Micro Credit Defaulter

Submitted by:

Dipak Someshwar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Khushboo Garg.

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thank and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Dipak Someshwar

INTRODUCTION

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

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

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

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

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

They are collaborating with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be defaulter if he deviates from the path of paying back the loaned amount within the time duration of 5 days. For

the loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).

The sample data is provided to us from our client database. It is hereby given to you for this exercise. In order to improve the selection of customers for the credit, the client wants some predictions that could help them in further investment and improvement in selection of customers.

1. **Model Building Phase:**

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

   Before model building do all data pre-processing steps.

   Try different models with different hyper parameters and select the best model.

   1. Data Cleaning

   2. Exploratory Data Analysis

   3. Data Pre-processing

   4. Model Building

   5. Model Evaluation

   6. Selecting the best model

Analytical Problem Framing

- Import library and load the dataset.

Data Reading and Analysis:

In [1]:
```
# Importing libraries.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load the dataset.
import pandas as pd
df = pd.read_csv(r'C:/Users/dipak/Desktop/Micro Credit Project/micro_credit.csv')
df
```

Out[2]:

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	med
0	1	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	...	6.0	
1	2	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	...	12.0	
2	3	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	...	6.0	
3	4	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	...	6.0	
4	5	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	...	6.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
209588	209589	1	22758I85348	404.0	151.872333	151.872333	1089.19	1089.19	1.0	0.0	...	6.0	
209589	209590	1	95583I84455	1075.0	36.936000	36.936000	1728.36	1728.36	4.0	0.0	...	6.0	
209590	209591	1	28556I85350	1013.0	11843.111667	11904.350000	5861.83	8893.20	3.0	0.0	...	12.0	
209591	209592	1	59712I82733	1732.0	12488.228333	12574.370000	411.83	984.58	2.0	38.0	...	12.0	
209592	209593	1	65061I85339	1581.0	4489.362000	4534.820000	483.92	631.20	13.0	0.0	...	12.0	

209593 rows × 37 columns

- Drop unnamed column and get the shape of data frame.

In [3]:
```
# drop unnamed column
df.drop('Unnamed: 0',axis=1,inplace=True)
df.head()
```

Out[3]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	maxamnt_loans30	me
0	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	...	6.0	
1	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	...	12.0	
2	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	...	6.0	
3	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	...	6.0	
4	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	...	6.0	

5 rows × 36 columns

In [4]:
```
# Get the numbers of rows and columns.
df.shape
```

Out[4]: (209593, 36)

- Display all column name of dataset.

In [5]:
```
# Check column of the dataframe.
df.columns
```

Out[5]:
```
Index(['label', 'msisdn', 'aon', 'daily_decr30', 'daily_decr90', 'rental30',
       'rental90', 'last_rech_date_ma', 'last_rech_date_da',
       'last_rech_amt_ma', 'cnt_ma_rech30', 'fr_ma_rech30',
       'sumamnt_ma_rech30', 'medianamnt_ma_rech30', 'medianmarechprebal30',
       'cnt_ma_rech90', 'fr_ma_rech90', 'sumamnt_ma_rech90',
       'medianamnt_ma_rech90', 'medianmarechprebal90', 'cnt_da_rech30',
       'fr_da_rech30', 'cnt_da_rech90', 'fr_da_rech90', 'cnt_loans30',
       'amnt_loans30', 'maxamnt_loans30', 'medianamnt_loans30', 'cnt_loans90',
       'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30',
       'payback90', 'pcircle', 'pdate'],
      dtype='object')
```

- Display datatypes and sum of null values.

In [6]:
```
# Get the column datatypes.
df.dtypes
```

Out[6]:
```
label                    int64
msisdn                  object
aon                    float64
daily_decr30           float64
daily_decr90           float64
rental30               float64
rental90               float64
last_rech_date_ma      float64
last_rech_date_da      float64
last_rech_amt_ma         int64
cnt_ma_rech30            int64
fr_ma_rech30           float64
sumamnt_ma_rech30      float64
medianamnt_ma_rech30   float64
medianmarechprebal30   float64
cnt_ma_rech90            int64
fr_ma_rech90             int64
sumamnt_ma_rech90        int64
medianamnt_ma_rech90   float64
medianmarechprebal90   float64
cnt_da_rech30          float64
fr_da_rech30           float64
cnt_da_rech90            int64
fr_da_rech90             int64
cnt_loans30              int64
amnt_loans30             int64
maxamnt_loans30        float64
medianamnt_loans30     float64
cnt_loans90            float64
amnt_loans90             int64
maxamnt_loans90          int64
medianamnt_loans90     float64
payback30              float64
payback90              float64
pcircle                 object
pdate                   object
dtype: object
```

In [8]:
```
# Get a count of the empty values for each column.
df.isna().sum()
```

Out[8]:
```
label                  0
msisdn                 0
aon                    0
daily_decr30           0
daily_decr90           0
rental30               0
rental90               0
last_rech_date_ma      0
last_rech_date_da      0
last_rech_amt_ma       0
cnt_ma_rech30          0
fr_ma_rech30           0
sumamnt_ma_rech30      0
medianamnt_ma_rech30   0
medianmarechprebal30   0
cnt_ma_rech90          0
fr_ma_rech90           0
sumamnt_ma_rech90      0
medianamnt_ma_rech90   0
medianmarechprebal90   0
cnt_da_rech30          0
fr_da_rech30           0
cnt_da_rech90          0
fr_da_rech90           0
cnt_loans30            0
amnt_loans30           0
maxamnt_loans30        0
medianamnt_loans30     0
cnt_loans90            0
amnt_loans90           0
maxamnt_loans90        0
medianamnt_loans90     0
payback30              0
payback90              0
pcircle                0
pdate                  0
dtype: int64
```

