PROJECT REPORT

XYZ Corporation Lending Data Project

Submitted towards the partial fulfillment of the criteria for award of Post Graduate Data Science Degree by Imarticus

Submitted By:

Saachi Mohanty
Prachi Gupta
Vaishnav Goregaonkar

Course and Batch: PGDA-05 June 2019

Abstract

People often save their money in the banks which offer security but with lower interest rates. Lending Club operates an online lending platform that enables borrowers to obtain a loan, and investors to purchase notes backed by payments made on loans. It is transforming the banking system to make credit more affordable and investing more rewarding. But this comes with a high risk of borrowers defaulting the loans. Hence there is a need to classify each borrower as defaulter or not using the data collected when the loan has been given.

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGDA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: June 28, 2019

Place: Mumbai

Saachi Mohanty

Prachi Gupta

Vaishnav Goregaonkar

Certificate of Completion

I hereby certify that the project titled "**XYZ Corporation Lending Data Project**" was undertaken and completed under my supervision by Saachi Mohanty, Prachi Gupta and Vaishnav Goregaonkar from the batch of PGDA-05 (June 2019)

Date: June 28, 2019

Place – Mumbai

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Title & Objective of the study

'**XYZ Corporation Lending Data Project'** is the project we are working upon which falls under the BFSI domain (Banking Financial services and Insurance sector). The text files contain complete loan data for all loans issued by XYZ Corp. through 2007-2015. The primary purpose of working on this project is to predict the probability of default, whether the customer will default the loan or not by using the past data. That means, given a set of new predictor variables, we need to predict the target variable as 1 -> Defaulter or 0 -> Non-Defaulter.

1.2 Need of the Study

In this project, the main purpose is to predict whether a borrower will default or not, so that investors can avoid such borrowers using manual investing feature provided by lending club. This, however, does not necessarily lead to highest return on investment (ROI) because by completely avoiding potential defaults, one is also avoiding riskier loans that may lead to higher ROI even though they'll default at some point in the future. In order to maximize ROI, one needs to optimize ROI instead. In this project, we work on the simpler problem that is to predict loan defaults.

1.3 Business or Enterprise under study

XYZ Corporation Lending Data is under the study. Data of Loans issued by XYZ Corp. through 2007-2015 is used for analysis. The data contains the indicator of default, payment information, credit history, etc.

1.4 Business Model of Enterprise

Selecting the relevant variables from the dataset and arranging their values in order of importance to create a models to predict the probability of default of an individual in the future by performing different types of algorithms on the data.

1.5 Data Sources

XYZ Corp Lending Data- Data contains the information about the status of the loan defaulter. The dataset contains the information like age, gender, annual income, grade of the customer paying capacity

Data Set Description:
Contains 855969 rows and 73 columns

The response variable is 'default_ind' with '0' for Non-Default and '1' for Default.

1.6 Tools & Techniques

Tools: Jupyter Notebook.

Techniques: Logistic Regression, Decision Tree Classification, Artificial Neural Networks, Gradient Boosting Classifier.

CHAPTER 2: DATA PREPARATION AND UNDERSTANDING

One of the first steps we engaged in was to outline the sequence of steps that we will be following for our project. Each of these steps are elaborated below

After importing the required libraries, a sequence of steps were followed to perform data preprocessing.

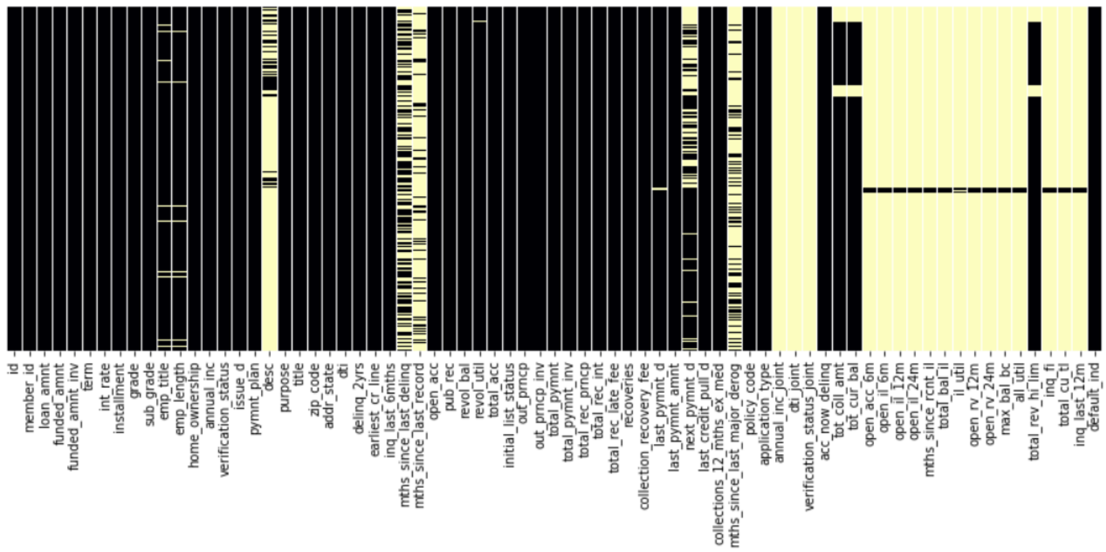
2.1 Phase I – Data Extraction and Cleaning:

- **Missing Value Analysis and Treatment**

After printing the shape of the data, we gain that the dataset consists of 855969 observations and 73 variables.

