FLIP ROBO

# MICRO CREDIT DEFAULTER

*Submitted by:*

**SAYAN KUMAR BHUNIA**

# ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped me and guided me in completion of the project.

- https://towardsdatascience.com/
- https://anshikaaxena.medium.com/
- https://medium.com/https://medium.com/

# INTRODUCTION

## ➢ Business Problem Framing

Collaborating with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be defaulter if he deviates from the path of paying back the loaned amount within the time duration of 5 days. For the loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).

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

## ➢ Conceptual Background of the Domain Problem

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

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

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

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

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

## ➢ Review of Literature

In real world, a loan in time enables the borrower to meet financial goals. At the same time, the interest associated with the loan generates revenues for the lender.

## ➢ Motivation for the Problem Undertaken

Every lending organization strives to assess the risk associated with the loan. Primarily, they want to assess their clients’ repayment abilities well in advance before deciding on approval and disbursement of loans it is a very realistic reason.

# Analytical Problem Framing

## ➢ Mathematical/ Analytical Modelling of the Problem

We shall build a supervised classification model to predict the risk of loan default.

Now, when we talk about building a supervised classifier catering to certain use-case, for example, classifying risk of loan default, following three things come into our minds:

- **Data** appropriate to the business requirement or use-case we are trying to solve
- A **classification model** which we think (or, rather assess) to be the best for our solution.
- **Optimize** the chosen model to ensure best performance.

## ➢ Data Sources and their formats

I am using CSV (comma-separated values) format file which is having 209593 rows × 37 columns.

In [2]:
```
df=pd.read_csv(r"C:\Users\sayan\OneDrive\Desktop\Data file.csv")
df
```

Out[2]:

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

## ➢ Data Pre-processing

Following steps have been performed on the data.

- **checking missing values**-
  - If there is any missing value present in your data set then for a better and correct accuracy you have to impute it.
  - If missing data present in object type column, then you have to take most frequent value for your missing data.
  - If missing data present in int or float type column then use mean/median for missing value.

In the following case no missing value present:

```
In [8]: df.isnull().sum()

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

- **Encoding categorical variables** -as we can see there are 3 object data type columns present so we will encode it into (int) format.
    - Apply **Label Encoding**, if number of categories in a categorical variable is equal to 2.
    - Apply **One-Hot Encoding**, if number of categories in a categorical variable is greater than 2.

In the following case Label Encoding is used.

```
In [9]: 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
```

```
In [10]: import sklearn
         from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

```
In [11]: le=LabelEncoder()
         list1=['msisdn','pcircle','pdate']
         for i in list1:
             df[i]=le.fit_transform(df[i].astype(str))
         df
```

Out[11]:

| | label | msisdn | aon | daily_decr30 | daily_decr90 | rental30 | rental90 | last_rech_date_ma | last_rech_date_da | last_rech_amt_ma | ... | maxamnt_loans30 | r |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 40191 | 272.0 | 3055.050000 | 3065.150000 | 220.13 | 260.13 | 2.0 | 0.0 | 1539 | ... | 6.0 | |
| **1** | 1 | 142291 | 712.0 | 12122.000000 | 12124.750000 | 3691.26 | 3691.26 | 20.0 | 0.0 | 5787 | ... | 12.0 | |
| **2** | 1 | 33594 | 535.0 | 1398.000000 | 1398.000000 | 900.13 | 900.13 | 3.0 | 0.0 | 1539 | ... | 6.0 | |
| **3** | 1 | 104157 | 241.0 | 21.228000 | 21.228000 | 159.42 | 159.42 | 41.0 | 0.0 | 947 | ... | 6.0 | |
| **4** | 1 | 6910 | 947.0 | 150.619333 | 150.619333 | 1098.90 | 1098.90 | 4.0 | 0.0 | 2309 | ... | 6.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **209588** | 1 | 42866 | 404.0 | 151.872333 | 151.872333 | 1089.19 | 1089.19 | 1.0 | 0.0 | 4048 | ... | 6.0 | |
| **209589** | 1 | 178248 | 1075.0 | 36.936000 | 36.936000 | 1728.36 | 1728.36 | 4.0 | 0.0 | 773 | ... | 6.0 | |
| **209590** | 1 | 53995 | 1013.0 | 11843.111667 | 11904.350000 | 5861.83 | 8893.20 | 3.0 | 0.0 | 1539 | ... | 12.0 | |
| **209591** | 1 | 111388 | 1732.0 | 12488.228333 | 12574.370000 | 411.83 | 984.58 | 2.0 | 38.0 | 773 | ... | 12.0 | |
| **209592** | 1 | 121263 | 1581.0 | 4489.362000 | 4534.820000 | 483.92 | 631.20 | 13.0 | 0.0 | 7526 | ... | 12.0 | |