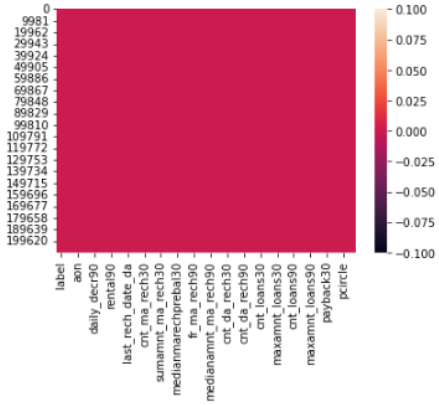
- Display null values of columns using heatmap.

```
In [9]:
# Check any missing/null values in the dataset.
df.isnull().values.any()
```

```
Out[9]: False
```

```
In [10]:
# Checking for null values using heatmap.
sns.heatmap(df.isnull())
```

```
Out[10]: <AxesSubplot:>
```



- Get the nunique values.

```
In [11]:
# Get the number of unique values.
df.nunique()
```

```
Out[11]: label                   2
msisdn                  186243
aon                       4507
daily_decr30            147025
daily_decr90            158669
rental30                132148
rental90                141033
last_rech_date_ma         1186
last_rech_date_da         1174
last_rech_amt_ma            70
cnt_ma_rech30               71
fr_ma_rech30              1083
sumamnt_ma_rech30        15141
medianamnt_ma_rech30       510
medianmarechprebal30     30428
cnt_ma_rech90              110
fr_ma_rech90                89
sumamnt_ma_rech90        31771
medianamnt_ma_rech90       608
medianmarechprebal90     29785
cnt_da_rech30             1066
fr_da_rech30              1072
cnt_da_rech90               27
fr_da_rech90                46
cnt_loans30                 40
amnt_loans30                48
maxamnt_loans30           1050
medianamnt_loans30           6
cnt_loans90               1110
amnt_loans90                69
maxamnt_loans90              3
medianamnt_loans90           6
payback30                 1363
payback90                 2381
pcircle                      1
pdate                       82
dtype: int64
```

- Data Preprocessing

Data Preprocessing:

Remove columns where number of unique value is only 1.

In [12]:
```
unique = df.nunique()
unique = unique[unique.values == 1]
unique
```

Out[12]:
```
pcircle    1
dtype: int64
```

In [13]:
```
df.drop('pcircle', axis=1, inplace=True)
df.head()
```

Out[13]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	amnt_loans30	maxam
0	0	21408170789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	...	12	
1	1	76462170374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	...	12	
2	1	17943170372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	...	6	
3	1	55773170781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	...	12	
4	1	03813182730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	...	42	

5 rows × 35 columns

- Summary Statistics and Drop Duplicates value.

