Micro Credit Defaulter Project

Submitted by:

Abhishek Mishra

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to team of Flip Robo as well as my my mentor Swati Rustagi who gave me the excellent opportunity to do this wonderful machine learning project ‘Micro Credit Defaulter’ , which also helped me in doing a lot of Research and I came to know about so many new things.

I am really thankful to them. Secondly, I would also like to thank Data-Trained Education who helped me a lot in finishing this project within the limited time. Just because of them I was able to create my project and make it a good and enjoyable experience.

There were some errors and problems I faced in between the project solution where I was able to rectify from the internet and different platforms such as google, youtube, kaggle, GitHub etc.

INTRODUCTION

- **Business Problem Framing**

Microfinance companies have problems in selecting the customers who can payback the loan in 5 days of issuance of loan. 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). Companies face problems in recovering loan amounts which are not recovered because these companies fail in selecting the right customers. Microfinance is widely accepted as a poverty-reduction tool, representing $70 billion in outstanding loans and a global outreach of 200 million clients.

- **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.

- **Review of Literature**

In this project we will be studying the dataset of the customers and will be performing different methodologies to come up with a resolution to the problem statement i.e. predict the nature of the customers if they will be able to pay back the loan amount. There have been about 12.5%

customers who default in paying back loans and have been classified as defaulters which are not able to pay the loan amount of 6 or 12 within 5 days of the issuance.

- **Motivation for the Problem Undertaken**

This project focuses on providing the services and products to lower income families or poor customers, thus it helps in defining these customer preferences of paying the loan back. So, building a prediction model for the organization which will help them to predict whether the loan provided to customers will be a defaulter or not, which will help the organization in providing the loan to the right customers.

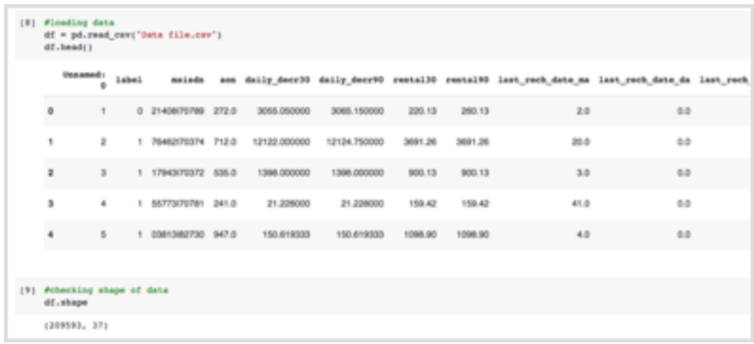
Analytical Problem Framing

- **Mathematical/ Analytical Modeling of the Problem**

MFI is an organization which provides financial services to low income groups.They are collaborating with MFI to provide the micro credit on mobile balances. These credits to be paid back in 5 days otherwise the customer is a defaulter.

There are two loan amounts : 5 and 10 Indonesian Rupiah which have a pay back amount of 6 and 12 Rupiah.

Data set has a shape of 209593 rows and 37 columns under which there is a column with a feature name of Unnamed: 0 which will be removed as it is of no use in predicting the target variable.



```
[8] #loading data
    df = pd.read_csv("Data file.csv")
    df.head()
```

	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech
0	1	0	21408070789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	
1	2	1	76482170374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	
2	3	1	17943170372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	
3	4	1	55773170781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	
4	5	1	03813682730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	

```
[9] #checking shape of data
    df.shape
```

```
(209593, 37)
```

Summary statistics

```
[17] #statistical summary of the data
     df2.describe()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_r
count	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	2(
mean	0.875177	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.847800	3712.202921	
std	0.330519	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.892230	53374.833430	
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	-29.000000	
25%	1.000000	246.000000	42.440000	42.692000	280.420000	300.260000	1.000000	0.000000	
50%	1.000000	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.000000	0.000000	
75%	1.000000	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.000000	0.000000	
max	1.000000	999860.755168	265926.000000	320630.000000	198926.110000	200148.110000	998650.377733	999171.809410	

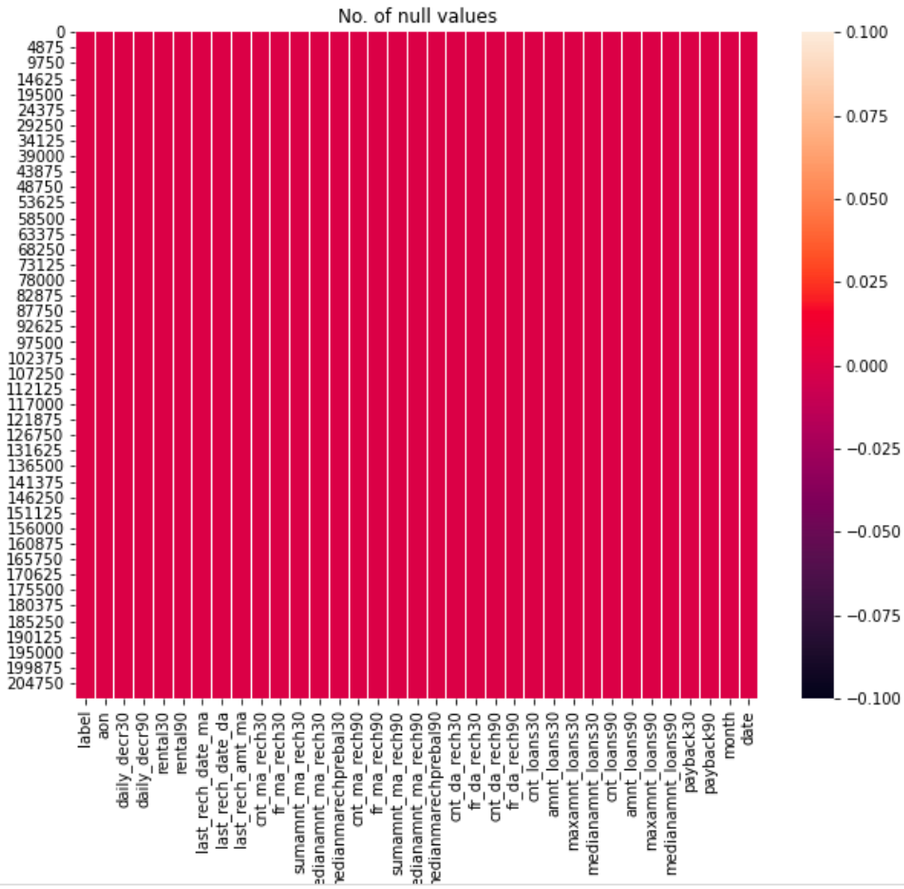
Removal of outliers and erroneous data : Since the dataset is expensive and we cannot lose more than 8-10 percent of the data will be replacing the outliers with the median and erroneous values with 0.

```
[47] # function to replace errorneous value with zero
     def replace_0(data,col):
         value = np.where(data[col]<0,0,data[col])
         data[col]=value
```

```
[48] #function to replace outliers
     def replace_outliers(data,col):
         Q1,Q3 = data[col].quantile(0.25) , data[col].quantile(0.75)
         IQR = Q3-Q1
         lower = Q1 - (1.5*IQR)
         upper = Q3 + (1.5*IQR)
         replace = np.where(((data[col]<lower) | (data[col]>upper)),data[col].median(),data[col])
         data[col]=replace
```

Null Values - There are no null values present in the dataset.

```
[21] plt.figure(figsize=(10,8))
     plt.title('No. of null values')
     sns.heatmap(df3.isnull())
     plt.show()
```



The above graph describes that if there are any null values in the above dataset, I have found out that the red color shows the null values here, it means that the values are not null.

- **Data Sources and their formats**

A. Data Types:

Data types in the dataset —> int : 12 attributes , float : 21 attributes, object : 1 attribute

```
Unnamed: 0                int64
label                     int64
msisdn                   object
aon                     float64
daily_decr30            float64
daily_decr90            float64
rental30                float64
rental90                float64
last_rech_date_ma       float64
last_rech_date_da       float64
last_rech_amt_ma          int64
cnt_ma_rech30             int64
fr_ma_rech30            float64
sumamnt_ma_rech30       float64
medianamnt_ma_rech30    float64
medianmarechpreba130    float64
cnt_ma_rech90             int64
fr_ma_rech90              int64
sumamnt_ma_rech90         int64
medianamnt_ma_rech90    float64
medianmarechpreba190    float64
cnt_da_rech30           float64
fr_da_rech30            float64
cnt_da_rech90             int64
fr_da_rech90              int64
cnt_loans30               int64
amnt_loans30              int64
maxamnt_loans30         float64
medianamnt_loans30      float64
cnt_loans90             float64
amnt_loans90              int64
maxamnt_loans90           int64
medianamnt_loans90      float64
payback30               float64
payback90               float64
pcircle                  object
pdate                    object
dtype: object
```

B. Data Description : Statistical Summary of the dataset

```
[17] #statistical summary of the data
     df2.describe()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_r
count	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	209593.000000	2(
mean	0.875177	8112.343445	5381.402289	6082.515068	2692.581910	3483.406534	3755.847800	3712.202921	
std	0.330519	75696.082531	9220.623400	10918.812767	4308.586781	5770.461279	53905.892230	53374.833430	
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	-29.000000	
25%	1.000000	246.000000	42.440000	42.692000	280.420000	300.260000	1.000000	0.000000	
50%	1.000000	527.000000	1469.175667	1500.000000	1083.570000	1334.000000	3.000000	0.000000	
75%	1.000000	982.000000	7244.000000	7802.790000	3356.940000	4201.790000	7.000000	0.000000	
max	1.000000	999860.755168	265926.000000	320630.000000	198926.110000	200148.110000	998650.377733	999171.809410	

In the above description we get the statistical overview of the dataset. Count is the number of values in the variable, for example let's take non field in consideration, no of values are 209593, mean is 8112, standard deviation 0.3305 which tells the spread of the data, min value is -48 which is negative and treated as erroneous data 25% & 75% are the quartiles Q1 and Q3, 50% indicates the median of the data, max is 999860 which is to far from 75% which indicates presence of outliers and erroneous data.

- **<u>Data Preprocessing Done</u>**

1. Dropping unnecessary columns : Dropped some of the unnecessary columns like "unnamed: 0","msisdn" "pcircle" as they are not necessary for prediction.

```
[14] #dropping columns which are not necessary for the target variable i.e. "unnamed: 0","msisdn" "pcircle"
     df2 = df1.drop(["Unnamed: 0","pcircle","msisdn"],axis=1)
     df2.head()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7

2. Parsing dates column: parsed dates column to date month and year

```
[18] #different columns for date time
     df3 = df2.copy()
     df3['year'] = pd.DatetimeIndex(df2.pdate).year
     df3['month'] = pd.DatetimeIndex(df2.pdate).month
     df3['date'] = pd.DatetimeIndex(df2.pdate).day
     df3.head()
```

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_ma_rech30
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539	2
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787	1
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539	1
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947	0
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309	7

```
[20] #dropping year as it has only one value and pdate as it is of no use now
     df3.drop(["pdate","year"],axis=1,inplace=True)
     df3.head()
```

axamnt_loans30	medianamnt_loans30	cnt_loans90	amnt_loans90	maxamnt_loans90	medianamnt_loans90	payback30	payback90	month	date
6.0	0.0	2.0	12	6	0.0	29.000000	29.000000	7	20
12.0	0.0	1.0	12	12	0.0	0.000000	0.000000	8	10
6.0	0.0	1.0	6	6	0.0	0.000000	0.000000	8	19
6.0	0.0	2.0	12	6	0.0	0.000000	0.000000	6	6
6.0	0.0	7.0	42	6	0.0	2.333333	2.333333	6	22

3. Checked for missing values : found no missing values

```
df3.isnull().sum()
```

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

4. Removing Outliers and Erroneous values :

Checking erroneous values

