University of Barishal

Micro Credit Defaulter

Submitted by:
MD RASHIDUNNABI

Problem statement :

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).

Methods and Ideas

Since the target variable is "label" ,this problem can be considered as a classification problem under RandomForestClassifier() algorithm because it gives the best accuracy.
We realized that the target variable is highly unbalanced in the data. 87.5% of the target variable are 1 (Default). As we've learned, these classifiers will be influenced by highly unbalanced data and be more likely to fail to classify the minority label in the test set. However, in the real-life scenario, these default loan (labeled as 0) will be more harmful to the financial institution. So, we decided to use SMOTE (Synthetic Minority Over-Sampling Technique) to over-sample the minority group in the data and make both labels occupied 50% of the training set. We will evaluate the performance of the classifiers trained with unbalanced data and balanced data.Loan Default Prediction Using RandomForestClassifier. We also conducted another experiment with the random forest classifier. Since some hyper-parameter influences the performance of the classifier, we made some changes on these hyper-parameters and evaluate the performance of these classifiers.

Exploratory Analysis :

Importing All necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading the data :

```
df=pd.read_csv('Data file.csv')
df.head()
```

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	medianamnt_loans30	cnt_loans90	amnt_loans90	maxamnt_loans90	medianam
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	...	6.0	0.0	2.0	12	6	
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	...	12.0	0.0	1.0	12	12	
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	...	6.0	0.0	1.0	6	6	
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	...	6.0	0.0	2.0	12	6	
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	...	6.0	0.0	7.0	42	6	

5 rows × 37 columns

"read_csv" is an important function of pandas which allows to read csv files and we can make various operations on the dataset. As my file is a CSV file that's why I have used "read_csv" function to load the data from the specific directory.The name of the dataset id df.

Description of Data :

```
df.info()
```

```
df.info()
```

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

Normaly to explore the data we can use various functions such as shape, columns, dtypes, info(), head(), tail(), describe(). Here, I have used df.info()
By using info() we can get a concise summary of a DataFrame. It includes the index dtype and column dtypes, non-null values and memory usage.
In our dataset we can see that there are 34 Numeric columns and three object type column.
Observations:

Numeric features
Numeric features = [`Unnamed: 0,label,aon,daily_decr30,daily_decr90,rental30, rental90 ,last_rech_date_ma,last_rech_date_da,last_rech_amt_ma,cnt_ma_r`

ech30 ,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
]

Catagorical features

Catagorical features =[msisdn ,pcircle ,pdate]

Missing Values

There are no missing values in the dataset

Exploratory the catagorical columns :

```
for column in df.columns:
    if df[column].dtype==object:
        print(str(column) + ':' + str(df[column].unique()))
        print(df[column].value_counts())
        print('\n\n\n**********************************************************************************')
        print("\n")
```

```
msisdn:['21408I70789' '76462I70374' '17943I70372' ... '22758I85348' '59712I82733'
 '65061I85339']
47819I90840    7
04581I85330    7
22038I88658    6
60744I91197    6
29191I82738    6
              ..
41698I90589    1
98495I89233    1
17267I85340    1
73146I90846    1
83144I70372    1
Name: msisdn, Length: 186243, dtype: int64



*******************************************************************************


pdate:['2016-07-20' '2016-08-10' '2016-08-19' '2016-06-06' '2016-06-22'
 '2016-07-02' '2016-07-05' '2016-08-05' '2016-06-15' '2016-06-08'
 '2016-06-12' '2016-06-20' '2016-06-29' '2016-06-16' '2016-08-03'
 '2016-06-24' '2016-07-04' '2016-07-03' '2016-07-01' '2016-08-08'
 '2016-06-26' '2016-06-23' '2016-07-06' '2016-07-09' '2016-06-10'
 '2016-06-07' '2016-06-27' '2016-08-11' '2016-06-30' '2016-06-19'
 '2016-07-26' '2016-08-14' '2016-06-14' '2016-06-21' '2016-06-25'
 '2016-06-28' '2016-06-11' '2016-07-27' '2016-07-23' '2016-08-16'
 '2016-08-15' '2016-06-02' '2016-06-05' '2016-08-02' '2016-07-28'
 '2016-07-18' '2016-08-18' '2016-07-16' '2016-07-29' '2016-07-21'
 '2016-06-03' '2016-06-13' '2016-08-01' '2016-07-13' '2016-07-10'
```

```
 '2016-06-09' '2016-07-15' '2016-07-11' '2016-08-09' '2016-08-12'
 '2016-07-22' '2016-06-04' '2016-07-24' '2016-06-18' '2016-08-13'
 '2016-06-17' '2016-08-07' '2016-07-12' '2016-08-06' '2016-07-19'
 '2016-08-21' '2016-08-04' '2016-07-25' '2016-07-30' '2016-08-17'
 '2016-07-08' '2016-07-14' '2016-06-01' '2016-07-07' '2016-07-17'
 '2016-07-31' '2016-08-20']
2016-07-04    3150
2016-07-05    3127
2016-07-07    3116
2016-06-20    3099
2016-06-17    3082
              ...
2016-06-04    1559
2016-08-18    1407
2016-08-19    1132
2016-08-20     788
2016-08-21     324
Name: pdate, Length: 82, dtype: int64


***************************************************************************
*******
```