In [14]:
```
# Summary statistics.
df.describe().transpose().style.background_gradient()
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
label	209593.000000	0.875177	0.330519	0.000000	1.000000	1.000000	1.000000	1.000000
aon	209593.000000	8112.343445	75696.082531	-48.000000	246.000000	527.000000	982.000000	999860.755168
daily_decr30	209593.000000	5381.402289	9220.623400	-93.012667	42.440000	1469.175667	7244.000000	265926.000000
daily_decr90	209593.000000	6082.515068	10918.812767	-93.012667	42.692000	1500.000000	7802.790000	320630.000000
rental30	209593.000000	2692.581910	4308.586781	-23737.140000	280.420000	1083.570000	3356.940000	198926.110000
rental90	209593.000000	3483.406534	5770.461279	-24720.580000	300.260000	1334.000000	4201.790000	200148.110000
last_rech_date_ma	209593.000000	3755.847800	53905.892230	-29.000000	1.000000	3.000000	7.000000	998850.377733
last_rech_date_da	209593.000000	3712.202921	53374.833430	-29.000000	0.000000	0.000000	0.000000	999171.809410
last_rech_amt_ma	209593.000000	2064.452797	2370.786034	0.000000	770.000000	1539.000000	2309.000000	55000.000000
cnt_ma_rech30	209593.000000	3.978057	4.256090	0.000000	1.000000	3.000000	5.000000	203.000000
fr_ma_rech30	209593.000000	3737.355121	53643.625172	0.000000	0.000000	2.000000	6.000000	999606.368132
sumamnt_ma_rech30	209593.000000	7704.501157	10139.621714	0.000000	1540.000000	4628.000000	10010.000000	810096.000000
medianamnt_ma_rech30	209593.000000	1812.817952	2070.864620	0.000000	770.000000	1539.000000	1924.000000	55000.000000
medianmarechprebal30	209593.000000	3851.927942	54006.374433	-200.000000	11.000000	33.900000	83.000000	999479.419319
cnt_ma_rech90	209593.000000	6.315430	7.193470	0.000000	2.000000	4.000000	8.000000	336.000000
fr_ma_rech90	209593.000000	7.716780	12.590251	0.000000	0.000000	2.000000	8.000000	88.000000
sumamnt_ma_rech90	209593.000000	12396.218352	16857.793882	0.000000	2317.000000	7226.000000	16000.000000	953038.000000
medianamnt_ma_rech90	209593.000000	1864.595821	2081.680664	0.000000	773.000000	1539.000000	1924.000000	55000.000000
medianmarechprebal90	209593.000000	92.025541	369.215658	-200.000000	14.600000	36.000000	79.310000	41456.500000
cnt_da_rech30	209593.000000	262.578110	4183.897978	0.000000	0.000000	0.000000	0.000000	99914.441420
fr_da_rech30	209593.000000	3749.494447	53885.414979	0.000000	0.000000	0.000000	0.000000	999809.240107
cnt_da_rech90	209593.000000	0.041495	0.397556	0.000000	0.000000	0.000000	0.000000	38.000000
fr_da_rech90	209593.000000	0.045712	0.951386	0.000000	0.000000	0.000000	0.000000	64.000000
cnt_loans30	209593.000000	2.758981	2.554502	0.000000	1.000000	2.000000	4.000000	50.000000
amnt_loans30	209593.000000	17.952021	17.379741	0.000000	6.000000	12.000000	24.000000	306.000000
maxamnt_loans30	209593.000000	274.658747	4245.264648	0.000000	6.000000	6.000000	6.000000	99864.560864
medianamnt_loans30	209593.000000	0.054029	0.218039	0.000000	0.000000	0.000000	0.000000	3.000000
cnt_loans90	209593.000000	18.520919	224.797423	0.000000	1.000000	2.000000	5.000000	4997.517944
amnt_loans90	209593.000000	23.645398	26.469861	0.000000	6.000000	12.000000	30.000000	438.000000
maxamnt_loans90	209593.000000	6.703134	2.103864	0.000000	6.000000	6.000000	6.000000	12.000000
medianamnt_loans90	209593.000000	0.046077	0.200692	0.000000	0.000000	0.000000	0.000000	3.000000
payback30	209593.000000	3.398826	8.813729	0.000000	0.000000	0.000000	3.750000	171.500000

In [15]:
```
# Summary statistics for object and datetime.
df.describe(include=['object','datetime']).transpose()
```

Out[15]:

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

In [16]:
```
df1 = df.copy()
```

In [17]:
```
# Drop duplicate values of msisdn column.
df.drop_duplicates(subset='msisdn',keep='first',inplace=True)
df.shape
```

Out[17]: (186243, 35)

- Data Exploration.

Data Exploration:

In [18]:
```
# Print all of the data types and their unique values.
for column in df.columns:
    if df[column].dtype == object:
        print(str(column) + ' : ' + str(df[column].unique()))
        print(df[column].value_counts())
        print('_____________________________________________')
```

```
msisdn : ['21408I70789' '76462I70374' '17943I70372' ... '22758I85348' '59712I82733'
 '65061I85339']
21408I70789    1
77953I95203    1
37896I85348    1
49727I70372    1
08288I88649    1
              ..
74074I82731    1
04445I85349    1
09031I70378    1
88188I90580    1
65061I85339    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']
2016-07-05    2825
2016-07-07    2792
2016-07-04    2790
2016-06-20    2785
2016-06-17    2770
              ...
2016-06-04    1395
2016-08-18    1226
2016-08-19     970
2016-08-20     675
```

In [19]:
```
# Print all of the data types and their unique values.
for column in df.columns:
    if df[column].dtype == np.number:
        print(str(column) + ' : ' + str(df[column].nunique()))
        print(df[column].value_counts())
        print('_____________________________________________')
```