209593 rows × 36 columns

- **Feature scaling-** Feature Scaling ensures that all features will get equal importance in supervised classifier models. Standard scaler was used to scale all features in the data.

```
]: from sklearn.preprocessing import StandardScaler
```

```
]: sc=StandardScaler()
   dfx=sc.fit_transform(x)
   dfx
```

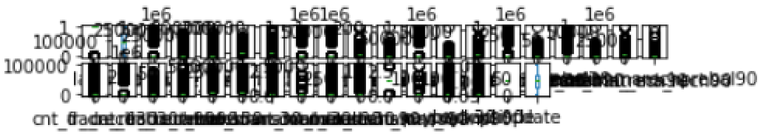
```
]: array([[-0.98275361, -0.74989904,  0.23913229, ...,  3.12519954,
            0.6044929 ,  0.12497221],
          [ 0.91488863,  0.14033762,  1.49733964, ..., -0.92348028,
            1.57292171,  1.9980835 ],
          [-1.1053662 , -0.21778031, -0.17124802, ..., -0.92348028,
            1.98796264,  0.12497221],
          ...,
          [-0.72619089,  0.74934043,  1.46810571, ...,  0.54850249,
            1.01953382,  0.12497221],
          [ 0.34052195,  2.20406805,  1.53522272, ...,  1.51269809,
            0.83507119, -0.45600331],
          [ 0.52405983,  1.89855502,  0.50832259, ..., -0.92348028,
            0.00498934,  2.5410792 ]])
```

- **Reducing dimension of the data-** Sklearn's  pca can be used to apply principal component analysis on the data. This helped in finding the vectors of maximal variance in the data.

- **Outliers detection-** In simple words, an outlier is an observation that diverges from an overall pattern on a sample.

  In the following case box plot is used to detect outliers.

```
maxamnt_loans30        AxesSubplot(0.359454,0.739237;0.032563x0.0639831)
medianamnt_loans30     AxesSubplot(0.398529,0.739237;0.032563x0.0639831)
cnt_loans90            AxesSubplot(0.437605,0.739237;0.032563x0.0639831)
amnt_loans90           AxesSubplot(0.476681,0.739237;0.032563x0.0639831)
maxamnt_loans90        AxesSubplot(0.515756,0.739237;0.032563x0.0639831)
medianamnt_loans90     AxesSubplot(0.554832,0.739237;0.032563x0.0639831)
payback30              AxesSubplot(0.593908,0.739237;0.032563x0.0639831)
payback90              AxesSubplot(0.632983,0.739237;0.032563x0.0639831)
pcircle                AxesSubplot(0.672059,0.739237;0.032563x0.0639831)
pdate                  AxesSubplot(0.711134,0.739237;0.032563x0.0639831)
dtype: object
```



There are many types of outlier detection techniques such as Z-Score or Extreme Value Analysis, Probabilistic and Statistical Modelling, Information Theory Models, Standard Deviation etc.

## ➢ Outliers Removal

In our dataset, we observed variations in the relation between values of some attributes.

So that these types of rows are dropped from the dataset.

