalanced and this is will balance the classes of that target column.

Sciypy and **xgboost** - Used xgboost to import XGBClassifier and remaining algorithms including ensemble are part of Scipy.

Explained Dashboard – This library helps in the creation of dashboards automatically specially to help laymen understand how the model works. I have used this to explain my ML model in the end of this project.

Model/s Development and Evaluation

- Testing of Identified Approaches (Algorithms)

For the model building I have considered the following 6 algorithms for the training and testing.

- ExtraTreeClassifier
- DecisionTreeClassifier
- RandomForestClassifier
- ExtraTreesClassifier
- HistGradientBoostingClassifier
- XGBClassifier

- Run and Evaluate selected models

  - I have used a total of 6 machine learning algorithms to find the best and suited model which also includes ensemble algorithms.
  - I have used all 4 metrics i.e Accuracy, Precision, Recall and AUC-ROC for all the algorithms.
  - But we cannot simply rely on these scores as we cannot have any scope for assumption. Hence post this I have also performed Cross Validation for all these algorithms to find the estimated performance metric when it's actually used in production.

1) ExtraTreeClassifier

- From the below algorithm we can see the accuracy score for ExtraTreeClassifier is approximately **88.23%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

```
In [93]: from sklearn.tree import ExtraTreeClassifier

         ex_tree_class = ExtraTreeClassifier()
         ex_tree_class.fit(x_train,y_train)

         y_pred = ex_tree_class.predict(x_test)

         print(accuracy_score(y_test, y_pred))
         print(confusion_matrix(y_test, y_pred))
         print(classification_report(y_test, y_pred))
```

```
0.8823185752816428
[[45353  5911]
 [ 6175 45262]]
              precision    recall  f1-score   support

           0       0.88      0.88      0.88     51264
           1       0.88      0.88      0.88     51437

    accuracy                           0.88    102701
   macro avg       0.88      0.88      0.88    102701
weighted avg       0.88      0.88      0.88    102701
```

2) DecisionTreeClassifier

- From the below algorithm we can see the accuracy score for DecisionTreeClassifier is approximately **90.97%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

In [94]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)

y_pred = dt.predict(x_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9097574512419548
[[46789  4475]
 [ 4793 46644]]
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     51264
           1       0.91      0.91      0.91     51437

    accuracy                           0.91    102701
   macro avg       0.91      0.91      0.91    102701
weighted avg       0.91      0.91      0.91    102701
```

3) RandomForestClassifier

- From the below algorithm we can see the accuracy score for RandomForestClassifier is approximately **94.40%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

In [95]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(x_train,y_train)

y_pred = rf.predict(x_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9440317036835084
```

```
[[47776  3488]
 [ 2260 49177]]
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     51264
           1       0.93      0.96      0.94     51437

    accuracy                           0.94    102701
   macro avg       0.94      0.94      0.94    102701
weighted avg       0.94      0.94      0.94    102701
```

4) ExtraTreesClassifier

- From the below algorithm we can see the accuracy score for ExtraTreesClassifier is approximately **94.49%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

```
In [96]: from sklearn.ensemble import ExtraTreesClassifier

ex_reg = ExtraTreesClassifier()
ex_reg.fit(x_train,y_train)

y_pred = ex_reg.predict(x_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9449177710051508
[[48121  3143]
 [ 2514 48923]]
              precision    recall  f1-score   support

           0       0.95      0.94      0.94     51264
           1       0.94      0.95      0.95     51437

    accuracy                           0.94    102701
   macro avg       0.94      0.94      0.94    102701
weighted avg       0.94      0.94      0.94    102701
```

5) HistGradientBoostingClassifier

- From the below algorithm we can see the accuracy score for HistGradientBoostingClassifier is approximately **93.75%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

In [97]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier

hist_class = HistGradientBoostingClassifier()
hist_class.fit(x_train,y_train)

y_pred = hist_class.predict(x_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.93754685932951
[[47586  3678]
 [ 2736 48701]]
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     51264
           1       0.93      0.95      0.94     51437

    accuracy                           0.94    102701
   macro avg       0.94      0.94      0.94    102701
weighted avg       0.94      0.94      0.94    102701
```

6) XGBClassifier

- From the below algorithm we can see the accuracy score for XGBClassifier is approximately **94.42%** better.
- Precision, Recall and f1-score are also used as a part of confusion matrix.

In [98]:
```
from xgboost import XGBClassifier
xgb_reg = XGBClassifier(eval_metric='mlogloss')
xgb_reg.fit(x_train,y_train)

y_pred = xgb_reg.predict(x_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9442556547648027
[[47924  3340]
 [ 2385 49052]]
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     51264
           1       0.94      0.95      0.94     51437

    accuracy                           0.94    102701
   macro avg       0.94      0.94      0.94    102701
weighted avg       0.94      0.94      0.94    102701
```

- The below code shows us the cross validation performed over all the algorithms and I have used the CV values as 5.
- If you observe the below screenshot, we get the difference in the values.

```
In [99]: from sklearn.model_selection import cross_val_score

In [100]: scr = cross_val_score(ex_tree_class, x, y, cv=5)
          print("Cross Validation score of ExtraTreeClassifier model is:", scr.mean())

          Cross Validation score of ExtraTreeClassifier model is: 0.8747306514759989

In [101]: scr = cross_val_score(dt, x, y, cv=5)
          print("Cross Validation score of DecisionTree model is:", scr.mean())

          Cross Validation score of DecisionTree model is: 0.8846417084017689

In [102]: scr = cross_val_score(rf, x, y, cv=5)
          print("Cross Validation score of RandomForestClassifier model is:", scr.mean())

          Cross Validation score of RandomForestClassifier model is: 0.921564321152457

In [103]: scr = cross_val_score(ex_reg, x, y, cv=5)
          print("Cross Validation score of ExtraTreesClassifier model is:", scr.mean())

          Cross Validation score of ExtraTreesClassifier model is: 0.9191235827711225

In [104]: scr = cross_val_score(hist_class, x, y, cv=5)
          print("Cross Validation score of HistGradientBoostingClassifier model is:", scr.mean())

          Cross Validation score of HistGradientBoostingClassifier model is: 0.9236425370653762

In [105]: scr = cross_val_score(xgb_reg, x, y, cv=5)
          print("Cross Validation score of XGBClassifier model is:", scr.mean())

          Cross Validation score of XGBClassifier model is: 0.9235302022891585
```

- From the above algorithms ExtraTreeClassifier seems to be an ideal algorithm in this scenario and for this type of dataset.
- The difference between the accuracy score and cross validation for this model is very less compared to other models.

Sr.No.	Models used	Model Accuracy	Cross Validation	Difference output
1	ExtraTreeClassifier	0.882318575281642	0.874730651475998	0.00758792380564399
2	DecisionTreeClassifier	0.909757451241954	0.884641708401768	0.0251157428401859
3	RandomForestClassifier	0.944031703683508	0.921564321152457	0.022467382531051
4	ExtraTreesClassifier	0.94491777100515	0.919123582771122	0.0257941882340279
5	HistGradientBoostingClassifier	0.93754685932951	0.923642537065376	0.013904322264134
6	XGBClassifier	0.944255654764802	0.923530202289158	0.0207254524756439

- Let us try to tune the proposed model (ExtraTreeClassifier) to get better accuracy, if possible.
- The "parameters" have been selected from the skicit library and I have considered 7 parameters.

```
In [114]: parameters = {"criterion":["gini", "entropy"],
                        "splitter":["random", "best"],
                        "max_features":["auto", "sqrt", "log2"],
                        "random_state":[50, 70, 100, 120, 130],
                        "min_samples_split":[1, 2, 3, 4, 5],
                        "min_samples_leaf":[1, 2, 3, 4, 5],
                        "max_leaf_nodes":[1, 2, 3, 4, 5]
                        }
```

- RandomizedSearchCV is used to tune the parameters by fitting the same to the training dataset and used the best parameters after selection

```
In [115]: from sklearn.model_selection import RandomizedSearchCV
          RCV = RandomizedSearchCV(ExtraTreeClassifier(), parameters, cv=5, n_iter=10)

In [116]: RCV.fit(x_train, y_train)

Out[116]: RandomizedSearchCV(cv=5, estimator=ExtraTreeClassifier(),
                             param_distributions={'criterion': ['gini', 'entropy'],
                                                  'max_features': ['auto', 'sqrt',
                                                                   'log2'],
                                                  'max_leaf_nodes': [1, 2, 3, 4, 5],
                                                  'min_samples_leaf': [1, 2, 3, 4, 5],
                                                  'min_samples_split': [1, 2, 3, 4, 5],
                                                  'random_state': [50, 70, 100, 120, 130],
                                                  'splitter': ['random', 'best']})

In [117]: RCV.best_params_

Out[117]: {'splitter': 'best',
           'random_state': 120,
           'min_samples_split': 4,
           'min_samples_leaf': 3,
           'max_leaf_nodes': 4,
           'max_features': 'auto',
           'criterion': 'entropy'}
```

- Rebuild the model using the appropriate params we received from best_params_