```
aon : 4282
95.000000        366
92.000000        342
96.000000        339
99.000000        335
94.000000        328
                ...
847385.682864      1
602000.328014      1
825740.918634      1
654148.566653      1
581435.484230      1
Name: aon, Length: 4282, dtype: int64
_____________________________________________
daily_decr30 : 130323
0.000000       4136
500.000000      823
1000.000000     557
700.000000      377
```

In [20]:
```
# Checking the number of defaulter and non-defaulter customer
df['label'].value_counts()
```

Out[20]:
```
1    160383
0     25860
Name: label, dtype: int64
```

In [21]:
```
# Checking the number of defaulter and non-defaulter customer percentage wise.
df['label'].value_counts(normalize=True) * 100
```

Out[21]:
```
1    86.114914
0    13.885086
Name: label, dtype: float64
```

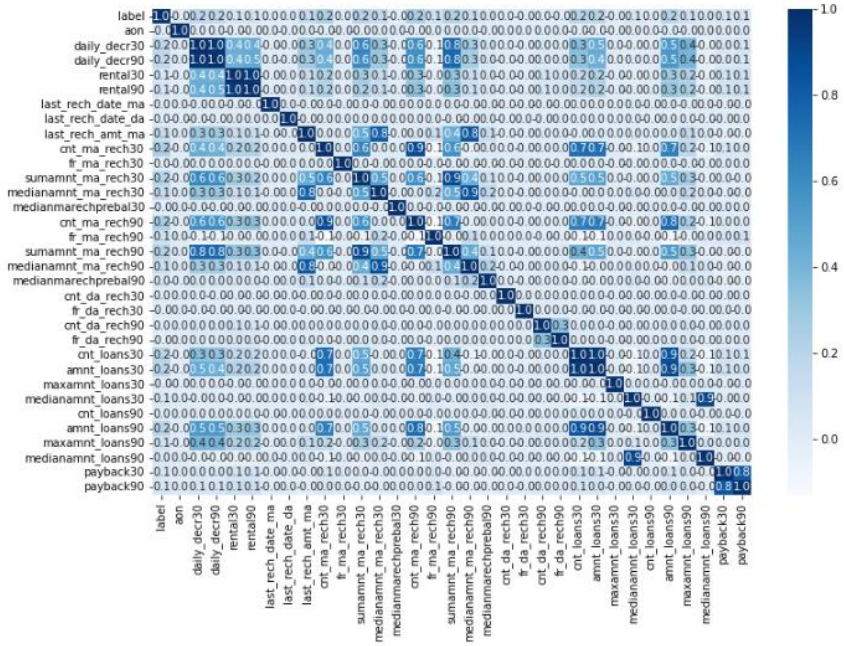
Correlation matrix

In [22]:
```
# Check correlation of columns.
df.corr()
```

Out[22]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma
label	1.000000	-0.004035	0.174901	0.173016	0.057207	0.075869	0.004113	0.001814	0.139969	0
aon	-0.004035	1.000000	0.000630	0.000052	-0.002930	-0.002618	0.001853	-0.001796	0.004102	-0
daily_decr30	0.174901	0.000630	1.000000	0.977659	0.427503	0.444932	-0.000171	-0.001311	0.287181	0
daily_decr90	0.173016	0.000052	0.977659	1.000000	0.420561	0.457443	0.000058	-0.001484	0.275195	0
rental30	0.057207	-0.002930	0.427503	0.420561	1.000000	0.955233	-0.000949	0.003294	0.128773	0
rental90	0.075869	-0.002618	0.444932	0.457443	0.955233	1.000000	-0.001758	0.002643	0.123436	0
last_rech_date_ma	0.004113	0.001853	-0.000171	0.000058	-0.000949	-0.001758	1.000000	0.002629	-0.000754	0
last_rech_date_da	0.001814	-0.001796	-0.001311	-0.001484	0.003294	0.002643	0.002629	1.000000	-0.000699	0
last_rech_amt_ma	0.139969	0.004102	0.287181	0.275195	0.128773	0.123436	-0.000754	-0.000699	1.000000	0
cnt_ma_rech30	0.244728	-0.004315	0.444365	0.419650	0.220472	0.218618	0.006491	0.002690	0.008012	1
fr_ma_rech30	0.001129	-0.000436	0.000766	0.001091	0.000272	0.001057	-0.001165	0.000958	0.002998	0
sumamnt_ma_rech30	0.207727	-0.000397	0.630202	0.597542	0.258656	0.246626	0.002544	0.000080	0.456707	0
medianamnt_ma_rech30	0.149780	0.004446	0.307440	0.294838	0.132083	0.122747	-0.002716	0.000184	0.796969	0
medianmarechprebal30	-0.004835	0.004221	-0.000854	-0.000688	-0.001112	-0.001047	0.004216	0.003673	-0.002597	0
cnt_ma_rech90	0.245941	-0.003957	0.576787	0.582115	0.295746	0.329330	0.006131	0.001924	0.028202	0
fr_ma_rech90	0.094709	0.005517	-0.061858	-0.063740	-0.022353	-0.024882	0.000881	0.001071	0.109126	-0
sumamnt_ma_rech90	0.212666	0.000160	0.754042	0.759865	0.324302	0.342772	0.002345	-0.000296	0.436776	0
medianamnt_ma_rech90	0.129527	0.005022	0.269721	0.262627	0.113115	0.106832	-0.001947	-0.000321	0.824654	-0

In [23]:
```
# Check correlation matrix with heatmap.
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, fmt = '.1f', cmap = 'Blues')
#sns.heatmap(df.corr(), cmap='coolwarm', annot=True, fmt='.1f', linewidths=.1)
plt.show()
```



In [24]:
```
# Dropping the columns which is highly correlated with each other do avoid multicolinearity problem.
df.drop(columns=['daily_decr30','rental30','amnt_loans30','medianamnt_loans30'],axis=1, inplace = True)
```

In [25]:
```
# Get the numbers of rows and columns.
df.shape
```

Out[25]: (186243, 31)

In [26]:
```
# Making the new column Day, Month and year from pdate column
df['pDay']=pd.to_datetime(df['pdate'],format='%Y/%m/%d').dt.day
df['pMonth']=pd.to_datetime(df['pdate'],format='%Y/%m/%d').dt.month
df['pYear']=pd.to_datetime(df['pdate'],format='%Y/%m/%d').dt.year
```

In [27]:
```
df.head()
```

Out[27]:

	label	msisdn	aon	daily_decr90	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rech30	...	cnt_loans90	ar
0	0	21408I70789	272.0	3065.150000	260.13	2.0	0.0	1539	2	21.0	...	2.0	
1	1	76462I70374	712.0	12124.750000	3691.26	20.0	0.0	5787	1	0.0	...	1.0	
2	1	17943I70372	535.0	1398.000000	900.13	3.0	0.0	1539	1	0.0	...	1.0	
3	1	55773I70781	241.0	21.228000	159.42	41.0	0.0	947	0	0.0	...	2.0	
4	1	03813I82730	947.0	150.619333	1098.90	4.0	0.0	2309	7	2.0	...	7.0	

5 rows × 34 columns

In [28]:
```
# Checking the number of months
df['pMonth'].unique()
```

Out[28]:
```
array([7, 8, 6], dtype=int64)
```

In [29]:
```
# After fetching the data from pdate column now we are going to drop it because it has not any significant role.
df.drop(columns=['pdate'],axis=1, inplace = True)
```

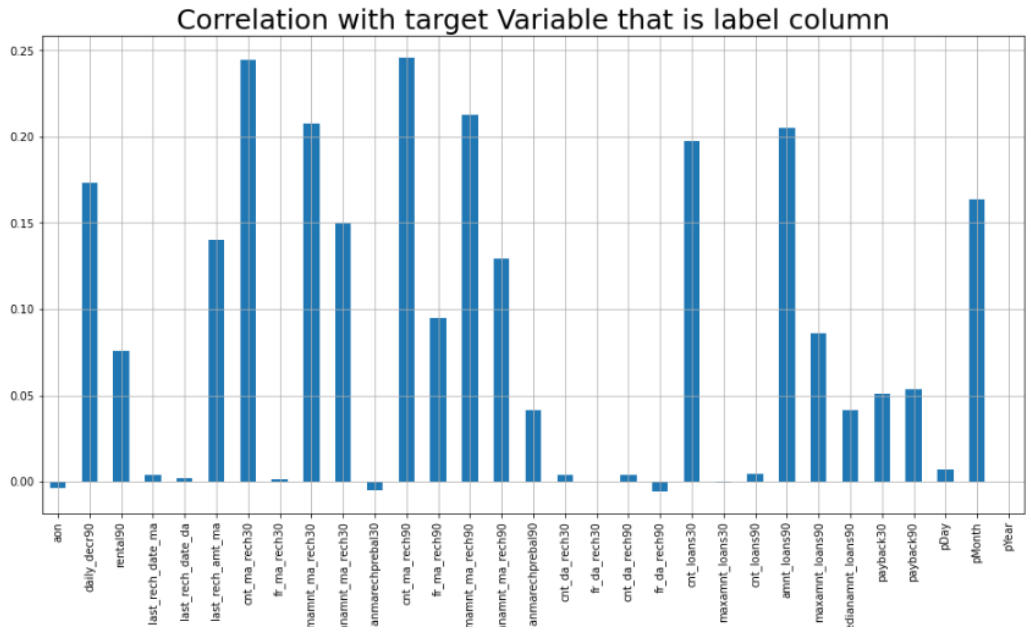
In [30]:
```
# Seprate the categorical columns and Numerical columns
cat_df,num_df=[],[]

