FLIP ROBO

MICRO CREDIT DEFAULTER MODEL

Submitted by:

Harish Dhami

ACKNOWLEDGEMENT

The internship opportunity I have with Flip Robo Technologies is a great chance for learning and professional development. I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills acknowledge in the best possible way.

I would like to extend my appreciation and thanks for the mentors from DataTrained and professionals from FlipRoboTechnologies who had extended their help and support.

References:

https://sklearn.org/supervised_learning.html#supervised-learning

https://www.datacamp.com/community

https://github.com/mxc19912008/Andrew-Ng-Machine-Learning-Notes

https://www.analyticsvidhya.com/blog/category/machine-learning/

INTRODUCTION

Business Problem:

A client in Telecom Industry is collaborating with an MFI (Microfinance Institution) 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.

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.

Background of domain:

- Microfinance is a banking service provided to unemployed or low-income individuals or groups who otherwise would have no other access to financial services.
- Microfinance allows people to take on reasonable small business loans safely, and in a manner that is consistent with ethical lending practices.
- The majority of micro financing operations occur in developing nations, such as Uganda, Indonesia, Serbia, and Honduras.
- Like conventional lenders, micro financiers charge interest on loans and institute specific repayment plans.
- The World Bank estimates that more than 500 million people have benefited from microfinance-related operations.

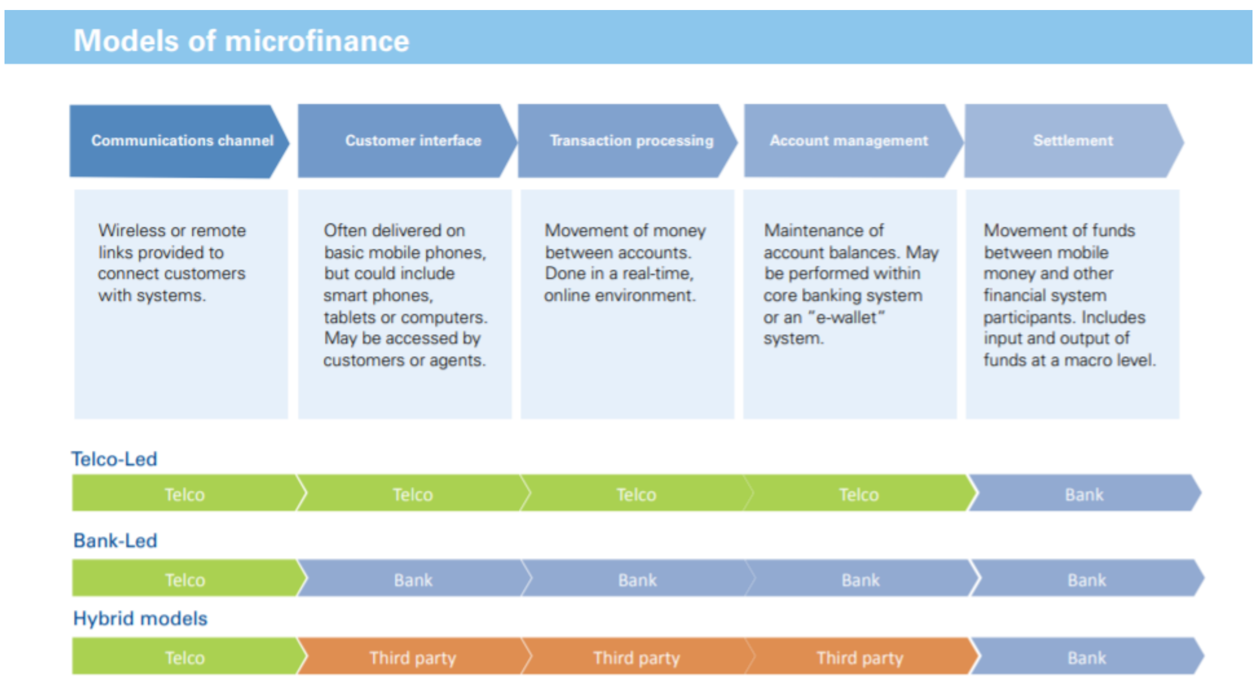
  Indonesia is renowned for its large scale microfinance sector, with a range of commercial banks. More than 56.5 million Micro Small Medium Enterprises1 (MSME), contributed greater than 50% of Gross Domestic Product (GDP) in 2014. However, many of them do not have adequate access to the bank financing they need to grow their businesses, particularly in rural areas.

Some rural communities in Indonesia have no choice but to seek out loans from unregulated moneylenders. Micro lenders, particularly those operating under Indonesian banks, as well as social enterprise startups, are also targeting these communities through their high mobile penetration rates and are developing the right digital platforms to reach out to them.

Only around 22% of Indonesians are connected to formal financial institutions.

Micro-finance is accessible for people in remote areas and on small islands, not just people in the cities.

In 2012, there were 143 million unique mobile subscribers, more than double the number of bank account holders (62 million). Telecommunication operators have more than 300,000 locations at which phone vouchers are sold. Most banks would like to have access to these distribution networks, which would enable them to access the poorest people requiring micro-finance.



MOTIVATION FOR PROBLEM UNDER TAKEN:

Based on data provided from our client database, customer's repayment of loan is assessed based on different factors. By building the model, we can assess which customers are highly likely to repay the loan, thereby it will be useful for those needy people who will repay the loan and also prevent the loss to the customer by avoiding loans to the defaulters.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL MODELLING OF PROBLEM:

Mathematical modeling is simply the method of implementing statistical analysis to a dataset where a Statistical Model is a mathematical representation of observed data.

While analyzing the data, there are an array of statistical models we can choose to utilize.

For the given project, we need to predict whether the customer is a defaulter or not.

This is a classification problem. There are wide varieties of classification models like decision trees, random forests, nearest neighbor, Logistic Regression.

DATA SOURCE AND FORMAT:

The data has been provided by client in a comma separated values(.csv) format.

1. The data will be loaded into pandas dataframe.

```
import pandas as pd
import numpy as np
```

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

2. Checking no. of rows and columns of the data frame and the data type of columns.

```
In [5]:  1 df.shape
Out[5]: (209593, 36)

In [6]:  1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   label                 209593 non-null  int64
 1   msisdn                209593 non-null  object
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64
 10  cnt_ma_rech30         209593 non-null  int64
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64
 16  fr_ma_rech90          209593 non-null  int64
 17  sumamnt_ma_rech90     209593 non-null  int64
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64
 23  fr_da_rech90          209593 non-null  int64
 24  cnt_loans30           209593 non-null  int64
 25  amnt_loans30          209593 non-null  int64
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64
 30  maxamnt_loans90       209593 non-null  int64
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
 34  pcircle               209593 non-null  object
 35  pdate                 209593 non-null  object
dtypes: float64(21), int64(12), object(3)
memory usage: 57.6+ MB
```

This data set has around 2 lakh rows and 36 columns.
There are 3 object columns namely msisdn, pcircle, pdate.
Msisdn is the mobile number of customer, Pcircle is the telecom circle and pdate is the date.

DATA PRE PROCESSING:

Data preprocessing is a technique of converting raw data into useful format.

Data cleaning is a part of preprocessing technique which involves filling missing values.
For the given dataset it has been mentioned that there are no null values.

Firstly, I dealt with object type columns.
Checking what columns are of object type and what type of data is stored in them.

In [7]:

```
#we can observe that almost all columns are of either integer or float type,very few are of object type
#storing object type columns into a separate dataframe

objecttypes=df.select_dtypes(include=['object'])
objecttypes
```

Out[7]:

	msisdn	pcircle	pdate
0	21408I70789	UPW	2016-07-20
1	76462I70374	UPW	2016-08-10
2	17943I70372	UPW	2016-08-19
3	55773I70781	UPW	2016-06-06
4	03813I82730	UPW	2016-06-22
...	...	...	...
209588	22758I85348	UPW	2016-06-17
209589	95583I84455	UPW	2016-06-12
209590	28556I85350	UPW	2016-07-29
209591	59712I82733	UPW	2016-07-25
209592	65061I85339	UPW	2016-07-07

209593 rows × 3 columns

OBSERVATION:

1) The msisdn has numeric entries,so converting this column into integer type.

2)Checking the Pcircle entries ,seems all the rows has same telecom provider name. If all the entries are same in pcircle column, will be deleting the column.

3)Will be converting the pdate column into like year ,month and date.

In [8]:

```
for col in objecttypes.columns:
    print("Number of unique value in ",col,"==>",objecttypes[col].nunique())
    print("\n",col,"\n",objecttypes[col].value_counts())
    print("************************************************")
```

```
Number of unique value in  msisdn ==> 186243

 msisdn
 04581I85330    7
47819I90840    7
30080I90588    6
55809I89238    6
22038I88658    6
              ..
36902I90840    1
17447I88689    1
59686I90584    1
00504I91190    1
65061I85339    1
Name: msisdn, Length: 186243, dtype: int64
************************************************
Number of unique value in  pcircle ==> 1

 pcircle
 UPW    209593
Name: pcircle, dtype: int64
************************************************
Number of unique value in  pdate ==> 82

 pdate
 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
************************************************
```

OBSERVATION:

1)msisdn happens to be cellphone number,but there is I in the 6th place.Usually a mobile number consists of 10 digits. by including I it will be 11 digits.so deleting I.

2)Deleting pcircle column as it has single value.

3)We can notice the data belong to year-2016 ,will be adding the month and date columns.