```
[45] #checking min values
     df4.min()
```

```
label                    0.000000
aon                    -48.000000
daily_decr30           -93.012667
daily_decr90           -93.012667
rental30             -23737.140000
rental90             -24720.580000
last_rech_date_ma      -29.000000
last_rech_date_da      -29.000000
last_rech_amt_ma         0.000000
cnt_ma_rech30            0.000000
fr_ma_rech30             0.000000
sumamnt_ma_rech30        0.000000
medianamnt_ma_rech30     0.000000
medianmarechprebal30  -200.000000
cnt_ma_rech90            0.000000
fr_ma_rech90             0.000000
sumamnt_ma_rech90        0.000000
medianamnt_ma_rech90     0.000000
medianmarechprebal90  -200.000000
cnt_da_rech30            0.000000
fr_da_rech30             0.000000
cnt_da_rech90            0.000000
fr_da_rech90             0.000000
cnt_loans30              0.000000
amnt_loans30             0.000000
maxamnt_loans30          0.000000
medianamnt_loans30       0.000000
cnt_loans90              0.000000
amnt_loans90             0.000000
maxamnt_loans90          0.000000
medianamnt_loans90       0.000000
payback30                0.000000
payback90                0.000000
month                    6.000000
date                     1.000000
dtype: float64
```

```
[ ] #checking max values
    df4.max()
```

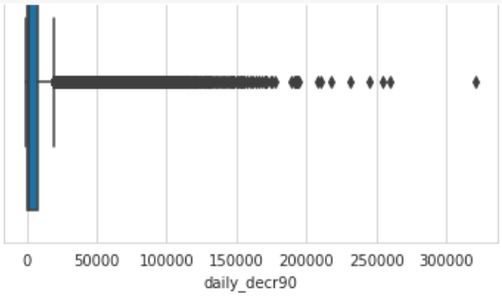
```
label                      1.000000
aon                   999860.755168
daily_decr30          265926.000000
daily_decr90          320630.000000
rental30              198926.110000
rental90              200148.110000
last_rech_date_ma     998650.377733
last_rech_date_da     999171.809410
last_rech_amt_ma       55000.000000
cnt_ma_rech30            203.000000
fr_ma_rech30          999606.368132
sumamnt_ma_rech30     810096.000000
medianamnt_ma_rech30   55000.000000
medianmarechprebal30  999479.419319
cnt_ma_rech90            336.000000
fr_ma_rech90              88.000000
sumamnt_ma_rech90     953036.000000
medianamnt_ma_rech90   55000.000000
medianmarechprebal90   41456.500000
cnt_da_rech30          99914.441420
fr_da_rech30          999809.240107
cnt_da_rech90             38.000000
fr_da_rech90              64.000000
cnt_loans30               50.000000
amnt_loans30             306.000000
maxamnt_loans30        99864.560864
medianamnt_loans30         3.000000
cnt_loans90             4997.517944
amnt_loans90             438.000000
maxamnt_loans90           12.000000
medianamnt_loans90         3.000000
payback30                171.500000
payback90                171.500000
month                      8.000000
date                      31.000000
dtype: float64
```

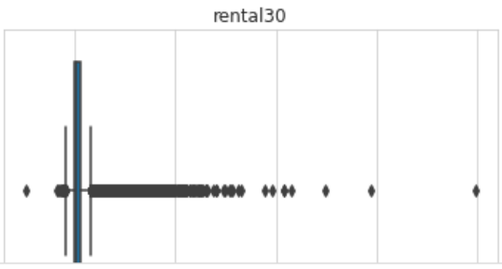
Removing Erroneous values and replacing it with zero

```
[27] # function to replace errorneous value with zero
     def replace_0(data,col):
         value = np.where(data[col]<0,0,data[col])
         data[col]=value
```

Checking outliers using box plot

```
#boxplot for every feature to visualise outliers
cols = ['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:
    sns.boxplot(df3[i])
    plt.title(i)
    plt.show()
```





Removing outliers and replacing it by median

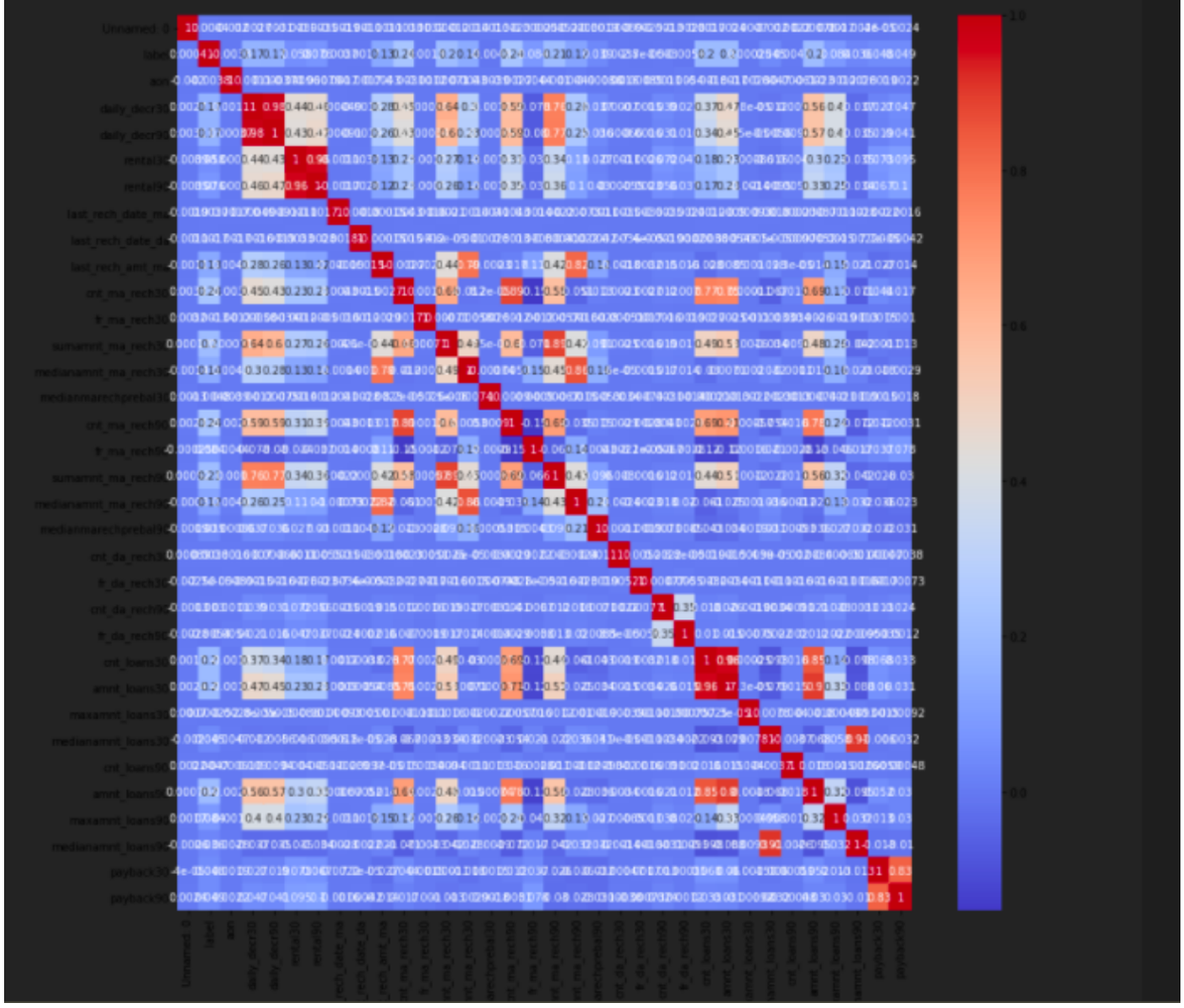
```
#function to replace outliers
def replace_outliers(data,col):
    Q1,Q3 = data[col].quantile(0.25) , data[col].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1 - (1.5*IQR)
    upper = Q3 + (1.5*IQR)
    replace = np.where(((data[col]<lower) | (data[col]>upper)),data[col].median(),data[col])
    data[col]=replace
```

5. Treating skewness : there is a lot of positive skewness i.e. skew > 0.5 in the data we will be treating it by applying square root function.

```
In [189]: #removing skewness

        for col in df5:
            if df5[col].skew()>=.5:
                df5[col] = np.sqrt(df5[col])
```

- **Data Inputs- Logic- Output Relationships**

```
plt.figure(figsize=(15,15))
sns.heatmap(corr,annot=True,cmap='coolwarm')
plt.show()
```



From the above heat map plot we can check the correlation of feature variables x with target variable y. Here, features with magnitude between -1 and 1 represent negative and positive correlation respectively i.e. 1 represents high correlation and -1 represents the least correlation between the variables.

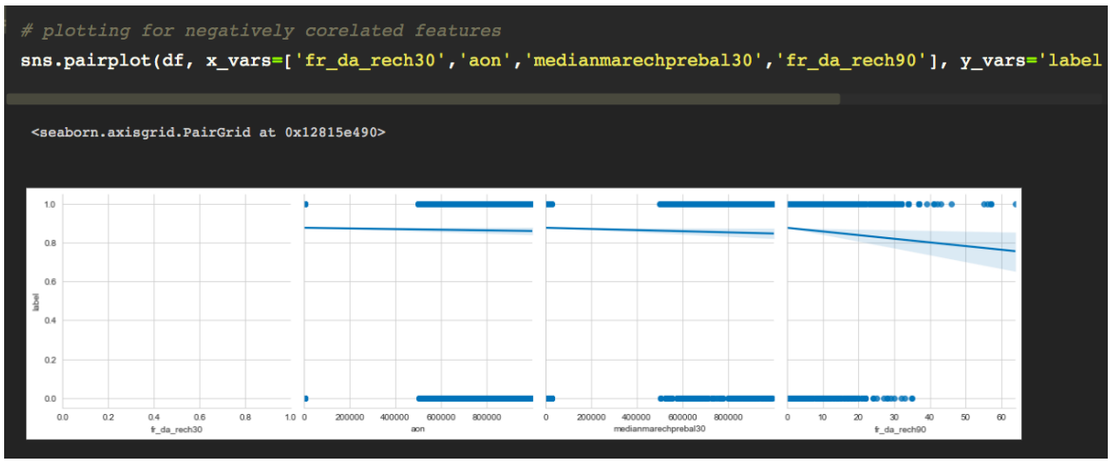
Correlation sorted representation.