for i in df.columns:
    if df[i].dtype==object:
        cat_df.append(i)
    elif (df[i].dtypes=='int64') | (df[i].dtypes=='float64') | (df[i].dtypes=='int32'):
        num_df.append(i)
    else: continue

print('>>> Total Number of Feature::', df.shape[1])
print('>>> Number of categorical features::', len(cat_df))
print('>>> Number of Numerical Feature::', len(num_df))
```

```
>>> Total Number of Feature:: 33
>>> Number of categorical features:: 1
>>> Number of Numerical Feature:: 32
```

- Data Visualization.

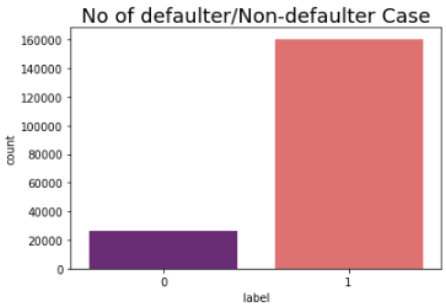
In [31]:
```
# Checking the correlation with target variable
plt.figure(figsize=(16,8))
df.drop('label', axis=1).corrwith(df['label']).plot(kind='bar',grid=True)
plt.xticks(rotation='vertical')
plt.title("Correlation with target Variable that is label column",fontsize=25)
```

Out[31]:
```
Text(0.5, 1.0, 'Correlation with target Variable that is label column')
```



In [32]:

```
# Checking the number of Fraud cases.
sns.countplot(x='label', data=df, palette='magma')
plt.title('No of defaulter/Non-defaulter Case',fontsize=18)
plt.show()

print(df['label'].value_counts())
```



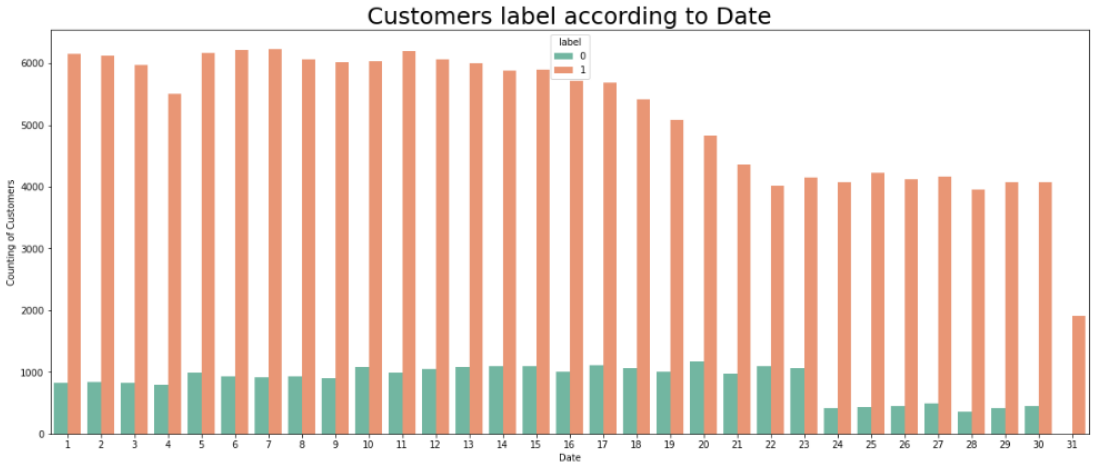