In [9]:
```
df.drop(['pcircle'],axis=1,inplace=True)
```

In [10]:
```
len(df['msisdn'][0])
```

Out[10]: 11

In [11]:
```
df['msisdn']=df['msisdn'].str.replace('I','')
df['msisdn']
```

Out[11]:
```
0         2140870789
1         7646270374
2         1794370372
3         5577370781
4         0381382730
             ...
209588    2275885348
209589    9558384455
209590    2855685350
209591    5971282733
209592    6506185339
Name: msisdn, Length: 209593, dtype: object
```

In [12]:
```
df['msisdn']=df['msisdn'].astype('int64')
```

In [13]:
```
df['Year']=df['pdate'].str.split('-').str[0]
df['Month']=df['pdate'].str.split('-').str[1]
df['Date']=df['pdate'].str.split('-').str[2]
```

In [14]:
```
df.head()
```

Out[14]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	cnt_loans90	amnt_loa
0	0	2140870789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	...	2.0	
1	1	7646270374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	...	1.0	
2	1	1794370372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	...	1.0	
3	1	5577370781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	...	2.0	
4	1	381382730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	...	7.0	

5 rows × 38 columns

As the date column is splitted into 3 columns,will be deleting the pdate column.

In [15]:
```
#since we have splitted the pdate column,into 3 columns,dropping pdate column
df.drop(['pdate'],axis=1,inplace=True)
```

In [16]:
```
#checking the unique values in year column.
df['Year'].nunique()
```

Out[16]: 1

In [17]:
```
#since all the data collected is about 2016 year,dropping the year column as weel
df.drop(['Year'],axis=1,inplace=True)
```

In [18]:
```
df['Month']=df['Month'].astype(int)
df['Date']=df['Date'].astype(int)
```

Also, the data gathered belongs to 2016 year , hence it won't be impacting the output due to same entry in all the columns. So dropped the year column.

Later converted the month and date columns to integer.

In [15]:
```
#since we have splitted the pdate column,into 3 columns,dropping pdate column
df.drop(['pdate'],axis=1,inplace=True)
```

In [16]:
```
#checking the unique values in year column.
df['Year'].nunique()
```

Out[16]: 1

In [17]:
```
#since all the data collected is about 2016 year,dropping the year column as weel
df.drop(['Year'],axis=1,inplace=True)
```

In [18]:
```
df['Month']=df['Month'].astype(int)
df['Date']=df['Date'].astype(int)
```

Checking whether all the columns are of integer type.

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   label                 209593 non-null  int64
 1   msisdn                209593 non-null  int64
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64
 10  cnt_ma_rech30         209593 non-null  int64
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64
 16  fr_ma_rech90          209593 non-null  int64
 17  sumamnt_ma_rech90     209593 non-null  int64
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64
 23  fr_da_rech90          209593 non-null  int64
 24  cnt_loans30           209593 non-null  int64
 25  amnt_loans30          209593 non-null  int64
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64
 30  maxamnt_loans90       209593 non-null  int64
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
 34  Month                 209593 non-null  int32
 35  Date                  209593 non-null  int32
dtypes: float64(21), int32(2), int64(13)
memory usage: 56.0 MB
```

We can see that all the columns are of numeric type.

Describe method is used to view some basic statistical details like percentile, mean, standard deviation etc. of a data frame or a series of numeric values.

In [20]:
```
1 df.describe()
```

Out[20]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_re
count	209593.000000	2.095930e+05	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209
mean	0.875177	4.974956e+09	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.847800	3712.202921	2
std	0.330519	2.890571e+09	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.892230	53374.833430	2
min	0.000000	4.827380e+05	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	-29.000000	
25%	1.000000	2.465991e+09	246.000000	42.440000	42.692000	280.420000	300.260000	1.000000	0.000000	
50%	1.000000	4.905684e+09	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.000000	0.000000	1
75%	1.000000	7.503370e+09	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.000000	0.000000	2
max	1.000000	9.999895e+09	999860.755168	265926.000000	320630.000000	198926.110000	200148.110000	998650.377733	999171.809410	55

8 rows × 36 columns

We can note that there is a huge difference in 75% value and max value for most of the columns which indicate presence of outliers.

COLUMNS WITH NEGATIVE MINIMUM VALUES:

1)aon

2)daily_decr30=>Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)

3)daily_decr90=>Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah)

4)rental30=>Average main account balance over last 30 days

5)rental90=>Average main account balance over last 90 days

6)last_rech_date_ma=>Number of days till last recharge of main account

7)last_rech_date_da=>Number of days till last recharge of data account

AON:

This column predicts age on cellular network in days.

This columns minimum value should be zero, instead there are negative values might be due to typos.

so checking the other columns values where aon has negative values.

In [21]:
```
df_aon=df[df['aon']<0]
df_aon
```

Out[21]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	maxamnt_loans30
30	1	7013090843	-42.0	8.864333	8.864333	780.71	780.71	780195.497093	0.0	773	...	6.0
197	1	4685890841	-36.0	32.075333	32.075333	1557.53	1557.53	1.000000	0.0	1539	...	6.0
322	1	302770379	-37.0	9.160000	9.160000	203.04	203.04	1.000000	0.0	770	...	6.0
504	1	7567084458	-36.0	11.470333	11.470333	40.04	40.04	1.000000	0.0	770	...	6.0
603	1	5849589235	-35.0	12.474000	12.474000	1823.08	1823.08	1.000000	0.0	1539	...	6.0
...	...	...	...	...	...	...	...	...	...	...	...	...
208716	1	1016290589	-34.0	19.260000	19.260000	714.15	714.15	1.000000	0.0	773	...	6.0
209121	1	6551184450	-18.0	700.580000	700.600000	1057.72	1090.76	1.000000	0.0	770	...	6.0
209240	1	3489670375	-28.0	10.640000	10.640000	133.35	133.35	1.000000	0.0	770	...	6.0
209421	1	5171784459	-44.0	7.166667	7.166667	368.55	368.55	1.000000	0.0	773	...	6.0
209514	0	5050870786	-2.0	1030.000000	1030.000000	634.91	634.91	1.000000	0.0	1539	...	6.0

1539 rows × 36 columns

Converting the aon column to positive.

In [23]:
```
df['aon']=abs(df['aon'])
```

In [24]:
```
#checking for minimum value.
df['aon'].min()
```

Out[24]: 1.0

last_rech_date_ma, last_rech_date_da : these two columns indicate no.of days till last recharge of main and data accounts. This count of days also can't be negative.

Converting them to positive.

In [25]:
```
#no.of days till last recharge of main and data accounts cant be negative.
#converting them into positive values.
df['last_rech_date_ma']=abs(df['last_rech_date_ma'])
df['last_rech_date_da']=abs(df['last_rech_date_da'])
```

In [26]:
```
df['last_rech_date_ma'].min()
```

Out[26]: 0.0

In [27]:
```
df['last_rech_date_da'].min()
```

Out[27]: 0.0

I Created two different data frames in respect to negative values in rental 30 column.

1)One being the people who failed to repay the loan.

2)Other being the people who did repay the loan .

```
In [29]:
#dataframe who failed to pay loan with respect to average rental balance of 30 days
df_renFail=df_ren30[df_ren30['label']==0]
df_renFail[['label','rental90','amnt_loans90']]
```

Out[29]:

	label	rental90	amnt_loans90
24	0	-2020.09	6
246	0	-229.77	6
1469	0	-163.90	24
1777	0	-121.21	6
2881	0	-443.04	18
...	...	...	...
207113	0	-278.64	6
207949	0	-3719.25	6
208543	0	-187.00	6
209175	0	-1128.18	6
209231	0	-158.70	12

398 rows × 3 columns

```
In [30]:
#dataframe who successfully pay loan with respect to average rental balance of 30 days
df_renPaid=df_ren30[df_ren30['label']==1]
df_renPaid[['label','rental90','amnt_loans90']]
```

Out[30]:

	label	rental90	amnt_loans90
41	1	-110.75	12
77	1	-919.80	6
118	1	-177.48	12
117	1	-40.20	12
125	1	-44.88	24
...	...	...	...
209304	1	-281.12	18
209332	1	-505.80	24
209441	1	-891.90	6
209466	1	-441.57	24
209574	1	-18.88	6

5230 rows × 3 columns

We can note that even though the average rental balance is in negatives which means the customer owe rent to company,

Even then they did repay their loans, which is most unlikely.

There might be other possibility that user will not be granted loan if they have negative balance. This might be due to erroneous entry. So converting them to positive.