The initial step was to check the missing values in each variable and for a better view, plot a heatmap of the dataset for visualizing the missing values as shown below:

```
In [11]: plt.figure(figsize=(15,5))
         sns.heatmap(data.isnull(), cbar = False, yticklabels=False, cmap="magma" )

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x23b8a70a2e8>
```



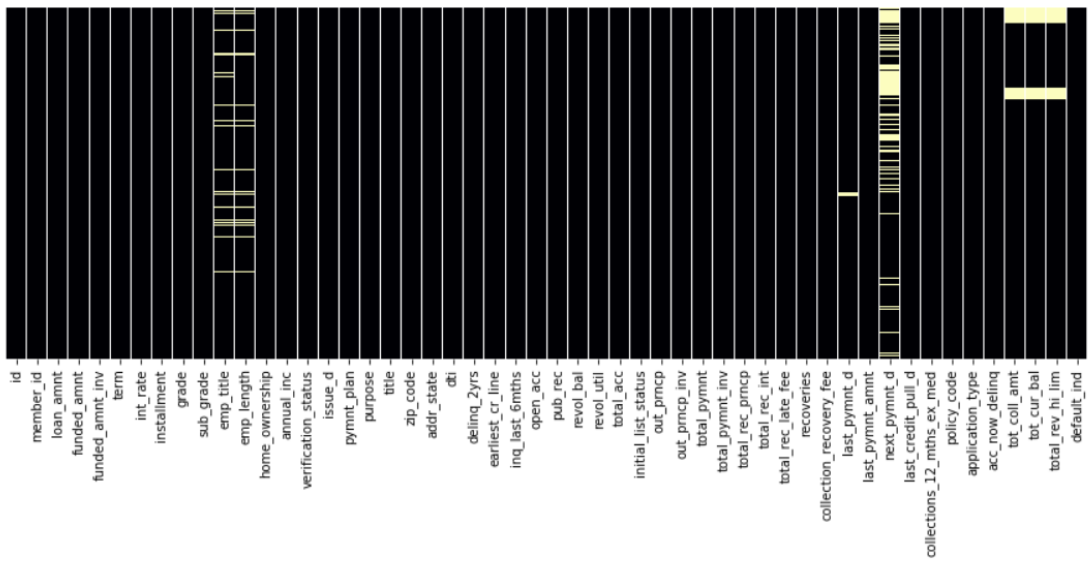
It is evident from the above heatmap that our dataset contains a lot of missing values and we cannot use feature that has so many missing values.

Above heatmap shows the intensity of values that are missing in every columns. All the light colored columns represents the amount of missing values present in that specific column.

Firstly, setting a threshold of 50%, i.e. dropping the columns which have more than or equal to 50% missing values. We are then left with **52 variables**.

Then visualizing the missing values in each column after dropping the variables, we get the following heatmap:

```
In [14]: # Visualising the missing values in each column after dropping the variables

         plt.figure(figsize=(15,5))
         sns.heatmap(data.isnull(), cbar = False, yticklabels=False, cmap="magma" )

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x23bc29dc780>
```



By comparing the above two heatmaps, it is clearly seen that the amount of missing values have been reduced drastically.

Also the dataset does not consists any duplicate records.

The next step was to drop the following irrelevant variables with proper reasoning:

- '**id**', '**member_id**', '**zip_code**' variables because they all are unique numbers.
- '**policy_code'** and '**payment_plan'** variables because they have same value for all observations.
- '**emp_title**' variable because it is a categorical variable with 290912 levels.
- '**last_credit_pull_d**' variable because it's a date variable with 102 levels.
- '**title**' variable because it's a categorical variable with 61000 levels.
- '**next_pymnt_d**' variable because it is a date variable with 3 levels and it contains 29% Missing records.
- '**earliest_cr_line**' variable because it is a date variable with 697 levels.
- '**addr_state**' and '**last_pymnt_d**' variables for trail purpose (51 levels each).
- '**application_type**' contains 'INDIVIDUAL' level for 99.94% of the records.
- '**acc_now_delinq**' contains '0' for 99.5% of the records.

- ‘**sub_grade**’ variable for trial purpose (35 levels).

After dropping the above columns, we are left with 40 variables of whose missing values will further be treated.

The remaining missing values present are treated by using **Mean** and **Mode**.

Missing values treatment with Mean:

The missing values of the following variables are treated with mean:

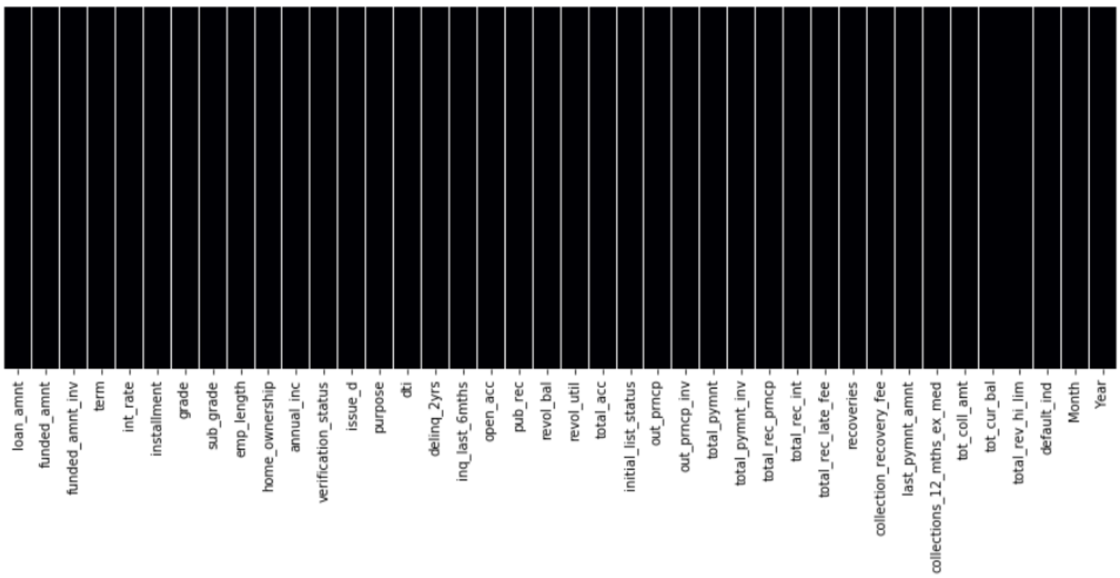
- tot_cur_bal
- tot_coll_amt
- total_rev_hi_lim
- revol_util

While the missing values of the following variables are treated with Mode:

- collections_12_mths_ex_med
- emp_length

After the complete treatment of the missing values, it is evident from the below heatmap that the dataset is now clean and ready for EDA.

```
In [49]: # Visualising the missing values in each column after dropping the variables

         plt.figure(figsize=(15,5))
         sns.heatmap(data.isnull(), cbar = False, yticklabels=False, cmap="magma" )

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x20bb84c7e48>
```



- **Handling Outliers**

Outlier Treatment was not done because of the following reasons:

- ❖ Presence of Clusters in the outliers.
- ❖ Less number of outliers as compared to the huge number of observations whose effect will be negligible.
- ❖ Lack of Domain knowledge.