```
In [33]: mod_ext_tree_class = ExtraTreeClassifier(splitter= "best", random_state= 120, min_samples_split= 4, min_samples_leaf= 3,
                                                 max_features= "auto", criterion= "entropy")

         mod_ext_tree_class.fit(x_train,y_train)
         pred = mod_ext_tree_class.predict(x_test)
         print(accuracy_score(y_test,pred)*100)

         90.07994079901852
```

- It's observed that the model accuracy was approximately **88.23%** earlier and post **Hyper Parameter tuning** its now approximately **90.28 %** better.

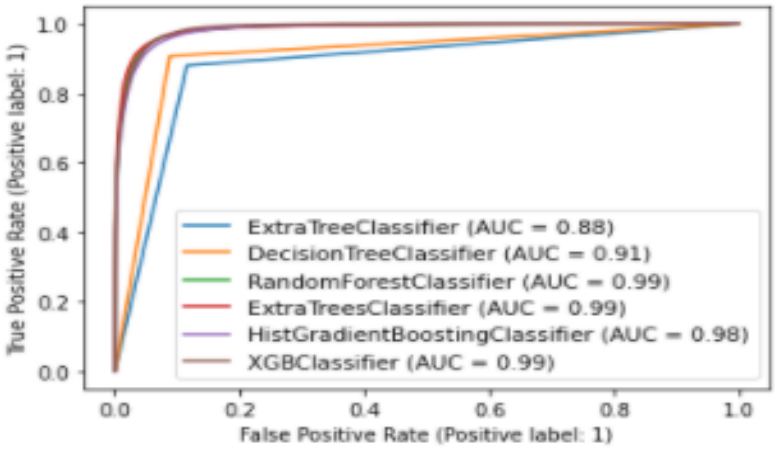
- Key Metrics for success in solving problem under consideration

  - Using sklearn.metrics I have used accuracy_score, confusion_matrix, classification_report to check for all possible.

  - AUC – ROC plots are also mapped for further clarification as below.

```
In [106]: from sklearn.metrics import roc_curve, roc_auc_score
          from sklearn.metrics import plot_roc_curve

In [107]: #ROC AUC score for ExtraTreeClassifier

          roc_auc_score(y_test, ex_tree_class.predict(x_test))
Out[107]: 0.8823225714940743

In [108]: #ROC AUC score for DecisionTreeClassifier

          roc_auc_score(y_test, dt.predict(x_test))
Out[108]: 0.9097624110240842

In [109]: #ROC AUC score for RandomForestClassifier

          roc_auc_score(y_test, rf.predict(x_test))
Out[109]: 0.9440114031595855

In [110]: #ROC AUC score for ExtraTreesClassifier

          roc_auc_score(y_test, ex_reg.predict(x_test))
Out[110]: 0.9449072978202587

In [111]: #ROC AUC score for HistGradientBoostingClassifier

          roc_auc_score(y_test, hist_class.predict(x_test))
Out[111]: 0.937531231389273

In [112]: #ROC AUC score for XGBClassifier

          roc_auc_score(y_test, xgb_reg.predict(x_test))
Out[112]: 0.9442398326345522
```

In [113]:
```
disp = plot_roc_curve(ex_tree_class, x_test, y_test)
plot_roc_curve(dt, x_test, y_test, ax=disp.ax_)
plot_roc_curve(rf, x_test, y_test, ax=disp.ax_)
plot_roc_curve(ex_reg, x_test, y_test, ax=disp.ax_)
plot_roc_curve(hist_class, x_test, y_test, ax=disp.ax_)
plot_roc_curve(xgb_reg, x_test, y_test, ax=disp.ax_)
plt.legend(prop={"size":11}, loc="lower right")
plt.show()
```



- I have used **SMOTE** technique for balance the target class so that we can have a better and unbiased accuracy.

In [90]:
```
#Handling class imbalance problem ny oversampling the moinority class