```
In [31]:
df['rental30']=abs(df['rental30'])
```

The same approach has been followed for rental 90 column.

In [33]:
```
#dataframe who failed to pay loan with respect to average rental balance of 90 days
df_ren90Fail=df_ren90[df_ren90['label']==0]
df_ren90Fail[['label','rental90','amnt_loans90']]
```
Out[33]:

	label	rental90	amnt_loans90
24	0	-2020.09	6
246	0	-229.77	6
1309	0	-83.02	12
1469	0	-163.90	24
1777	0	-121.21	6
...	...	...	...
207113	0	-278.64	6
207949	0	-3719.25	6
208543	0	-187.00	6
209175	0	-1128.16	6
209231	0	-158.70	12

414 rows × 3 columns

In [34]:
```
#dataframe who failed to pay loan with respect to average rental balance of 90 days
df_ren90Pass=df_ren90[df_ren90['label']==1]
df_ren90Pass[['label','rental90','amnt_loans90']]
```
Out[34]:

	label	rental90	amnt_loans90
41	1	-110.75	12
77	1	-919.80	6
116	1	-177.48	12
117	1	-40.20	12
125	1	-44.88	24
...	...	...	...
209304	1	-281.12	18
209332	1	-505.80	24
209441	1	-891.90	6
209466	1	-441.57	24
209574	1	-16.86	6

5515 rows × 3 columns

Converting the rental 90 column negative values to positive values.

In [35]:
```
#converting rental 90 column to positive.As there are negative values for people who had repaid their loans.
df['rental90']=abs(df['rental90'])
```

Checking the entries of maxamnt_loans30 column.

In [36]:
```
df['maxamnt_loans30'].value_counts()
```
Out[36]:
```
6.000000         179193
12.000000         26109
0.000000           3244
17083.998141          1
62511.750702          1
                  ...
30346.385852          1
66821.819056          1
55716.817238          1
41580.156627          1
96927.243252          1
Name: maxamnt_loans30, Length: 1050, dtype: int64
```

It has been mentioned that this columns values has to be either 6 or 12.we can notice that there are huge no.of entries other than 6,12. Ignoring 0 because there might be users who hasn't taken loans. Converting the other numbers to zero beacuse there is no probability of loan repay amount other than 6 ad 12.There are 1047 rows that has values other than 6,12 and0.

In [38]:
```python
#checking the values which have entries other than 6,12,0
df.loc[(df['maxamnt_loans30']!=6.0) & (df['maxamnt_loans30']!=12.0) & (df['maxamnt_loans30']!=0.0),'maxamnt_loans30']
```

Out[38]:
```
118       61907.697372
125       22099.413732
146       98745.934048
369       58925.364061
374       78232.464324
              ...
209189    50824.996349
209262    17324.994582
209331    92864.501728
209392    54259.265687
209424    96927.243252
Name: maxamnt_loans30, Length: 1047, dtype: float64
```

There are 1047 records of values that are other than 6,12 and 0.

Converting these 1047 records to zero because we can't predict their repayment amount.

In [39]:
```python
#converting them to zero
df.loc[(df['maxamnt_loans30']!=6.0) & (df['maxamnt_loans30']!=12.0) & (df['maxamnt_loans30']!=0.0),'maxamnt_loans30']=0.0
```

In [40]:
```python
df['maxamnt_loans30'].value_counts()
```

Out[40]:
```
6.0     179193
12.0     26109
0.0       4291
Name: maxamnt_loans30, dtype: int64
```

Checking the users who haven't taken any loan.

In [41]:
```python
#checking the users who havent taken any Loan.
dff=pd.DataFrame(np.where(df['amnt_loans90']==0))
dff
```

Out[41]:

	0	1	2	3	4	5	6	7	8	9	...	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042
0	127	149	187	212	262	431	441	475	488	570	...	208137	208148	208231	208818	209213	209337	209343	209401	209406	209580

1 rows × 2043 columns

Amt_loans90 column describes the total amount of loans taken by the user in span of 90 days. The presence of zero in this column indicates that the user hasn't taken any loans.

There are 2043 rows in the dataframe with zero in amt_loans90 column.Dropping the rows which has zero in the amt_loans 90 column because such rows wont be useful in predicting the loan repayment.

```
In [42]:  1  #deleting the info of users who havent taken any loan.
          2  df.drop(df[df['amnt_loans90']==0].index,inplace=True)

In [43]:  1  np.where(df['amnt_loans90']==0)
Out[43]: (array([], dtype=int64),)
```

msisdn is nothing but phone number of the user, it has nothing to do with the predictions of loan payment. Hence dropping it.

```
#msisdn is nothing but phone number of the user,it has nothing to do with the predictions of loan payment
#so dropping the msisdn column
df.drop(['msisdn'],axis=1,inplace=True)
```

Hardware and Softwares Used:

Software requirement: Anaconda, Jupyter notebook

Libraries and packages used: Numpy, Pandas, Sklearn,seaborn,Matplotlib,imblearn,scipy.

Model/s Development and Evaluation

Problem-solving approach:

The data set is imbalanced since it has large no. of records which contains data about those repaid the loan and less no. of records of those who defaulted loan.

This might result in biased predictions. So, used imblearn library to reduce the imbalances. The imblearn library provides different approaches one is Random under sampling. In contextof this problem, RandomUnderSampling reduces the no.of records of those who paid the loan. To be precise,random under sampling deletes data from the majority class such that there will be equal no.of samples of both the classes. Hence reduces the bias.

Imbalanced learn

In [99]:
```
#!pip3 install imblearn --trusted-host pypi.org --trusted-host pypi.python.org --trusted-host files.pythonhosted.org --user
```

In [100]:
```
from imblearn.under_sampling import RandomUnderSampler

UnderSample=RandomUnderSampler(sampling_strategy='majority')
x_us,y_us=UnderSample.fit_resample(x,y)

print("Original target database shape:",y.shape)
print("Resample target database shape:",y_us.shape)
```

```
Original target database shape: (207550,)
Resample target database shape: (52324,)
```

In [103]:
```
from collections import Counter
print("Target distribution before sampling:",Counter(y))
print("\nTarget distribution after sampling:",Counter(y_us))
```

```
Target distribution before sampling: Counter({1: 181388, 0: 26162})

Target distribution after sampling: Counter({0: 26162, 1: 26162})
```

OBSERVATION:

Statistical methods used:

Outlier removal : Mostly outliers are removed by either z score or IQR(Inter Quartile Range).Tried both these approaches first but, the data loss is high in both these approaches. It has been mentioned in guidelines that the data loss should not exceed 7%.So applied capping technique which is also called as winsorization.

OUTLIER REMOVAL:

In [86]:
```
from scipy.stats import zscore

z=np.abs(zscore(df))
print(np.where(z>3))
```

```
(array([    21,     22,     22, ..., 207543, 207544, 207544], dtype=int64), array([15, 15, 32, ..., 28, 26, 30], dtype=int64))
```

In [87]:
```
df1=df[(z<3).all(axis=1)]
print("With outliers==>",df.shape)
print("After removing outliers==>",df1.shape)
```

```
With outliers==> (207550, 35)
After removing outliers==> (160365, 35)
```

OBSERVATION:

22% of data removed through z score.

IQR METHOD:

In [88]:
```
from scipy import stats
IQR=stats.iqr(df)
IQR
```

Out[88]: 117.44

In [89]:
```
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
```

In [90]:
```
df_out=df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
print(df_out.shape)
```

```
(31988, 35)
```

OBSERVATION:

Huge amounts of data is removed through IQR ,hence can say IQR is not recommended for outlier removal

In [92]:

```
cols=['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']
for i in cols:
            FloorQ=df[i].quantile(0.10)
            CeilQ=df[i].quantile(0.90)
            df[i] = np.where(df[i] <FloorQ,FloorQ,df[i])
            df[i] = np.where(df[i] >CeilQ,CeilQ,df[i])
            print(i,"->",df[i].skew())
```

```
aon -> 0.5284793206435929
daily_decr30 -> 1.0935453721579609
daily_decr90 -> 1.1543224310681035
rental30 -> 1.1265741276622285
rental90 -> 1.1558700165194147
last_rech_date_ma -> 1.1987272241492852
last_rech_date_da -> 0
last_rech_amt_ma -> 0.8359061321231329
cnt_ma_rech30 -> 0.6277453849914029
fr_ma_rech30 -> 1.0070623988221368
sumamnt_ma_rech30 -> 0.7588124242088926
medianamnt_ma_rech30 -> 0.9295986608192282
medianmarechprebal30 -> 1.3584623130969407
cnt_ma_rech90 -> 0.7670588467394291
fr_ma_rech90 -> 1.5893409860791652
sumamnt_ma_rech90 -> 0.8543081510966238
medianamnt_ma_rech90 -> 0.9684104269012621
medianmarechprebal90 -> 1.258477107838736
cnt_da_rech30 -> 0
fr_da_rech30 -> 0
cnt_da_rech90 -> 0
fr_da_rech90 -> 0
cnt_loans30 -> 0.8910588387118499
amnt_loans30 -> 0.7816448336192705
maxamnt_loans30 -> 2.2568424305116785
medianamnt_loans30 -> 0
cnt_loans90 -> 1.017641178749645
amnt_loans90 -> 0.9564054554835316
maxamnt_loans90 -> 2.224470801656892
medianamnt_loans90 -> 0
payback30 -> 0.9902087443404338
payback90 -> 1.0130726215353174
```

Testing of Identified Approaches (Algorithms):

List of algorithms used:

- Logistic Regression
- Decision Tree Classifier
- KNeighborsClassifier
- RandomForestClassifier
- AdaboostClassifier
- BaggingC;assifier
- GradientBoostingClassifier

Run and Evaluate selected models:

Cross-validation is used to test the model's ability to predict new data that was not used in estimating it. Cross validation used in scenarios where we need to avoid over fitting.

LOGISTIC REGRESSION

In [252]:
```
logreg=LogisticRegression()
logreg_score=cross_val_score(logreg,x_us,y_us,cv=5,scoring='accuracy')
print("cross validation score for svm:",np.mean(logreg_score))
```

```
cross validation score for svm: 0.7720548033272404
```

In [253]:
```
logreg.fit(x_train,y_train)
predicted_logreg=logreg.predict(x_test)
print("Accuracy score::",accuracy_score(y_test,predicted_logreg))
print('Precision: ', precision_score(y_test, predicted_logreg))
print('Recall: ',recall_score(y_test, predicted_logreg))
print('F-measure:',f1_score(y_test, predicted_logreg))
print("Training accuracy::",logreg.score(x_train,y_train))
print("Test accuracy::",logreg.score(x_test,y_test))
```

```
Accuracy score:: 0.7733248392888168
Precision:  0.7854984894259819
Recall:  0.7524018983678666
F-measure: 0.7685940640889204
Training accuracy:: 0.7716290612431184
Test accuracy:: 0.7733248392888168
```

DECISION TREE CLASSIFIER:

In [119]:
```
dtc=DecisionTreeClassifier()
dtc_score=cross_val_score(dtc,x_us,y_us,cv=5,scoring='accuracy')
print("Cross validation score for DECISION TREE CLASSIFIER:",np.mean(dtc_score))
```

```
Cross validation score for dtc: 0.7776355435549691
```

In [115]:
```
dtc.fit(x_train,y_train)
predicted_dtc=dtc.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_dtc))
print("Precision:",precision_score(y_test,predicted_dtc))
print("Recall:",recall_score(y_test,predicted_dtc))
print("F-measure",f1_score(y_test,predicted_dtc))
print("Training accuracy==>",dtc.score(x_train,y_train))
print("Test accuracy==>",dtc.score(x_test,y_test))
```