```
In [129]: corr['label'].sort_values(ascending=False)
```

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

From the above observation we can check the highest correlated feature and least correlated feature easily.

Here, we can also see the last four features are negatively correlated, so we will take them into consideration since we can not lose more than 8-10%of data. Below is the plot of negatively correlated features.



- **<u>Hardware and Software Requirements and Tools Used</u>**

Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

Numpy : Numpy is used in the project for working with arrays and multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Pandas **:** pandas is used in the project for working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Matplotlib.pyplot : Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.It is has been used in project for visualization of the dataset.

Seaborn : Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Standard Scaler **:** *sklearn.preprocessing.StandardScaler,* was used for Standardising the data so that all the values can Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data.

PCA Analysis **:** *sklearn.decomposition.PCA*, Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Oversampling : *from imblearn.over_sampling import SMOTE,* SMOTE is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them.

Train-test-split : *from sklearn.model_selection import train_test_split*, the procedure involves taking a dataset and dividing it into two subsets of training an testing sets of feature and target variables

Cross validation score **:** *from sklearn.model_selection import cross_val_score,* in cross-validation, we run our modeling process on different subsets of the data to get multiple measures of model quality.

Logistic Regression **:** *from sklearn.linear_model import LogisticRegression,* used in predicting the model with logistic regression where , Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression)

Decision Tree **:** *from sklearn.tree import DecisionTreeClassifier,* A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

K Nearest Neighbors **:** *from sklearn.neighbors import KNeighborsClassifier,* The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data

Support Vector Machines **:** *from sklearn.svm import SVC,* The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

Bernoulli Naive Bayes **:** *from sklearn.naive_bayes import BernoulliNB*, BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable.

Random Forest **:** *from sklearn.ensemble import RandomForestClassifier,* A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Adaboost : *from sklearn.ensemble import AdaBoostClassifier,* is a Boosting technique that is used as an Ensemble Method in Machine Learning, as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances.

Classification_report **:** *from sklearn.metrics import classification_report,* The classification report visualizer displays the precision, recall, F1, and support scores for the model.

Confusion Matrix **:** *from sklearn.metrics import confusion_matrix,* A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

Accuracy Score **:** *from sklearn.metrics import accuracy_score*, In multi label classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.

Roc curve : *from sklearn.metrics import roc_curve* , ROC is a plot of signal (True Positive Rate) against noise (False Positive Rate). ... The model performance is determined by looking at the area under the ROC curve (or AUC). The best possible AUC is 1 while the worst is 0.5 (the 45 degrees random line).

Roc_auc_score : *from sklearn.metrics import roc_auc_score* , ROC stands for curves receiver or operating characteristic curve. It illustrates in a binary classifier system the discrimination threshold created by plotting the true positive rate vs false positive rate. ... The roc_auc_score always runs from 0 to 1, and is sorting predictive possibilities.

Precision : *from sklearn.metrics import precision* , Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Recall : *from sklearn.metrics import recall* , Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1-Score : *from sklearn.metrics import f1_score* , F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

Randomized search cv **:** *from sklearn.model_selection import RandomizedSearchCV*, RandomizedSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters

Model/s Development and Evaluation

- **Identification of possible problem-solving approaches (methods)**

  Describe the approaches you followed, both statistical and analytical, for solving of this problem.

- **Testing of Identified Approaches (Algorithms)**

  Listing down all the algorithms used for the training and testing.

  1.Train/Test split:

Splitting the dataset in training and testing sets. Distributing train and test in 75-25 ratio, Enabled stratify for better prediction.

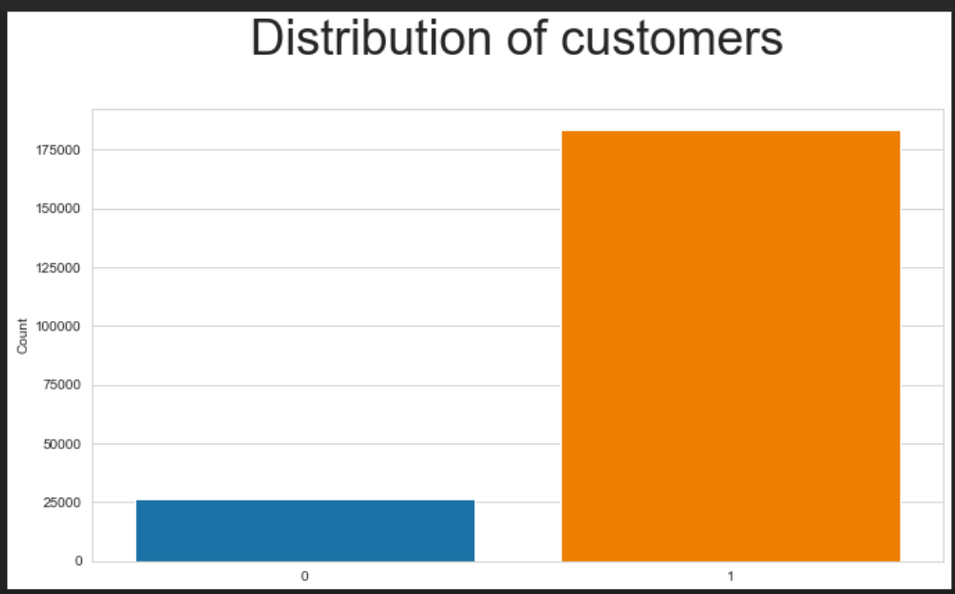
```
from sklearn.model_selection import train_test_split,cross_val_score
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=45,test_size=.25,stratify=y)
```

2. Oversampling:

As from the below plot we can see that the target variable is unbalanced which could result in a low accurate model we will balance the dataset. We will be using the oversampling method SMOTE here, to oversample the minority class.

```
# Distribution of Target Variable
plt.figure(figsize=(11,6))
sns.countplot(df3.label)
sns.set_style('whitegrid')
plt.ylabel("Count")
plt.xlabel("Defaulter")
plt.title("Distribution of customers",fontsize=35,y=1.1)
plt.show()
```



Observation : It is Clearly visible that our target dataset is imbalance .

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE(sampling_strategy='minority')
x_train,y_train = sm.fit_resample(xtrain,ytrain)
print(x_train.shape," \n ",y_train.shape)
```

```
(275146, 20)
  (275146,)
```

3. Model Building :

Will be building some models and then we will fit it onto our training set and then further we will evaluate them using the right metrics.

Algorithms used for model building are :

Logistic Regression : Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression)

Support vector Machines : The objective of a Linear SVC (Support Vector Classifier) is to fit the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

Decision Tree Classifiers : A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

K Nearest Neighbors : The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithm. KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data

Bernoulli Naive Bayes : BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable.

Random Forest Classifiers : A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

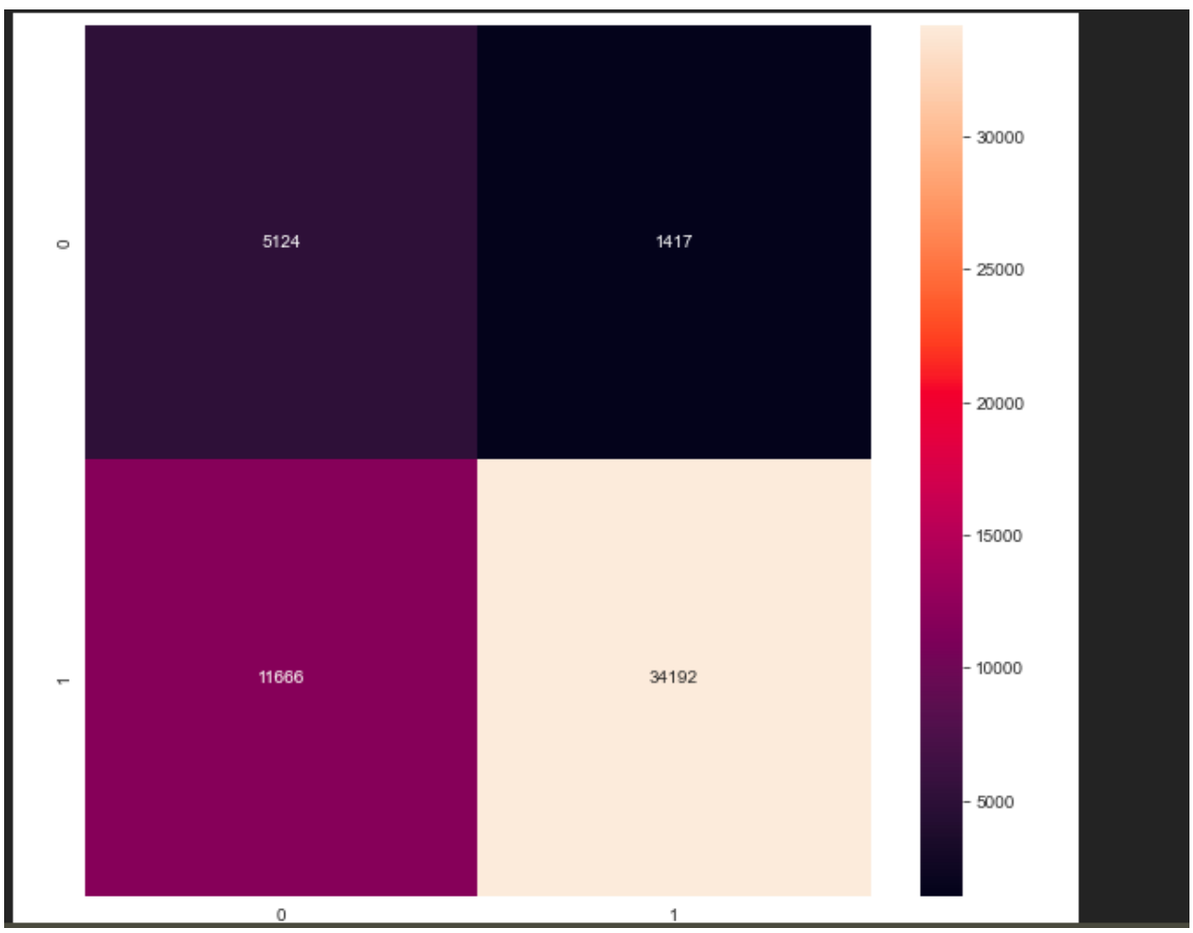
Adaboost Classifiers : Adaboost a Boosting technique that is used as an Ensemble Method in Machine Learning, as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances.

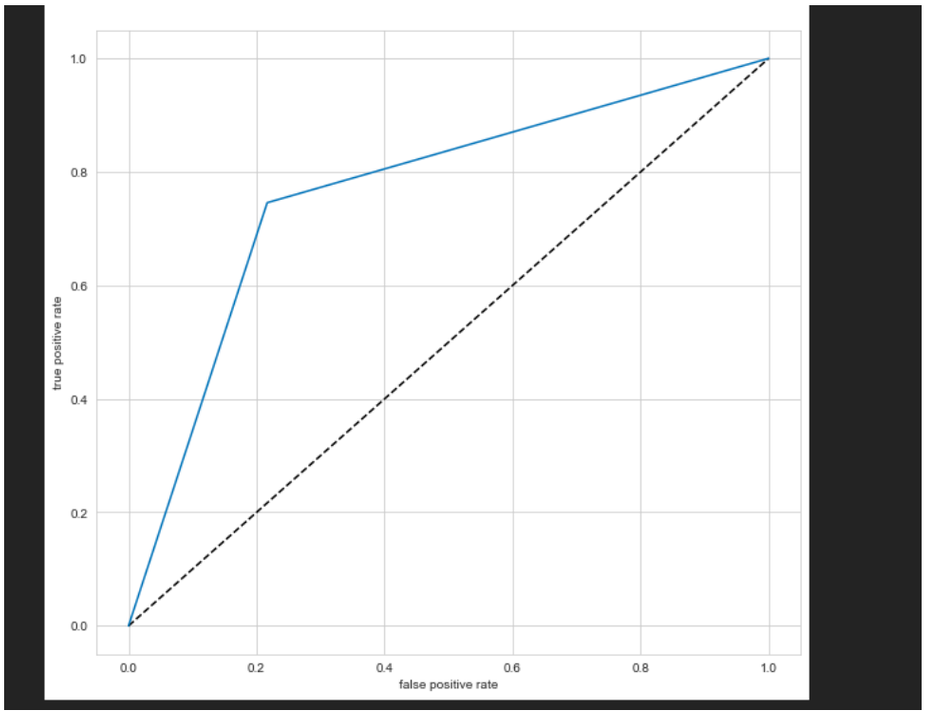
- **Run and Evaluate selected models**

Logistic Regression

```
LR = LogisticRegression()
print("*****Logistic Regression*****")
LR.fit(x_train, y_train)
pred = LR.predict(xtest)
accuracy = accuracy_score(ytest,pred)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(LR,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred)
roc_auc = auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred))
precision = precision_score(ytest, pred)
print('Precision: ', precision)
recall = recall_score(ytest, pred)
print('Recall: ', recall)
f1 = f1_score(ytest, pred)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Logistic Regression*****

 accuracy score :  0.7503196625889808

 cross validation score :  0.7688173125839057

 [[ 5124  1417]
 [11666 34192]]

 ROC_AUC_SCORE :  0.7644862294310556
              precision    recall  f1-score   support

           0       0.31      0.78      0.44      6541
           1       0.96      0.75      0.84     45858

    accuracy                           0.75     52399
   macro avg       0.63      0.76      0.64     52399
weighted avg       0.88      0.75      0.79     52399

Precision:  0.9602066893201157
Recall:  0.7456060011339352
F1 score:  0.8394073674002971
```