from imblearn.over_sampling import SMOTE
SM = SMOTE()
x_over, y_over = SM.fit_resample(x,y)
```

Balanced dataset after SMOTE ¶

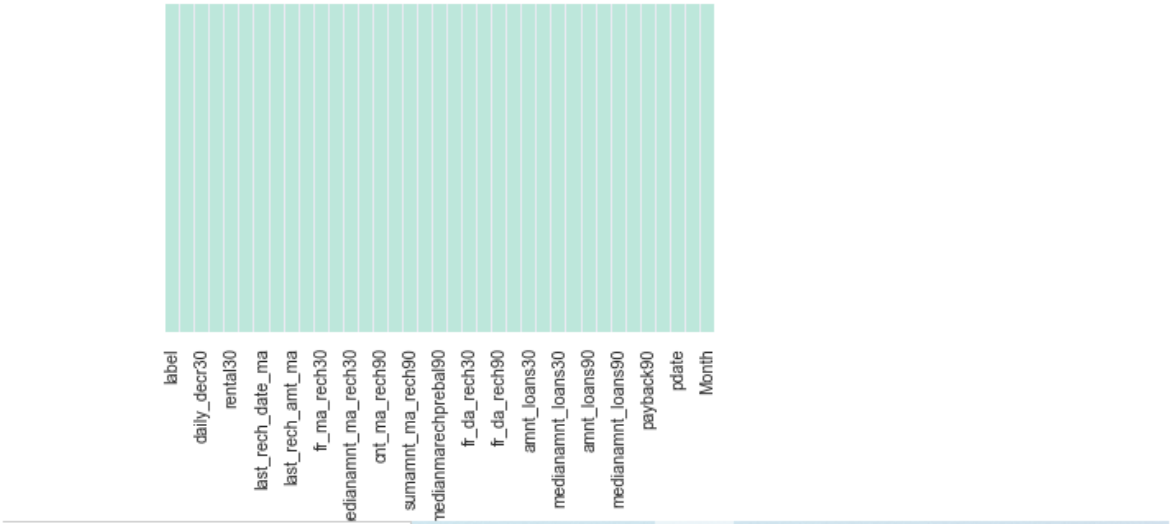
In [91]:
```
y_over.value_counts()
```

Out[91]:
```
0    171167
1    171167
Name: label, dtype: int64
```

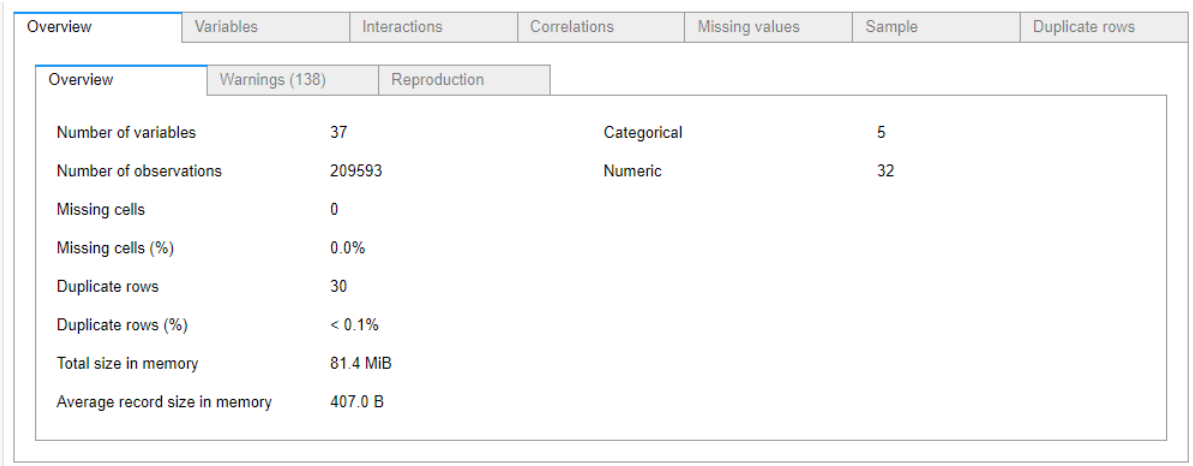
- Visualizations

- We don't have any null / missing values in any columns
- This plot is only to help you see graphically and it is not really required as it's already checked for earlier.

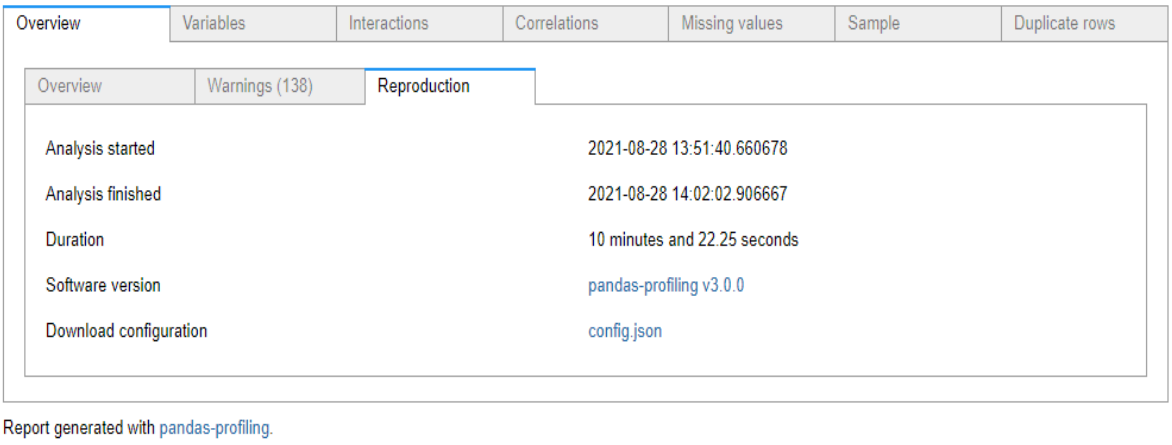
```
In [137]: sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap="icefire")
Out[137]: <AxesSubplot:>
```



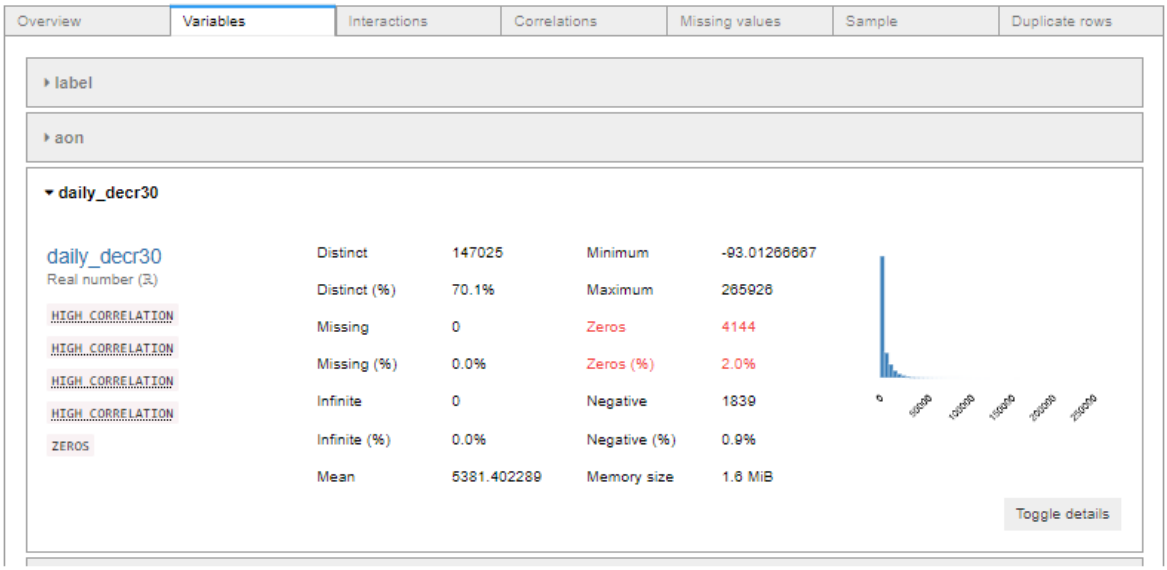
- Explore the dataset with Pandas Profiling
- I have attached all the screenshots for better understanding.
- The below screenshot given us overall understanding about the dataset.
- It gives us the type of features, total features, what features are categorical or numerical etc. Basically this window is a combination of all the options available on the plot.



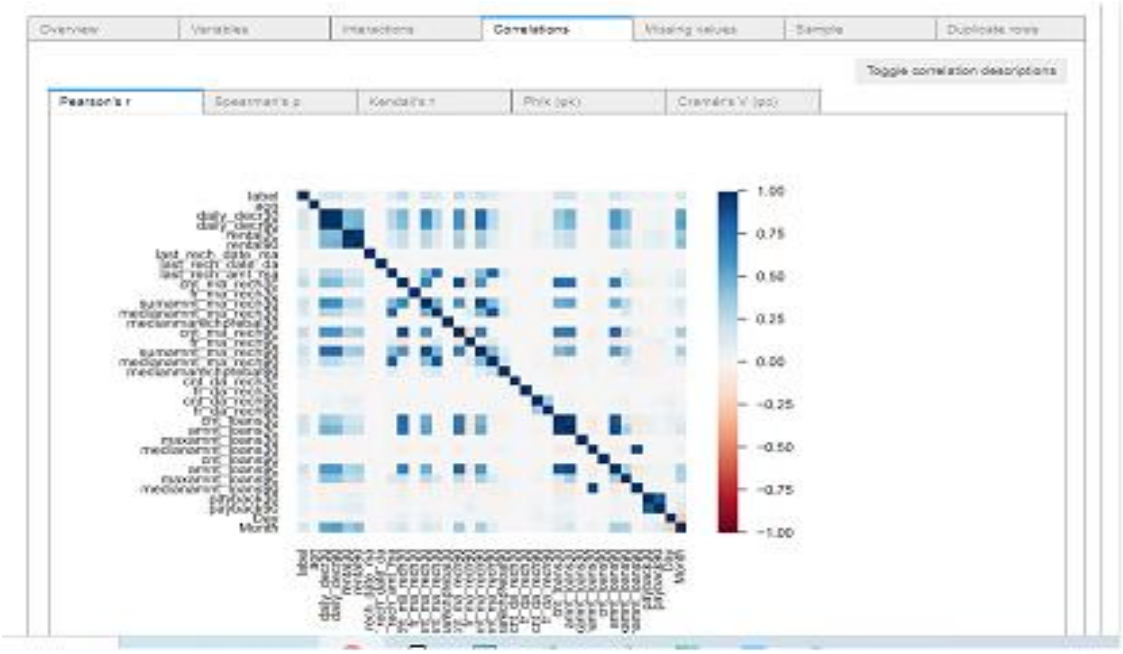
- If you select “Reproduction” option we can see relationships or information on time taken to run this code.
- In this case, it’s taken approximately 10 minutes to run this dataset under pandas profiling. It’s depends on the type of dataset, size of dataset and also on the system one uses to run these plots and models.



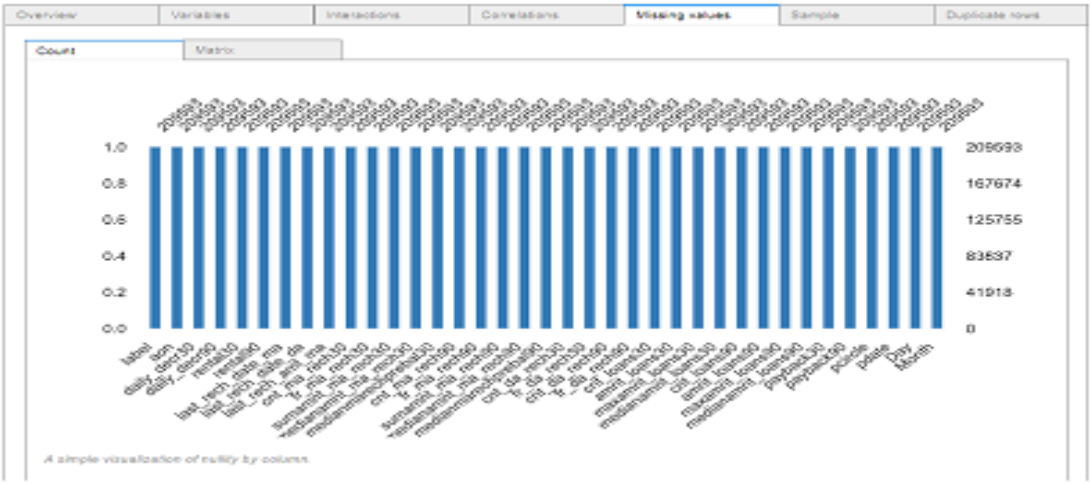
- If you select “Variables” option we can see relationships or information on individual value or feature.
- In the below screenshot we can see the information related to the feature “daily_decr30”.



- The following screenshot shows us the correlation of the dataset. Pandas profiling supports almost 4 to 5 correlation techniques and the following is Pearson correlation.

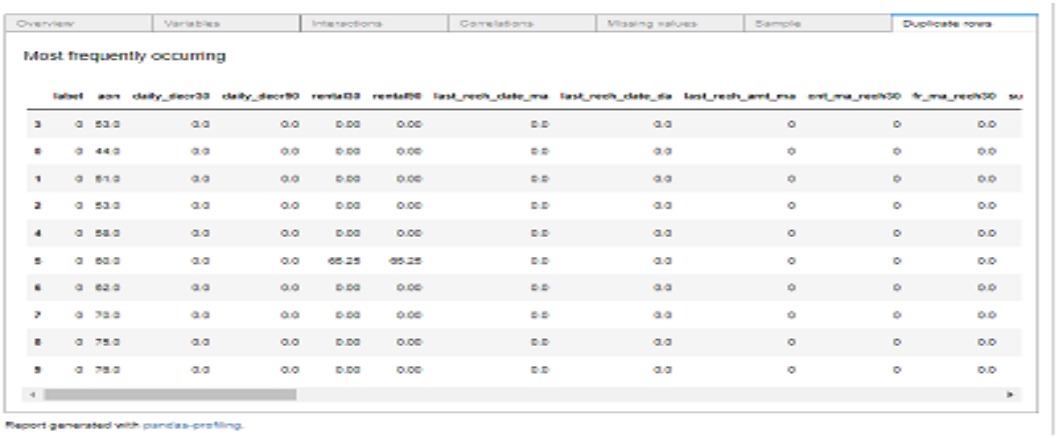


- The “missing values” option by name gives us details of any missing data for all the features of the dataset. In this case we do not have any missing values a shown below.



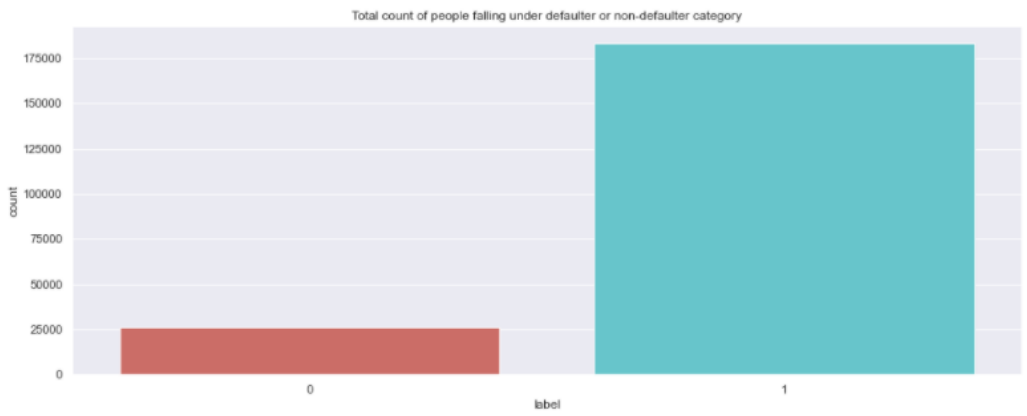
- The “Duplicate Values” option by name gives us details of any duplicate values in the dataset. It’s confusing to understand as the systems considers

any repeated value in a row as duplicate value. In real world datasets such data is likely to occur when a number, count etc could be same.



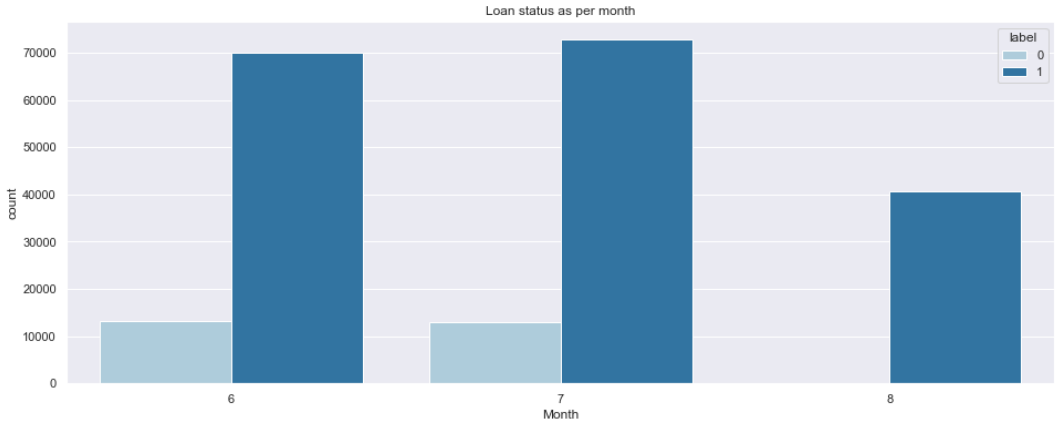
- Let's observe the relationship between the defaulter and non-defaulters.
- Customers that have paid back the loaned amount within 5 days of insurance of loan are "non - defaulters" and if they haven't then they are "defaulters",
- We can see that approximately 87.5% of customers have paid back and remaining 12.4% of customers are considered "defaulters",
- This also tells us the dataset is highly imbalanced and we may have to take necessary steps to overcome this.

