“MICRO CREDIT DEFAULTER PROJECT”

Submitted by:

SUJIT KUMAR

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://www.geeksforgeeks.org/machine-learning/

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 microfinancing operations occur in developing nations, such as Uganda, Indonesia, Serbia, and Honduras.
- Like conventional lenders, microfinanciers 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

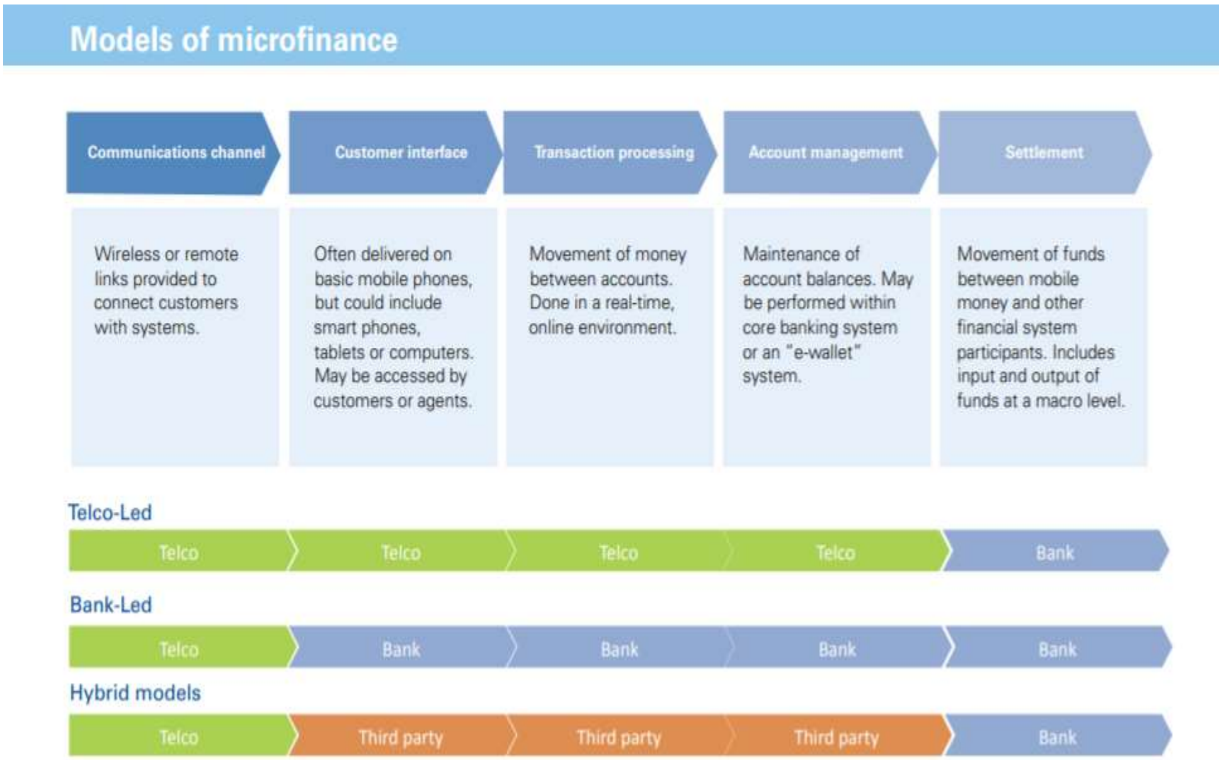
2014. However, many of them do not have adequateacces to the bank financing they need to grow their businesses, particularly in rural areas.

Some rural communities in Indonesia have no choice but to seek out loans from unregulated money lenders. 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.