In [16]:
```
from scipy.stats import zscore
```

In [17]:
```
z=np.abs(zscore(df))
z
```

Out[17]:

| | label | msisdn | aon | daily_decr30 | daily_decr90 | rental30 | rental90 | last_rech_date_ma | last_rech_date_da | last_rech_amt_ma | ... | amnt_loans3( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.647896 | 0.984213 | 0.103577 | 0.252299 | 0.276346 | 0.573844 | 0.558583 | 0.069637 | 0.069550 | 0.221637 | ... | 0.34247( |
| 1 | 0.377658 | 0.915027 | 0.097764 | 0.731037 | 0.553380 | 0.231788 | 0.036020 | 0.069303 | 0.069550 | 1.570178 | ... | 0.34247( |
| 2 | 0.377658 | 1.106929 | 0.100102 | 0.432011 | 0.429033 | 0.416020 | 0.447674 | 0.069619 | 0.069550 | 0.221637 | ... | 0.68770( |
| 3 | 0.377658 | 0.205668 | 0.103986 | 0.581326 | 0.555125 | 0.587935 | 0.576036 | 0.068914 | 0.069550 | 0.471344 | ... | 0.34247( |
| 4 | 0.377658 | 1.603298 | 0.094660 | 0.567293 | 0.543274 | 0.369886 | 0.413227 | 0.069600 | 0.069550 | 0.103151 | ... | 1.38368: |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 209588 | 0.377658 | 0.934453 | 0.101833 | 0.567157 | 0.543159 | 0.372140 | 0.414910 | 0.069656 | 0.069550 | 0.836664 | ... | 0.34247( |
| 209589 | 0.377658 | 1.583891 | 0.092969 | 0.579622 | 0.553686 | 0.223791 | 0.304144 | 0.069600 | 0.069550 | 0.544737 | ... | 0.00276 |
| 209590 | 0.377658 | 0.727434 | 0.093788 | 0.700790 | 0.533194 | 0.735567 | 0.937500 | 0.069619 | 0.069550 | 0.221637 | ... | 1.38368: |
| 209591 | 0.377658 | 0.340177 | 0.084289 | 0.770755 | 0.594558 | 0.529352 | 0.433039 | 0.069637 | 0.068838 | 0.544737 | ... | 0.00276 |
| 209592 | 0.377658 | 0.523869 | 0.086284 | 0.096744 | 0.141746 | 0.512620 | 0.494278 | 0.069433 | 0.069550 | 2.303692 | ... | 0.00276 |

209593 rows × 35 columns

In [18]:
```
threshold=3
print(np.where(z>3))
```

```
(array([    21,     22,     22, ..., 209586, 209587, 209587], dtype=int64), array([16, 16, 33, ..., 29, 27, 31], dtype=int64))
```

In [19]:
```
df_new=df[(z<3).all(axis=1)]
df_new
```

Out[19]:

| | label | msisdn | aon | daily_decr30 | daily_decr90 | rental30 | rental90 | last_rech_date_ma | last_rech_date_da | last_rech_amt_ma | ... | amnt_loans30 | maxa |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 40191 | 272.0 | 3055.050000 | 3065.150000 | 220.13 | 260.13 | 2.0 | 0.0 | 1539 | ... | 12 | |
| 1 | 1 | 142291 | 712.0 | 12122.000000 | 12124.750000 | 3691.26 | 3691.26 | 20.0 | 0.0 | 5787 | ... | 12 | |
| 2 | 1 | 33594 | 535.0 | 1398.000000 | 1398.000000 | 900.13 | 900.13 | 3.0 | 0.0 | 1539 | | 6 | |