```
Accuracy score: 0.7780158684195286
Precision: 0.779874213836478
Recall: 0.7750897094571131
F-measure 0.7774746008708272
Training accuracy==> 0.9984026014775936
Test accuracy==> 0.7780158684195286
```

KNeighborsClassifier:

In [116]:
```
knn=KNeighborsClassifier()
knn_score=cross_val_score(knn,x_us,y_us,cv=5,scoring='accuracy')
print("cross validation score for K-Neighbors Classifier:",np.mean(knn_score))
```

```
cross validation score for knn: 0.7809990944768568
```

In [117]:
```
knn.fit(x_train,y_train)
predicted_knn=knn.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_knn))
print("Precision:",precision_score(y_test,predicted_knn))
print("Recall:",recall_score(y_test,predicted_knn))
print("F-measure",f1_score(y_test,predicted_knn))
print("Training accuracy==>",knn.score(x_train,y_train))
print("Test accuracy==>",knn.score(x_test,y_test))
```

```
Accuracy score: 0.7801586841952858
Precision: 0.7982510161349919
Recall: 0.7502025697418683
F-measure 0.7734813223535028
Training accuracy==> 0.8413440967567105
Test accuracy==> 0.7801586841952858
```

RandomForestClassifier:

In [147]:
```
rfc=RandomForestClassifier()
rfc_score=cross_val_score(rfc,x_us,y_us,cv=5,scoring='accuracy')
print("Cross validation score for Random Forest Classifier:",np.mean(rfc_score))
```

```
Cross validation score for Random Forest Classifier: 0.8378755199726481
```

In [122]:
```
rfc.fit(x_train,y_train)
predicted_rfc=rfc.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_rfc))
print("Precision:",precision_score(y_test,predicted_rfc))
print("Recall:",recall_score(y_test,predicted_rfc))
print("F-measure",f1_score(y_test,predicted_rfc))
print("Training accuracy==>",rfc.score(x_train,y_train))
print("Test accuracy==>",knn.score(x_test,y_test))
```

```
Accuracy score: 0.8391150749985522
Precision: 0.8366456059735784
Recall: 0.843037388586642
F-measure 0.8398293357933578
Training accuracy==> 0.9983740765039792
Test accuracy==> 0.7801586841952858
```

Ensemble models in machine learning operate on a similar idea. They combine the decisions from multiple models to improve the overall performance.

The idea behind bagging is combining the results of multiple models to get a generalized result.

Here I have used the following ensemble techniques.

1.ADA BOOST CLASSIFIER

In [123]:
```
adb=AdaBoostClassifier()
adb_score=cross_val_score(adb,x_us,y_us,cv=10,scoring='accuracy')
print("Cross validation score for Ada boost:",np.mean(adb_score))
```

```
Cross validation score for Ada boost: 0.8132979383949541
```

In [125]:
```
adb.fit(x_train,y_train)
predicted_adb=adb.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_adb))
print("Precision:",precision_score(y_test,predicted_adb))
print("Recall:",recall_score(y_test,predicted_adb))
print("F-measure",f1_score(y_test,predicted_adb))
print("Training accuracy==>",adb.score(x_train,y_train))
print("Test accuracy==>",adb.score(x_test,y_test))
```

```
Accuracy score: 0.8120692650721029
Precision: 0.8264342774146696
Recall: 0.7903692557008913
F-measure 0.8079995266552276
Training accuracy==> 0.8141883218758023
Test accuracy==> 0.8120692650721029
```

2.BAGGING CLASSIFIER

In [126]:
```
bgc=BaggingClassifier()
bgc_score=cross_val_score(bgc,x_us,y_us,cv=10,scoring='accuracy')
print("Cross validation score for BAGGING Classifier:",np.mean(bgc_score))
```

```
Cross validation score for BAGGING Classifier: 0.8276700153577246
```

In [128]:
```
bgc.fit(x_train,y_train)
predicted_bgc=bgc.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_bgc))
print("Precision:",precision_score(y_test,predicted_bgc))
print("Recall:",recall_score(y_test,predicted_bgc))
print("F-measure:",f1_score(y_test,predicted_bgc))
print("Training accuracy==>",bgc.score(x_train,y_train))
print("Test accuracy==>",bgc.score(x_test,y_test))
```

```
Accuracy score: 0.8241153645682516
Precision: 0.8493389872786231
Recall: 0.7882856812131034
F-measure: 0.8176742510656181
Training accuracy==> 0.9871922868471347
Test accuracy==> 0.8241153645682516
```

3.Gradient Boosting classifier

In [130]:
```
grbc=GradientBoostingClassifier()
grbc_score=cross_val_score(grbc,x_us,y_us,cv=10,scoring='accuracy')
print("Cross validation score for Gradient Boosting Classifier:",np.mean(grbc_score))
```

```
Cross validation score for Gradient Boosting Classifier: 0.8400159998211774
```

In [131]:
```
grbc.fit(x_train,y_train)
predicted_grbc=grbc.predict(x_test)
print("Accuracy score:",accuracy_score(y_test,predicted_grbc))
print("Precision:",precision_score(y_test,predicted_grbc))
print("Recall:",recall_score(y_test,predicted_grbc))
print("F-measure:",f1_score(y_test,predicted_grbc))
print("Training accuracy==>",grbc.score(x_train,y_train))
print("Test accuracy==>",grbc.score(x_test,y_test))
```

```
Accuracy score: 0.8396363004575201
Precision: 0.8513287048120661
Recall: 0.8232434309526565
F-measure: 0.8370505502265639
Training accuracy==> 0.8441965941181504
Test accuracy==> 0.8396363004575201
```

Key Metrics for success in solving problem under consideration:

A confusion matrix helps us gain an insight into how correct our predictions were and how they hold up against the actual values.

The following metrics are used :

1)Accuracy : Accuracy is the ratio of the total number of correct predictions and the total number of predictions.

2)Precision: Precision is the ratio between the True Positives and all the Positives

3)Recall: The recall is the measure of our model correctly identifying True Positives

4)F1 score: F1 Score is needed when you want to seek a balance between Precision and Recall.

HYPER PARAMETER TUNING:

Hyper parameter tuning is used to increase the performance of the algorithm.

HYPER PARAMETER TUNING:

In [132]:
```
parameters={'n_estimators':[100,200],
    'learning_rate':[0.001,0.01,0.1,0.2,0.5],
    'algorithm':['SAMME', 'SAMME.R']}
```

In [133]:
```
adb_grid=GridSearchCV(AdaBoostClassifier(),parameters,cv=10,scoring='accuracy')
```

In [134]:
```
adb_grid.fit(x_train,y_train)
adb_pred=adb_grid.best_estimator_.predict(x_test)
print("Accuracy after parameter tuning==>",accuracy_score(y_test,adb_pred))
```

```
Accuracy after parameter tuning==> 0.8182081426999479
```

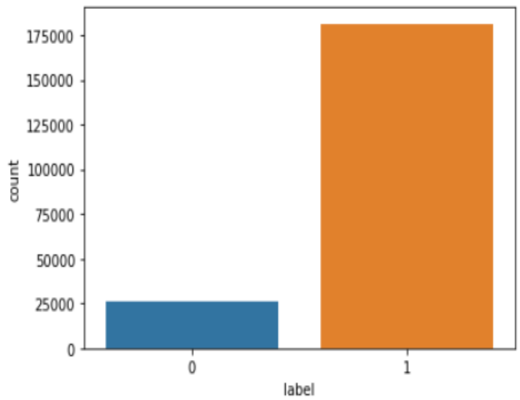
Accuracy after parameter tuning:: 0.8186135402791452 OBSERVATION:

We can observe that accuracy score increased after tuning hyper Parameters

Visualizations:

In [196]:
```
sns.countplot(df['label'])
```

Out[196]: <matplotlib.axes._subplots.AxesSubplot at 0x25a328dce48>



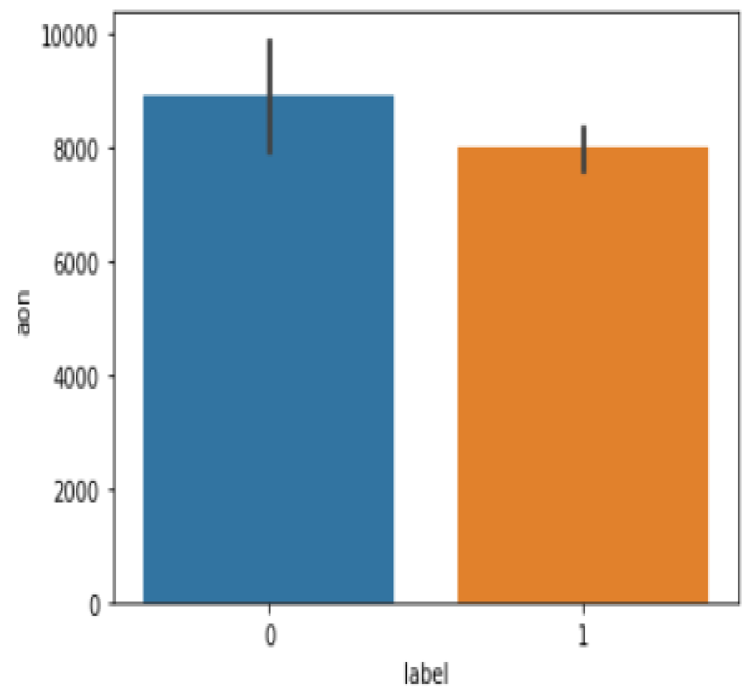
OBSERVATION:

We can note that there is less data about defaulters and more about those who did repay their loan.

Hence can say that the data is imbalanced.

In [197]:
```
sns.barplot(x='label',y='aon',data=df)
```

Out[197]: <matplotlib.axes._subplots.AxesSubplot at 0x25a328b9548>

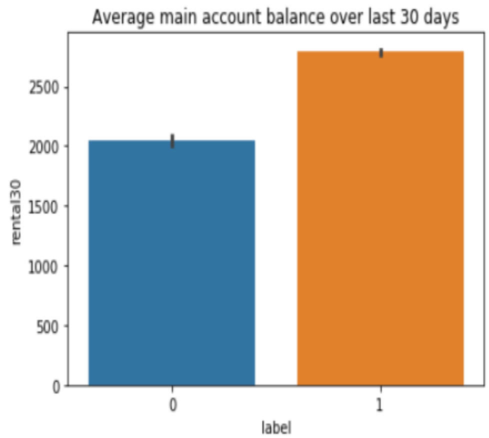


OBSERVATION:

With increase in Age on Network,defaulting rate is higher.