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]: df=pd.read_csv("Micro Credit Defaulter Data file.csv")
```

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

```
In [7]: 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
```

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 [8]:

```
#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[8]:

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

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 [9]:

```
for col in objecttypes.columns:

      print("No.of unique values in ",col,"::",objecttypes[col].nunique())

      print("\n",col," \n ",objecttypes[col].value_counts())
      print("************************************************")
```

```
No.of unique values in  msisdn :: 186243

 msisdn
 04581I85330    7
47819I90840    7
43096I88688    6
22038I88658    6
30080I90588    6
               ..
51297I89231    1
98688I70377    1
99064I90840    1
47211I90580    1
69063I90849    1
Name: msisdn, Length: 186243, dtype: int64
************************************************
No.of unique values in  pcircle :: 1

 pcircle
 UPW    209593
Name: pcircle, dtype: int64
************************************************
No.of unique values in  pdate :: 82

 pdate
 04-07-2016    3150
05-07-2016    3127
07-07-2016    3116
20-06-2016    3099
17-06-2016    3082
              ...
04-06-2016    1559
18-08-2016    1407
19-08-2016    1132
```

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 [10]:
```
df.drop(['pcircle'], axis=1,inplace=True)
```

In [11]:
```
len(df['msisdn'][0])
```

Out[11]: 11

In [12]:
```
df['msisdn'] = df['msisdn'].str.replace('I', '')
```

In [13]:
```
df['msisdn']
```

Out[13]:
```
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 [14]:
```
df['msisdn'] =df['msisdn'].astype('int64')
```

In [15]:
```
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 [16]:
```
df.head()
```

Out[16]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	cnt_loans9
0	0	2140870789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	...	2.
1	1	7646270374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	...	1.
2	1	1794370372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	...	1.
3	1	5577370781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	...	2.
4	1	381382730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	...	7.

5 rows × 38 columns

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

```
In [17]: #since we have splitted the pdate column,into 3 columns,dropping pdate column
         df.drop(['pdate'],axis=1,inplace=True)

In [18]: #checking the unique values in year column.
         df['Year'].nunique()

Out[18]: 31
```

```
In [19]: #since all the data collected is about 2016 year,dropping the year column as weel.
         df.drop(['Year'],axis=1,inplace=True)

In [20]: 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.

Checking whether all the columns are of integer type.

```
In [21]:  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  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 [22]:
```
df.describe()
```

Out[22]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_
count	209593.000000	2.095930e+05	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.00000	209593.000
mean	0.875177	4.974956e+09	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.84780	3712.202
std	0.330519	2.890571e+09	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.89223	53374.833
min	0.000000	4.827380e+05	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.00000	-29.000
25%	1.000000	2.465991e+09	246.000000	42.440000	42.692000	280.420000	300.260000	1.00000	0.000
50%	1.000000	4.905684e+09	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.00000	0.000
75%	1.000000	7.503370e+09	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.00000	0.000
max	1.000000	9.999895e+09	999860.755200	265926.000000	320630.000000	198926.110000	200148.110000	998650.37770	999171.809

8 rows × 36 columns

In [23]:
```
df_aon = df[df['aon'] < 0]
```

In [24]:
```
df_aon
```

Out[24]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	maxam
30	1	7013090843	-42.0	8.864333	8.864333	780.71	780.71	780195.4971	0.0	773	...	
197	1	4685890841	-36.0	32.075333	32.075333	1557.53	1557.53	1.0000	0.0	1539	...	
322	1	302770379	-37.0	9.160000	9.160000	203.04	203.04	1.0000	0.0	770	...	
504	1	7567084458	-36.0	11.470333	11.470333	40.04	40.04	1.0000	0.0	770	...	
603	1	5849589235	-35.0	12.474000	12.474000	1823.08	1823.08	1.0000	0.0	1539	...	
...	...	...	...	...	...	...	...	...	...	...	...	
208716	1	1016290589	-34.0	19.260000	19.260000	714.15	714.15	1.0000	0.0	773	...	
209121	1	6551184450	-18.0	700.580000	700.600000	1057.72	1090.76	1.0000	0.0	770	...	
209240	1	3489670375	-28.0	10.640000	10.640000	133.35	133.35	1.0000	0.0	770	...	
209421	1	5171784459	-44.0	7.166667	7.166667	368.55	368.55	1.0000	0.0	773	...	
209514	0	5050870786	-2.0	1030.000000	1030.000000	634.91	634.91	1.0000	0.0	1539	...	