## ➢ Software Requirements and library Used

```
[1]: import pandas
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')
```

```
10]: import sklearn
     from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

- **NumPy**
  NumPy is a popular Python library for multi-dimensional array and matrix processing because it can be used to perform a great variety of mathematical operations. Its capability to handle linear algebra, Fourier transform, and more, makes NumPy ideal for machine learning and artificial intelligence (AI) projects, allowing users to manipulate the matrix to easily improve machine learning performance. NumPy is faster and easier to use than most other Python libraries.

- **Scikit-learn**
  Scikit-learn is a very popular machine learning library that is built on NumPy and SciPy. It supports most of the classic supervised and unsupervised learning algorithms, and it can also be used for data mining, modelling, and analysis.

- **Seaborn**
  Seaborn is another open-source Python library, one that is based on Matplotlib (which focuses on plotting and data visualization) but features Pandas' data structures. Seaborn is often used in ML projects because it can generate plots of learning data. Of all the Python libraries, it produces the most aesthetically pleasing graphs and plots, making it an effective choice if you'll also use it for marketing and data analysis.

- **Pandas**

  Pandas is another Python library that is built on top of NumPy, responsible for preparing high-level data sets for machine learning and training. It relies on two types of data structures, one-dimensional (series) and two-dimensional (Data Frame). This allows Pandas to be applicable in a variety of industries including finance, engineering, and statistics. Unlike the slow-moving animals themselves, the Pandas library is quick, compliant, and flexible.

## ➢ Class imbalance problem

The first challenge we hit upon exploring the data, is class imbalanced problem. Imbalance data will lead to a bad accuracy of a model. To achieve better accuracy, we'll balance the data by using Smote Over Sampling Method.

```
58]: y.value_counts()

58]: 1    132773
     0     20316
     Name: label, dtype: int64

178]: from imblearn import under_sampling, over_sampling

179]: from imblearn.over_sampling import SMOTE

180]: smt=SMOTE()
      dfx,dfy=smt.fit_resample(x,y)

181]: dfy.value_counts()

181]: 0    132773
      1    132773
      Name: label, dtype: int64
```

# Model/s Development and Evaluation

## ➢ Run and evaluate selected models

Let's select our classification model for this project:

- Random Forest Classifier
- Gradient Boosting Classifier
- Adaboost Classifier

## ➢ Testing of Identified Approaches (Algorithms)

- For random forest classifier:

```
: import sklearn
  from sklearn.ensemble import RandomForestClassifier
  from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
  from sklearn.model_selection import train_test_split
  import warnings
  warnings.filterwarnings('ignore')
```

```
: x_train,x_test,y_train,y_test=train_test_split(dfx,dfy,test_size=.30,random_state=45)
```

```
199]: rf=RandomForestClassifier(n_estimators=300,random_state=42)
      rf.fit(x_train,y_train)
      predrf=rf.predict(x_test)
      print(accuracy_score(y_test,predrf))
      print(confusion_matrix(y_test,predrf))
      print(classification_report(y_test,predrf))

      0.943788913436433
      [[37472  2533]
       [ 1945 37714]]
                    precision    recall  f1-score   support

                 0       0.95      0.94      0.94     40005
                 1       0.94      0.95      0.94     39659

          accuracy                           0.94     79664
         macro avg       0.94      0.94      0.94     79664
      weighted avg       0.94      0.94      0.94     79664
```

```
In [201]: predrf=rf.predict(x_test)
          from sklearn.model_selection import cross_val_score
          rfs=accuracy_score(y_test,predrf)
          for j in range(4,7):
              rfscore=cross_val_score(rf,dfx,dfy,cv=j)
              rfcv=rfscore.mean()
              print('at cv:-',j)
              print('crossvalidation score:',rfcv*100)
              print('accuracy_score is:-',rfs*100)
              print('\n')

          at cv:- 4
          crossvalidation score: 93.53937258174234
          accuracy_score is:- 94.3788913436433


          at cv:- 5
          crossvalidation score: 93.58910641809159
          accuracy_score is:- 94.3788913436433


          at cv:- 6
          crossvalidation score: 93.65234835928119
          accuracy_score is:- 94.3788913436433
```

- For Adaboost Classifier:

```
202]: from sklearn.tree import DecisionTreeClassifier
      from sklearn.ensemble import AdaBoostClassifier
      #ad=AdaBoostClassifier(base_estimator=DecisionTreeClassifier,n_estimators=50,learning_rate=1.0)
      ad=AdaBoostClassifier()
      ad.fit(x_train,y_train)
      predad=ad.predict(x_test)
      print(accuracy_score(y_test,predad))
      print(confusion_matrix(y_test,predad))
      print(classification_report(y_test,predad))
```

```
0.887502510544286
[[35952  4053]
 [ 4909 34750]]
              precision    recall  f1-score   support

           0       0.88      0.90      0.89     40005
           1       0.90      0.88      0.89     39659

    accuracy                           0.89     79664
   macro avg       0.89      0.89      0.89     79664
weighted avg       0.89      0.89      0.89     79664
```

- For Gradient Boosting Classifier

```
[205]:
from sklearn.ensemble import GradientBoostingClassifier
gb=GradientBoostingClassifier()
gb.fit(x_train,y_train)
predgb=gb.predict(x_test)
print(accuracy_score(y_test,predgb))
print(confusion_matrix(y_test,predgb))
print(classification_report(y_test,predgb))
```

```
0.9122062663185379
[[36856  3149]
 [ 3845 35814]]
              precision    recall  f1-score   support

           0       0.91      0.92      0.91     40005
           1       0.92      0.90      0.91     39659

    accuracy                           0.91     79664
   macro avg       0.91      0.91      0.91     79664
weighted avg       0.91      0.91      0.91     79664
```

## ➢ Key Metrics for success in solving problem under consideration

Selection of a model requires evaluation and evaluation requires a good metric. This is indeed important. If we optimize a model based on incorrect metric, then, our model might not be suitable for the business goals.

We have a number of metrics, for example, accuracy, recall, precision, F1 score, area under receiver operating characteristic curve, to choose from.

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

```
0.943788913436433
[[37472  2533]
 [ 1945 37714]]
              precision    recall  f1-score   support

           0       0.95      0.94      0.94     40005
           1       0.94      0.95      0.94     39659

    accuracy                           0.94     79664
   macro avg       0.94      0.94      0.94     79664
weighted avg       0.94      0.94      0.94     79664
```

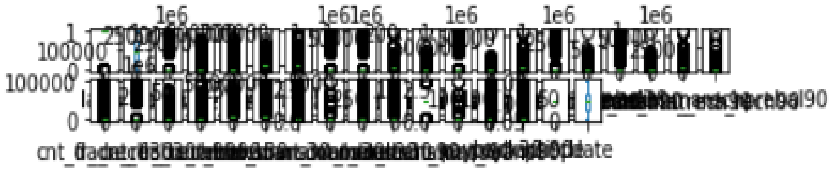
## ➤ Visualizations

For better understanding of outliers, I have used boxplot.