There are two catagorical columns named "pdate" and "msisdn"

To explore the catagorical columns and to count the number of values we can use the following code. We can get the following observations after using this code :

Checking Unique values :

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

To explore the dataset it’s also necessary to explore the unique values. If we see our dataset we can see that in some columns there are more unique values and some columns contains less unique values. Columns with less unique values normally effect more to predict the outcome. But if there are constant value in this case there will have no use of that column. From the dataset we can see that ‘pcircle’,has only one unique value. So, this column should be deleted. It will not effect our dataset.

Other observations :

1. All the values in " `Unnamed: 0`" are unique.So, these columns may not effect much to predict the outcome.

Split Date Column :

```
df['pdate']=pd.to_datetime(df["pdate"])
```

```
df["pdate"]
```

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

```
df["year"]=df["pdate"].dt.year
df["month"]=df["pdate"].dt.month
df["day"]=df["pdate"].dt.day
```

To split the 'pdate' column into year, month and day in different columns firstly I have changed the datatype of Date column to datetime64. Then I have splited into different columns by using the previous code.

Summary Statistics :

```
df.describe()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30
count	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000
mean	0.875177	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.847800	3712.202921	2064.452797	3.978057
std	0.330519	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.892230	53374.833430	2370.786034	4.256090
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	-29.000000	0.000000	0.000000
25%	1.000000	246.000000	42.440000	42.692000	280.420000	300.260000	1.000000	0.000000	770.000000	1.000000
50%	1.000000	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.000000	0.000000	1539.000000	3.000000
75%	1.000000	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.000000	0.000000	2309.000000	5.000000
max	1.000000	999860.755168	265926.000000	320630.000000	198926.110000	200148.110000	998650.377733	999171.809410	55000.000000	203.000000

maxamnt_loans30	medianamnt_loans30	cnt_loans90	amnt_loans90	maxamnt_loans90	medianamnt_loans90	payback30	payback90	month	day