2.2 Phase II - Feature Engineering

After building the Logistic Regression, Decision Tree and the ANN (on balanced and unbalanced dataset) as well as applying tuning and cross validation, we created a new dataframe with different variable selections to check the effect on the model and also decrease the errors.

```
In [77]: #Feature Selection
         # Out of 73 , few variables are not helpful or impactful in order to build a predictive model, hence dropping.

         data.drop(['id','member_id','funded_amnt_inv','grade','emp_title','pymnt_plan','desc','title','addr_state',
                    'inq_last_6mths','mths_since_last_record','initial_list_status','mths_since_last_major_derog','policy_code
         ',
                    'dti_joint','verification_status_joint','tot_coll_amt','tot_cur_bal','open_acc_6m','open_il_6m','open_il_12
         m'
                    ,'open_il_24m','mths_since_rcnt_il','total_bal_il','il_util','open_rv_12m','open_rv_24m',
                    'max_bal_bc','all_util','inq_fi','total_cu_tl','inq_last_12m'],axis=1,inplace=True)

         data.shape #(855969, 41)

Out[77]: (855969, 41)
```

After imputing the missing data for categorical variable with mode and for numerical variable with mean value/zeros, we split the dataset into Train and Test.

In [82]:

```
#Train and Test split

# issue_d is object datatype to make use for split converting issue_d in Date

data.issue_d = pd.to_datetime(data.issue_d)    #%y-%m-%d
col_name = 'issue_d'
print (data[col_name].dtype)

#split data in train and test

split_date = "2015-05-01"

train = data.loc[data['issue_d'] <= split_date]
train=train.drop(['issue_d'],axis=1)
#train.head()
train.shape     #(598978, 40)

test = data.loc[data['issue_d'] > split_date]
test=test.drop(['issue_d'],axis=1)
#test.head()
test.shape   #(256991, 40)
```

```
datetime64[ns]
```

Out[82]:

```
(256991, 40)
```

In [84]:

```
#selecting X and Y

X_train=train.values[:,:-1]
Y_train=train.values[:,-1]
Y_train=Y_train.astype(int)
print(Y_train)

X_test=test.values[:,:-1]
Y_test=test.values[:,-1]
Y_test=Y_test.astype(int)
print(Y_test)
```

```
[0 1 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
```

2.3 Exploratory Data Analysis:

EDA is the process of performing initial investigations on data to discover patterns, to test hypothesis and to check assumptions with the help of descriptive statistics and graphical representations.

The response variable in this data is '**default_ind**' which indicates that the customer will Default ('1') or Non-Default ('2')

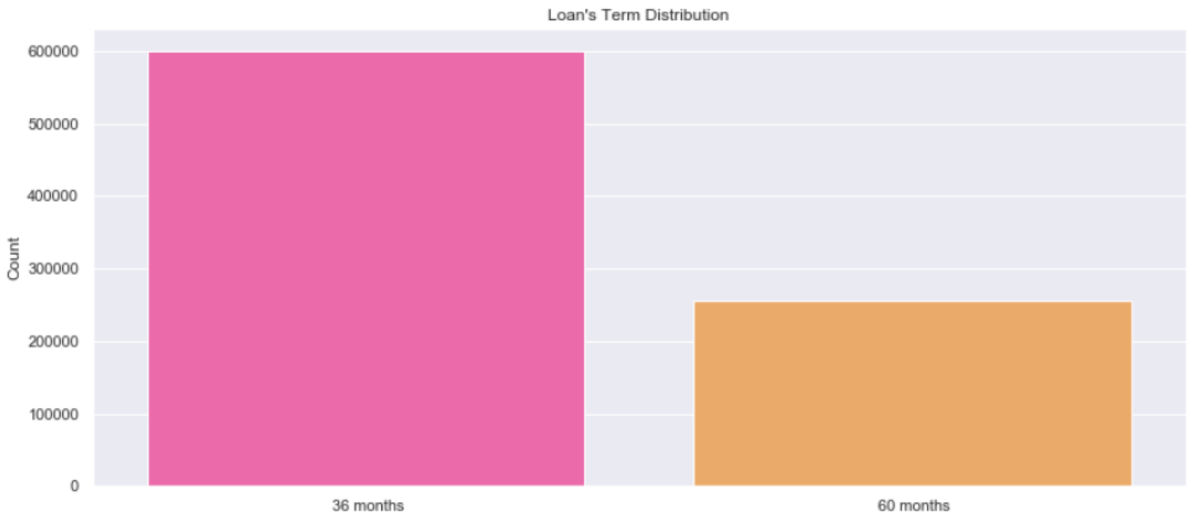
- Plot showing the count of the Default customers and Non-default customers in '**default_ind**' variable.
  -

Non-Default Customer: 94.57 % of the dataset.

Default Customer: 5.43 % of the dataset.