```
1    160383
0     25860
Name: label, dtype: int64
```

In [33]:

```
# Plotting the Histogram
df.hist(figsize=(20,20),color='g')
plt.show()
```

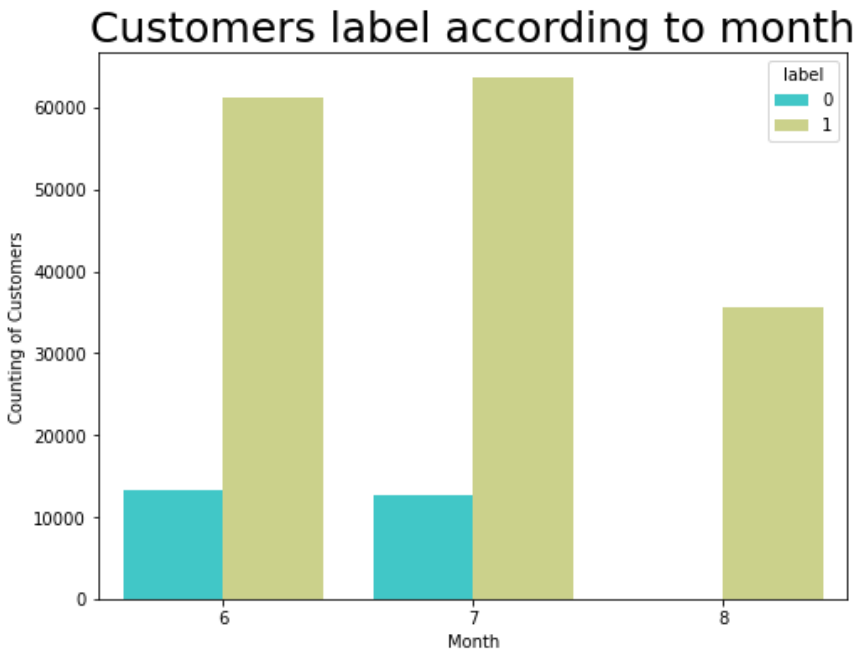
In [34]:

```
# Customer label according to Date.
plt.figure(figsize=(20,8))
sns.countplot(x="pDay", hue='label', data=df, palette='Set2')
plt.title("Customers label according to Date", fontsize=25)
plt.xlabel('Date')
plt.ylabel('Counting of Customers')
plt.show()
```

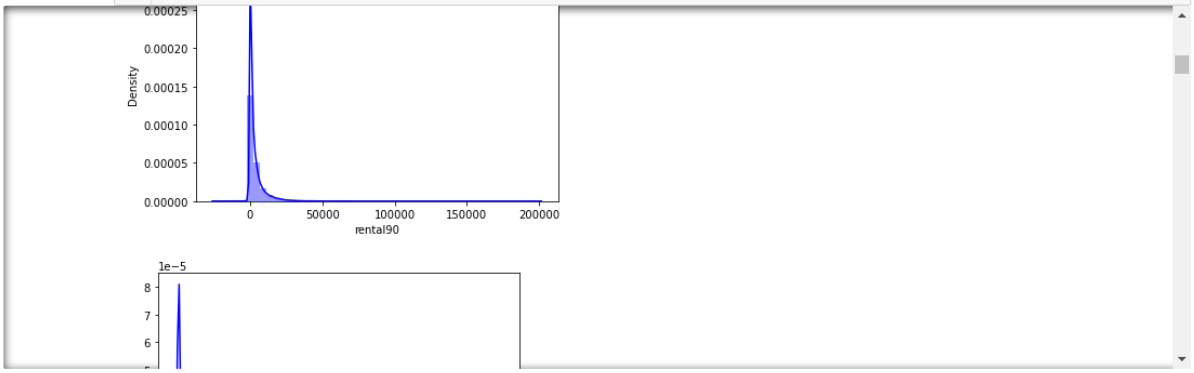


In [35]:

```
# Customer label according to Month.
plt.figure(figsize=(8,6))
sns.countplot(x="pMonth", hue='label', data=df, palette='rainbow')
plt.title("Customers label according to month", fontsize=25)
plt.xlabel('Month')
plt.ylabel('Counting of Customers')
plt.show()
```



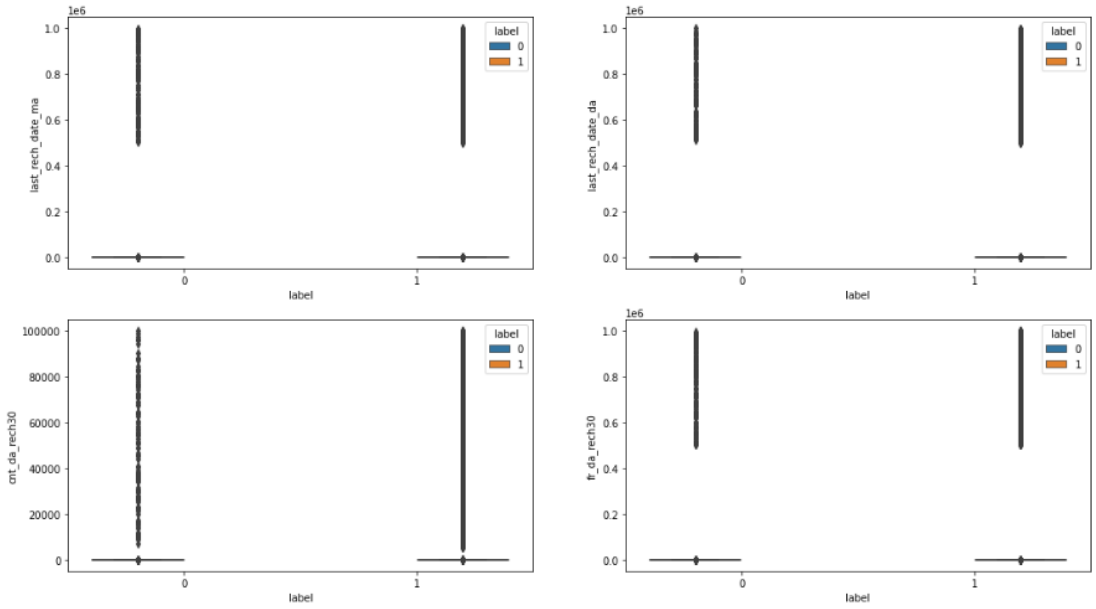
In [36]:
```
# Checking skewness with distplot.
for col in df.describe().columns:
    sns.distplot(df[col],color='b')
    plt.show()
```



- Display boxplot of columns to compare with label.

In [41]:
```
# Plotting outliers
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize = (18, 10))
sns.boxplot(ax=ax1, x = 'label', y = 'last_rech_date_ma', hue = 'label', data = df)
sns.boxplot(ax=ax2, x = 'label', y = 'last_rech_date_da', hue = 'label', data = df)
sns.boxplot(ax=ax3, x = 'label', y = 'cnt_da_rech30', hue = 'label', data = df)
sns.boxplot(ax=ax4, x = 'label', y = 'fr_da_rech30', hue = 'label', data = df)
```

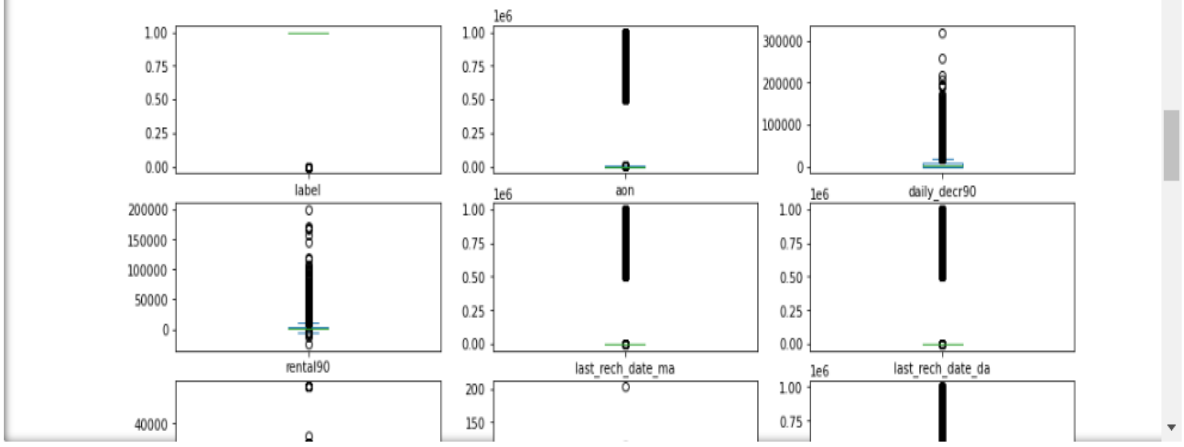
Out[41]: <AxesSubplot:xlabel='label', ylabel='fr_da_rech30'>



- Display outliers of all columns.

In [40]:
```
# checking outliers with boxplot.
df.plot(kind='box', subplots=True, layout=(11,3), figsize=(15,25))
```

```
dtype: object
```



- Label Encoding.

In [48]:
```
#Converting the categorical data into numeric variables
# Transform Non numeric columns into Numeric columns

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for column in df.columns:
    if df[column].dtype==np.number:
        continue
    df[column]=le.fit_transform(df[column])
```

In [49]:
```
df.head()
```

Out[49]:

	label	msisdn	aon	daily_decr90	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30	fr_ma_rech30	...	maxamnt_loans30
0	0	40191	272.0	3065.150000	260.13	2.0	0.0	14	2	21.0	...	6.0
1	1	142291	712.0	12124.750000	3691.26	20.0	0.0	38	1	0.0	...	12.0
2	1	33594	535.0	1398.000000	900.13	3.0	0.0	14	1	0.0	...	6.0
3	1	104157	241.0	21.228000	159.42	41.0	0.0	10	0	0.0	...	6.0
4	1	6910	947.0	150.619333	1098.90	4.0	0.0	23	7	2.0	...	6.0

5 rows × 33 columns

- Data Pre-processing and Scalling the data.

In [50]:
```
#feature importance
#Splitting the data into x and y
x = df.drop(['label'], axis=1)