209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.00000
274.658747	0.054029	18.520919	23.645398	6.703134	0.046077	3.398826	4.321485	6.797321	14.39894
4245.264648	0.218039	224.797423	26.469861	2.103864	0.200692	8.813729	10.308108	0.741435	8.43890
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.000000	1.00000
6.000000	0.000000	1.000000	6.000000	6.000000	0.000000	0.000000	0.000000	6.000000	7.00000
6.000000	0.000000	2.000000	12.000000	6.000000	0.000000	0.000000	1.666667	7.000000	14.00000
6.000000	0.000000	5.000000	30.000000	6.000000	0.000000	3.750000	4.500000	7.000000	21.00000
99864.560864	3.000000	4997.517944	438.000000	12.000000	3.000000	171.500000	171.500000	8.000000	31.00000

By using describe() function we can explore the count, mean , median, standard deviation, minimum value, 25th, 50th and 75th percentile , maximum value.

We can find the following observations from the dataset :

1.Maximum values of 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,medianamnt_loans30,cnt_loans90,amnt_loans90,maxamnt_loans90,medianamnt_loans90 ,payback30,payback90,month,day are : 1.000000, 999860.755168,265926.000000, 320630.000000, 198926.110000, 200148.110000, 998650.377733, 999171.809410, 55000.000000, 203.000000, 99864.560864 ,3.000000,4997.517944, 438.000000,12.000000,3.000000,171.500000,171.500000,8.000000,31.00000

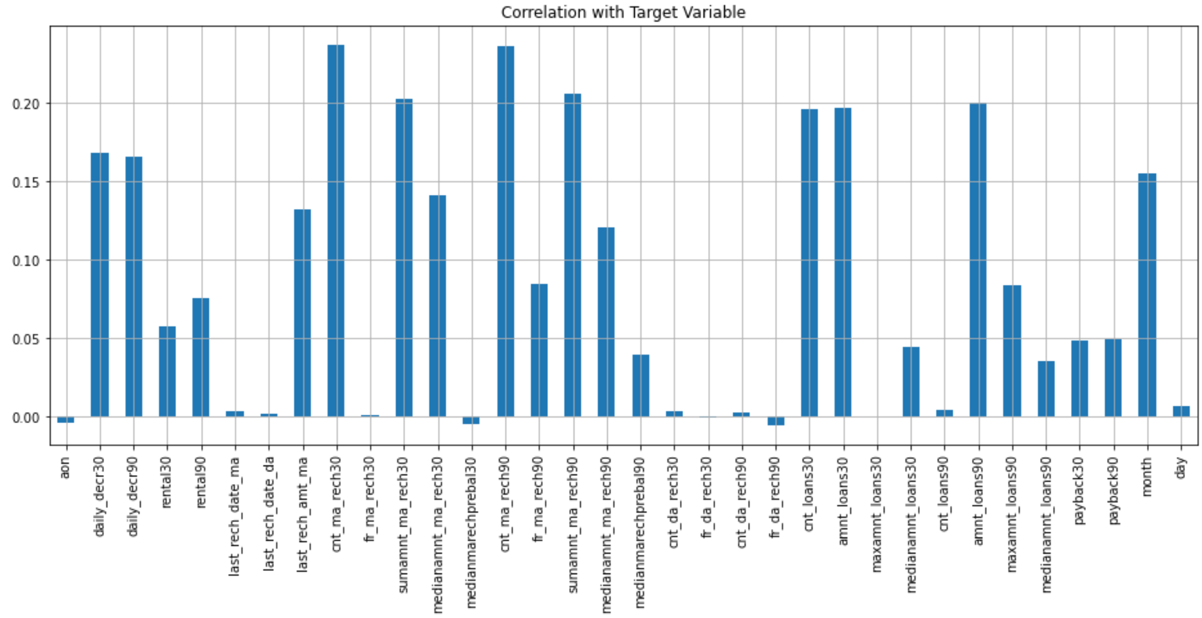
2. Minimum values of label,aon ,daily_decr30,daily_decr90, rental30, rental90, last_rech_date_ma,last_rech_date_da,last_rech_amt_ma,cnt_ma_rech30, maxamnt_loans30,medianamnt_loans30,cnt_loans90,amnt_loans90,maxamnt_loans90,medianamnt_loans90 ,payback30,payback90,month,day are 0.000000,-48.000000,-93.012667, -93.012667,-23737.140000, -24720.580000,-29.000000, -29.000000,0.000000 0.000000,0.000000,0.000000,0.000000,0.000000,0.000000 ,0.000000,0.000000,0.000000,6.000000,1.00000

75 percentile and max value

1. In the most of the columns 75 percentile and max value has huge diffenece. Most probably there are outliers.

Correlation:

Correlation only with target variable :



We can see the correlation of every column with the target variable by using barplot. The peak points which are above 0 are positively correlated with the target variable. And the peak points which are under 0 are negatively correlated with the target variable. But the problem is we can no know the exact value of correlation by using barplot. To know the exact correlation we have another technique.