```
In [199]: sns.barplot(x=df['label'],y=df['rental30'])
          plt.title('Average main account balance over last 30 days')

Out[199]: Text(0.5, 1.0, 'Average main account balance over last 30 days')
```



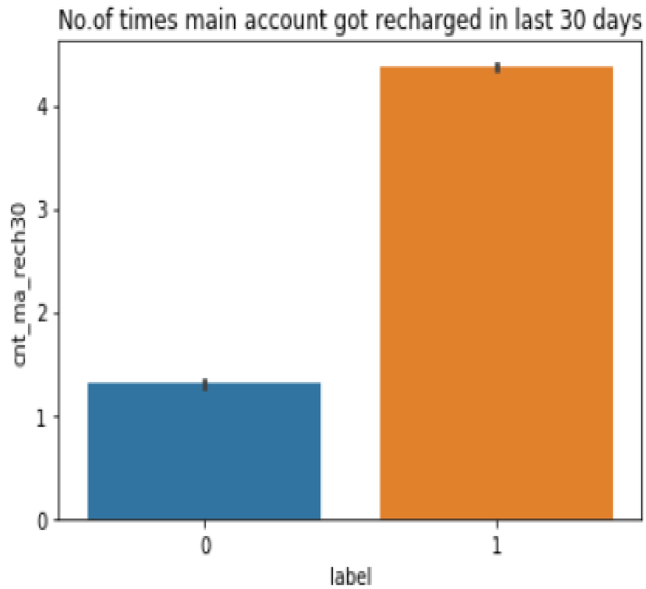
OBSERVATION:

```
There is huge imbalance in the data collected,
when compared to the imbalances,we can note that there is less difference between loan default and repayment.
Hence can say that with the increase in Average Main balance,there is a probability of defaulting.

Defaulters have  max average balance of 2000,repayers has an avg main balance over 2500
```

In [200]:
```
sns.barplot(x=df['label'],y=df['cnt_ma_rech30'])
plt.title('No.of times main account got recharged in last 30 days')
```

Out[200]: Text(0.5, 1.0, 'No.of times main account got recharged in last 30 days')



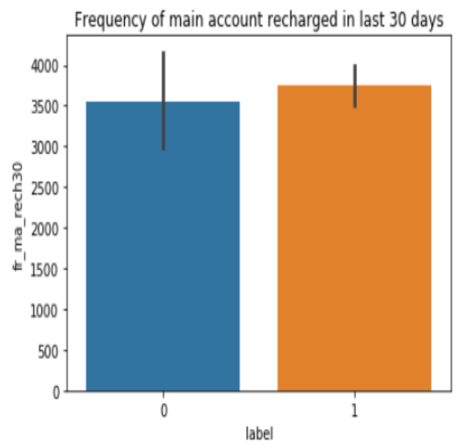
OBSERVATION:

Defaulters recharged Main account max number between 1 and 2 times.

whereas repayers recharged for 4 plus times.

In [201]:
```
sns.barplot(x=df['label'],y=df['fr_ma_rech30'])
plt.title('Frequency of main account recharged in last 30 days')
```

Out[201]: Text(0.5, 1.0, 'Frequency of main account recharged in last 30 days')

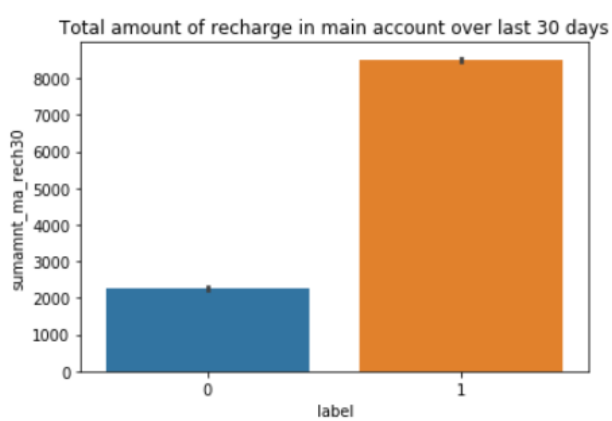


OBSERVATION:

With increase in frequency of Recharge in last 30 days,equal probabilities of defaulting and repayment.
Even though there is less data about defaulting,there is high chance of defaulting with incresed recharge fdrequency.

In [202]:
```
sns.barplot(x=df['label'],y=df['sumamnt_ma_rech30'])
plt.title('Total amount of recharge in main account over last 30 days')
```

Out[202]: Text(0.5, 1.0, 'Total amount of recharge in main account over last 30 days')

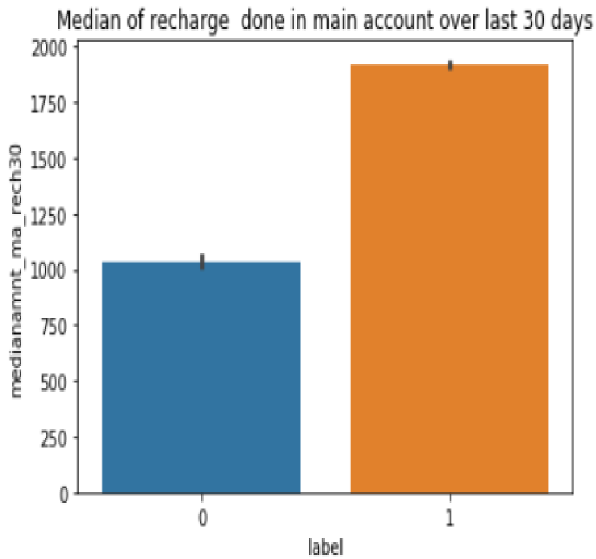


OBSERVATION:

the defaulters has max limit ranging between 2000 and 3000 of Total recharge amount.

In [203]:
```
sns.barplot(x=df['label'],y=df['medianamnt_ma_rech30'])
plt.title('Median of recharge  done in main account over last 30 days')
```

Out[203]:
```
Text(0.5, 1.0, 'Median of recharge  done in main account over last 30 days')
```

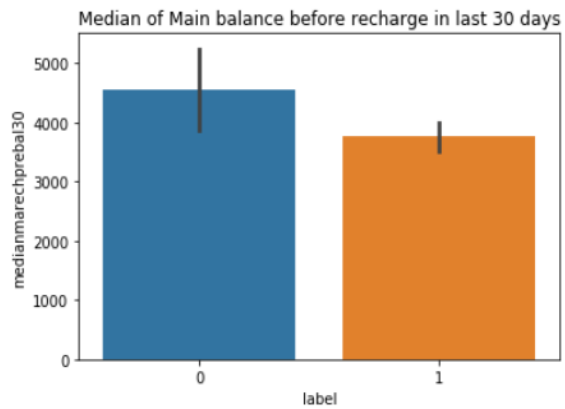


OBSERVATION:

On an average the defaulters has recharged for a max of 1000 indonesian rupaiah.

```
In [204]: sns.barplot(x=df['label'],y=df['medianmarechprebal30'])
          plt.title('Median of Main balance before recharge in last 30 days')
```

```
Out[204]: Text(0.5, 1.0, 'Median of Main balance before recharge in last 30 days')
```



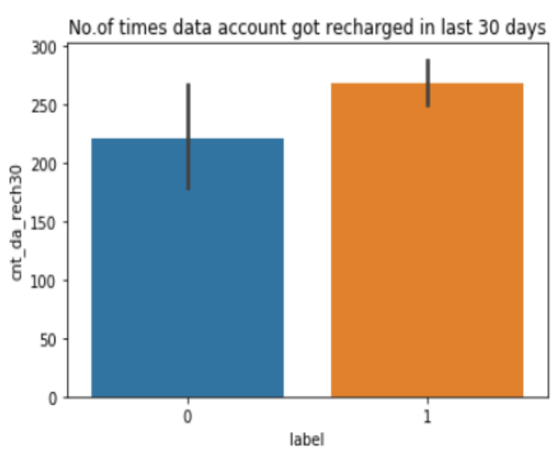
OBSERVATION:

Defaulters has a medianMain account recharge amount ranging between 4000 and 5000.

2)With increase in Median of Main balance recharge,probability of defaulting is very high.