y = df['label']
```

In [51]:
```
# Scalling the data
from sklearn.preprocessing import StandardScaler

st = StandardScaler()
x = st.fit_transform(x)
x
```

Out[51]:
```
array([[-0.98449283, -0.10377791, -0.25447802, ...,  0.66650703,
         0.28076267,  0.        ],
       [ 0.9145572 , -0.09797818,  0.62163549, ..., -0.51976436,
         1.63266114,  0.        ],
       [-1.10719639, -0.10031125, -0.41570066, ...,  0.54787989,
         1.63266114,  0.        ],
       ...,
       [-0.93473809, -0.10203799, -0.53620811, ...,  0.31062561,
        -1.07113581,  0.        ],
       [ 0.33976442, -0.08453335,  0.66511624, ...,  1.25964272,
         0.28076267,  0.        ],
       [ 0.52343845, -0.08652371, -0.11235279, ..., -0.87564578,
         0.28076267,  0.        ]])
```

- Train-Test Splitting.

In [55]:
```
# Splitting the data into training and testing data
from sklearn.model_selection import train_test_split,cross_val_score
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=42,stratify=y)
```

- Run and evaluate selected models.

In [56]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
KNN = KNeighborsClassifier(n_neighbors=10)
LR = LogisticRegression()
DT = DecisionTreeClassifier(random_state=20)
GNB = GaussianNB()
RF = RandomForestClassifier()
```

In [58]:
```
models = []
models.append(('KNeighborsClassifier', KNN))
models.append(('LogisticRegression', LR))
models.append(('DecisionTreeClassifier',DT))
models.append(('GaussianNB', GNB))
models.append(('RandomForestClassifier', RF))
```

In [59]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,roc_curve,auc
```

In [61]:
```
Model=[]
score=[]
cvs=[]
rocscore=[]

for name,model in models:
    print('**************************',name,'******************************')
    print('\n')

    Model.append(name)
    model.fit(x_train,y_train.values.ravel())
    print(model)

    pre=model.predict(x_test)
    print('\n')

    AS=accuracy_score(y_test,pre)
    print('Accuracy score = ', AS)
    score.append(AS*100)
    print('\n')

    sc=cross_val_score(model,x,y,cv=10,scoring='accuracy').mean()
    print('Cross_val_Score = ', sc)
    cvs.append(sc*100)
    print('\n')

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,pre)
    roc_auc= auc(false_positive_rate, true_positive_rate)
    print('roc_auc_score = ',roc_auc)
    rocscore.append(roc_auc*100)
    print('\n')

    print('classification_report\n',classification_report(y_test,pre))
    print('\n')
```

```
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n')

    plt.figure(figsize=(10,40))
    plt.subplot(911)
    plt.title(name)
    print(sns.heatmap(cm,annot=True))

    plt.subplot(912)
    plt.title(name)
    plt.plot(false_positive_rate, true_positive_rate, label = 'AUC= %0.2f'%roc_auc)
    plt.legend(loc='lower right')

    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    print('\n\n')
```

```
Cross_val_Score =  0.8746583457298369


roc_auc_score =  0.740822571393308


classification_report
              precision    recall  f1-score   support

           0       0.54      0.56      0.55      5172
           1       0.93      0.92      0.93     32077

    accuracy                           0.87     37249
   macro avg       0.73      0.74      0.74     37249
weighted avg       0.87      0.87      0.87     37249
```

- Hypertuning of the Model.

In [63]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [64]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [65]:
```
parameters = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
```

In [67]:
```
rf_random = RandomizedSearchCV(estimator = RF, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

Hypertuning of the model:

In [63]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [64]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [65]:
```
parameters = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
```

In [67]:
```
rf_random = RandomizedSearchCV(estimator = RF, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

In [68]:
```
rf_random.fit(x_train,y_train)
```

```
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.5min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 4.1min
```

Out[68]:
```
RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [69]:
```
rf_random.best_params_
```

Out[69]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [*]:
```
rfc = RandomForestClassifier(n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features='sqrt', max_depth=25)
rfc.fit(x_train,y_train)
rfc.score(x_train,y_train)
pred_decision = rfc.predict(x_test)

rfca = accuracy_score(y_test,pred_decision)
print("Accuracy Score is:",rfca*100)

rfscore = cross_val_score(rfc,x,y,cv=10)
rfc = rfscore.mean()
print("Cross Val Score is:",rfc*100)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,pred_decision)
roc_auc= auc(false_positive_rate, true_positive_rate)
print('roc_auc_score = ',roc_auc)
rocscore.append(roc_auc*100)
print('\n')
```

```
Accuracy Score is: 91.46017342747457
```

- Hardware and Software Requirements and Tools Used

- **Language :-** Python
- **Tool:-** Jupyter
- **OS:-** Windows 10
- **RAM:-** 8gb

CONCLUSION:

- This Kernel investigates different models for car price prediction.
- Different types of Machine Learning methods including LinearRegression, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor and DecisionTreeRegressor in machine learning are compared and analysed for optimal solutions.
- Even though all of those methods achieved desirable results, different models have their own pros and cons.
- The RandomForestRegressor is probably the best one and has been selected for this problem.
- Finally, the RandomForestRegressor is the best choice when parameterization is the top priority.