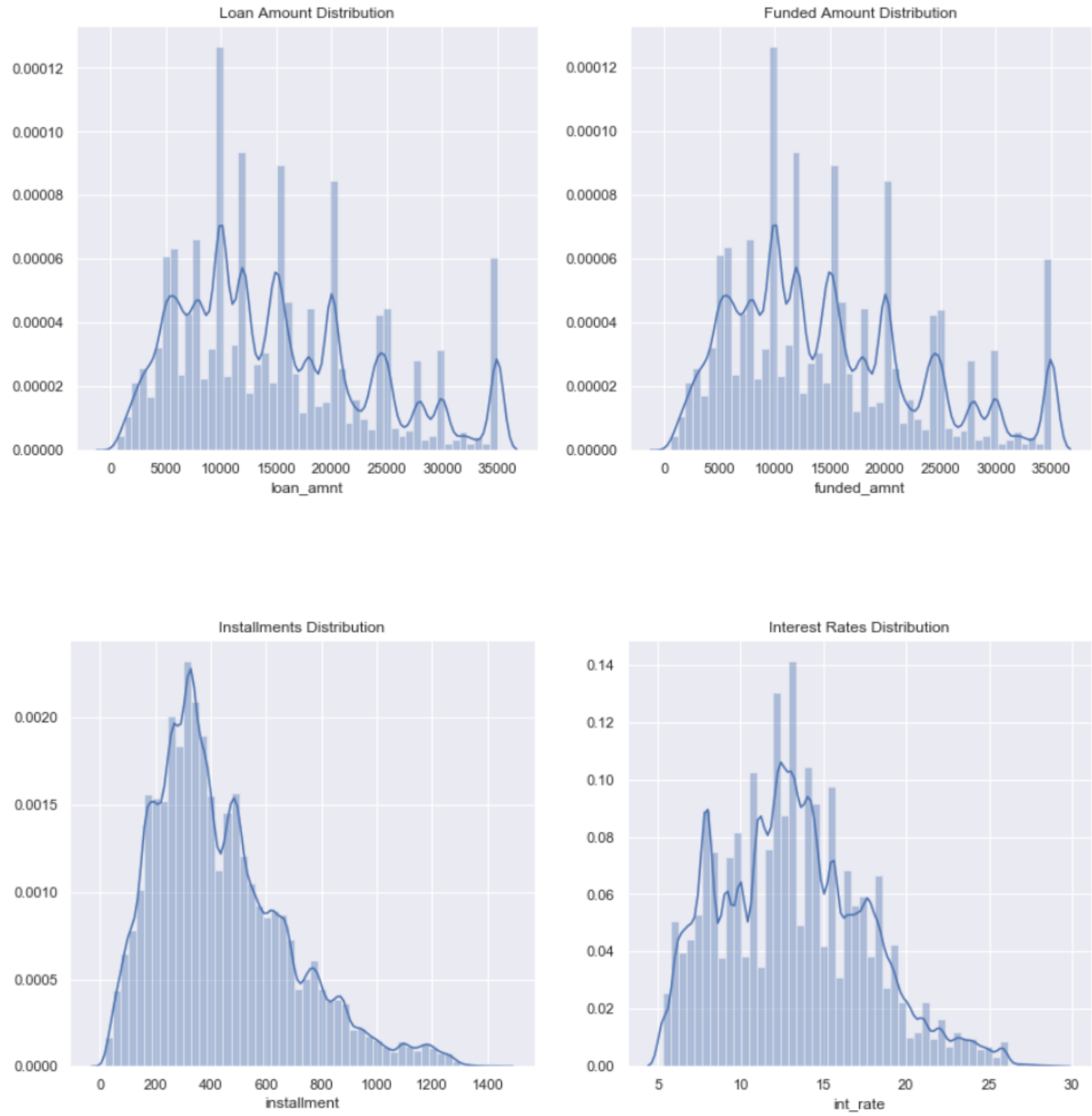
From the above graph, we gain that the dataset is highly unbalanced.

- Plot showing the distribution of 'term' variable.
  -

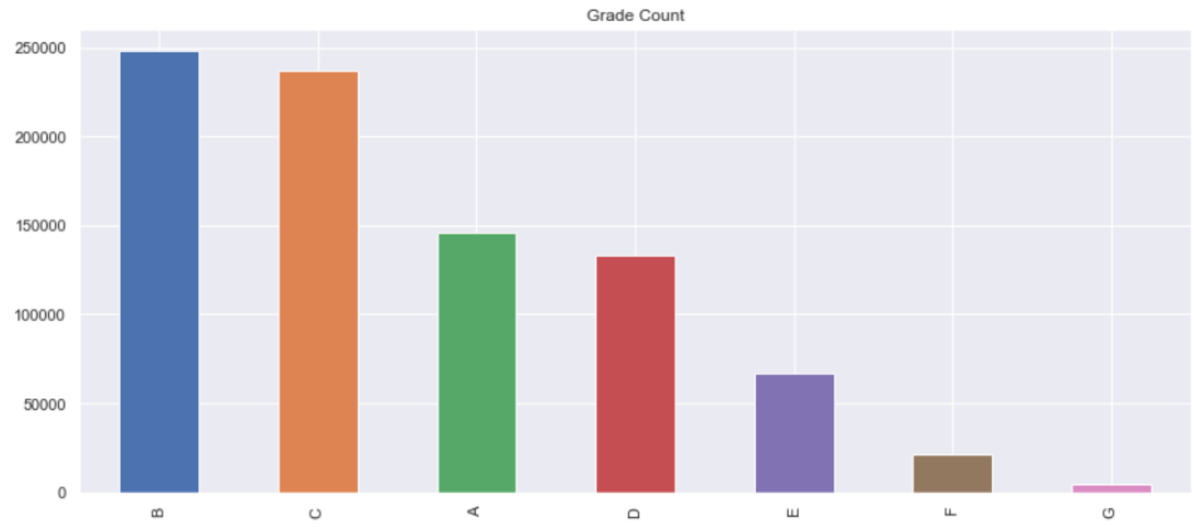


- Plot showing the distribution of loan amount, funded amount, Installments distribution and Interest rates distribution.