```
df.corrwith(df["label"])
```

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

From the above figure we we be able to know the exact correlation of each column with the target variable.
The findings are mentioned below:

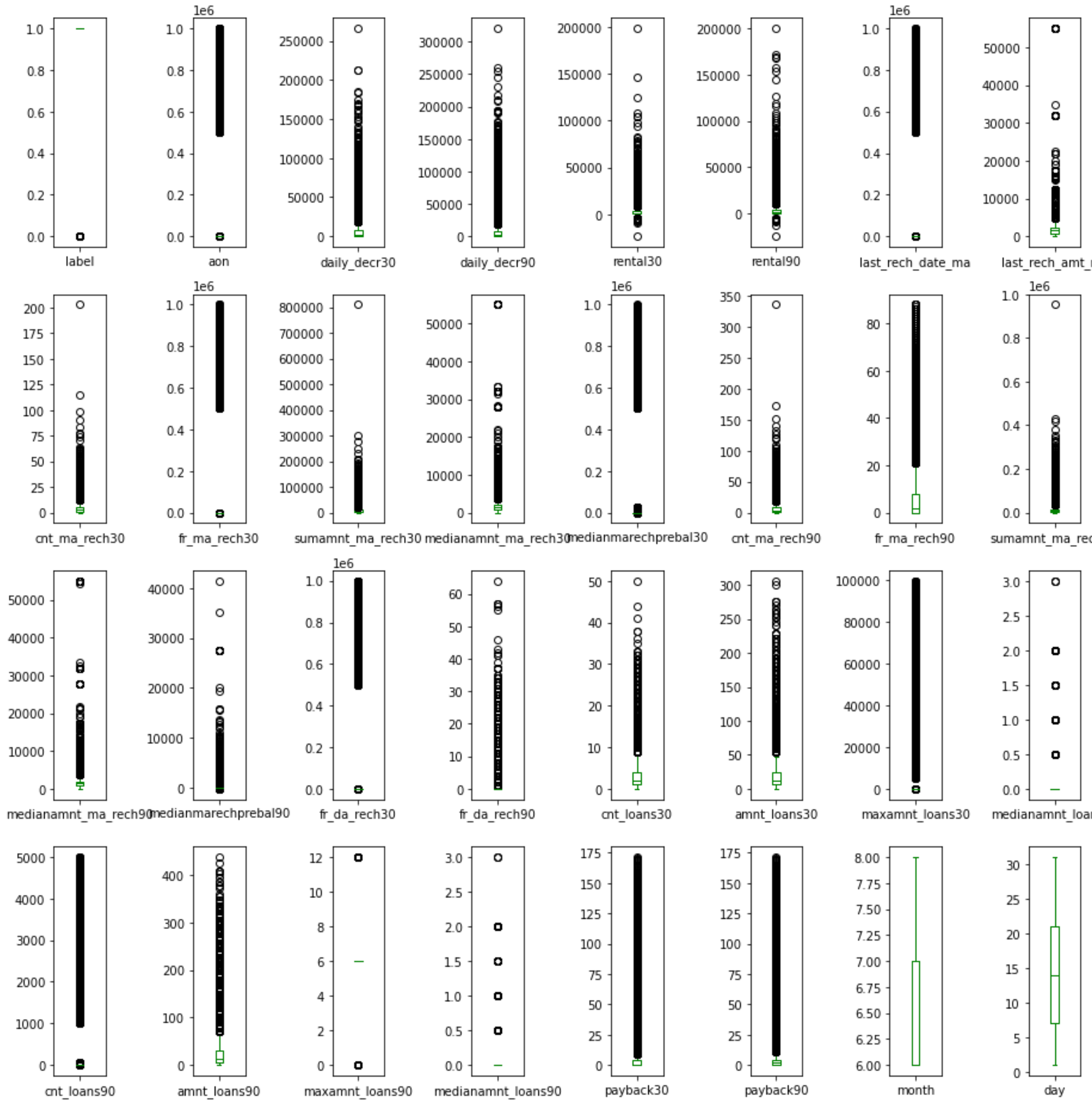
1. Negative correlation with Average price : aon, medianmarechprebal30, fr_da_rech30, fr_da_rech90,

2. Positive correlation : All the columns except aon, medianmarechprebal30, fr_da_rech30, fr_da_rech90,

3. Strong correlation : daily_decr30, daily_decr90 , last_rech_amt_ma ,cnt_ma_rech30 , sumamnt_ma_rech30 ,medianamnt_ma_rech30,cnt_ma_rech90 ,sumamnt_ma_rech90 ,medianamnt_ma_rech90 ,cnt_loans30 ,amnt_loans30 ,amnt_loans90 ,month

Check Outliers :

```
df.plot(kind='box',subplots=True,layout=(4,5),color='green',figsize=(13,
13))
plt.tight_layout()
```



After illustrating the above figure we can explore that most of the columns except month and dat having outliers. So, we need to treat the outliers.

We can also check the outliers individually.

Dropping columns :