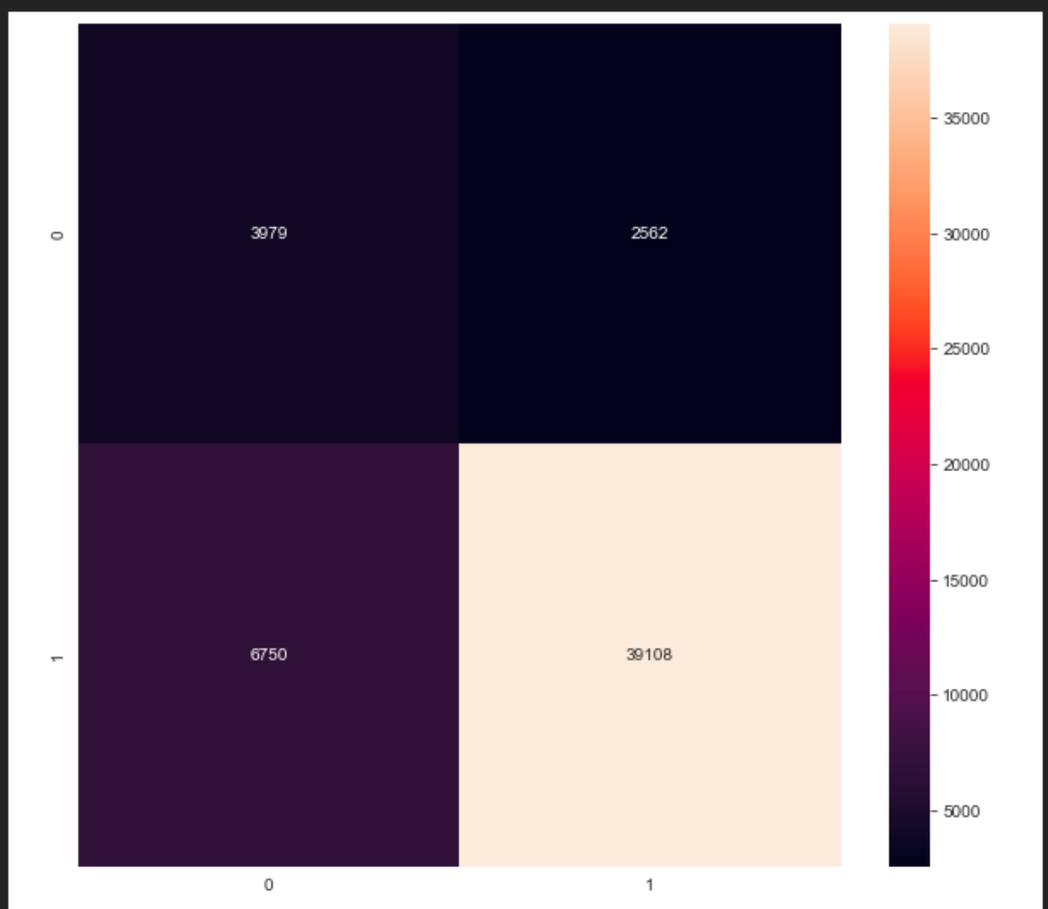


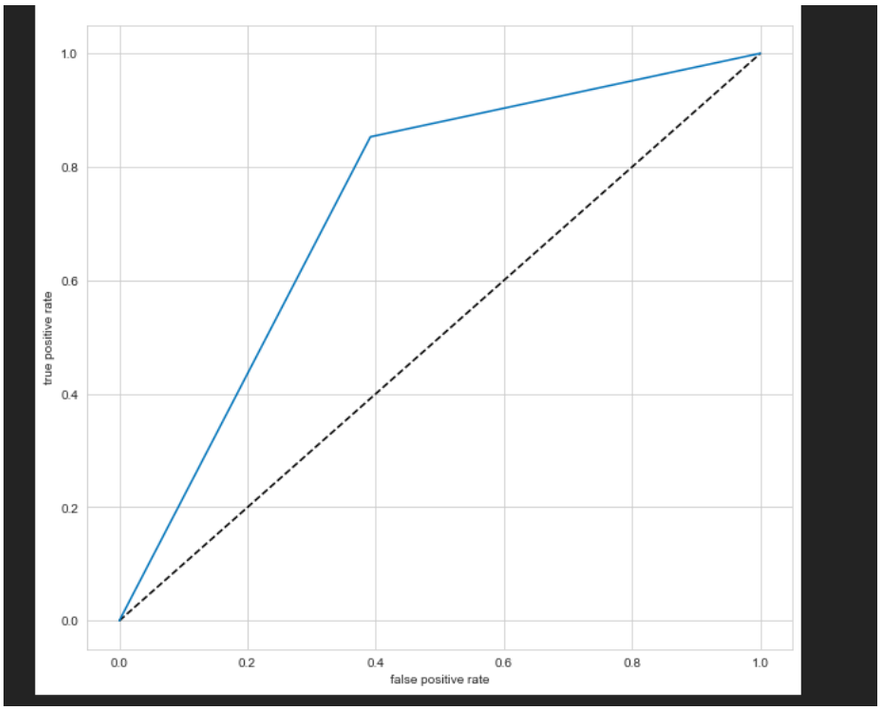


Decision Tree Classifier

```
DTC = DecisionTreeClassifier()
print("*****Decision Tree Classifier*****")
DTC.fit(x_train, y_train)
pred1 = DTC.predict(xtest)
accuracy = accuracy_score(ytest,pred1)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(DTC,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred1)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred1)
roc_auc = auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred1))
precision = precision_score(ytest, pred1)
print('Precision: ', precision)
recall = recall_score(ytest, pred1)
print('Recall: ', recall)
f1 = f1_score(ytest, pred1)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Decision Tree Classifier*****

 accuracy score :  0.8222866848603981

 cross validation score :  0.8716899624463563

 [[ 3979  2562]
 [ 6750 39108]]

 ROC_AUC_SCORE :  0.7305616303671185
              precision    recall  f1-score   support

           0       0.37      0.61      0.46      6541
           1       0.94      0.85      0.89     45858

    accuracy                           0.82     52399
   macro avg       0.65      0.73      0.68     52399
weighted avg       0.87      0.82      0.84     52399

Precision:  0.9385169186465083
Recall:  0.8528064895983253
F1 score:  0.8936111872772141
AxesSubplot(0.125,0.536818;0.62x0.343182)
```

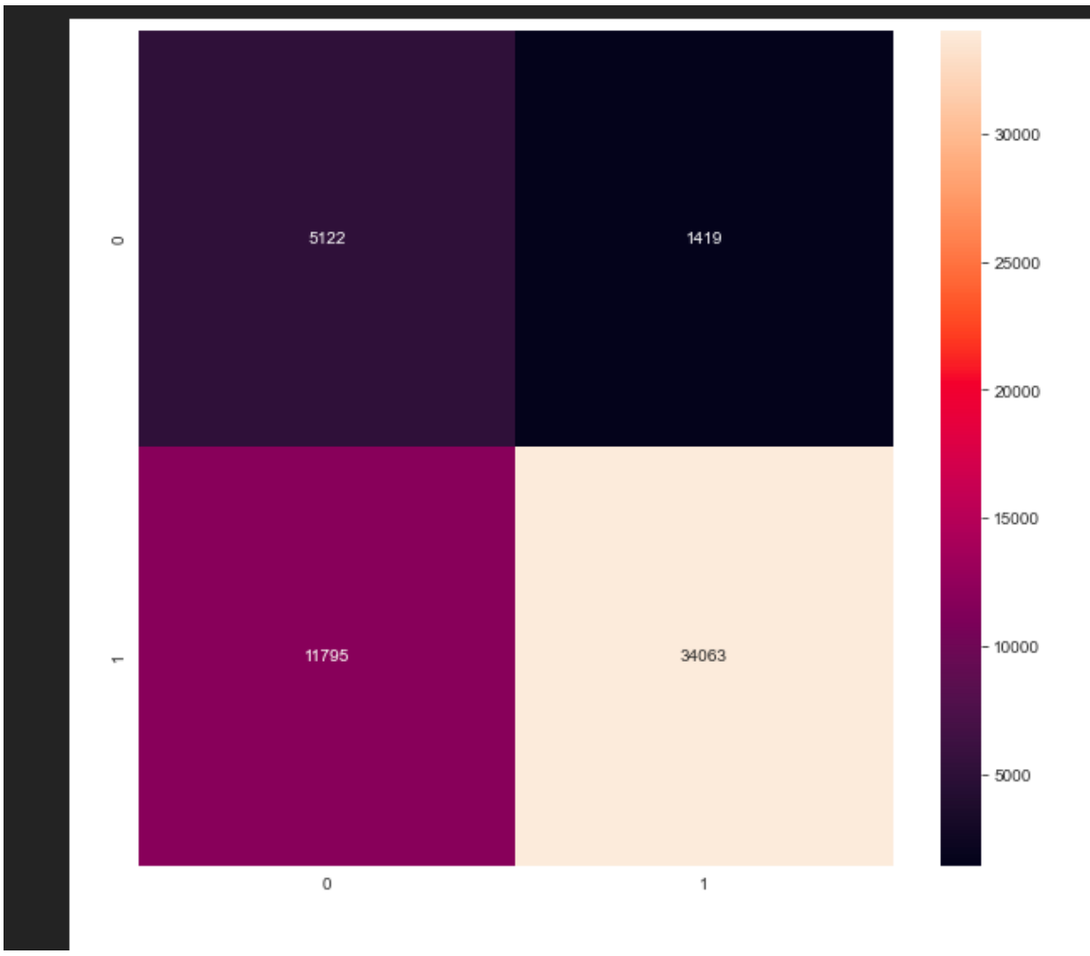


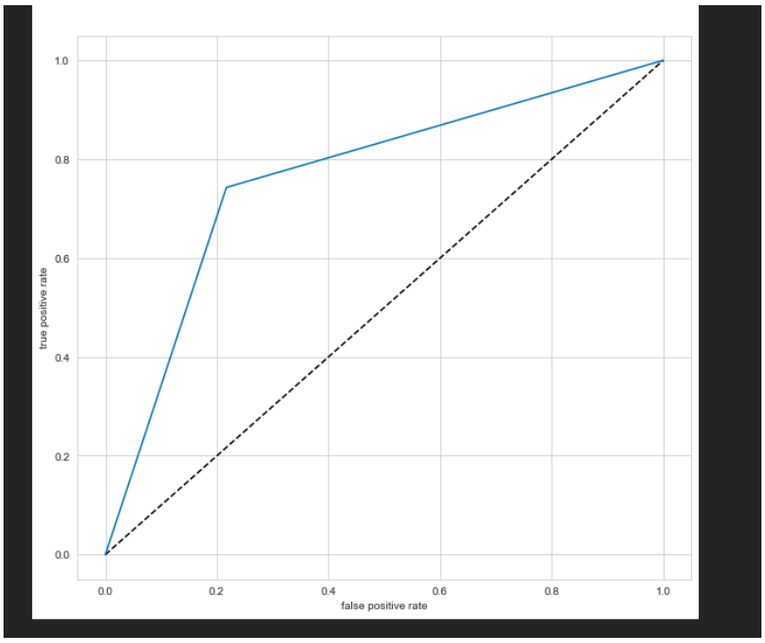


Support Vector Machines

```
svc = LinearSVC(random_state=0, tol=1e-5)
print("*****Linear SVC*****")
svc.fit(x_train, y_train.ravel())
pred2 = svc.predict(xtest)
accuracy = accuracy_score(ytest,pred2)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(svc,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred2)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred2)
roc_auc= auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred2))
precision = precision_score(ytest, pred2)
print('Precision: ', precision)
recall = recall_score(ytest, pred2)
print('Recall: ', recall)
f1 = f1_score(ytest, pred2)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Linear SVC*****

 accuracy score :  0.7478196148781465

 cross validation score :  0.768577434835646

 [[ 5122  1419]
 [11795 34063]]

 ROC_AUC_SCORE :  0.7629268318426439
              precision    recall  f1-score   support

           0       0.30      0.78      0.44      6541
           1       0.96      0.74      0.84     45858

    accuracy                           0.75     52399
   macro avg       0.63      0.76      0.64     52399
weighted avg       0.88      0.75      0.79     52399

Precision:  0.9600078913251789
Recall:  0.7427929696018143
F1 score:  0.8375461027784608
AxesSubplot(0.125,0.536818;0.62x0.343182)
```