•

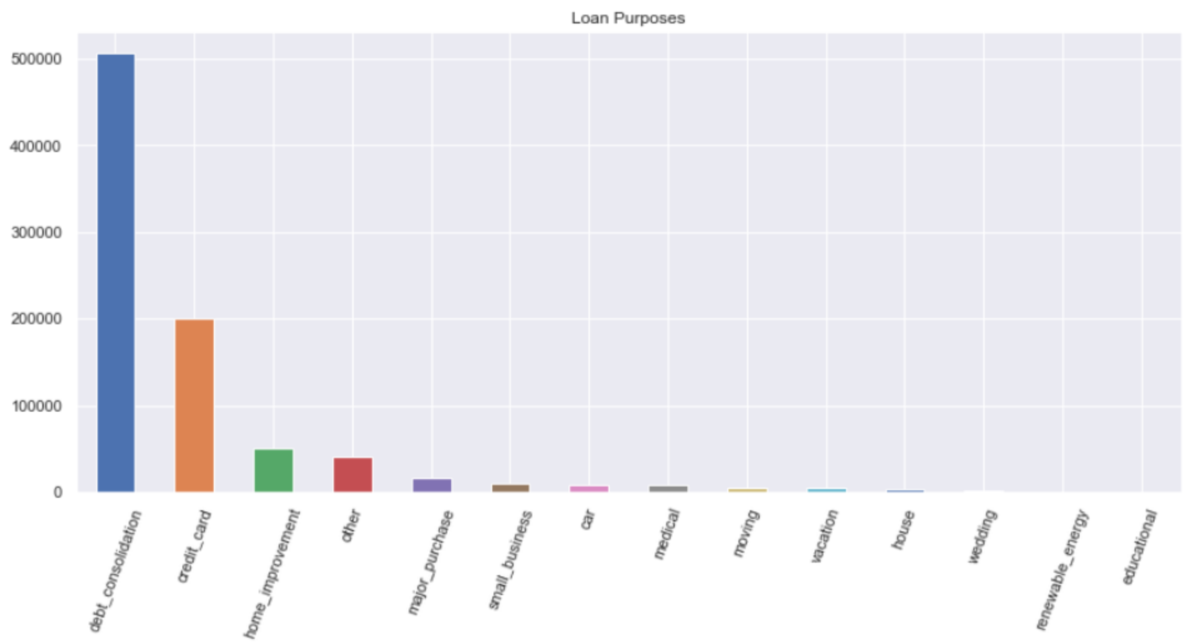


• Plot showing the Grade count.



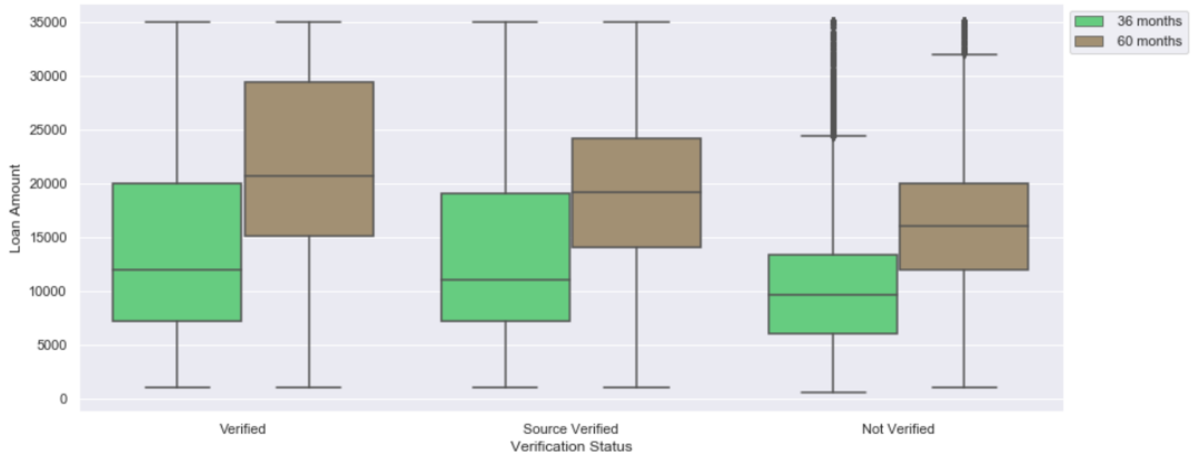
It appears that B and C are the dominant grades.

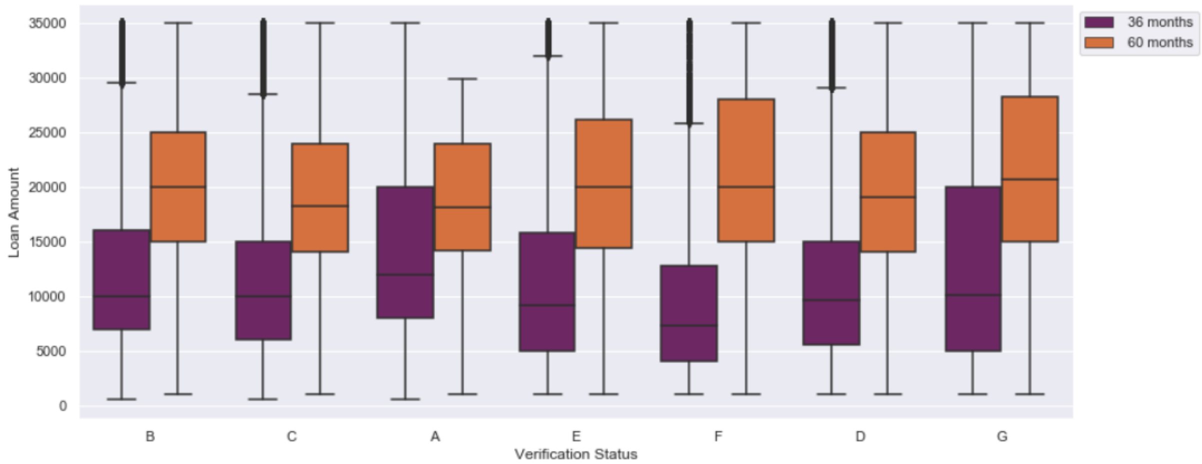
- Plot showing the purpose for which the loan was taken by every individual.



From the above figure, it is observed that huge loans are taken for debt consolidation.