```
df.drop(['msisdn','balance_group','frequency_group','loan_frequency_group','loanamnt_frequency_group'],axis=1,inplace=True)
df
```

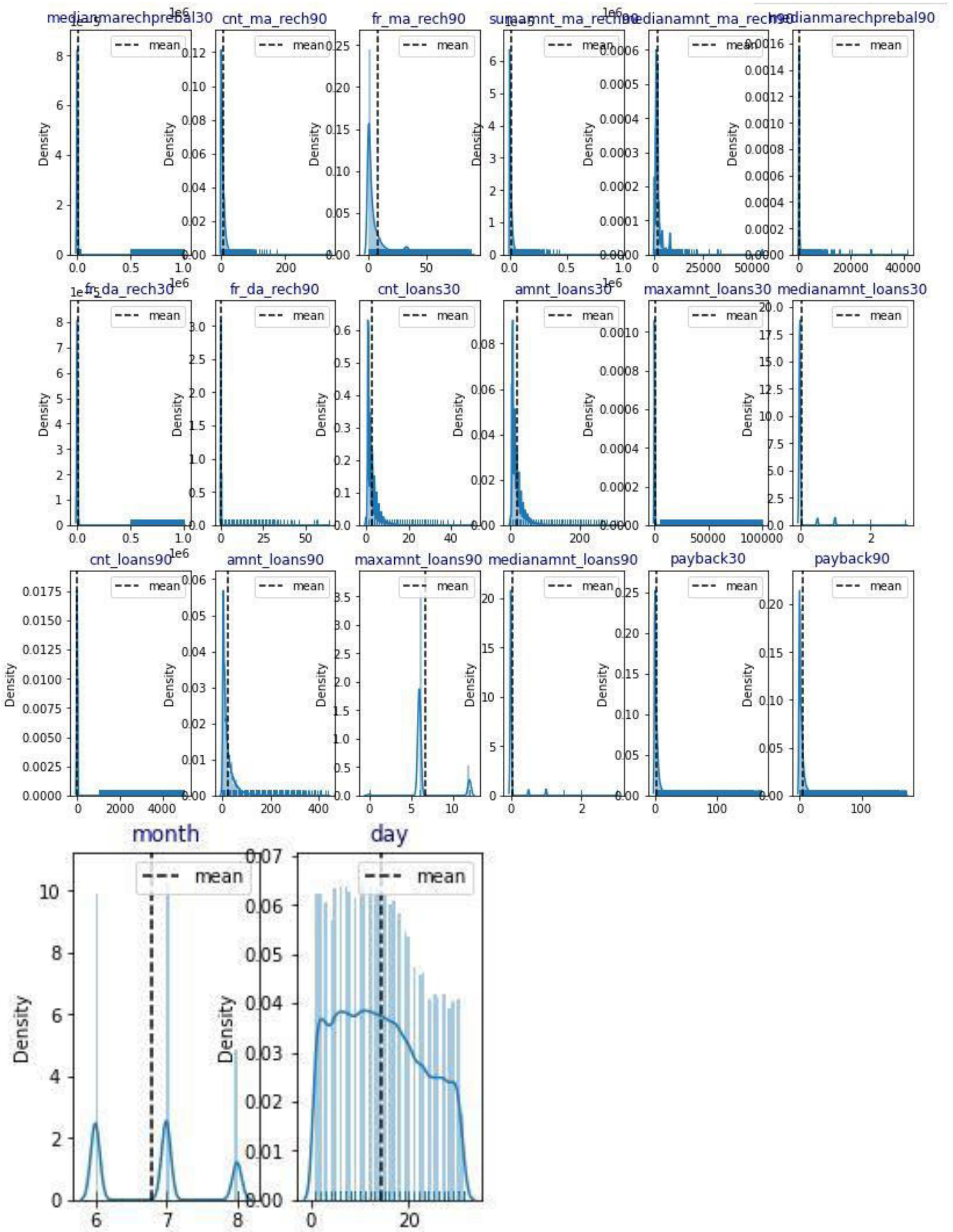
```
df.drop(['year'],axis=1,inplace=True)
```

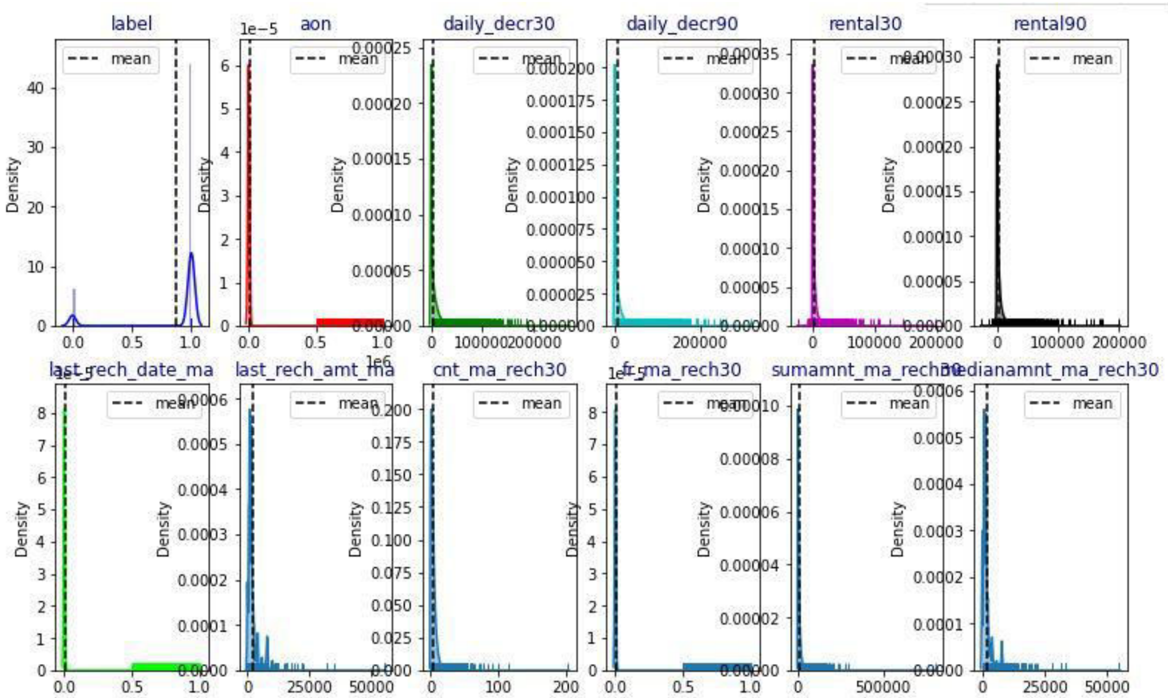
```
df.drop(['pdate'],axis=1,inplace=True)
```