1539 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 [23]:  1 df_aon = df[df['aon'] < 0]
```

```
In [24]:  1 df_aon
```

Out[24]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	...	maxam
30	1	7013090843	-42.0	8.864333	8.864333	780.71	780.71	780195.4971	0.0	773	...	
197	1	4685890841	-36.0	32.075333	32.075333	1557.53	1557.53	1.0000	0.0	1539	...	
322	1	302770379	-37.0	9.160000	9.160000	203.04	203.04	1.0000	0.0	770	...	
504	1	7567084458	-36.0	11.470333	11.470333	40.04	40.04	1.0000	0.0	770	...	
603	1	5849589235	-35.0	12.474000	12.474000	1823.08	1823.08	1.0000	0.0	1539	...	
...	...	...	...	...	...	...	...	...	...	...	...	
208716	1	1016290589	-34.0	19.260000	19.260000	714.15	714.15	1.0000	0.0	773	...	
209121	1	6551184450	-18.0	700.580000	700.600000	1057.72	1090.76	1.0000	0.0	770	...	
209240	1	3489670375	-28.0	10.640000	10.640000	133.35	133.35	1.0000	0.0	770	...	
209421	1	5171784459	-44.0	7.166667	7.166667	368.55	368.55	1.0000	0.0	773	...	
209514	0	5050870786	-2.0	1030.000000	1030.000000	634.91	634.91	1.0000	0.0	1539	...	

1539 rows × 36 columns

Converting the aon column to positive.

```
In [26]:  1 df['aon']=abs(df['aon'])
```

```
In [27]:  1 #checking for minimum value.
          2 df['aon'].min()
Out[27]: 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 [28]:  1 #no.of days till last recharge of main and data accounts cant be negative.
          2 #converting them into positive values.
          3 df['last_rech_date_ma']=abs(df['last_rech_date_ma'])
          4 df['last_rech_date_da']=abs(df['last_rech_date_da'])
```

```
In [29]:  1 df['last_rech_date_ma'].min()
Out[29]: 0.0
```

```
In [30]:  1 df['last_rech_date_da'].min()
Out[30]: 0.0
```

I Created two different dataframesin 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 [32]:
```
#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[32]:

	label	rental90	amnt_loans90
24	0	-2020.09	6
246	0	-229.77	6
1469	0	-163.90	24
1777	0	-121.21	6
2681	0	-443.04	18
...	...	...	...
207113	0	-278.64	6
207949	0	-3719.25	6
208543	0	-187.00	6
209175	0	-1126.16	6
209231	0	-158.70	12

398 rows × 3 columns

In [33]:
```
#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[33]:

	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	-691.90	6
209466	1	-441.57	24
209574	1	-16.86	6

5230 rows × 3 columns

We can note that eventhough 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.

The same approach has been followed for rental 90 column.

In [34]:
```
df['rental30']=abs(df['rental30'])
```

In [35]:
```
#creating a dataframe which has negative values in rental90 column
df_ren90=df[df['rental90']<0]
df_ren90[['label','rental90','amnt_loans90']]
```

Out[35]:

	label	rental90	amnt_loans90
24	0	-2020.09	6
41	1	-110.75	12
77	1	-919.80	6
116	1	-177.48	12
117	1	-40.20	12
...	...	...	...
209304	1	-281.12	18
209332	1	-505.80	24
209441	1	-691.90	6
209466	1	-441.57	24
209574	1	-16.86	6

5929 rows × 3 columns

In [36]:
```
#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[36]:

	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	-1126.16	6
209231	0	-158.70	12

414 rows × 3 columns

Converting the rental 90 column negative values to positive values.

```
In [38]:  1 #converting rental 90 column to positive.As there are negative values for people who had repaid their loans.
          2 df['rental90']=abs(df['rental90'])
```