```
In [138]: label_percent = (df["label"].value_counts()/df.shape[0])*100
          print(label_percent)

          1    87.517713
          0    12.482287
          Name: label, dtype: float64

In [14]:  plt.figure(figsize=(16, 6))
          sns.set_theme(style="darkgrid")
          ax = sns.countplot(x="label", data=df, palette="hls").set(title='Total count of people falling under defaulter or non-defaulter
```



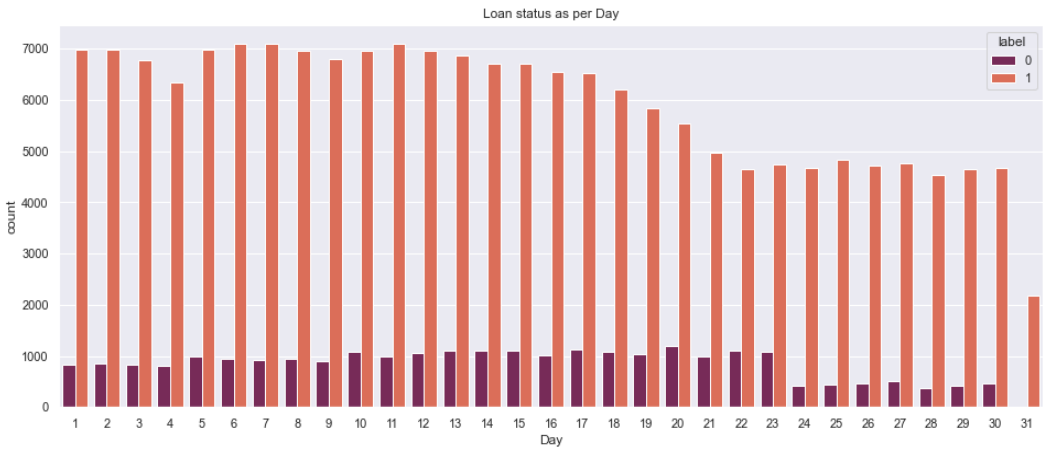
- From the below plot, we can see that we have data for the months June, July and August and in general the outstanding loan amount count is less compared to the ones who have paid.
- Also for the month of August we can see there is no defaulted amount but also the total loan received seems to be less in count.
- Either the data is as is or data was only for half of August month.

```
In [59]: plt.figure(figsize=(16, 6))
         sns.set_theme(style="darkgrid")
         ax = sns.countplot(x="Month", data=df, palette="Paired", hue="label").set(title='Loan status as per month')
```



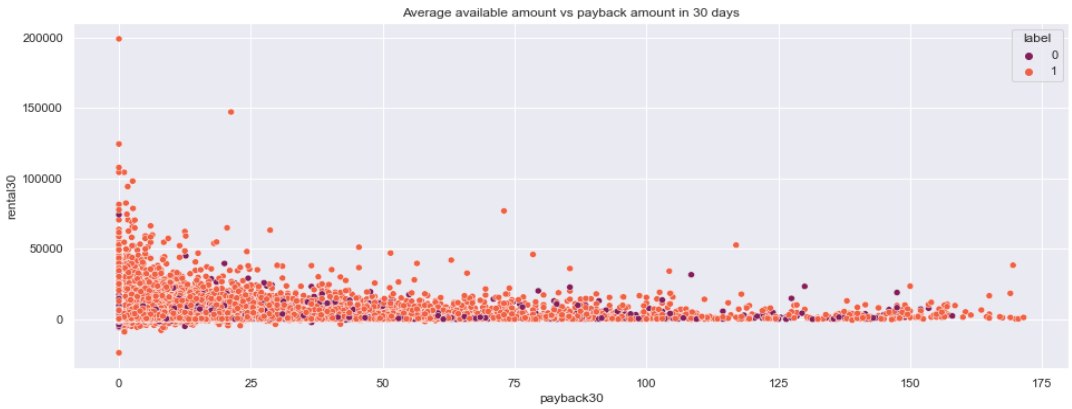
- From the below plot we can see the count of days and status of loan amount.
- We can observe that from day 1 to 17 the payment of loan seems consistent and after 17th the inflow of loan amount seems to have reduced and is on same scale. Maybe customers are likely to pay in the first 2 weeks and as the month end approaches the pattern seems to change.
- We can observe in this plot for 31st day we have less count of inflow. This shows that majority of customers pay within the month except for few customers.