```
df.drop(['pcircle','Unnamed: 0'],axis=1,inplace=True)
```

1. As the correlation of 'last_rech_date_da','cnt_da_rech30','cnt_da_rech90' columns with the target column is not good thet's why I have dropped those columns.
2. In the year column there are only single value. So, it will not effect our dataset.
3. As we have splitted pdate to year, month and day that's why pdate is not necessary.
4. Pcircle has constant value and Unnamed: 0 has all the unique values. That's why these two columns will not effect our dataset.

Skewness:


medianmarechprebal30
cnt_ma_rech90
fr_ma_rech90
sumamnt_ma_rech90
medianamnt_ma_rech90
medianmarechprebal90
fr_da_rech30
fr_da_rech90
cnt_loans30
amnt_loans30
maxamnt_loans30
medianamnt_loans30
cnt_loans90
amnt_loans90
maxamnt_loans90
medianamnt_loans90
payback30
payback90
month
day
mean
Density



In most of the columns skewness is present.

```
df.skew()
```

```
df.skew()
```

```
label                    -2.270254
aon                      10.392949
daily_decr30              3.946230
daily_decr90              4.252565
rental30                  4.521929
rental90                  4.437681
last_rech_date_ma        14.790974
last_rech_amt_ma          3.781149
cnt_ma_rech30             3.283842
fr_ma_rech30             14.772833
sumamnt_ma_rech30         6.386787
medianamnt_ma_rech30      3.512324
medianmarechprebal30     14.779875
cnt_ma_rech90             3.425254
fr_ma_rech90              2.285423
sumamnt_ma_rech90         4.897950
medianamnt_ma_rech90      3.752706
medianmarechprebal90     44.880503
fr_da_rech30             14.776430
fr_da_rech90             28.988083
cnt_loans30               2.713421
amnt_loans30              2.975719
maxamnt_loans30          17.658052
medianamnt_loans30        4.551043
cnt_loans90              16.594408
amnt_loans90              3.150006
maxamnt_loans90           1.678304
medianamnt_loans90        4.895720
payback30                 8.310695
payback90                 6.899951
month                     0.343242
day                       0.199845
dtype: float64
```

If we want to know the exact value then skew() function is the best way to know the skewness of the variavles. Here, The standard value I have used is 0.56. If the value is not in between -0.56 and 0.56 that means skewness is present in those columns.

Removing Outliers :

Z-Score

```
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(x))
threshold=3
new_x=x[(z<3).all(axis=1)]
```

```
new_x.shape
```

```
(163331, 31)
```

```
new2_x=x[~((x<(q1-1.5*IQR))|(x>(q3+1.5*IQR))).any(axis=1)]
print(new2_x.shape)
```

```
(72436, 31)
```

I have used both z-score and IQR method to remove the outliers. But using both of the methods we can see that we are loosing huge amount of data. So, we will create our model with outliers.

Split the data into x and y

```
new2_x=x[~((x<(q1-1.5*IQR))|(x>(q3+1.5*IQR))).any(axis=1)]
print(new2_x.shape)
```

```
(72436, 31)
```

```
y=df['label']
y.head()
```

I have splitted the dataset into x and y where x represents all the columns except the target variable labeland y represents the target variable.

Treating Skewness via yeo-johnson method:

Yeo Jonson Method ¶

```
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
```

Model Training :

Spilitting the data into input and output variable :

```
x=df_new.drop('AveragePrice',axis=1)
y=df_new['AveragePrice']
```

We can split the data into x and y. x is having all the columns except the target variable. Y is having only the target column.

Spliting the data into training and testing set :

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=42)
```

Build Model :

Scaling :

As I have used yeo-johnson method to remove the skewness that's why there is no need to scale the dataset. It automatically scale the data as well.