Checking the entries of maxamnt_loans30 column.

```
In [39]:  1 df['maxamnt_loans30'].value_counts()

Out[39]: 6.00000        179193
         12.00000        26109
         0.00000          3244
         42638.64832         1
         43961.22397         1
                          ...
         95609.88240         1
         18728.01220         1
         64645.93468         1
         17347.61221         1
         71036.31091         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 [41]:  1 #checking the values which have entries other than 6,12,0
          2 df.loc[(df['maxamnt_loans30'] != 6.0) & (df['maxamnt_loans30'] != 12.0) & (df['maxamnt_loans30']!=0.0), 'maxamnt_lo

Out[41]: 118       61907.69737
         125       22099.41373
         146       98745.93405
         369       58925.36406
         374       78232.46432
                       ...
         209189    50824.99635
         209262    17324.99458
         209331    92864.50173
         209392    54259.26569
         209424    96927.24325
         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 [42]:
#converting them to zero
df.loc[(df['maxamnt_loans30'] != 6.0) & (df['maxamnt_loans30'] != 12.0) & (df['maxamnt_loans30']!=0.0), 'maxamnt_loa
```

```
In [43]:
df['maxamnt_loans30'].value_counts()

Out[43]: 6.0     179193
12.0     26109
0.0       4291
Name: maxamnt_loans30, dtype: int64
```

Checking the users who haven't taken any loan.

```
In [44]:
#checking the users who havent taken any loan.
dff=pd.DataFrame(np.where(df['amnt_loans90']==0))
```

```
In [45]:
dff
```

Out[45]:

	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 [46]:
#deleting the info of users who havent taken any loan.
df.drop(df[df['amnt_loans90']==0].index, inplace = True)
```

```
In [47]:
np.where(df['amnt_loans90']==0)

Out[47]: (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.

PLOTTING CORRELATIONS

In [88]:
```
#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 [104]:
```
from imblearn.under_sampling import RandomUnderSampler

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

print('original  Target dataset shape:',y.shape)
print('Resample Target dataset shape',y_us.shape)
```
```
original  Target dataset shape: (207550,)
Resample Target dataset shape (52324,)
```

In [105]:
```
print('original  Independent dataset shape:',x.shape)
print('Resample Independent dataset shape',x_us.shape)
```
```
original  Independent dataset shape: (207550, 34)
Resample Independent dataset shape (52324, 34)
```

In [106]:
```
from collections import Counter
```

In [107]:
```
print(" target distribution before sampling :",Counter(y))
```
```
target distribution before sampling : Counter({1: 181388, 0: 26162})
```

In [108]:
```
print(" target distribution after sampling :",Counter(y_us))
```
```
target distribution after sampling : Counter({0: 26162, 1: 26162})
```

OBSERVATION:

We can note that before performing Random under sampling, the data is unevenly distributed.

there are 181388 records of repayers and 26162 record of defaulters.If we feed this data into the model, it may give biased prediction.

So,performed undersampling.After performing undersampling we can note that equal no.of records are fetched for b oth defauters and repayers.

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 [91]:
```
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 [92]:
```
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:: (0, 35)
```

OBSERVATION:

22% of data removed through z score.

IQR METTHOD:

In [93]:
```
from scipy import stats
IQR = stats.iqr(df)
IQR
```
Out[93]: 393.0

In [94]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
```

In [95]:
```
df_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
print(df_out.shape)
```
```
(52807, 35)
```

OBSERVATION:

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

In [97]:

```python
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 -> 1.0935453721579915
daily_decr90 -> 1.1543224310680835
rental30 -> 1.1265741276622285
rental90 -> 1.1558700165194151
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 -> 1.3584623130969395
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
```

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 [111]:

```
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: 0.7737366454513444
```

In [112]:

```
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:: 0.7756993108241154
Precision:  0.7869701348747592
Recall:  0.7564532932052321
F-measure: 0.7714100218379273
Training accuracy:: 0.7729697350029952
Test accuracy:: 0.7756993108241154
```

DECISION TREE CLASSIFIER:

In [113]:

```
dtc=DecisionTreeClassifier()
dtc_score=cross_val_score(dtc,x_us,y_us,cv=5,scoring='accuracy')
print("cross validation score for svm:",np.mean(dtc_score))

```

```
cross validation score for svm: 0.7494648628528765
```

In [114]:

```
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:: 0.7467423408814502
Precision:  0.7499414108272792
Recall:  0.7408264845468225
F-measure: 0.7453560822220929
Training accuracy:: 0.9967196280343441
Test accuracy:: 0.7467423408814502
```

KNeighborsClassifier:

In [115]:

```
knn=KNeighborsClassifier()
knn_score=cross_val_score(knn,x_us,y_us,cv=5,scoring='accuracy')
print("cross validation score for knn:",np.mean(knn_score))
```

```
cross validation score for knn: 0.7739469814190597
```

In [116]:

```
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:: 0.7747726877859501
Precision:  0.7912680892813343
Recall:  0.7468456997337655
F-measure: 0.7684154111832311
Training accuracy:: 0.837236500556237
Test accuracy:: 0.7747726877859501
```

RandomForestClassifier:

In [117]:
```
rfc=RandomForestClassifier()
rfc_score=cross_val_score(rfc,x_us,y_us,cv=5,scoring='accuracy')
print("cross validation score for rfc:",np.mean(rfc_score))

```

```
cross validation score for rfc: 0.817254058599292
```

In [118]:
```
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::",rfc.score(x_test,y_test))
```

```
Accuracy score:: 0.8179764869404066
Precision:  0.8165169315825846
Recall:  0.8205810857738164
F-measure: 0.8185439639743664
Training accuracy:: 0.9966911030607297
Test accuracy:: 0.8179764869404066
```

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 [119]:
```
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: 0.8069146576857872
```

In [120]:
```
adb.fit(x_train,y_train)
predicted_adb=adb.predict(x_test)
print("Accuracy score::",accuracy_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:: 0.8086523426188683
Training accuracy:: 0.8081410274695496
Test accuracy:: 0.8086523426188683
```

In [121]:
```
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))
```

```
Precision:  0.8243161094224924
Recall:  0.7848130570667902
F-measure: 0.804079696394687
```

2.BAGGING CLASSIFIER

In [122]:
```
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: 0.7969003241017514
```

In [123]:
```
bgc.fit(x_train,y_train)
predicted_bgc=bgc.predict(x_test)
print("Accuracy score::",accuracy_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:: 0.7969537267620316
Training accuracy:: 0.9835696151981059
Test accuracy:: 0.7969537267620316
```

3.Gradient Boosting classifier

In [124]:
```
grbc=BaggingClassifier()
grbc_score=cross_val_score(grbc,x_us,y_us,cv=10,scoring='accuracy')
print("cross validation score for BAGGING Classifier:",np.mean(grbc_score))
```

```
cross validation score for BAGGING Classifier: 0.7973781565003556
```

In [125]:
```
grbc.fit(x_train,y_train)
predicted_grbc=grbc.predict(x_test)
print("Accuracy score::",accuracy_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:: 0.7962008455435223
Training accuracy:: 0.9845679892746099
Test accuracy:: 0.7962008455435223
```

OBSERVATION:

    Choosing Adaboost classifier because there both train and test accuracies are same.
    Rest of the two models there is huge difference between train and test accuracies so not considering them.

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 Positive

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 [126]:

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

In [127]:

```
adb_grid=GridSearchCV(AdaBoostClassifier(),parameters,cv=10,scoring='accuracy')
```

In [128]:

```
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:: 0.8113163838535936
```

OBSERVATION:

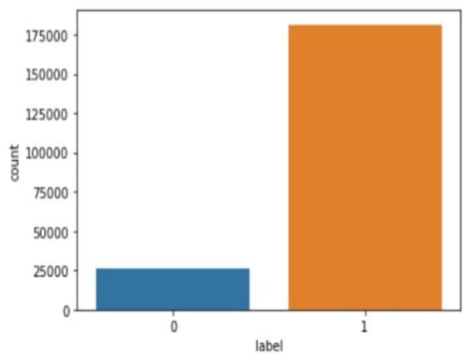
```
We can observe that accuracy score increased after tuning hyper Parameters
```

Visualizations:

In [55]:
```
1 sns.countplot(df['label'])
```

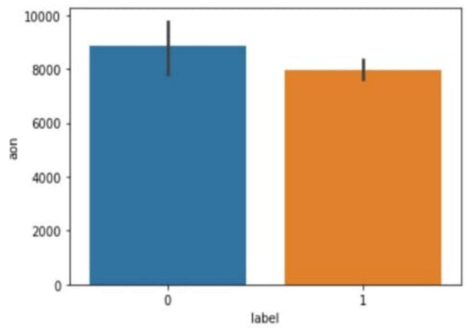
```
E:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg:
x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[55]: <AxesSubplot:xlabel='label', ylabel='count'>