```
In [120]: plt.figure(figsize=(16, 6))
          sns.set_theme(style="darkgrid")
          ax = sns.countplot(x="Day", data=df, palette="rocket", hue="label").set(title='Loan status as per Day')
```



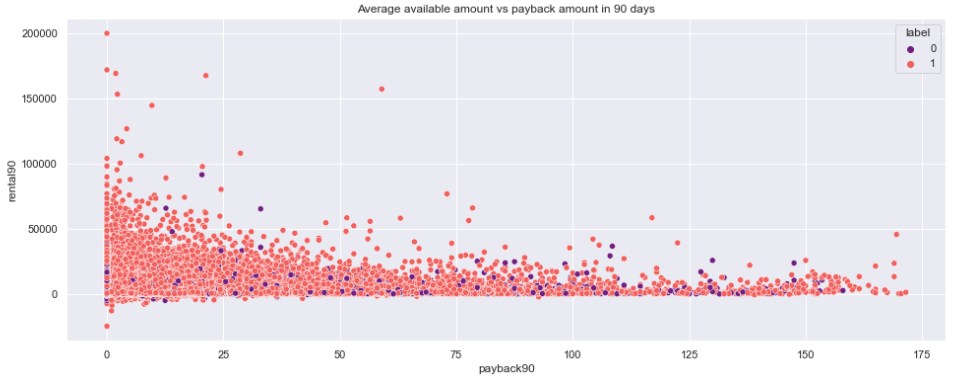
- Lets now observe the pattern of payback of loan amount in 30 days
- We previously observed that majority of people pay within month and some of them even on the last day of the month. From the below plot we can observe that as and when their payback amount is increased, their average account balance also gets decreased except for a few observations where they appear to be outliers.
- Since the average balance amount itself is less, repayment becomes a chaos but still very few customers have ended up defaulting and majority of them have already paid the loan amount.

```
In [121]: plt.figure(figsize=(16, 6))
          sns.set_theme(style="darkgrid")
          ax = sns.scatterplot(x="payback30", y="rental30", data=df, hue="label", palette="rocket").set(title='Average available amount vs
```



- Let's see if the same pattern applies in the case of 90 days payback period.
- From the plot, the distribution of amount seems as same as amount taken in 30 days but majority of customers seem to fall in category between 0 to 50000 Rupiah at least. Also the outlier amount appears in the initial stages compared to later stages. Maybe customers start filing in their account initially through salary etc hoping to return the amount as much as possible and also trying to keep the account balance above average.
- Also just like in payback 30, here also customers prefer paying as early as possible and delay in payment is as same as in 30 days.

```
In [122]: plt.figure(figsize=(16, 6))
          sns.set_theme(style="darkgrid")
          ax = sns.scatterplot(x="payback90", y="rental90", data=df, hue="label", palette="magma").set(title='Average available amount vs p
```

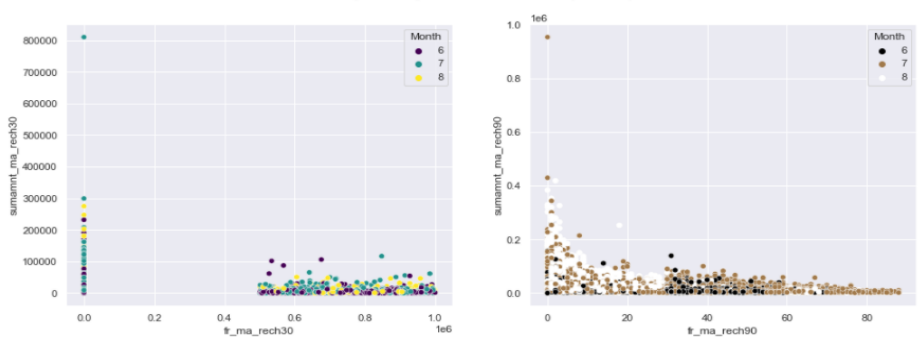


- The following subplot gives us decrease in amount for both 30 and 90 days and how much of average amount balance is left post that.
- We can see that although defaulters are quite less as per plot, in case of 90 days we can find more defaulters compared to 30 days plot.
- Here also we can see outliers more in case of 90 days plot. But they gradually decrease as the account balance is reduced. But in plot 1, outliers are scattered throughout.

```
In [47]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

         fig.suptitle('Average available amount vs amount spent in both 30 and  90 days')

         sns.scatterplot(ax=axes[0], x="daily_decr30", y="rental30", data=df, hue="label", palette="viridis")
         sns.scatterplot(ax=axes[1], x="daily_decr90", y="rental90", data=df, hue="label", palette="magma")

         plt.show()
```

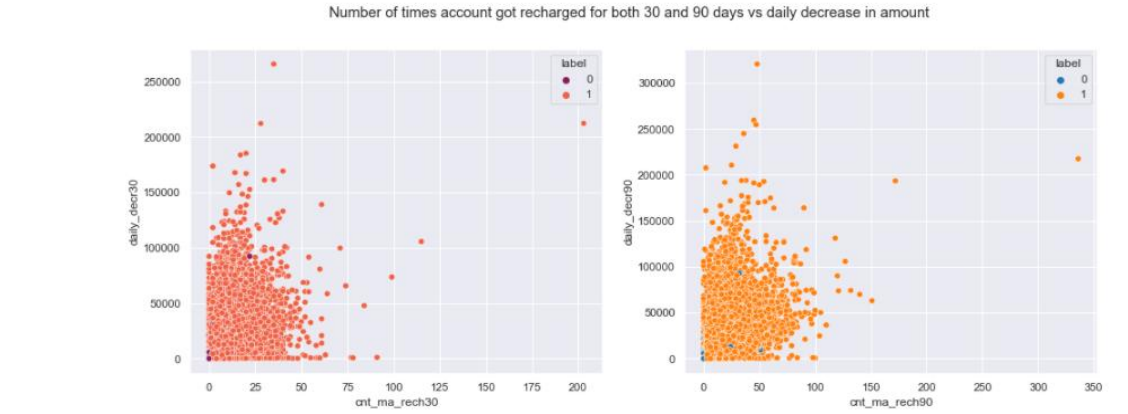
- From the below plot we can see how frequent customers recharge / or add amount to the account. From 1st plot we can see those customers either recharge in the 2nd month i.e., July compared to June or August. Also, the frequency of recharge seems to reduce in the last month i.e., August. We can also see there are a few customers right in the beginning who may not have recharged at all.
- From the 2nd plot we can see a person taking 90 days loan, then July and August months seems to have high frequency of recharges done and customers that are paying even at the end of July month have account balance managed till the month end.

```
In [123]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

          fig.suptitle('Number of times account got recharged for both 30 and 90 days vs daily decrease in amount')

          sns.scatterplot(ax=axes[0], x="fr_ma_rech30", y="sumamnt_ma_rech30", data=df, hue="Month", palette="viridis")
          sns.scatterplot(ax=axes[1], x="fr_ma_rech90", y="sumamnt_ma_rech90", data=df, hue="Month", palette="cubehelix")

          plt.show()
```



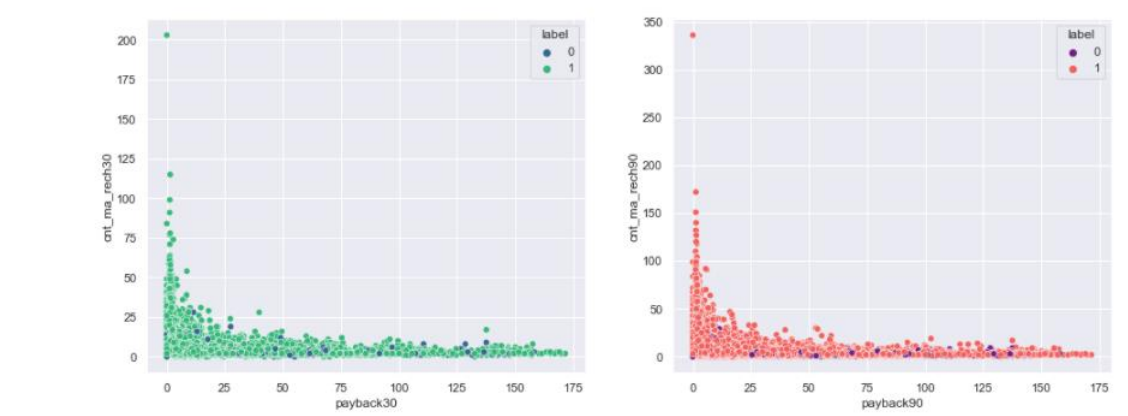
- From the below plot the daily decrease of account balance is in relation with number of times the account got recharged
- We can see that the more the number of times account got recharged, the less amount started to decrease. This makes sense and it's similar to both 30 days and 90 days.
- We can see the majority of outliers lie in the range 1,50,000 to 2,70,000 Rupiah and defaulters' counts are also minimum is both cases.