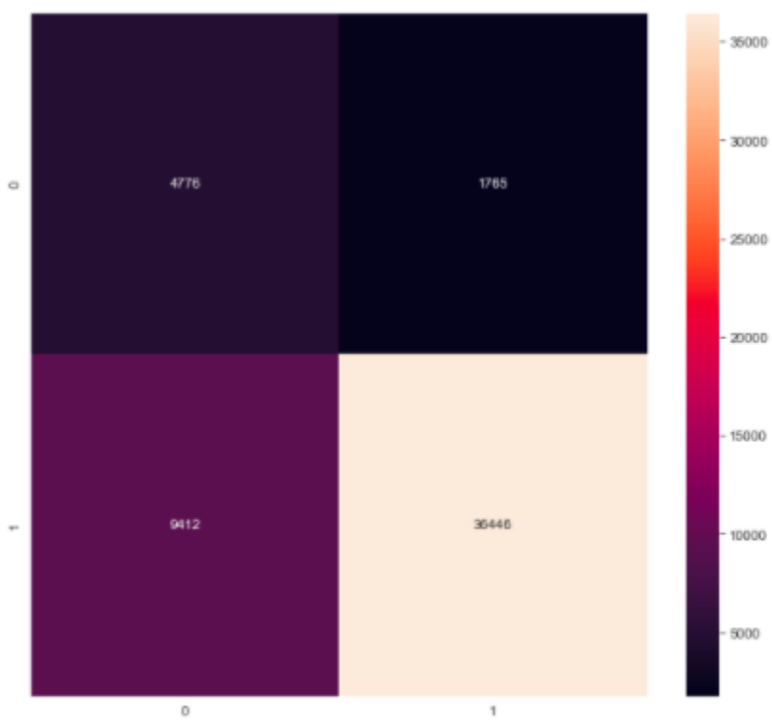


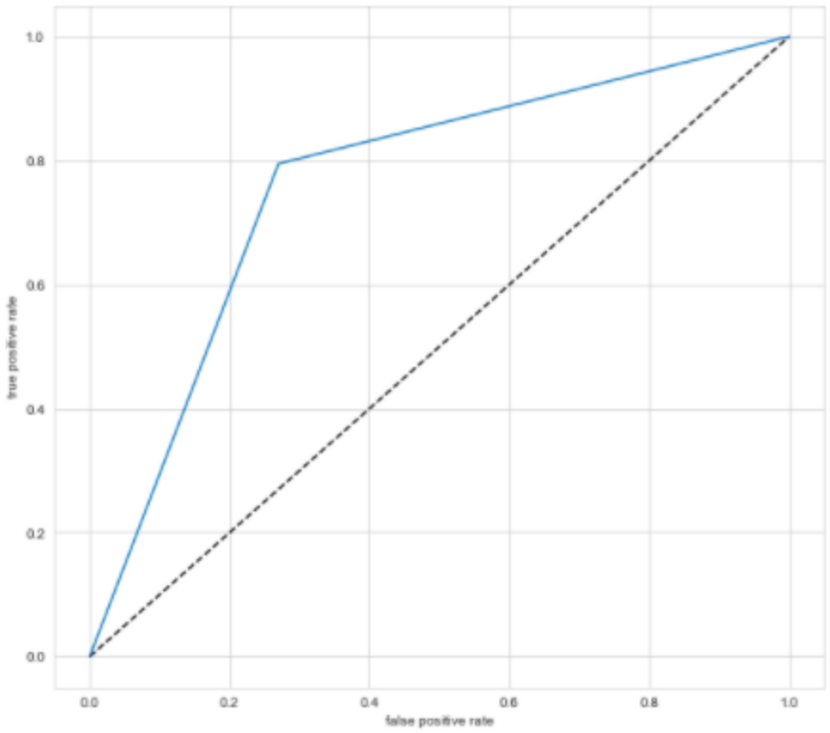


K Nearest Neighbors Classifier

```
knn = KNeighborsClassifier()
print("*****K Neighbors Classifier*****")
knn.fit(x_train, y_train.ravel())
pred3 = knn.predict(xtest)
accuracy = accuracy_score(ytest,pred3)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(knn,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred3)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred3)
roc_auc= auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred3))
precision = precision_score(ytest, pred3)
print('Precision: ', precision)
recall = recall_score(ytest, pred3)
print('Recall: ', recall)
f1 = f1_score(ytest, pred3)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****K Neighbors Classifier*****

 accuracy score :  0.7866944025649344

 cross validation score :  0.8938636457932091

 [[ 4776  1765]
 [ 9412 36446]]

 ROC_AUC_SCORE :  0.7624606569675089
              precision    recall  f1-score   support

           0       0.34      0.73      0.46      6541
           1       0.95      0.79      0.87     45858

    accuracy                           0.79     52399
   macro avg       0.65      0.76      0.66     52399
weighted avg       0.88      0.79      0.82     52399

Precision:  0.9538091125592107
Recall:  0.7947577303851019
F1 score:  0.8670496853774876
AxesSubplot(0.125,0.536818;0.62x0.343182)
```

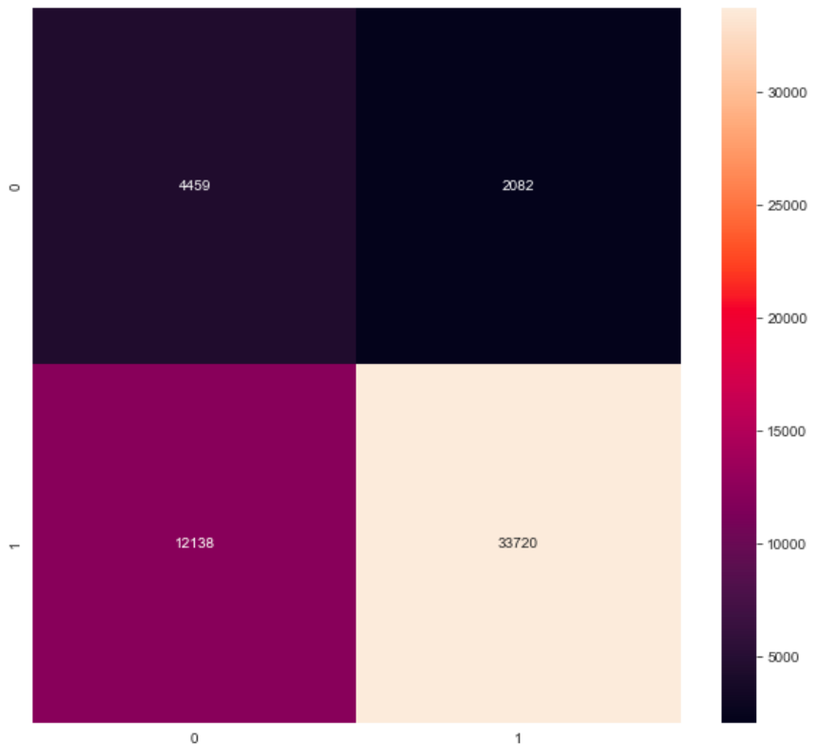


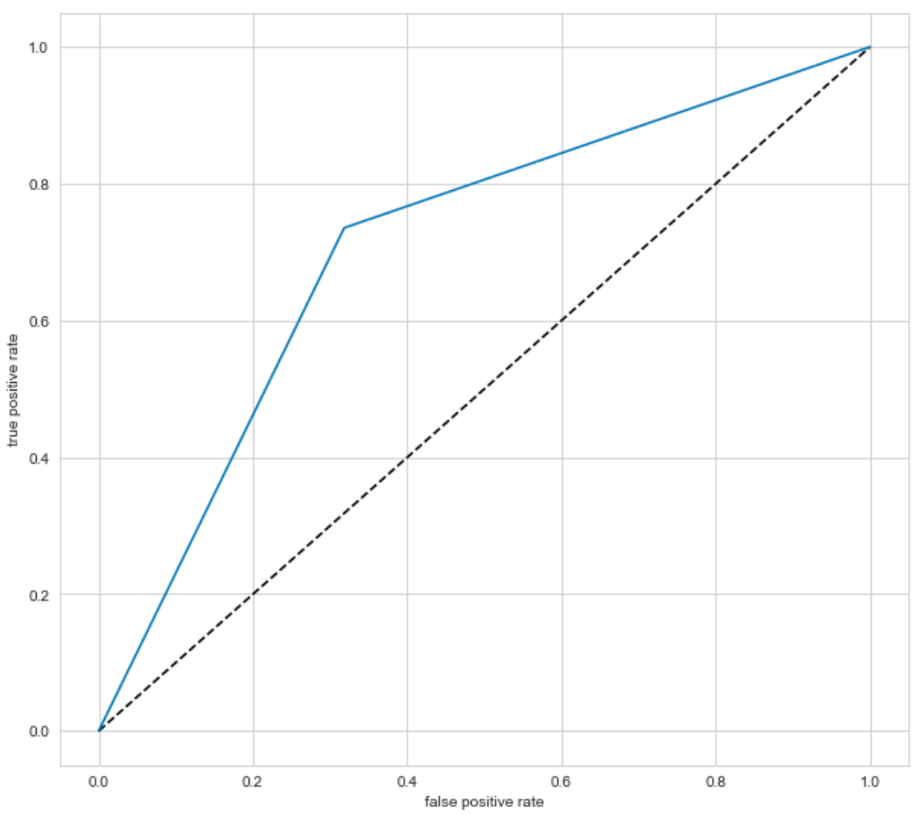

1.0
0.8
0.6
0.4
0.2
0.0
true positive rate
0.0
0.2
0.4
0.6
0.8
1.0
false positive rate

Bernoulli Naive Bayes Classifiers

```
BNB = BernoulliNB()
print("*****Bernoulli Nave Bayes*****")
BNB.fit(x_train, y_train)
pred4 = BNB.predict(xtest)
accuracy = accuracy_score(ytest,pred4)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(BNB,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred4)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred4)
roc_auc = auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred4))
precision = precision_score(ytest, pred4)
print('Precision: ', precision)
recall = recall_score(ytest, pred4)
print('Recall: ', recall)
f1 = f1_score(ytest, pred4)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Bernoulli Nave Bayes*****

 accuracy score :  0.7286207752056337

 cross validation score :  0.7154420049695416

 [[ 4459  2082]
 [12138 33720]]

 ROC_AUC_SCORE :  0.7085067022466788
              precision    recall  f1-score   support

           0       0.27      0.68      0.39      6541
           1       0.94      0.74      0.83     45858

    accuracy                           0.73     52399
   macro avg       0.61      0.71      0.61     52399
weighted avg       0.86      0.73      0.77     52399

Precision:  0.9418468241997654
Recall:  0.7353133586288106
F1 score:  0.8258633357825128
AxesSubplot(0.125,0.536818;0.62x0.343182)
```