In [56]:
```
1 sns.barplot(x='label',y='aon',data=df)
```

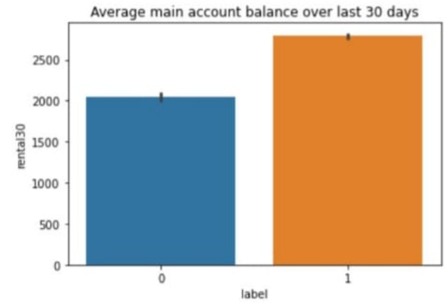
Out[56]: <AxesSubplot:xlabel='label', ylabel='aon'>



OBSERVATION:

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

In [58]:
```
sns.barplot(x=df['label'],y=df['rental30'])
plt.title('Average main account balance over last 30 days')
```
Out[58]: Text(0.5, 1.0, 'Average main account balance over last 30 days')

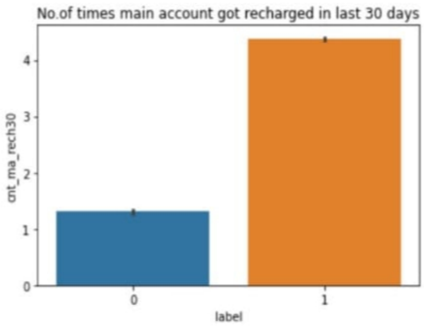


OBSERVATION:

```
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 [59]:
```
sns.barplot(x=df['label'],y=df['cnt_ma_rech30'])
plt.title('No.of times main account got recharged in last 30 days')
```
Out[59]: Text(0.5, 1.0, 'No.of times main account got recharged in last 30 days')



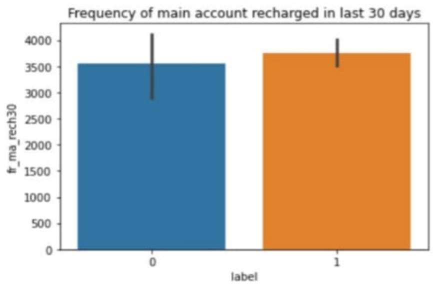
OBSERVATION:

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

whereas repayers recharged for 4 plus times.
```

In [60]:
```
sns.barplot(x=df['label'],y=df['fr_ma_rech30'])
plt.title('Frequency of main account recharged in last 30 days')
```

Out[60]: Text(0.5, 1.0, 'Frequency of main account recharged in last 30 days')

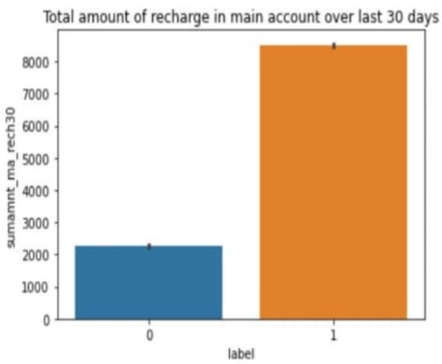


OBSERVATION:

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 [61]:
```
sns.barplot(x=df['label'],y=df['sumamnt_ma_rech30'])
plt.title('Total amount of recharge in main account over last 30 days')
```

Out[61]: Text(0.5, 1.0, 'Total amount of recharge in main account over last 30 days')

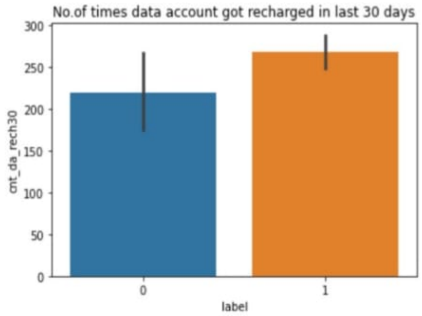


OBSERVATION:

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

In [64]:
```
sns.barplot(x=df['label'],y=df['cnt_da_rech30'])
plt.title('No.of times data account got recharged in last 30 days')
```