- Plot showing Loan amount by verification status.



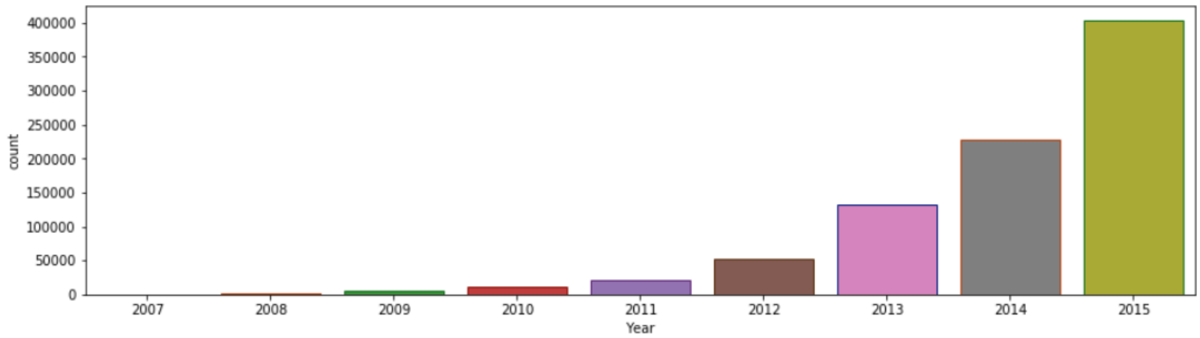


- Plot showing Issue date of the loan amount

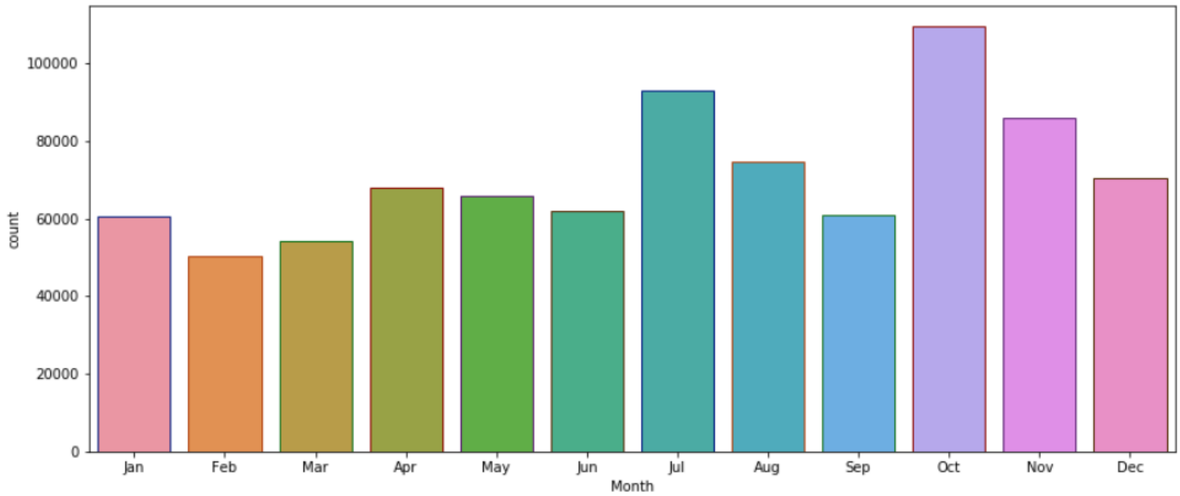
A function is created that will split the '**issue_d**' variable which is nothing but the month in which the loan was funded.

In [15]:

```
def getMonth(x):
    return x.split('-')[0]


def getYear(x):
    return x.split('-')[1]


data['Month'] = data.issue_d.apply(getMonth)
data['Year'] = data.issue_d.apply(getYear)
```

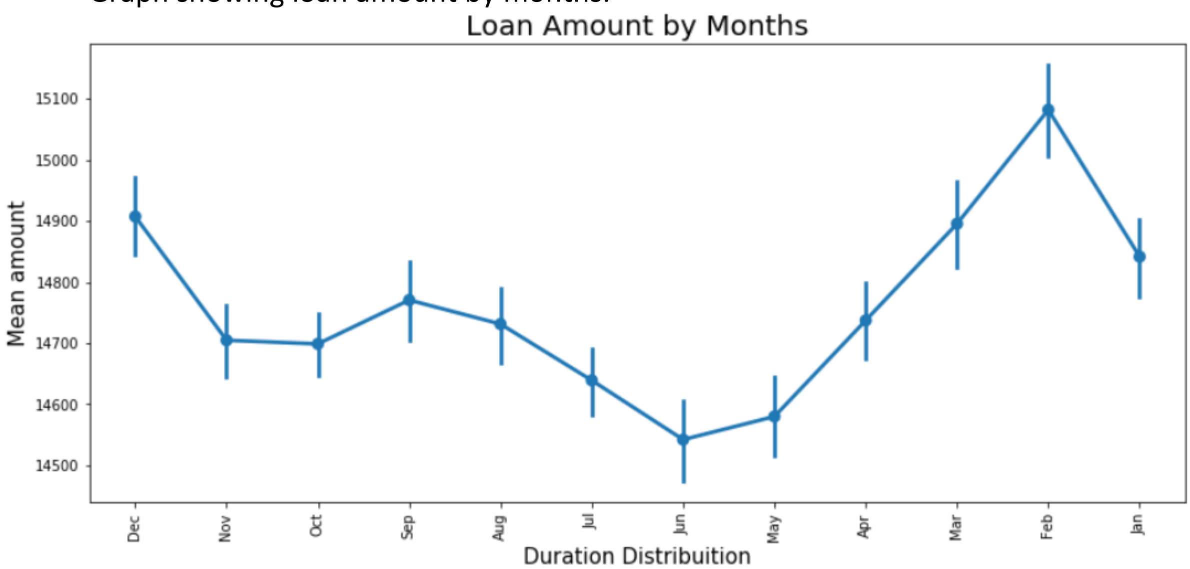


An exponential rise is observed in the number of applications for loan over a period of years.