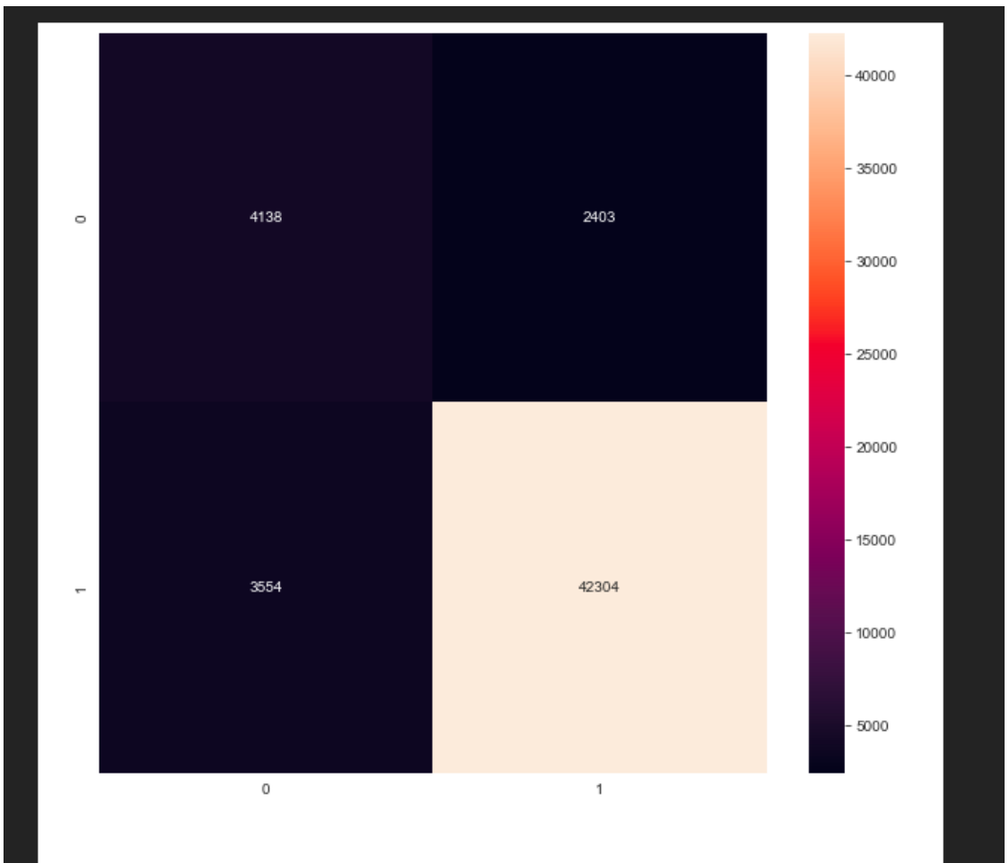

4459
2082
12138
33720
0
1
0
1
30000
25000
20000
15000
10000
5000

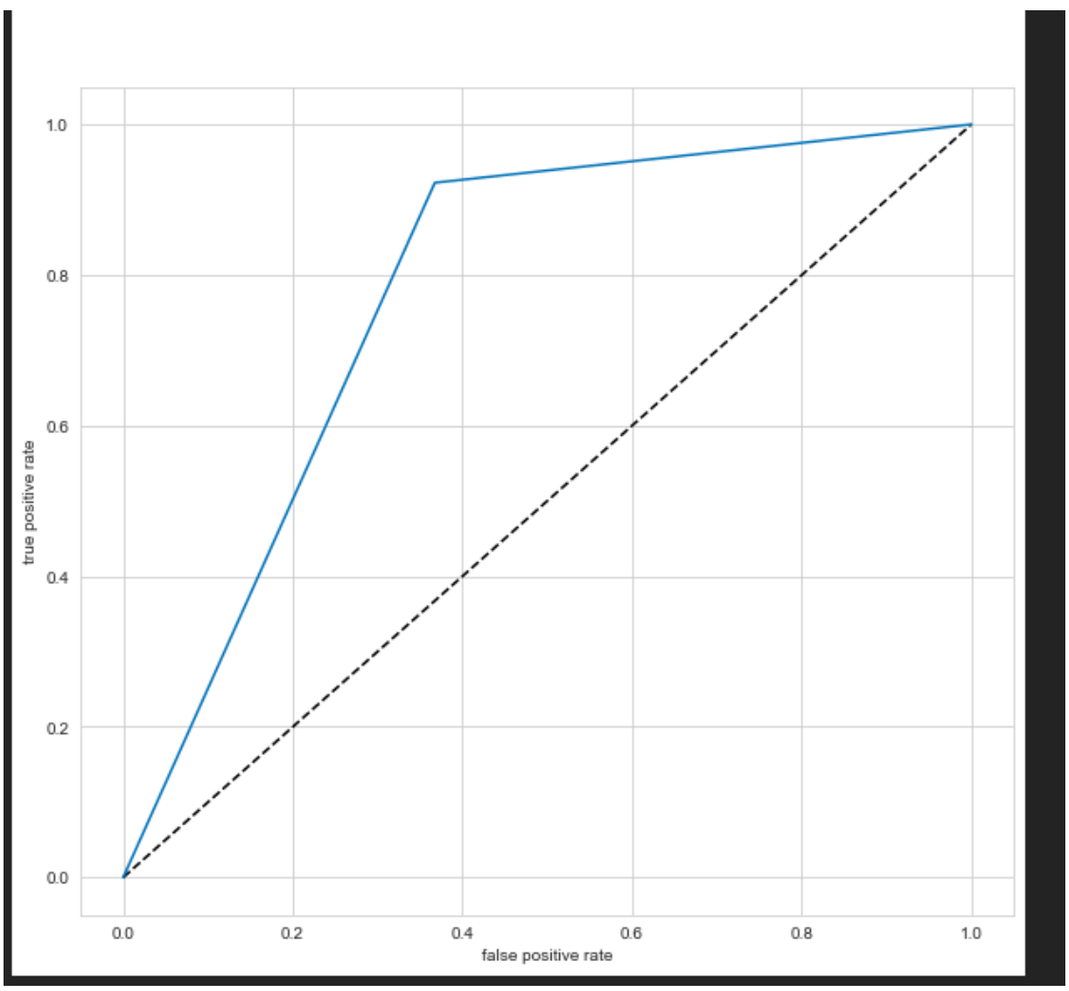

1.0
0.8
0.6
0.4
0.2
0.0
true positive rate
0.0
0.2
0.4
0.6
0.8
1.0
false positive rate

Random forest Classifier

```
RFC = RandomForestClassifier(n_estimators=100)
print("*****Random Forest Classifier*****")
RFC.fit(x_train, y_train.ravel())
pred5 = RFC.predict(xtest)
accuracy = accuracy_score(ytest,pred5)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(RFC,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred5)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred5)
roc_auc= auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred5))
precision = precision_score(ytest, pred5)
print('Precision: ', precision)
recall = recall_score(ytest, pred5)
print('Recall: ', recall)
f1 = f1_score(ytest, pred5)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Random Forest Classifier*****

 accuracy score :  0.8863146243248917

 cross validation score :  0.9356523498367096

 [[ 4138  2403]
 [ 3554 42304]]

 ROC_AUC_SCORE :  0.7775624359287711
              precision    recall  f1-score   support

           0       0.54      0.63      0.58      6541
           1       0.95      0.92      0.93     45858

    accuracy                           0.89     52399
   macro avg       0.74      0.78      0.76     52399
weighted avg       0.90      0.89      0.89     52399

Precision:  0.9462500279598273
Recall:  0.9224998909677701
F1 score:  0.9342240379837685
AxesSubplot(0.125,0.536818;0.62x0.343182)
```

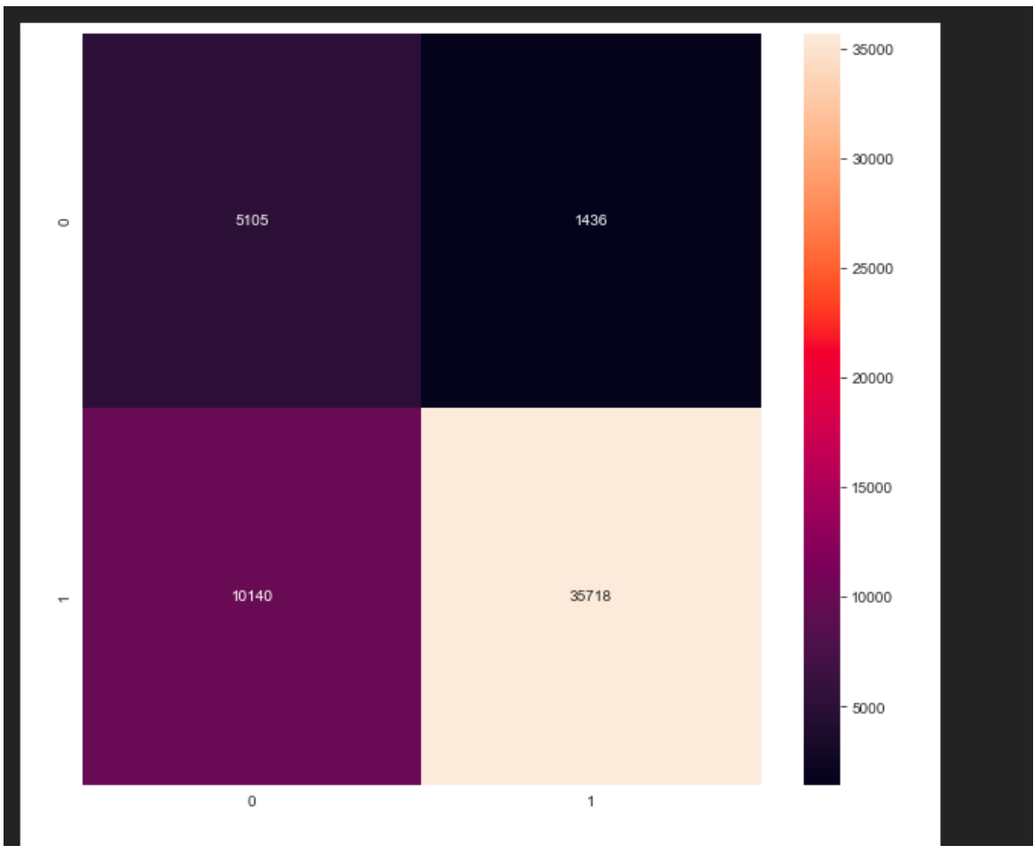

4138
2403
3554
42304
0
1
0
1
40000
35000
30000
25000
20000
15000
10000
5000

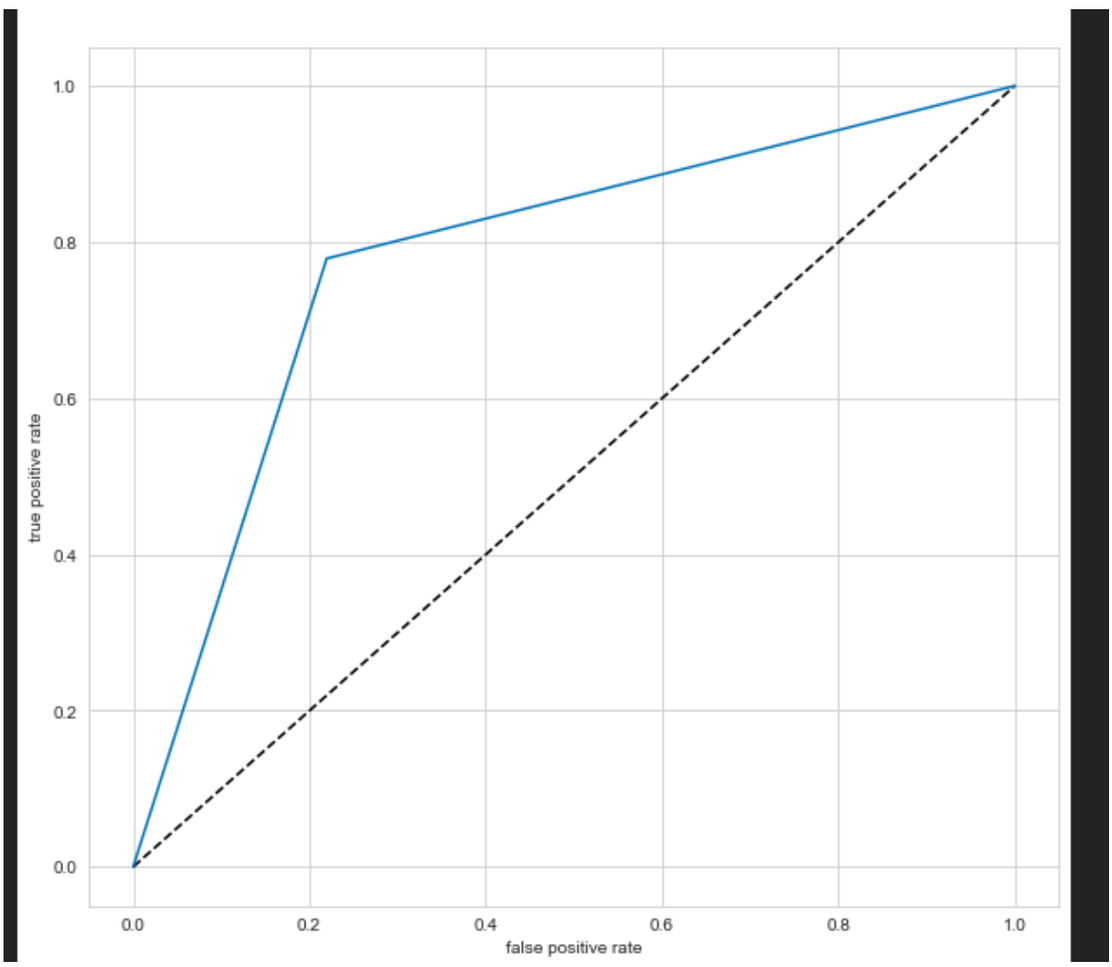

true positive rate
false positive rate
0.0
0.2
0.4
0.6
0.8
1.0

AdaBoost Classifier

```
ADB = AdaBoostClassifier(n_estimators=150)
print("*****Ada boost classifier*****")
ADB.fit(x_train, y_train.ravel())
pred6 = ADB.predict(xtest)
accuracy = accuracy_score(ytest,pred6)
print("\n","accuracy score : ",accuracy)
cross=cross_val_score(ADB,x_train,y_train,cv=10,scoring="accuracy").mean()
print("\n","cross validation score : ",cross)
cm = confusion_matrix(ytest,pred6)
print("\n",cm)
fpr,tpr,thresholds=roc_curve(ytest,pred6)
roc_auc= auc(fpr,tpr)
print("\n","ROC_AUC_SCORE : ",roc_auc)
print(classification_report(ytest,pred6))
precision = precision_score(ytest, pred6)
print('Precision: ', precision)
recall = recall_score(ytest, pred6)
print('Recall: ', recall)
f1 = f1_score(ytest, pred6)
print('F1 score: ', f1)
plt.figure(figsize=(10,20))
plt.subplot(211)
print(sns.heatmap(cm,annot=True,fmt='d'))
plt.subplot(212)
plt.plot([0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