```
In [205]: sns.barplot(x=df['label'],y=df['cnt_da_rech30'])
          plt.title('No.of times data account got recharged in last 30 days')
```

```
Out[205]: Text(0.5, 1.0, 'No.of times data account got recharged in last 30 days')
```



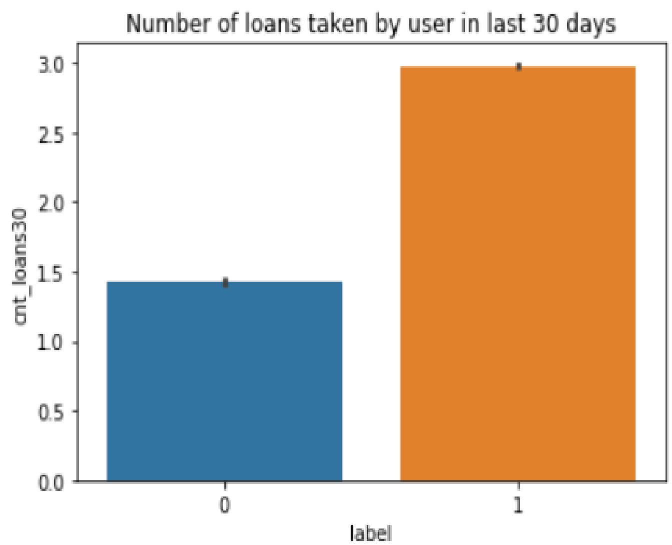
OBSERVATION:

1)Defaulters has recharged the data account for a maximum of 200 to 250 times.

2)With increase in No.of times data accounts recharge,probability of defaulting is high.

In [207]:
```
sns.barplot(x=df['label'],y=df['cnt_loans30'])
plt.title('Number of loans taken by user in last 30 days')
```

Out[207]: Text(0.5, 1.0, 'Number of loans taken by user in last 30 days')



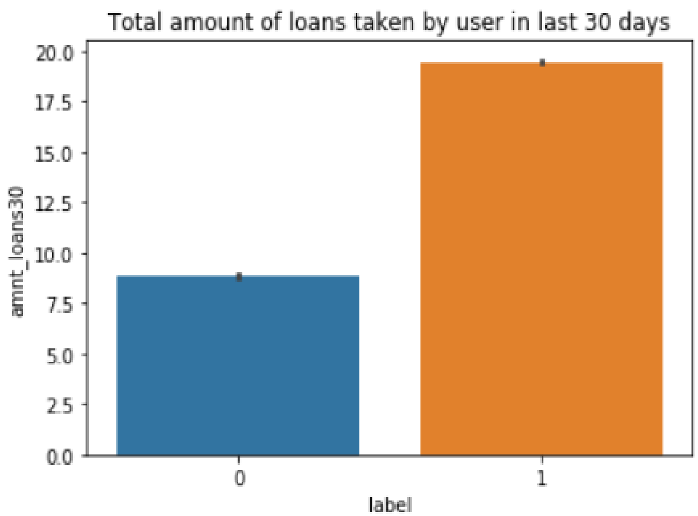
OBSERVATION:

1)Defaulters has taken between 1 to 1.5 no.of loans.
As practically there will be no 1.5 loan,considering only 1 loan.

2)Those who repaid had taken maxof 3 loans.

In [208]:
```
sns.barplot(x=df['label'],y=df['amnt_loans30'])
plt.title('Total amount of loans taken by user in last 30 days')
```

Out[208]:
```
Text(0.5, 1.0, 'Total amount of loans taken by user in last 30 days')
```



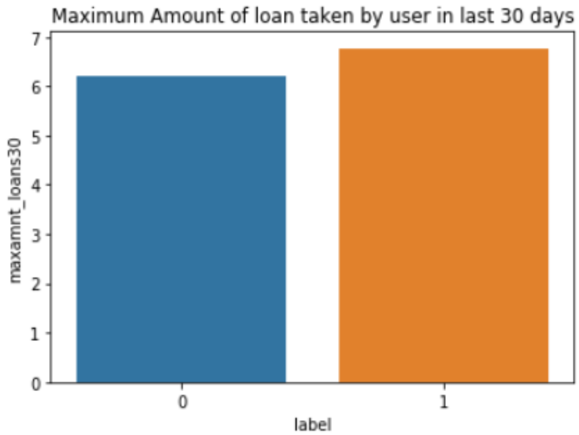
OBSERVATION:

```
1)Total Amount of loans took by Defaulters varies between 7.5 and 10.

2)Repayers has took 20 loans which tends to be the max limit.
```

n [209]:
```
sns.barplot(x=df['label'],y=df['maxamnt_loans30'])
plt.title('Maximum Amount of loan taken by user in last 30 days')
```

ut[209]:
```
Text(0.5, 1.0, 'Maximum Amount of loan taken by user in last 30 days')
```



OBSERVATION:

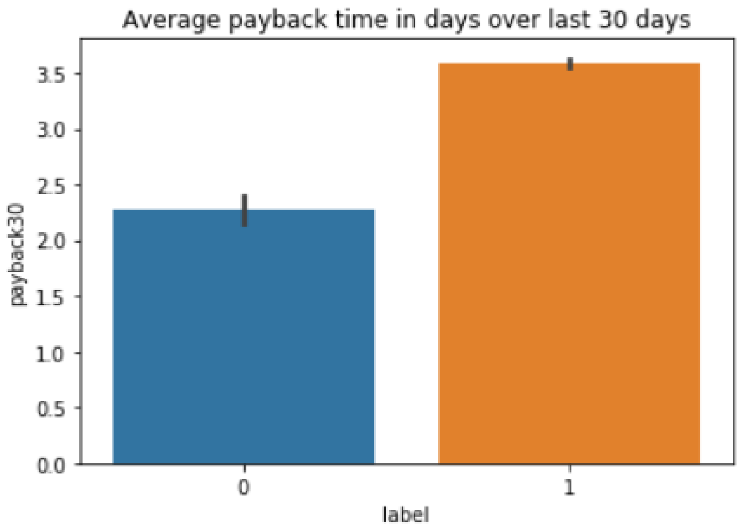
A user can take Maximum of 7 loans in 30 days.

Defaulters took 6 loans whereas repayers took 7 loans.

Can say that there not a much difference in loans took by both defaulters and repayers.

In [211]:
```
sns.barplot(x=df['label'],y=df['payback30'])
plt.title('Average payback time in days over last 30 days')
```

Out[211]:
```
Text(0.5, 1.0, 'Average payback time in days over last 30 days')
```



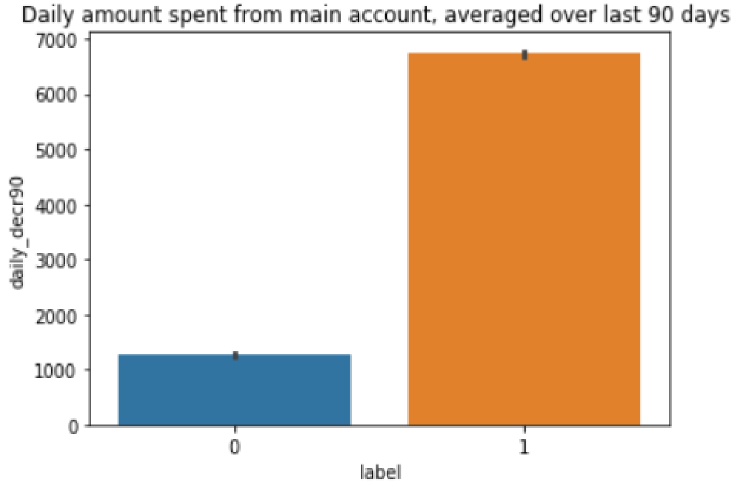
OBSERVATION:

```
A potentail defaulter might repay in 2 days.

Repayers took average of 3.5 days.
```

In [212]:
```
sns.barplot(x=df['label'],y=df['daily_decr90'])
plt.title('Daily amount spent from main account, averaged over last 90 days ')
```

Out[212]:
```
Text(0.5, 1.0, 'Daily amount spent from main account, averaged over last 90 days ')
```



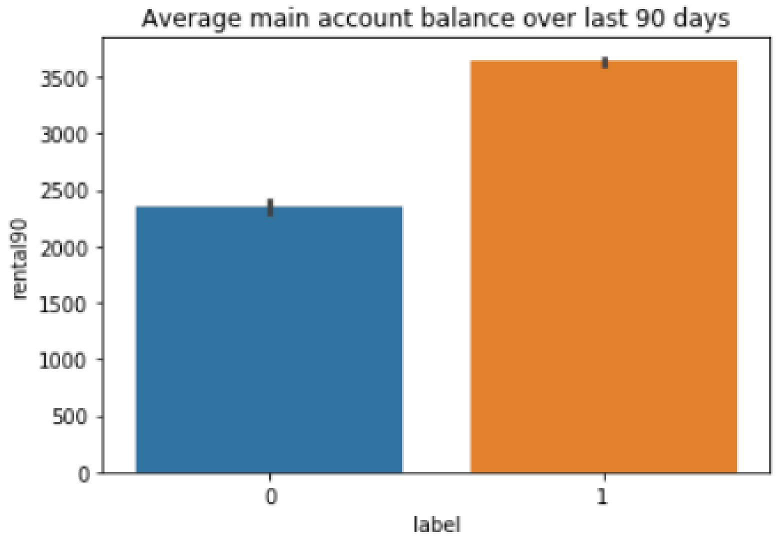
OBSERVATION: In terms of daily spending from main account in span of 90 days,

```
1)the defaulters has spent a little above 1000

2)Repayers has spent 7000 rupaiah.
```

In [213]:
```
sns.barplot(x=df['label'],y=df['rental90'])
plt.title('Average main account balance over last 90 days')
```

Out[213]: `Text(0.5, 1.0, 'Average main account balance over last 90 days')`



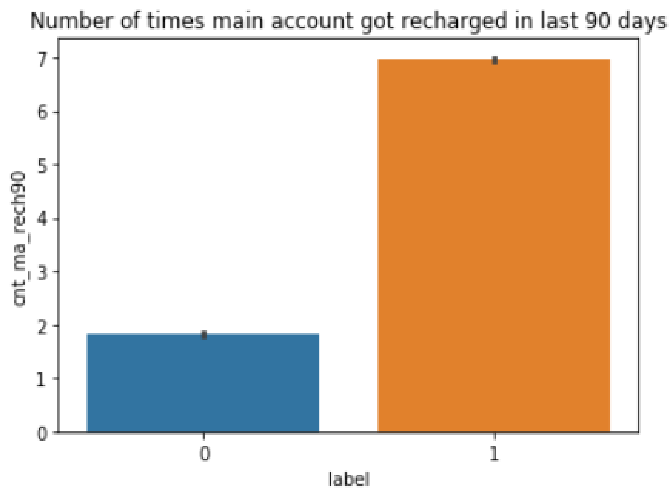
OBSERVATION: In terms of Average balance over 90 days,