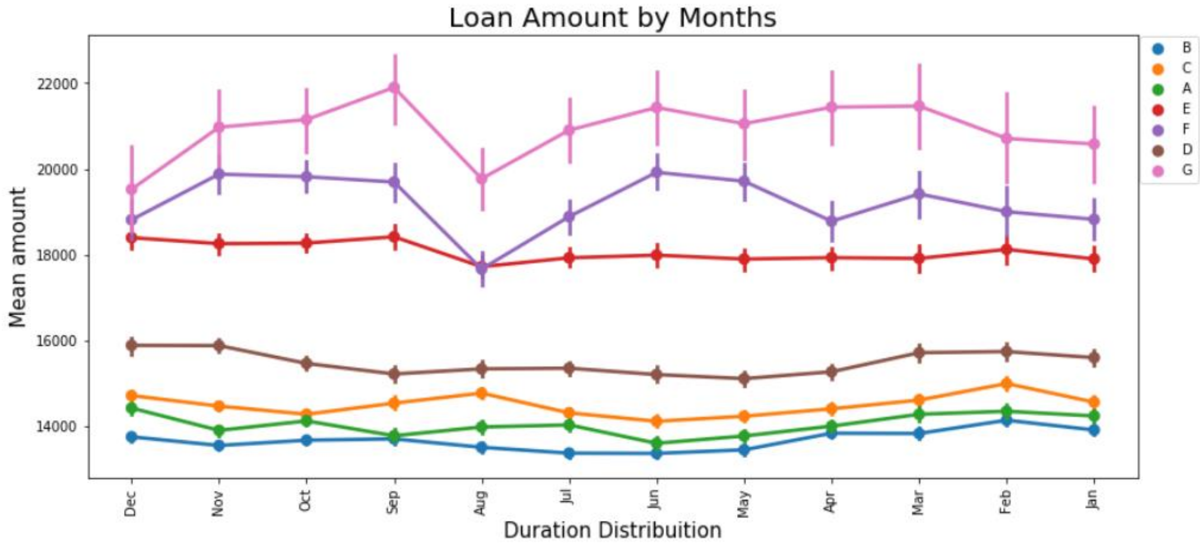
When sorted by months, we can clearly observe that the month of October and July have the highest number of applications.

- Graph showing loan amount by months.



The amount for loan applied is highest in the month of February while it is lowest in June and May.

- Graph showing loan amount by months and grade.



From the above graph, following inferences can be achieved:

- Customers with Grade G have the highest amount of loan applied.
- Customers with Grade B have the lowest amount of loan applied.

- Plot showing distribution of length of employment in years for the issued loans.

2.4 ENCODING

LABEL ENCODING

The SciKit Learn library in Python consists of two encoders which are used to convert categorical data or text data into numbers which will help our model to understand.

The two encoders are **Label Encoder** and **One Hot Encoder.**

By importing the LabelEncoder class from the sklearn library, a categorical data or text data can be converted to numbers, fit and transform the respective categorical variable data and then replace the existing text data with the new encoded data.

Now when the data has been encoded into numbers, the model might get confused into thinking that a column has data with some kind of order or hierarchy. Therefore, to overcome this One Hot Encoder is used.

MANUAL LABEL ENCODING

- Employee length in years has 11 levels. The possible values we can assign is from 0 to 10 with 0 indicating less than one year and 10 indicating experience of ten or more years.

```
In [61]: data['emp_length'] = data['emp_length'].map({'< 1 year':0, '1 year':1, '2 years':2,
                                                       '3 years':3, '4 years':4, '5 years':5,
                                                       '6 years':6, '7 years':7, '8 years':8,
                                                       '9 years':9, '10+ years':10})
```

```
In [62]: data['emp_length'].value_counts()
```

```
Out[62]: 10    325151
         2      75986
         0      67597
         3      67392
         1      54855
         5      53812
         4      50643
         7      43204
         8      42421
         6      41446
         9      33462
         Name: emp_length, dtype: int64
```

- Similarly the term which consists of 2 levels (36months and 60 months) are label encoded with 1 and 2 respectively.

```
In [68]: # map function not working
         data['term'] = data['term'].replace({'36 months':1,'60 months':2},regex = True)
```

```
In [69]: data['term'].value_counts()
```

```
Out[69]: 1    600221
         2    255748
         Name: term, dtype: int64
```

- Initial list status which indicates whether the loan is an individual application or a joint application with two co-borrowers. Replacing f and w with 1 and 2 respectively.

```
In [71]: data['initial_list_status'] = data['initial_list_status'].map({'f':1,'w':2})

In [72]: data['initial_list_status'].value_counts()

Out[72]: 1    442555
         2    413414
         Name: initial_list_status, dtype: int64
```

- Verification status with 3 levels: Source Verified, Verified and Not Verified replaced with 1, 2 and 3 respectively.

```
In [74]: data['verification_status'] = data['verification_status'].map({'Source Verified':1,'Verified':2, 'Not Verified':3})

In [75]: data['verification_status'].value_counts()

Out[75]: 1    318178
         2    280049
         3    257742
         Name: verification_status, dtype: int64
```

- Home ownership which has 6 levels such as 'Mortgage',' Rent', 'Own', 'Other', 'None' and 'Any' have been label encoded as well.

```
In [77]: data['home_ownership'] = data['home_ownership'].map({'MORTGAGE':1,'RENT':2,'OWN':3,'OTHER':4,'NONE':5,'ANY':6})

In [78]: data['home_ownership'].value_counts()

Out[78]: 1    429106
         2    342535
         3     84136
         4       144
         5        45
         6         3
         Name: home_ownership, dtype: int64
```

- 7 levels of Grades which was assigned by XYZ Corp also needed label encoding as well as the purpose variable with 14 levels provided by the borrower for the loan request. Grade:

```
In [80]: data['grade'] = data['grade'].map({'A':1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7})

In [81]: data['grade'].value_counts()

Out[81]: 2    247998
         3    236855
         1    145665
         4    132802
         5     66448
         6     21328
         7      4873
         Name: grade, dtype: int64
```