Importing all the model Library :

```
# Libraries for data modelling
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

#Importinf boosting models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

#Importing error metrics
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import GridSearchCV,cross_val_score
```

All algorithms are in one code :

```
#LogisticRegression(),GaussianNB(),SVC(),DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesCl
model=[LogisticRegression(),GaussianNB(),SVC(),DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraT

for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    predm=m.predict(x_test)
    print("\033[1m"+ 'Accuracy score of',m,'is : ' + "\033[0m" )  # Make the line bold
    print(accuracy_score(y_test,predm))
    print(confusion_matrix(y_test,predm))
    print(classification_report(y_test,predm))
    print('********************************************************************')
    print('\n')
```

By using all the algoriths one by one we can use one function to implement all the algorithms.
If we summarize the result we get the following r2 score :
LogisticRegression(),GaussianNB(),SVC(),DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()

Accuracy score of LogisticRegression() is :
0.8828216321954245

Accuracy score of GaussianNB() is :
0.7401655573844796

Accuracy score of SVC() is :
0.9059614971731196

Accuracy score of DecisionTreeClassifier() is :
0.8831794651589971

Accuracy score of KNeighborsClassifier() is :
0.9036952217371598

Accuracy score of RandomForestClassifier() is :
0.921133614828598

Accuracy score of AdaBoostClassifier() is :
0.9088003053507956

Accuracy score of GradientBoostingClassifier() is :
0.9169111858584412

Accuracy score of BaggingClassifier() is :
0.9143586440516234

Accuracy score of ExtraTreesClassifier() is :
0.9183425177127317

We have got good Accuracy Score by using the following algorithms :
KNeighborsClassifier() , SVC() , RandomForestClassifier (),
AdaBoostClassifier(), **GradientBoostingClassifier** (), **BaggingClassifier** (),
ExtraTreesClassifier () is giving the best result.
But **RandomForestClassifier() is givving the highest accuracy rate.**
We will do the hyperparameter tuning to reduce the overfitting.

Using Best Parameter :

```
parameters={'random_state':range(42,100)}
RFC=RandomForestClassifier()
clf=GridSearchCV(RFC,parameters)
clf.fit(x,y)
print(clf.best_params_)
```

Here the best parametes for **RandomForestClassifier** () 'random_state': 100.

Using Best Parameter :

```
RFC=RandomForestClassifier(random_state=42)
RFC.fit(x_train,y_train)
RFC.score(x_train,y_train)
predRFC=RFC.predict(x_test)
print('Accuracy score of',RFC,'is : ')
print(accuracy_score(y_test,predRFC))
print(confusion_matrix(y_test,predRFC))
print(classification_report(y_test,predRFC))
```

```
Accuracy score of RandomForestClassifier(random_state=42) is :
0.9213244590758367
[[ 2764  2533]
 [  765 35857]]
              precision    recall  f1-score   support

           0       0.78      0.52      0.63      5297
           1       0.93      0.98      0.96     36622

    accuracy                           0.92     41919
   macro avg       0.86      0.75      0.79     41919
weighted avg       0.91      0.92      0.91     41919
```

After using the best parametes we have got the accuracy for `RandomForestClassifier` () is `0.9213244590758367`

Cross Validation score :

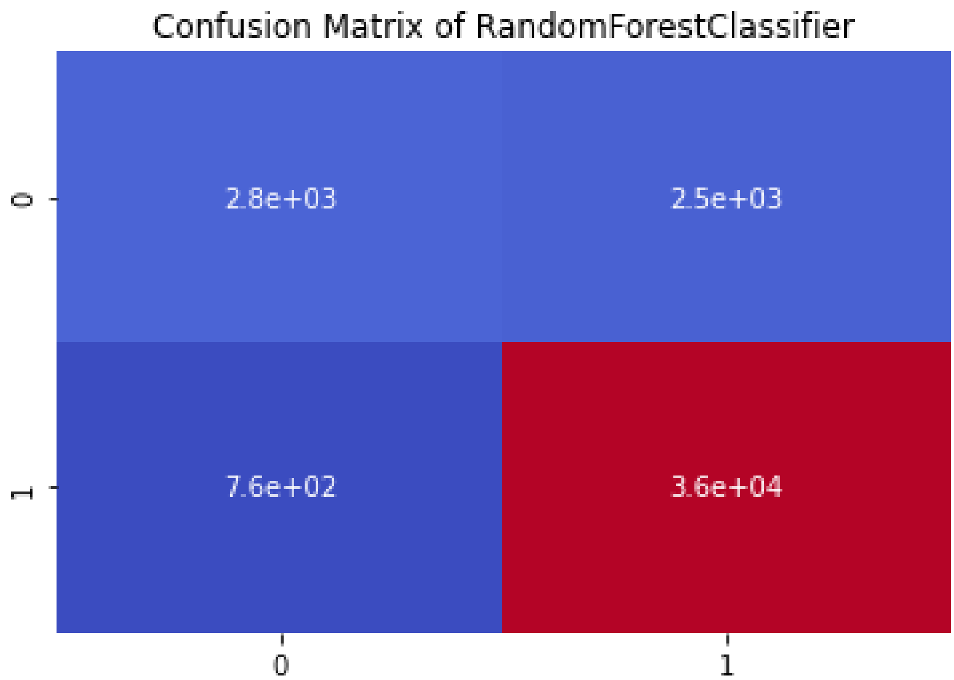
```
score=cross_val_score(RFC,x,y,scoring='accuracy')
print("model:",RFC)
print("Score:",score)
print("Mean score:",score.mean())
print("Standard deviation:",score.std())
```