```
1)Defaullters average=2000 to 2500

2)Repayers average= 3500
```

In [214]:
```
sns.barplot(x=df['label'],y=df['cnt_ma_rech90'])
plt.title('Number of times main account got recharged in last 90 days')
```

Out[214]:
```
Text(0.5, 1.0, 'Number of times main account got recharged in last 90 days')
```



OBSERVATION:

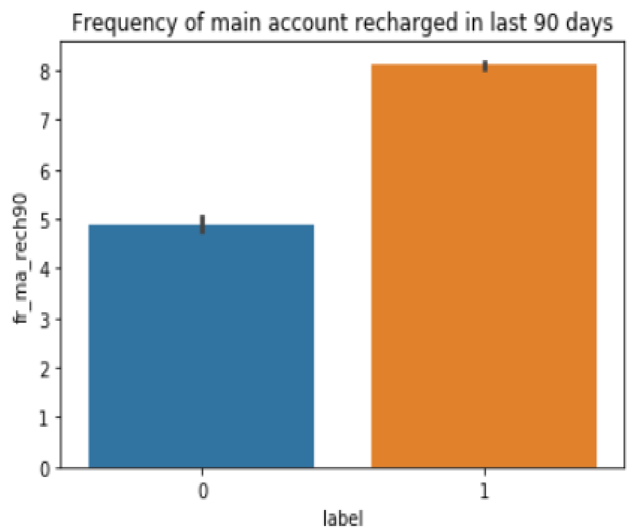
```
In terms of No.of times Main accounts recharged in 90 days,

defaulters recharged for 2 times.

repayers recharged for 7 times.
```

In [215]:
```
sns.barplot(x=df['label'],y=df['fr_ma_rech90'])
plt.title('Frequency of main account recharged in last 90 days')
```

Out[215]:
```
Text(0.5, 1.0, 'Frequency of main account recharged in last 90 days')
```



OBSERVATION:

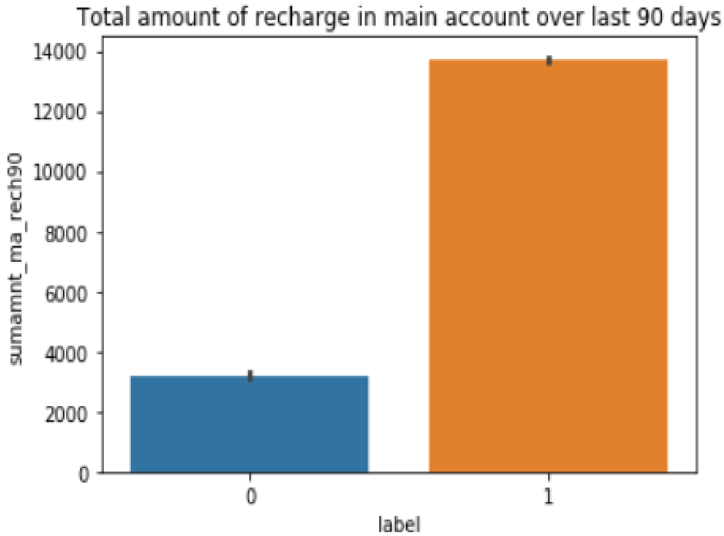
```
In terms of Frequency of Main Account recharge in 90 days period,

defaulters frequency is 5.

repayers frequency is 8.
```

In [216]:
```
sns.barplot(x=df['label'],y=df['sumamnt_ma_rech90'])
plt.title('Total amount of recharge in main account over last 90 days')
```

Out[216]:
```
Text(0.5, 1.0, 'Total amount of recharge in main account over last 90 days')
```



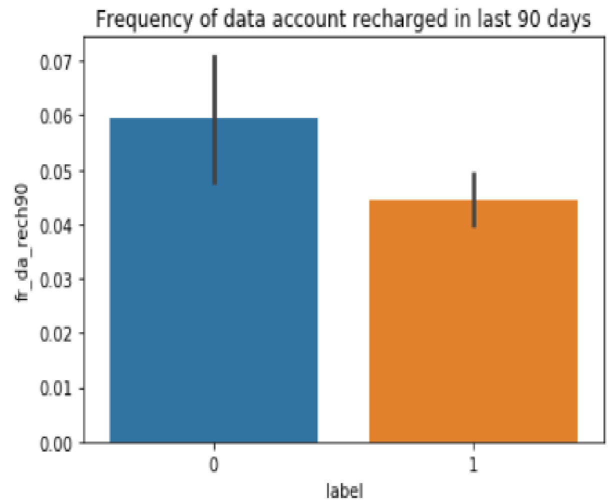
OBSERVATION:

In terms of Total recharge amount in 90 days,

defaulters recharge amount varies from 2000 to 4000.
Repayers recharged for 14000.

```
In [219]: sns.barplot(x=df['label'],y=df['fr_da_rech90'])
          plt.title('Frequency of data account recharged in last 90 days')

Out[219]: Text(0.5, 1.0, 'Frequency of data account recharged in last 90 days')
```

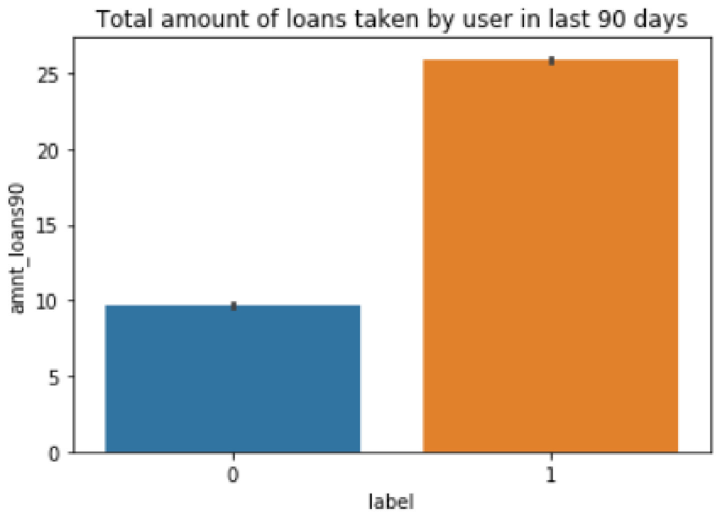


OBSERVATION:

With increase in frequency of Data account recharge in 90 days,defaulting rate is high.

```
In [221]: sns.barplot(x=df['label'],y=df['amnt_loans90'])
          plt.title('Total amount of loans taken by user in last 90 days')

Out[221]: Text(0.5, 1.0, 'Total amount of loans taken by user in last 90 days')
```



OBSERVATION:

In total amount of loans users took,

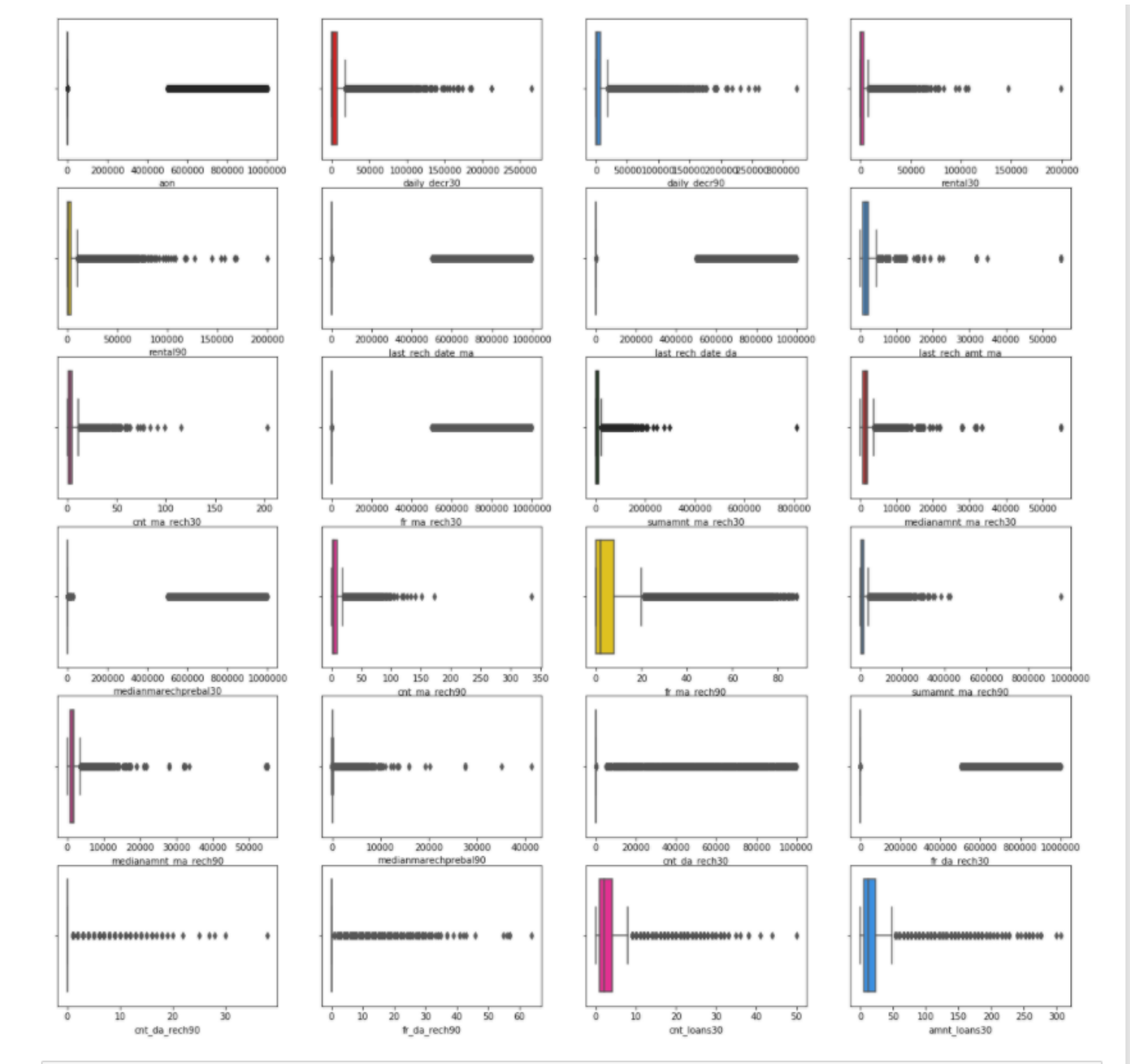
1)defaulters took max of 10 loans.

2)Repayers took 25 max of loans.

CHECKING FOR OUTLIERS:

[226]:
```
#checking outliers in columns
fig, ((ax1, ax2,ax3,ax4),(ax5,ax6,ax7, ax8),(ax9,ax10,ax11,ax12),(ax13,ax14,ax15,ax16),(ax17,ax18,ax19,ax20),(ax21,ax22,ax23,ax24