```
*****Ada boost classifier*****

 accuracy score :  0.779079753430409

 cross validation score :  0.7973476477460053

 [[ 5105  1436]
 [10140 35718]]

 ROC_AUC_SCORE :  0.7796721704056038
              precision    recall  f1-score   support

           0       0.33      0.78      0.47      6541
           1       0.96      0.78      0.86     45858

    accuracy                           0.78     52399
   macro avg       0.65      0.78      0.66     52399
weighted avg       0.88      0.78      0.81     52399

Precision:  0.9613500565215051
Recall:  0.7788826377077064
F1 score:  0.860550281886956
AxesSubplot(0.125,0.536818;0.62x0.343182)
```


5105
1436
10140
35718
0
1
0
1
35000
30000
25000
20000
15000
10000
5000


true positive rate
false positive rate
0.0
0.2
0.4
0.6
0.8
1.0

- **Key Metrics for success in solving problem under consideration**

There are few metrics which have been used to identify and predicting any of the best model are :

1) Accuracy score

2) Cross_val_score

3) Roc_AUC_score

4) Classification report (Recall)

5) Confusion Matrix

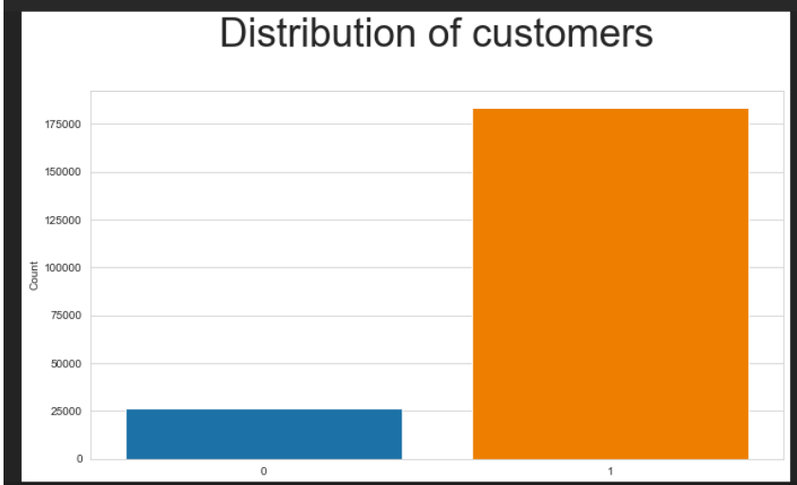
6) True positive rate and False positive rate Statistical graph.

7) F1-score

- **Visualizations**

a.Distribution of target variables

```
# Distribution of Target Variable
plt.figure(figsize=(11,6))
sns.countplot(df3.label)
sns.set_style('whitegrid')
plt.ylabel("Count")
plt.xlabel("Defaulter")
plt.title("Distribution of customers",fontsize=35,y=1.1)
plt.show()
```

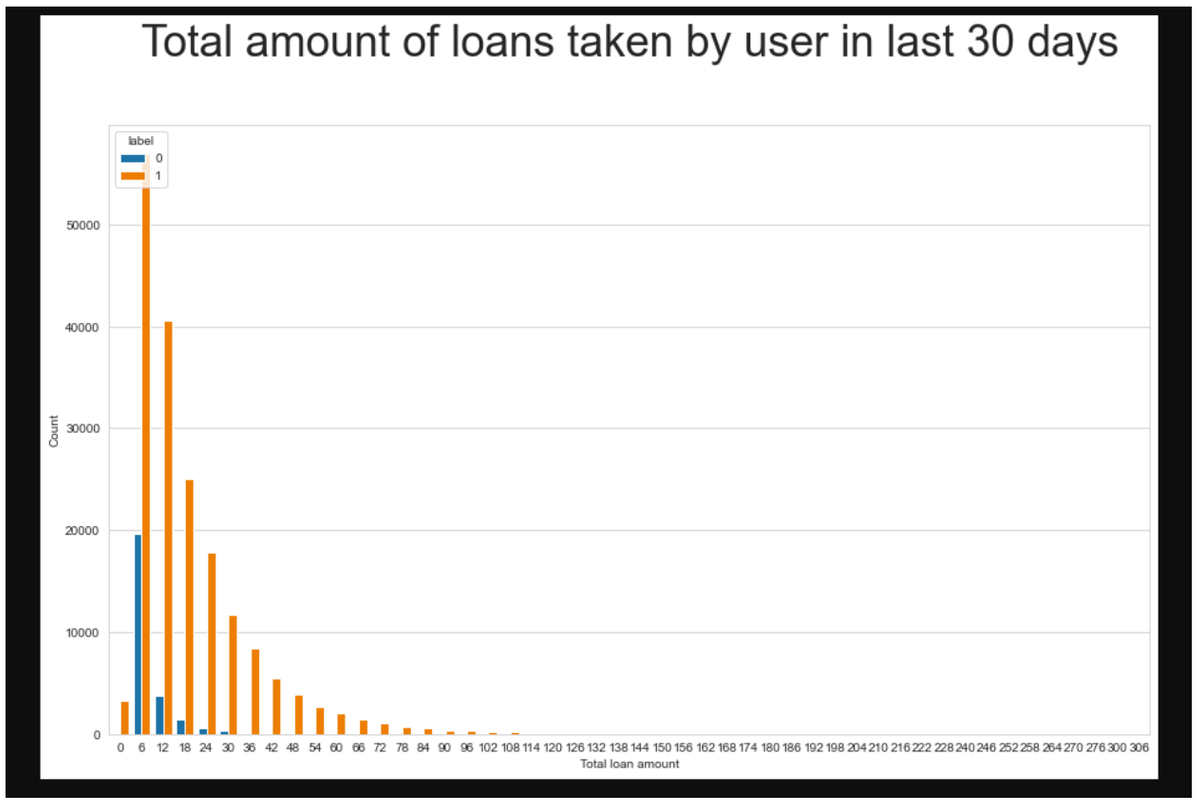


Observation : It is Clearly visible that our target dataset is imbalance .

The above plot depicts imbalance in the dataset

b. Total amount of loans taken by user in last 30 days

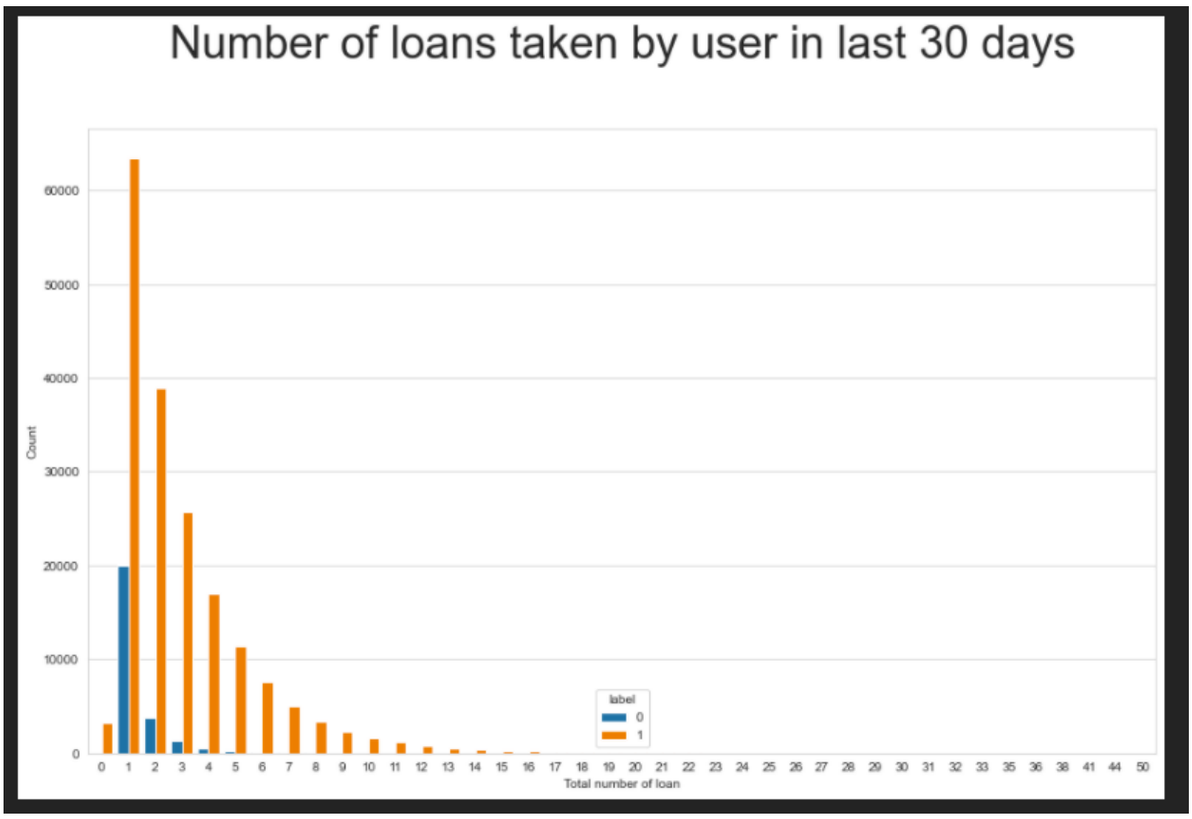
```
plt.figure(figsize=(15,9))
sns.countplot(df3.amnt_loans30,hue=df3.label)
plt.ylabel("Count")
plt.xlabel("Total loan amount")
plt.title("Total amount of loans taken by user in last 30 days",fontsize=35,y=1.1)
plt.show()
```



Observations : Total amount of loan taken by the defaulters is very low as compared to non- defaulters who tend to take mostly 6 rupees loan

c. Number of loans taken by user in last 30 days