```
In [42]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

fig.suptitle('Number of times account got recharged for both 30 and 90 days vs daily decrease in amount')

sns.scatterplot(ax=axes[0], x="cnt_ma_rech30", y="daily_decr30", data=df, hue="label", palette="rocket")
sns.scatterplot(ax=axes[1], x="cnt_ma_rech90", y="daily_decr90", data=df, hue="label", palette="tab10")

plt.show()
```



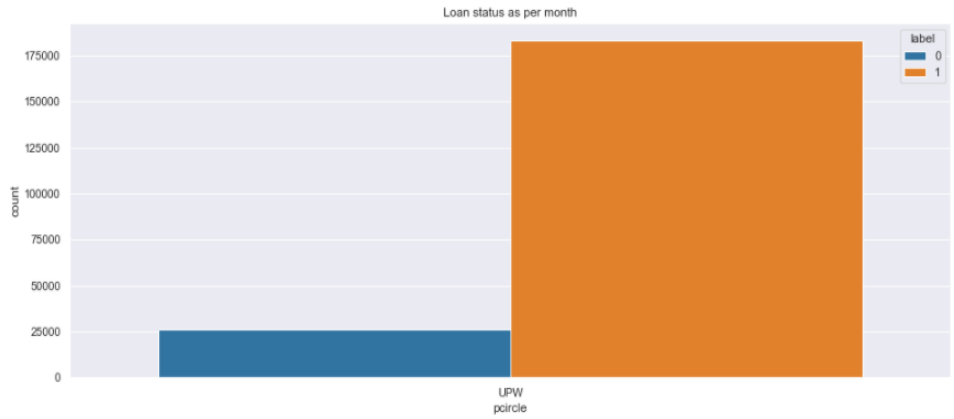
- The below plot shows us the pattern of payback for both 30 days and 90 days with regards to recharge of amounts.
- We can see from subplots for both the plots it's consistent. If loan amount / recharge is taken then it's paid immediately in both the cases and few numbers of customers seem to fall under defaulter category.

```
In [53]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

fig.suptitle('Payback time of loans vs median amout available in both 30 and  90 days')

sns.scatterplot(ax=axes[0], x="payback30", y="cnt_ma_rech30", data=df, hue="label", palette="viridis")
sns.scatterplot(ax=axes[1], x="payback90", y="cnt_ma_rech90", data=df, hue="label", palette="magma")

plt.show()
```



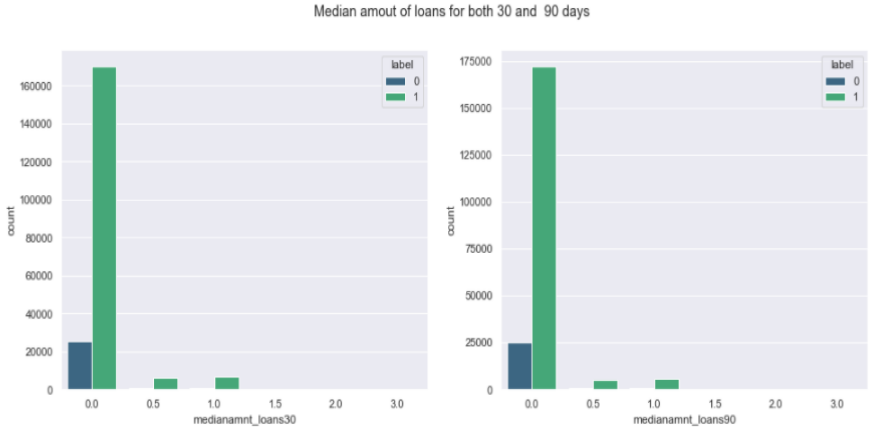
- The following plot shows us the telecom circle that is associated with the MFI institutions. In this case its only UPW which is one of the telecom providers of Indonesia.

```
In [96]: plt.figure(figsize=(16, 6))
         sns.set_theme(style="darkgrid")
         ax = sns.countplot(x="pcircle", data=df, palette="tab10", hue="label").set(title='Loan status as per month')
```



- From the below plot we can see there is no much difference in the median amount of loan taken in both 30 and 90 days and with minor variations with respect to default loans.

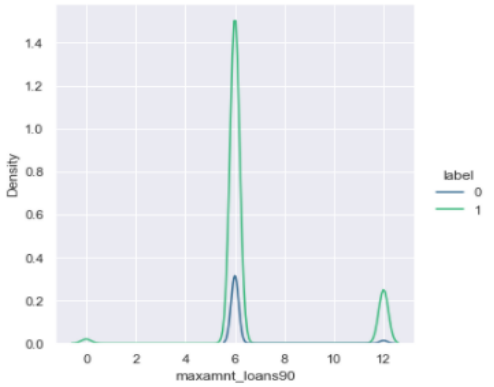
```
In [124]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

          fig.suptitle('Median amout of loans for both 30 and  90 days')

          sns.countplot(ax=axes[0], x="medianamnt_loans30", data=df, hue="label", palette="viridis")
          sns.countplot(ax=axes[1], x="medianamnt_loans90",data=df, hue="label", palette="viridis")

          plt.show()
```



- From the following plot we can observe the maximum amount returned is 6 Rupiah and 12 Rupiah.
- This shows that this amount was paid as a fine for defaulting the payment by long time or simply because they delayed it by 5 days.
- Since the bulge is high at Rupiah 6 because the amount recharged was Rupiah 5 compared to Rupiah 10. Not many people prefer getting recharge of more amount hence we can find customers with Rupiah 5.

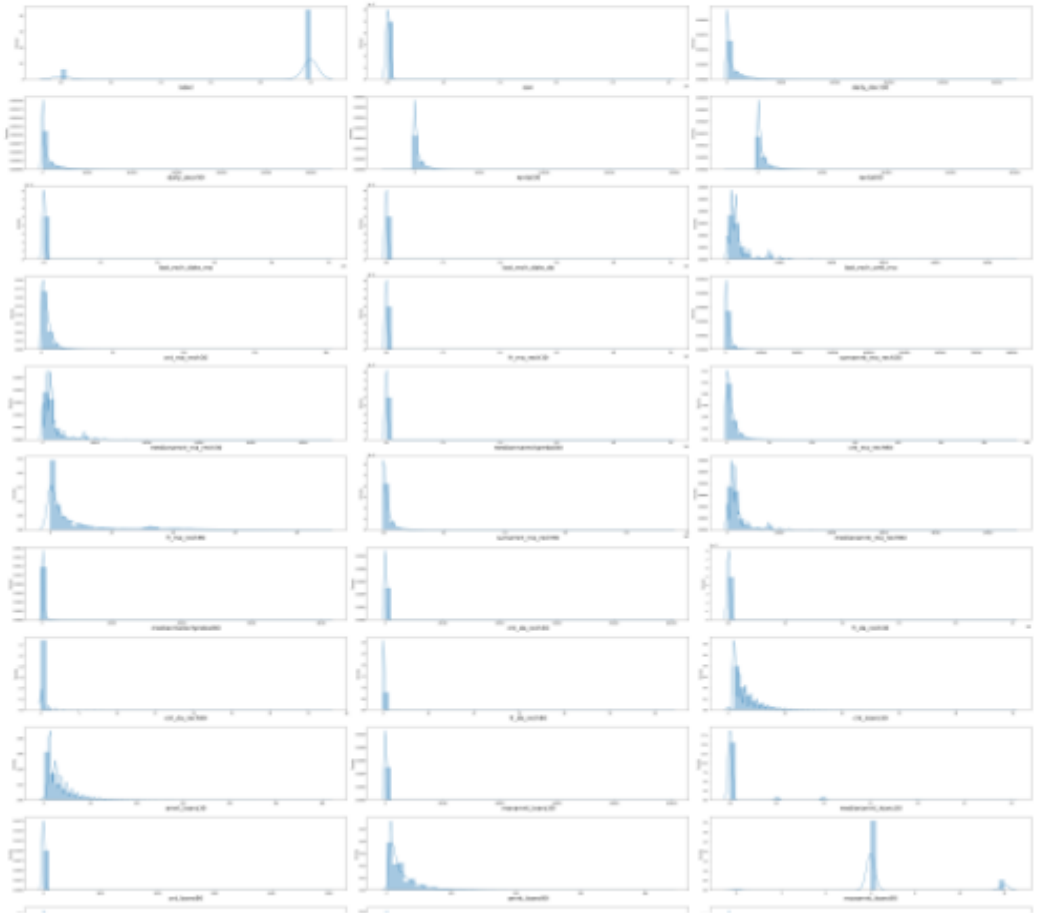
In [119]:
```
plt.figure(figsize=(16, 6))
sns.set_theme(style="darkgrid")
sns.displot(x="maxamnt_loans90", data=df, hue="label", palette="viridis", kind="kde")