sns.boxplot(df['aon'] , color="g",ax=ax1)
sns.boxplot(df['daily_decr30'] , color="r",ax=ax2)
sns.boxplot(df['daily_decr90'] , color="dodgerblue",ax=ax3)
sns.boxplot(df['rental30'] , color="deeppink",ax=ax4)
sns.boxplot(df['rental90'] , color="gold",ax=ax5)
sns.boxplot(df['last_rech_date_ma'] , color="dodgerblue",ax=ax6)
sns.boxplot(df['last_rech_date_da'] , color="deeppink",ax=ax7)
sns.boxplot(df['last_rech_amt_ma'] , color="dodgerblue",ax=ax8)
sns.boxplot(df['cnt_ma_rech30'] , color="deeppink",ax=ax9)
sns.boxplot(df['fr_ma_rech30'] , color="dodgerblue",ax=ax10)
sns.boxplot(df['sumamnt_ma_rech30'] , color="g",ax=ax11)
sns.boxplot(df['medianamnt_ma_rech30'] , color="r",ax=ax12)
sns.boxplot(df['medianmarechprebal30'] , color="dodgerblue",ax=ax13)
sns.boxplot(df['cnt_ma_rech90'] , color="deeppink",ax=ax14)
sns.boxplot(df['fr_ma_rech90'] , color="gold",ax=ax15)
sns.boxplot(df['sumamnt_ma_rech90'] , color="dodgerblue",ax=ax16)
sns.boxplot(df['medianamnt_ma_rech90'] , color="deeppink",ax=ax17)
sns.boxplot(df['medianmarechprebal90'] , color="dodgerblue",ax=ax18)
sns.boxplot(df['cnt_da_rech30'] , color="deeppink",ax=ax19)
sns.boxplot(df['fr_da_rech30'] , color="dodgerblue",ax=ax20)
sns.boxplot(df['cnt_da_rech90'] , color="deeppink",ax=ax21)
sns.boxplot(df['fr_da_rech90'] , color="dodgerblue",ax=ax22)
sns.boxplot(df['cnt_loans30'] , color="deeppink",ax=ax23)
sns.boxplot(df['amnt_loans30'] , color="dodgerblue",ax=ax24)
```



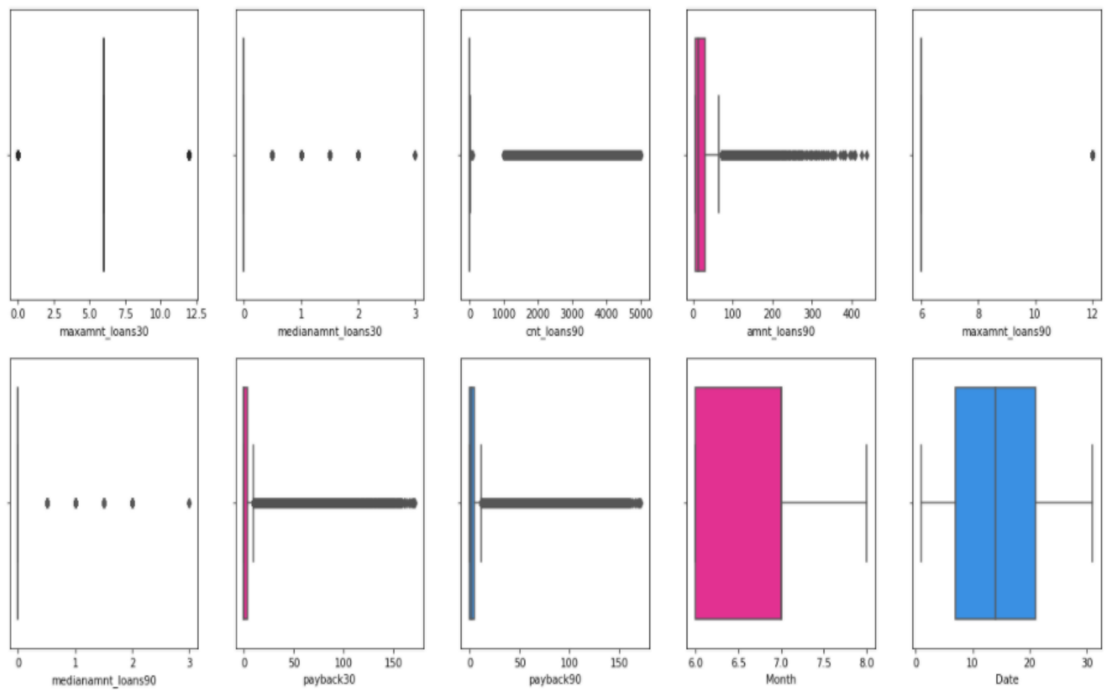
We can note that there are outliers in almost every column.

In [227]:

```
fig, ((ax1, ax2,ax3,ax4,ax5),(ax6,ax7, ax8,ax9,ax10)) = plt.subplots(nrows=2, ncols=5, figsize = (20, 10))


sns.boxplot(df['maxamnt_loans30'] , color="g",ax=ax1)
sns.boxplot(df['medianamnt_loans30'] , color="r",ax=ax2)
sns.boxplot(df['cnt_loans90'] , color="dodgerblue",ax=ax3)
sns.boxplot(df['amnt_loans90'] , color="deeppink",ax=ax4)
sns.boxplot(df['maxamnt_loans90'] , color="gold",ax=ax5)
sns.boxplot(df['medianamnt_loans90'] , color="dodgerblue",ax=ax6)
sns.boxplot(df['payback30'] , color="deeppink",ax=ax7)
sns.boxplot(df['payback90'] , color="dodgerblue",ax=ax8)
sns.boxplot(df['Month'] , color="deeppink",ax=ax9)
sns.boxplot(df['Date'] , color="dodgerblue",ax=ax10)
```

Out[227]: <matplotlib.axes._subplots.AxesSubplot at 0x25a19e26348>



Interpretation of the Results:

- We can note that there is less data about defaulters and more about those who did repay their loan. Hence can say that the data is imbalanced
- With increase in Age on Network, defaulting rate is higher.
- The data is collected based on different parameters for two time periods. One observation is for 30 days and other is for 90 days. Analyzing the parameters separately.
- **For 30 days:**
  1) With the increase in Average Main balance, there is a probability of defaulting.
  2) Defaulters recharged Main account max number between 1 and 2 times. Whereas re-payers recharged for 4 plus times.
  3) On an average the defaulters has recharged for a max of 1000 Indonesian Rupaiah for Main Balance.
  4) Defaulters has recharged the data account for a maximum of 200 to 250 times. With increase in No.of times data accounts recharge, probability of defaulting is high
  5) A defaulter may default after 2 days , re payers took average of 3.5 days.
  6) Defaulters took 1 loan, re payers took 3 loans.
- **FOR 90 DAYS:**
  1) the defaulters has spent a max of 1000 from main account, Repayers has spent 7000 rupaiah.
  2) Defaulters average main account balance =2000 to 2500 Repayers average main account balance = 3500
  3) Defaulters recharged main account for 2 times. Re-payers recharged main account for 7 times.
  4) Defaulters frequency of main account recharge is 5, Re-payers frequency of main account recharge is 8.

CONCLUSION

- **Key Findings and Conclusions of the Study**:

  The defaulting rate is higher in old customers. Defaulters recharge for the main account less no.of times but does recharge for data account more no.of times.

  Re payers recharge the main account more no.of times when compared to defaulters.

- **Learning Outcomes of the Study in respect of Data Science**

One of the challenge i faced while data cleaning is outlier removal, in most of the scenarios Z-score will be used as outlier removal technique since it performs quite well with less data loss. In our data set, Z-score has caused 22% data loss. Then I tried another famous technique called InterQuartileRange it caused around 80% data loss.

Another technique is replacing the outlier data with mean or median. But when we observe this data set there is a huge difference between minimum and maximum values. If we calculate mean or median it won't give appropriate values as it includes the outlier value (maximum ones).So not using this approach.

As we are not dropping the outliers, another approach is capping or winsorization of outliers .Using percentile capping. Values that are less than the value at 10th percentile are replaced by 10th percentile value, and the value greater than 90th percentile are replaced by 90th percentile value.

The other challenge is when I used the imbalanced data, the accuracy was very high but there was bias in predictions. So I used imblearn to reduce the imbalances in the target variable.

Limitations of this work and Scope for Future Work:

This data set contains data of the year 2016 belonging to psw telecom circle.

If we get data of other years along with other telecom companies we can predict on varied scenarios.