Result :

```
model: RandomForestClassifier(random_state=42)
Score: [0.9215153  0.92058494 0.92046566 0.92199055 0.92103631]
Mean score: 0.921118552574546
Standard deviation: 0.0005719394309432784
```

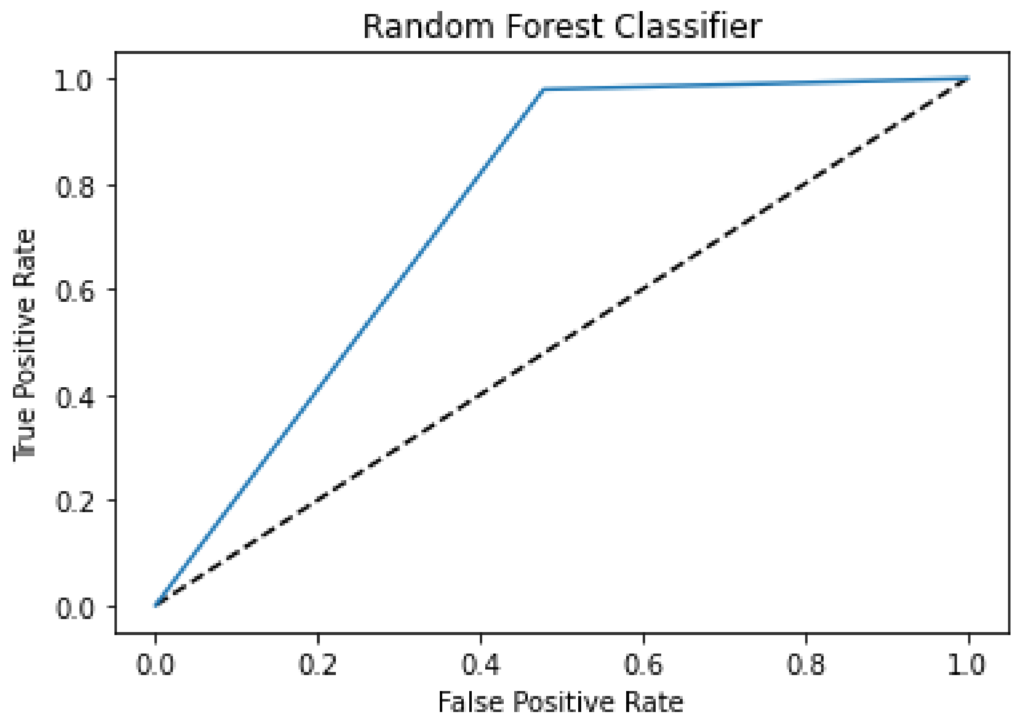
Plotting Confusion matrix for RandomForestClassifier()

```
cm=confusion_matrix(y_test,predRFC)
sns.heatmap(cm,annot=True,cbar=False,cmap='coolwarm')

plt.title("Confusion Matrix of RandomForestClassifier")
plt.show()
```



Auc_Roc Curve and finding auc score

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
y_pred_prob=RFC.predict(x_test)
fpr,tpr,thredholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label='Random Forest Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest Classifier')
plt.show()

auc_score=roc_auc_score(y_test,predRFC)
print(auc_score)
```



The predicted data and the original are almost on the same line. So, this model will be accepted.

Saving the model

Saving the model

```
import joblib
```

Save the model as a pickle in a file

```
joblib.dump(RFC,'credit_defaulter.pkl')
```

Name : Md Rashidunnabi
Email : rashidunnabi11@gmail.com
Phone : +8801319527349
Whatsapp : +8801319527349