Out[64]: `Text(0.5, 1.0, 'No.of times data account got recharged in last 30 days')`



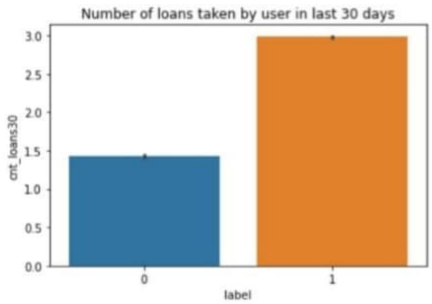
OBSERVATION:

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 [66]:
```
sns.barplot(x=df['label'],y=df['cnt_loans30'])
plt.title('Number of loans taken by user in last 30 days')
```

Out[66]: `Text(0.5, 1.0, 'Number of loans taken by user in last 30 days')`



OBSERVATION:

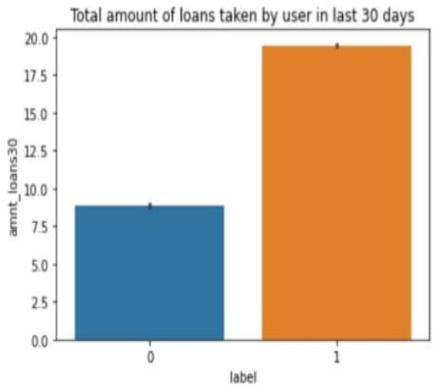
1)Defaulters has taken between 1 to 1.5 no.of loans.
As practically there will be no 1.5 loan,considering only 1 loan.

2)Those who repaid had taken maxof 3 loans.

In [67]:

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

Out[67]: Text(0.5, 1.0, 'Total amount of loans taken by user in last 30 days')



OBSERVATION:

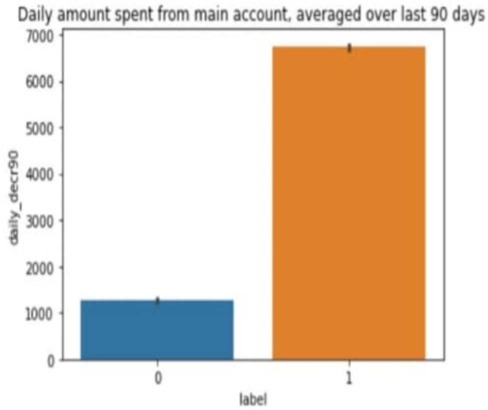
1)Total Amount of loans took by Defaulters varies between 7.5 and 10.

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

90 DAYS DATA

In [71]:
```
sns.barplot(x=df['label'],y=df['daily_decr90'])
plt.title('Daily amount spent from main account, averaged over last 90 days ')
```

Out[71]: Text(0.5, 1.0, 'Daily amount spent from main account, averaged over last 90 days ')



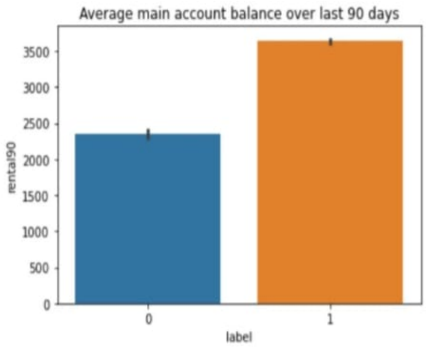
OBSERVATION: In terms of daily spending from main account in span of 90 days,

1)the defaulters has spent a little above 1000

2)Repayers has spent 7000 rupaiah.

```
In [72]: sns.barplot(x=df['label'],y=df['rental90'])
         plt.title('Average main account balance over last 90 days')

Out[72]: Text(0.5, 1.0, 'Average main account balance over last 90 days')
```



OBSERVATION: In terms of Average balance over 90 days,

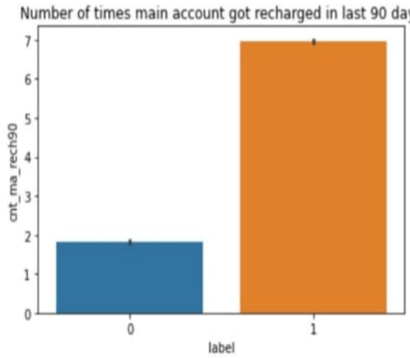
```
1)Defaullters average=2000 to 2500

2)Repayers average= 3500
```