```
12]: df.plot(kind='box',subplots=True,layout=(10,20))
```

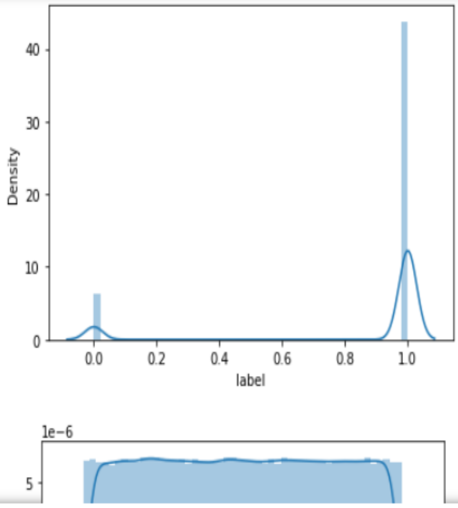
```
12]: label                      AxesSubplot(0.125,0.816017;0.032563x0.0639831)
     msisdn                  AxesSubplot(0.164076,0.816017;0.032563x0.0639831)
     aon                     AxesSubplot(0.203151,0.816017;0.032563x0.0639831)
     daily_decr30            AxesSubplot(0.242227,0.816017;0.032563x0.0639831)
     daily_decr90            AxesSubplot(0.281303,0.816017;0.032563x0.0639831)
     rental30                AxesSubplot(0.320378,0.816017;0.032563x0.0639831)
     rental90                AxesSubplot(0.359454,0.816017;0.032563x0.0639831)
     last_rech_date_ma       AxesSubplot(0.398529,0.816017;0.032563x0.0639831)
     last_rech_date_da       AxesSubplot(0.437605,0.816017;0.032563x0.0639831)
     last_rech_amt_ma        AxesSubplot(0.476681,0.816017;0.032563x0.0639831)
     cnt_ma_rech30           AxesSubplot(0.515756,0.816017;0.032563x0.0639831)
     fr_ma_rech30            AxesSubplot(0.554832,0.816017;0.032563x0.0639831)
     sumamnt_ma_rech30       AxesSubplot(0.593908,0.816017;0.032563x0.0639831)
     medianamnt_ma_rech30    AxesSubplot(0.632983,0.816017;0.032563x0.0639831)
     medianmarechprebal30    AxesSubplot(0.672059,0.816017;0.032563x0.0639831)
     cnt_ma_rech90           AxesSubplot(0.711134,0.816017;0.032563x0.0639831)
     fr_ma_rech90             AxesSubplot(0.75021,0.816017;0.032563x0.0639831)
     sumamnt_ma_rech90       AxesSubplot(0.789286,0.816017;0.032563x0.0639831)
     medianamnt_ma_rech90    AxesSubplot(0.828361,0.816017;0.032563x0.0639831)
     medianmarechprebal90    AxesSubplot(0.867437,0.816017;0.032563x0.0639831)
     cnt_da_rech30              AxesSubplot(0.125,0.739237;0.032563x0.0639831)
     fr_da_rech30            AxesSubplot(0.164076,0.739237;0.032563x0.0639831)
     cnt_da_rech90           AxesSubplot(0.203151,0.739237;0.032563x0.0639831)
     fr_da_rech90            AxesSubplot(0.242227,0.739237;0.032563x0.0639831)
     cnt_loans30             AxesSubplot(0.281303,0.739237;0.032563x0.0639831)
     amnt_loans30            AxesSubplot(0.320378,0.739237;0.032563x0.0639831)
```

```
maxamnt_loans30        AxesSubplot(0.359454,0.739237;0.032563x0.0639831)
medianamnt_loans30     AxesSubplot(0.398529,0.739237;0.032563x0.0639831)
cnt_loans90            AxesSubplot(0.437605,0.739237;0.032563x0.0639831)
amnt_loans90           AxesSubplot(0.476681,0.739237;0.032563x0.0639831)
maxamnt_loans90        AxesSubplot(0.515756,0.739237;0.032563x0.0639831)
medianamnt_loans90     AxesSubplot(0.554832,0.739237;0.032563x0.0639831)
payback30              AxesSubplot(0.593908,0.739237;0.032563x0.0639831)
payback90              AxesSubplot(0.632983,0.739237;0.032563x0.0639831)
pcircle                AxesSubplot(0.672059,0.739237;0.032563x0.0639831)
pdate                  AxesSubplot(0.711134,0.739237;0.032563x0.0639831)
dtype: object
```



For better understanding of skewness, I have used distribution plot.

```
In [13]: for i in df.columns:
             plt.figure()
             sns.distplot(df[i])
```




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

# CONCLUSION

Key aspects of building successful classifier are:

- Selecting correct data according to the purpose or problem statement.
- Proper processing and understanding of the data
- Selecting the model and optimizing the model.

In this project I have dealt with outliers and using z score I removed those outliers for better accuracy.

I have used label encoder for encoding object data type into int datatype as machine doesn’t understand object type data.

I have performed Smote operation as data was imbalanced.

I have used three classification model and found Random Forest Classifier to be the best fit. It is giving 94% accuracy.