plt.show()
```

```
<Figure size 1152x432 with 0 Axes>
```

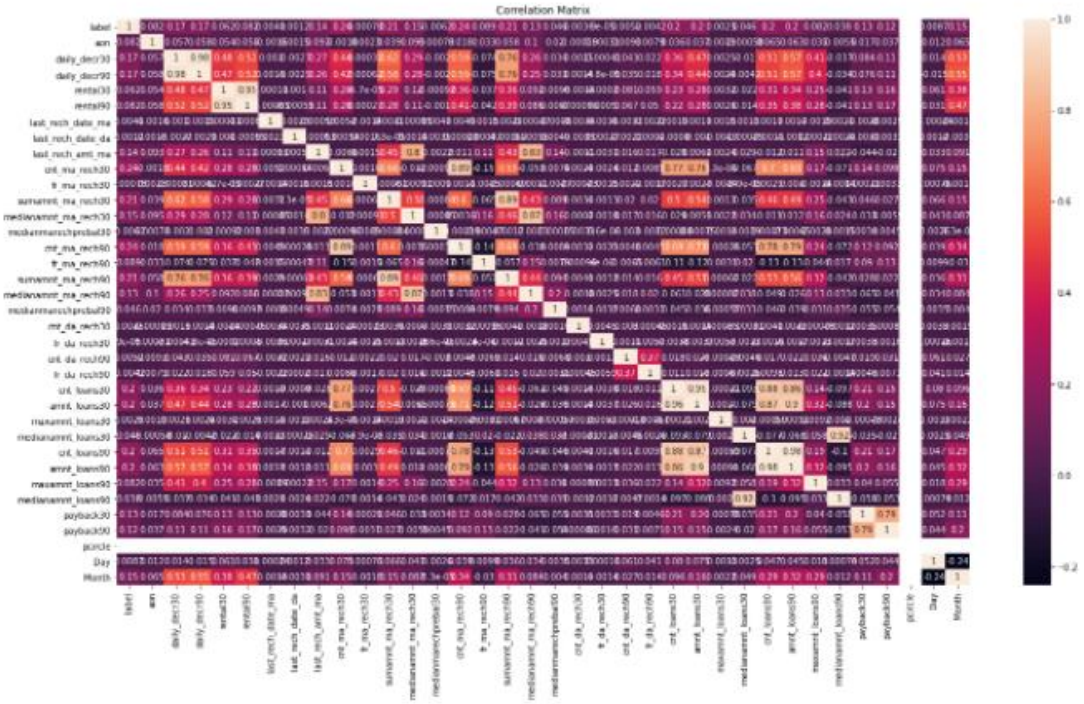


- Majority if features are not normally distributed. Since we have multiple features, let's try to identify the distribution

In [73]:
```
# Let us now see the distribution of the dataset we have

plt.figure(figsize=(50,55), facecolor="white")
plotnumber = 1

for column in df:
    if plotnumber <=36:
        ax = plt.subplot(12,3, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column, fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

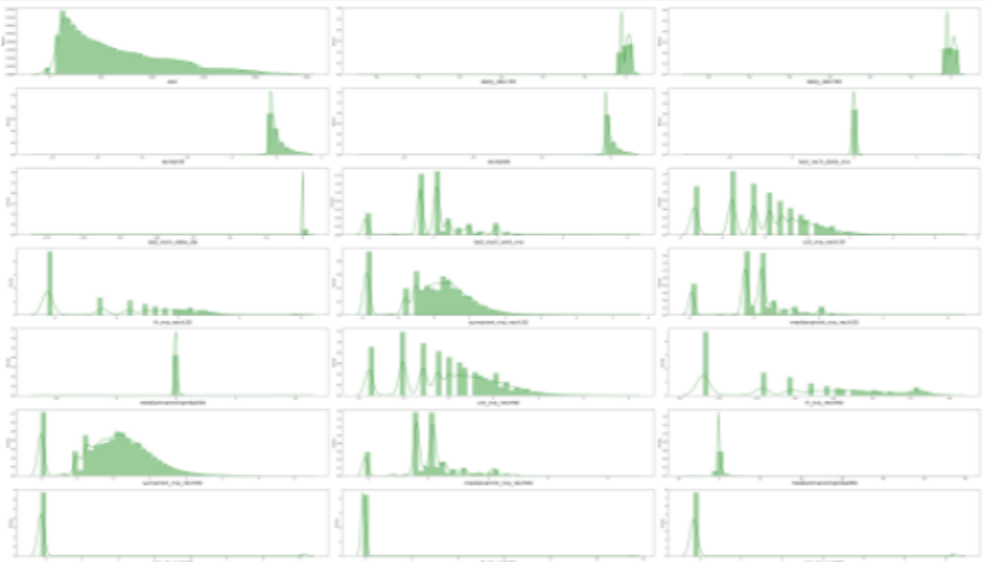


- Let’s us now examine correlation using a "heatmap" for further clarification
- Since there are multiple features, going through this plot may seem difficult.

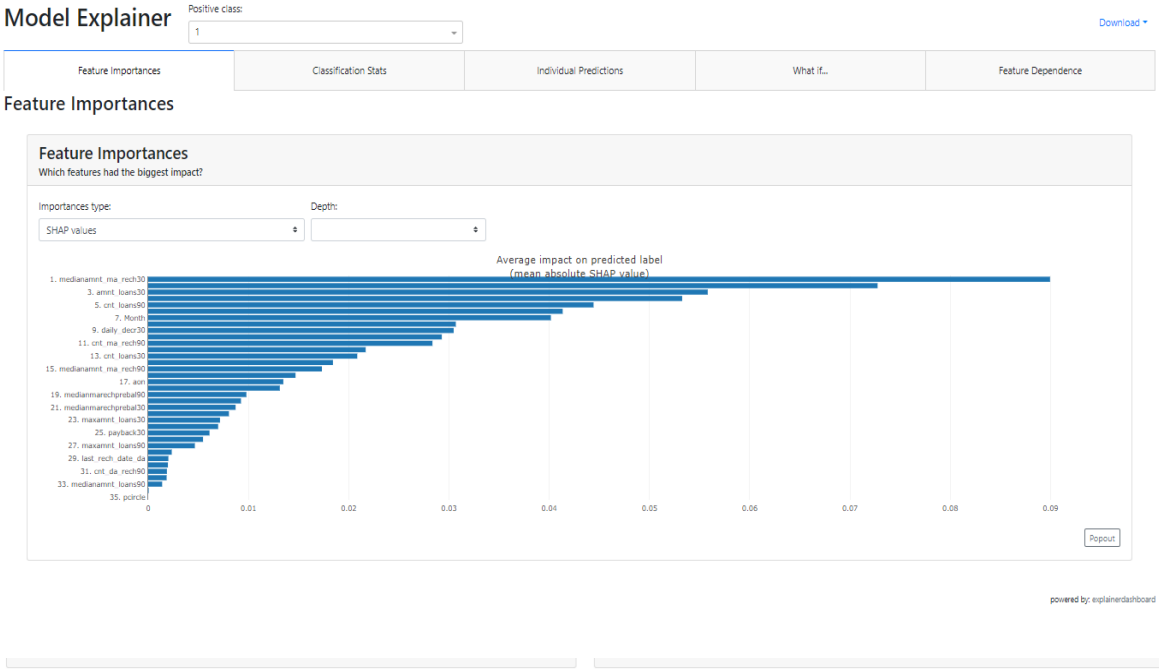


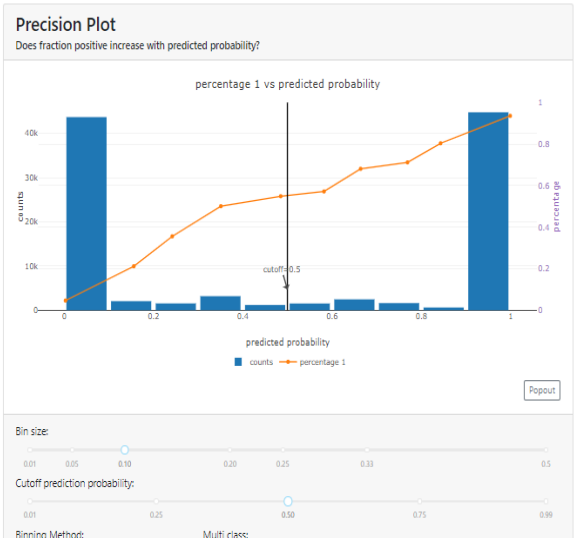
- Distribution after skewness is removed to some extent.