```
plt.figure(figsize=(15,9))
sns.countplot(df3.cnt_loans30,hue=df3.label)
plt.ylabel("Count")
plt.xlabel("Total number of loan")
plt.title("Number of loans taken by user in last 30 days",fontsize=35,y=1.1)
plt.show()
```

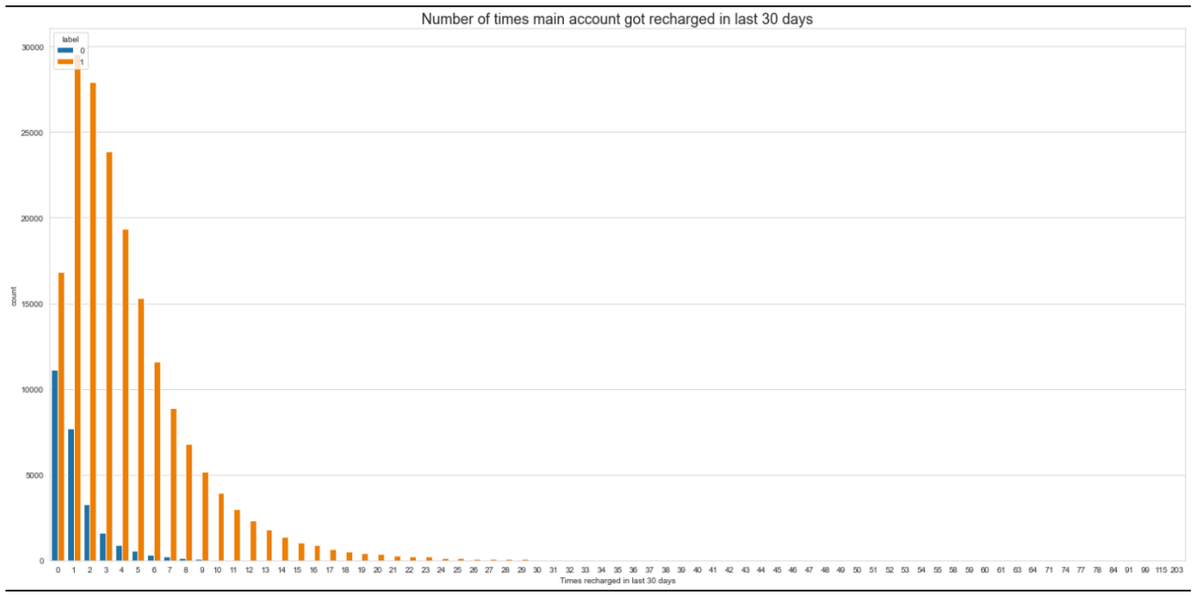


Observations : Since the number of loans taken is decreasing in a month tendcy of being a non-defaulter also goes down.

Risk in granting loan to customers taking loans less number of times in a month.

d. Number of times main account got recharged in last 30 days

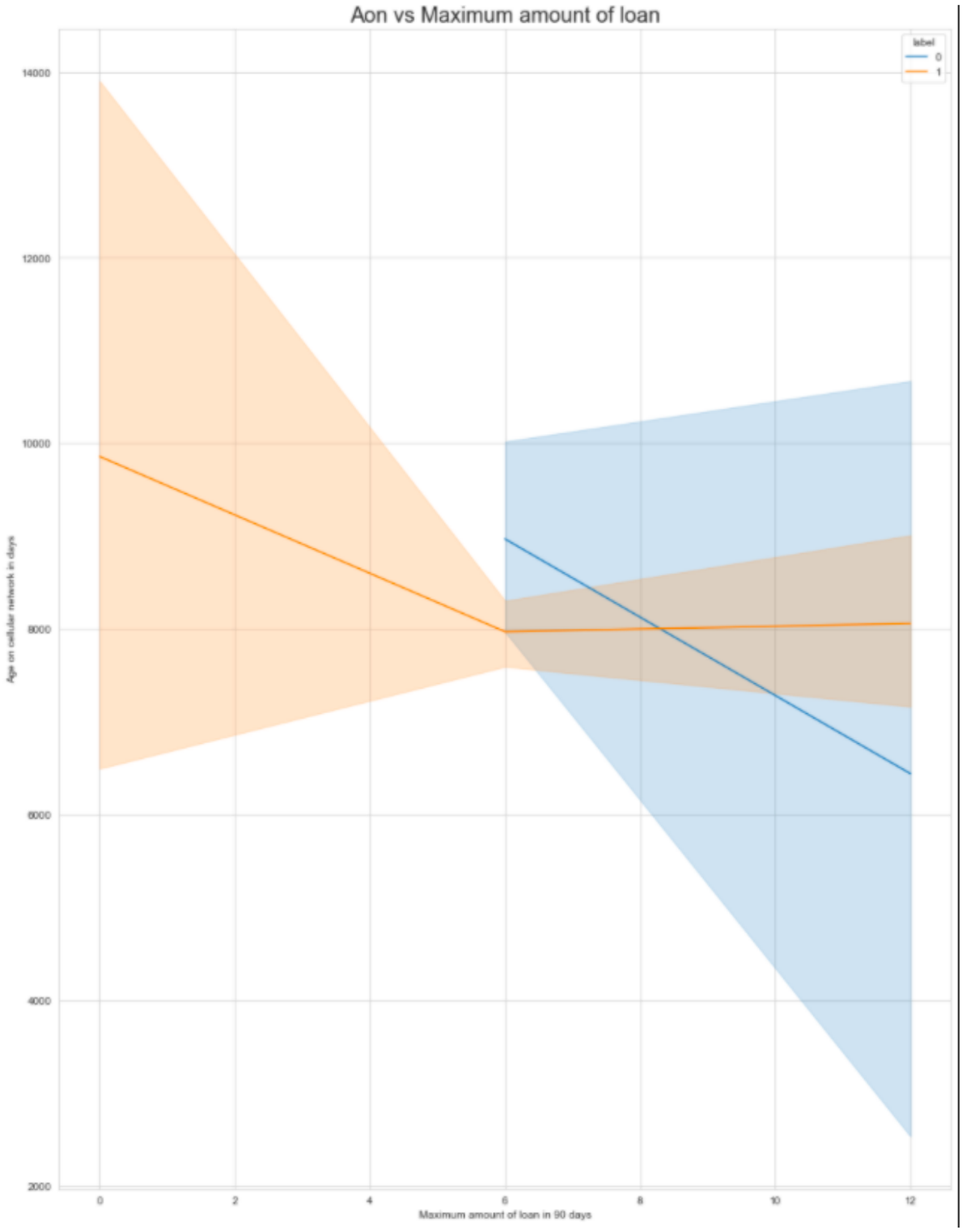
```
plt.figure(figsize=(25,12))
plt.title('Number of times main account got recharged in last 30 days',fontsize=18)
sns.countplot(df.cnt_ma_rech30,hue='label',data=df3)
plt.xlabel("Times recharged in last 30 days")
plt.show()
```



Observations : People who recharge the account frequently in a month have low tendency to be a defaulter. Defaulters do not recharge more than 5 times a month where as non-defaulters do.

e. Aon vs Maximum amount of loan

```
plt.figure(figsize=(15,20))
sns.lineplot(x='maxamnt_loans90', y='aon', data=df, hue='label')
plt.xlabel('Maximum amount of loan in 90 days')
plt.ylabel('Age on cellular network in days')
plt.title('Aon vs Maximum amount of loan',fontsize=20)
plt.show()
```



Observation : From above plot we can infer that customers with aon above 6000 - 8000 tend to be non-defaulters whereas customers ranging between 8000-10000 can be defaulters and customers below aon 6000 are majorly defaulters.

- **Interpretation of the Results**

  Evaluation matrix for the models

```
In [240]: #creating a dataframe for the evaluation metrics of models
data = [['Logistic Regression',75.03,76.88,96.02,74.56,76.44,83.94],
        ['Linear SVC',74.78,76.85,96.00,74.27,76.29,83.75],
        ['Decision Tree',82.22,87.16,93.85,85.28,73.05,89.36],
        ['K Neighbors Classifier',78.66,89.38,95.38,79.47,76.24,86.70],
        ['Bernoulli NB',72.86,71.54,94.18,73.53,70.85,82.58],
        ['Random Forest',88.63,93.56,94.62,92.24,77.75,93.42],
        ['Adaboost',77.90,79.73,96.13,77.88,77.96,86.05]]
selection = pd.DataFrame(data,columns=['Model','Accuracy','Cross validation score','Precision','Rec
selection
```

	Model	Accuracy	Cross validation score	Precision	Recall	Roc_auc score	F1-score
0	Logistic Regression	75.03	76.88	96.02	74.56	76.44	83.94
1	Linear SVC	74.78	76.85	96.00	74.27	76.29	83.75
2	Decision Tree	82.22	87.16	93.85	85.28	73.05	89.36
3	K Neighbors Classifier	78.66	89.38	95.38	79.47	76.24	86.70
4	Bernoulli NB	72.86	71.54	94.18	73.53	70.85	82.58
5	Random Forest	88.63	93.56	94.62	92.24	77.75	93.42
6	Adaboost	77.90	79.73	96.13	77.88	77.96	86.05

Selecting Random Forest as the best model as the accuracy is highest among all other models and Cross validation score, precision, f1-score is also the highest so random forest will perform the best here.

From the above representation we can pick random forest as the best model because it has the highest Accuracy of 88.63 and cross validation score 0f 93.56. In addition to that it has higher precision and F1-score as well.

Hyperparameter Tuning : Did hyperparameter tuning to find the best parameters for the model using Randomized search cv since Gridsearch was running endlessly and was taking a lot of time.

Hyperparameter tuning

```
In [215]: from sklearn.model_selection import RandomizedSearchCV

In [232]: # performing random search cv on random forest model for hypertuning
search_param = {'n_estimators':[100, 150 ,200],
                'criterion':['gini','entropy'],
                'max_depth':[6,8,10,12],
                'max_features':['auto', 'sqrt', 'log2'],
                'class_weight':['balanced','balanced_subsample']
                }
Rsearch = RandomizedSearchCV(RFC, search_param, n_iter=5, cv=5, verbose=5)
# fitting the training data into ranomized search
Rsearch.fit(x_train, y_train)
```

The best parameters for our model

```
#best parameters after tuning
Rsearch.best_params_
```

```
{'n_estimators': 200,
 'max_features': 'log2',
 'max_depth': 10,
 'criterion': 'gini',
 'class_weight': 'balanced'}
```

Tuned the model with the best parameter, but since with these new parameters the accuracy is decreasing will be taking our default Random forest model in consideration, to get a good accuracy of 88.63 which is higher than tuned model 81.41

Tuning with best parameters

```
tune_model = RandomForestClassifier(n_estimators=200,max_features='log2',max_depth=10,criterion='gi
tune_model.fit(x_train, y_train.ravel())
tuned_pred = tune_model.predict(xtest)

accuracy = accuracy_score(ytest,tuned_pred)
print("\n","accuracy score : ",accuracy)

print(classification_report(ytest,tuned_pred))
```

```
 accuracy score :  0.8141758430504399
              precision    recall  f1-score   support

           0       0.38      0.78      0.51      6541
           1       0.96      0.82      0.89     45858

    accuracy                           0.81     52399
   macro avg       0.67      0.80      0.70     52399
weighted avg       0.89      0.81      0.84     52399
```

CONCLUSION

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

Statistical Conclusion :

Maximum amount of loan taken by defaulters is majorly 6 Rupiah as compared to 12 Rupiah

Customers with age on cellular network more than 6000 or 8000 days are generally not defaulters as they pay back the loan on time whereas customers who joined recently or have less age on cellular network have more chances of being defaulter

Loan amount of 6 Rupiah is more in respect to 12 Rupiah in last 90 days

Loans are not majorly paid in the case of 6 Rupiah i.e. loan of 6 Rupiah is majorly taken by customers.

Final Model :

Our final model for this problem will be Random forest since it is giving an accuracy of 88.63 and the f1 score is 93.42 for default parameters. Precision is 94.62 which means false positive is very low i.e. we are able to identify non-defaulters with a good precision. Therefore, will be selecting untuned Random forest model for this problem statement

	Model	Accuracy	Cross validation score	Precision	Recall	Roc_auc score	F1-score
0	Logistic Regression	75.03	76.88	96.02	74.56	76.44	83.94
1	Linear SVC	74.78	76.85	96.00	74.27	76.29	83.75
2	Decision Tree	82.22	87.16	93.85	85.28	73.05	89.36
3	K Neighbors Classifier	78.66	89.38	95.38	79.47	76.24	86.70
4	Bernoulli NB	72.86	71.54	94.18	73.53	70.85	82.58
5	Random Forest	88.63	93.56	94.62	92.24	77.75	93.42
6	Adaboost	77.90	79.73	96.13	77.88	77.96	86.05

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

1. Learned to work with a larger dataset which I haven't worked on before.

2. Learned a few new concepts related to oversampling and model building algorithms.

3. Learned to work with doing less loss to the dataset as it is very expensive and important for precise predictions.

4. Learned to work with metrics in a practical way and got a better idea of how to evaluate the model with help of metrics.

5. Faced some challenges while building and running the algorithm as the dataset was large the algorithms were taking too much time to run.

6. Faced challenges when tried to do hyperparameter tuning it was taking hours and hours to execute since I was applying grid search method, but when used randomised

- **Limitations of this work and Scope for Future Work**

1. Since the dataset was large it was taking a lot of time to run some algorithms, such as ensemble techniques because they use multiple base learners which takes a lot of time to run.

2. Faced endless execution time in hypertuning as well Grid search was never able to execute due to large dataset.

3. There were customers with no loan history due to which the target label was imbalance and the scores would have been more perfect but, since we cannot incur loss of 8-10% for the dataset.

Thank you