In [73]:

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

Out[73]: 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,

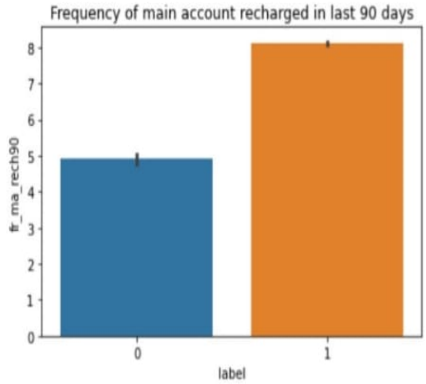
defaulters recharged for 2 times.

repayers recharged for 7 times.

In [74]:

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

Out[74]: 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.

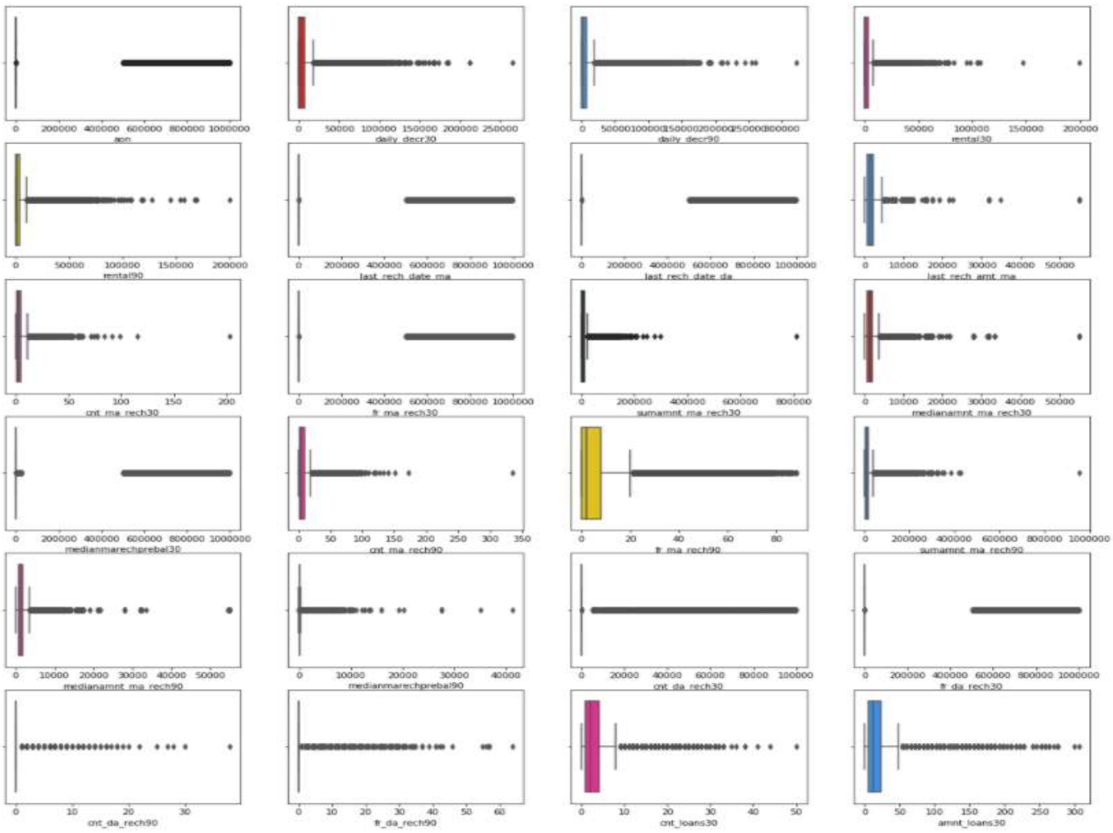
repayers frequency is 8.

CHECKING FOR OUTLIERS:

In [85]:

```
#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,a


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)
```


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

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 ,repayers 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 winsorisation 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.