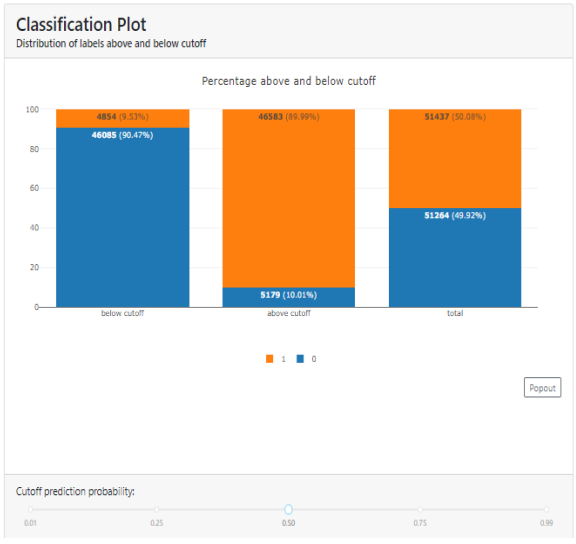
```
In [85]: # Let us now see the distribution of the dataset we have

plt.figure(figsize=(50,55), facecolor="white")
plotnumber = 1

for column in x:
    if plotnumber <=36:
        ax = plt.subplot(12,3, plotnumber)
        sns.distplot(x[column], color="green")
        plt.xlabel(column, fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

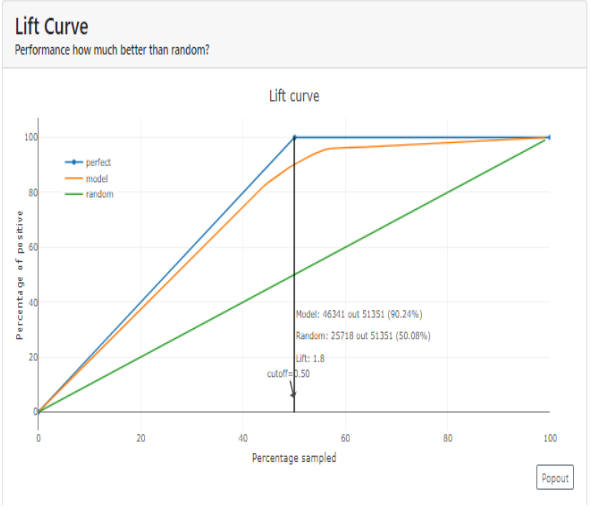


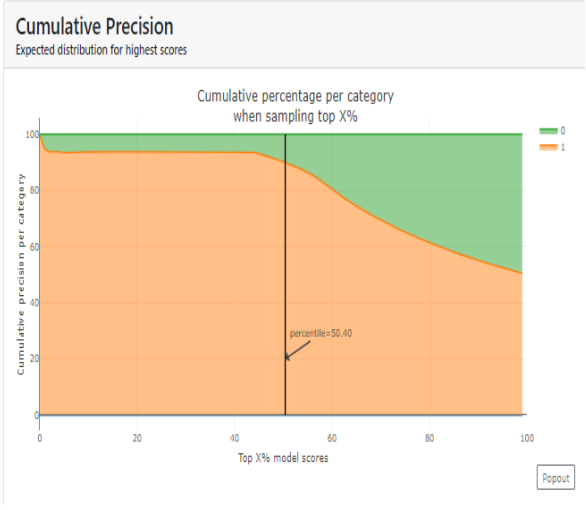
Model Dashboard











Model Explainer

Positive class: 1

Download

Feature Importances | Classification Stats | Individual Predictions | What if... | Feature Dependence

Select Index

Select from list or pick at random

240143

Random Index

Observed label: 0 1

Range: probability

probability
percentile

Predicted probability range:

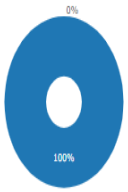
0.2 0.4 0.6 0.8

Prediction

Index: 240143

label	probability
0*	100.0 %
1	0.0 %

* indicates observed label



Feature Input

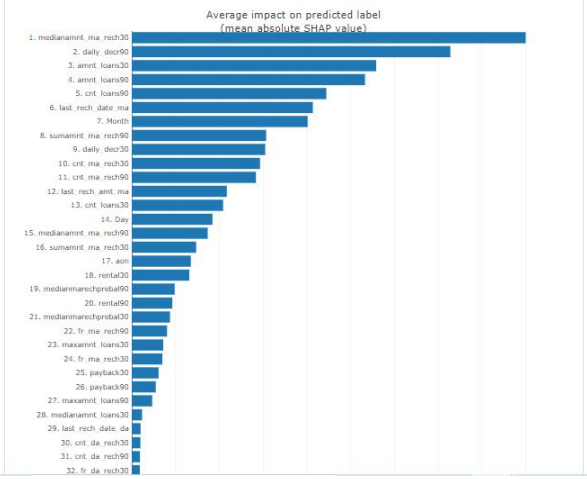
Adjust the feature values to change the prediction

medianamnt_ma_rech30	daily_decr90	amnt_loans30	amnt_loans90
0.11034579328947902	-0.19453483553700135	-0.8923399988741779	-0.4064421280210381
Range: -1.94-5.19	Range: -55.29-3.39	Range: -3.43-3.8	Range: -3.7-3.33
cnt_loans90	last_rech_date_ma	Month	sumamnt_ma_rech90
-0.36593124585819825	-0.013374707106039422	6	-0.80349517368398
Range: -3.01-2.59	Range: -13.44-8.69	Range: 6-8	Range: -2.0-5.22
daily_decr30	cnt_ma_rech30	cnt_ma_rech90	last_rech_amt_ma
-0.18082671287601185	-0.7833645638579282	-1.0113051883782407	-0.016764859330154078
Range: -53.91-3.15	Range: -1.67-4.26	Range: -1.77-3.99	Range: -2.09-6.12
cnt_loans30	Day	medianamnt_ma_rech90	sumamnt_ma_rech30
-0.9103418225186835	9	0.038573482598379505	-0.5817590237928582
Range: -3.08-2.83	Range: 1-31	Range: -2.11-6.22	Range: -1.88-4.47
aon	rental30	medianmarechprebal90	rental90
1329.5535440467163	1.3025959073714692	-0.16633112802229 77	1.117432010497922
Range: -48.0-7417.0	Range: -73.38-3.91	Range: -9.57-59.78	Range: -20.83-4.03

Shap Summary

Ordering features by shap value

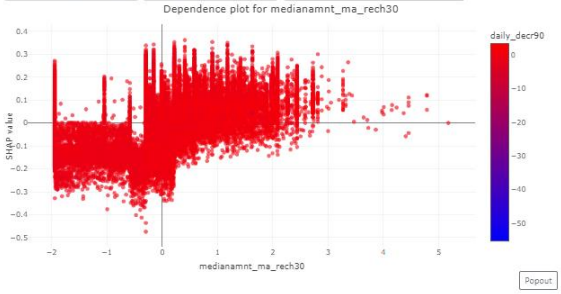
Depth: Summary Type: Aggregate



Shap Dependence

Relationship between feature value and SHAP value

Feature: medianamnt_ma_rech30 Color feature: daily_decr90 Index: Select...



Popout

CONCLUSION

- Key Findings and Conclusions of the Study

    - From the EDA of the Micro-Finance-Dataset, we can see that there are numerous Micro Finances that have partnered with small scale industries including Telecom operators.
    - From one of the article's, I got to know that in Indonesia volume of microcredit disbursed by the country's largest state-owned bank rose to 42.48 trillion IDR as of the end of December 2015, up 22.9% over December 2014.
    - The Financial Services Authority (OJK) additionally cited that Indonesian bank in fashionable have complied with the OJK guidelines that mandate them to allocate 10% in their mortgage portfolio to small and medium enterprises (SMEs) in 2016 with a purpose to be multiplied to 20% in 2018.
    - So, I think many financial institutions even though are approximately under outstanding loans up to $70 Billion, they are obliged to invest in these sectors if they themselves are to thrive in the market.

- Interpretation of the Results

    - It's the first time I got to work on a very large dataset having more than 200,000 records.
    - We have come across multiple and very high outliers. This could be an outcome of misinterpretation by people handling the data as majority of features if not all had similar issues.
    - Although if all the outliers were to be treated, we would lose 22% of data. I think it would have been an optimal model rather than restricting to only 7 to 8% data loss.
    - Through visualizations I got to see how 2 or more individual features are related to each other and how it impacts one another.

- I came across only one categorical feature that was not required to be in any order hence used LabelEncoder to convert.
- For me challenge was to try to understand in detail about Micro Finance of Indonesian geography as a whole.
- Majority of articles I tried referring to did not really help me get complete picture of the dataset.

- Learning Outcomes of the Study in respect of Data Science

  - With the help of this dataset, I was able to work on multiple classification algorithms and also got to work on large real-world dataset.
  - I also got some understating on how a company would provide datasets and not necessarily datasets are perfect all the time.
  - I realised the importance of working on datasets that have too many outliers and we have to take a decision whether we need to remove those completely or not.
  - I built multiple models several times where in one scenario I have considered model with 22% data loss, in one scenario I have considered model by dropping columns based on Variation Inflation Factors.
  - At one point I got into confusion if I need to consider only continuous or also discrete values for outlier and skewness removal and upon confirmation and doubt clearance from my SME I got better understanding about this dataset.
  - But the final model was considered that gave better accuracy after hyper tuning and model with 7 to 8% data loss is an ideal model.
  - The dashboard that I built using ExplainerDashbaord took me approximately 23 hours. This could be because the dataset is very large and also because I use older system and processing power is very less for such applications.

- Limitations of this work and Scope for Future Work

    - This dataset has multiple outliers and since based on management's advice only up to certain percent of outliers are removed, I am not able to predict how this model would give output on different type of similar datasets.
    - Also this dataset contains information only for 3 months of the year 2016 and I believe having at least an annual information can help in formulating even better approaches.
    - As per one of the articles I referred to, the government is investing heavily in Spectrum Holding, IoT Markets and Digital Infrastructure (Fibre, Telecom Towers, Data Centres, Submarine Cables) are being innovated in Indonesia as part of 2021 to 2026 (5-year plan project).
    - Due to this there are multiple Telecom operators offering similar or better financial assistance due to the availability of 5G, M&A and e-Commerce.
    - Since this dataset is of the year 2016, I think due to lot of changes there could be some or major impacts in how prediction would work.