Purpose:

```
In [83]: data['purpose'] = data['purpose'].map({'debt_consolidation':1, 'credit_card':2,
                                               'home_improvement':3, 'other':4, 'major_purchase':5,
                                               'small_business':6, 'car':7, 'medical':8,
                                               'moving':9, 'vacation':10, 'house':11,'wedding':12,
                                               'renewable_energy':13,'educational':14})
```

```
In [84]: data['purpose'].value_counts()

Out[84]: 1     505392
         2     200144
         3      49956
         4      40949
         5      16587
         6       9785
         7       8593
         8       8193
         9       5160
         10      4542
         11      3513
         12      2280
         13       549
         14       326
         Name: purpose, dtype: int64
```

The final data is prepared and we are left with 37 variables.

```
In [88]: data.shape

Out[88]: (855969, 37)
```

CHAPTER 3: FITTING MODELS TO DATA

3.1 Data Partition:

The data is divided based on the 'issue_d' variable from which the records from **June-2007 to May-2015** will go into Training data while the records from **June-2015 to Dec-2015** will fall in the Testing data.

So to treat the date column i.e. 'issue_d', Split the column into two different columns and replace the values as per the requirement. Then with the help of map function join the split columns and merge them one with a different name ('period'). Followed by sorting the 'period' column and making it an index for slicing according to the requirement.

```
In [6]: data['str_split'] = data.issue_d.str.split('-')

In [7]: data['m'] = data.str_split.str.get(0)

In [8]: data['y']=data.str_split.str.get(1)

In [9]: data['m'] = data['m'].replace({'Jan':'01','Feb':'02','Mar':'03','Apr':'04','May':'05','Jun':'06',
                                       'Jul':'07','Aug':'08','Sep':'09','Oct':'10','Nov':'11','Dec':'12'}
                                 ,regex = True)

In [10]: data["period"] = data["y"].map(str) + data["m"]
```

Followed by dropping the irrelevant columns such as 'issue_d', 'str_split', 'm' and 'y', we are left with 36 variables.

Slicing the data into train and test

Train

In [6]:
```
train_data = data.loc['200706':'201505',:]
```

In [7]:
```
train_data.shape
```

Out[7]:
```
(598978, 36)
```

In [ ]:
```

```

In [8]:
```
train_data.head()
```

Test

In [10]:
```
test_data = data.loc['201506':'201512',:]
```

In [11]:
```
test_data.shape
```

Out[11]:
```
(256991, 36)
```

In [12]:
```
test_data.head()
```

Creating the x_train, y_train, x_test and y_test dataframes:

In [67]:
```
x_train = pd.DataFrame(train_data.values[:,:-1])
```

In [68]:
```
y_train = pd.DataFrame(train_data.values[:,-1])
```

In [69]:
```
x_test = pd.DataFrame(test_data.values[:,:-1])
```

In [70]:
```
y_test = pd.DataFrame(test_data.values[:,-1])
```

3.2 Feature Scaling

Feature scaling involves rescaling the features so as to limit the range of variables so that they can be compared on common grounds. Using the sklearn library and importing the StandardScaler class, we can use feature scaling.

```
In [74]: from sklearn.preprocessing import StandardScaler

         sc = StandardScaler()

         x_train_scale = pd.DataFrame(sc.fit_transform(x_train))
         x_test_scale = pd.DataFrame(sc.transform(x_test))
```

4.1 MODEL BUILDING

Firstly, we created a custom function for **Confusion Matrix** for better understanding and organized look.

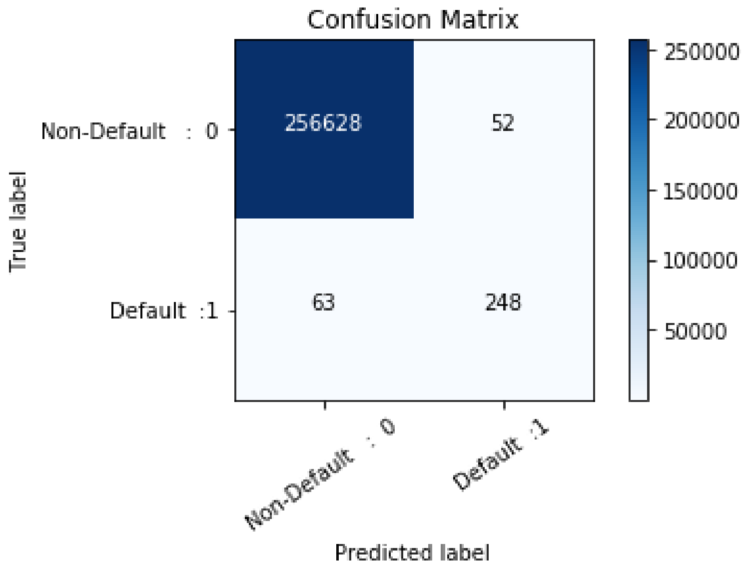
Custom function for Confusion matrix

```
In [1]: import matplotlib.pyplot as plt
        from sklearn.metrics import confusion_matrix
        from sklearn.utils.multiclass import unique_labels
        import itertools


        def plot_confusion_metrix(cm,classes,
                                  normalize=False,
                                  title='Confusion Matrix',
                                  cmap=plt.cm.Blues):
            """this function prints and plot the confusion matirx
            Normalization can be applied by setting 'normalize=True'
            """
            if normalize:
                cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                print("Normalized Confusion Matrix")
            else:
                print("Confusion Matrix, Without Normalisation")

            print(cm)

            plt.imshow(cm, interpolation='nearest',cmap=cmap)
            plt.title(title)
            plt.colorbar()
            tick_marks = np.arange(len(classes))
            plt.xticks(tick_marks,classes,rotation=35)
            plt.yticks(tick_marks,classes)

            fmt = '.2f' if normalize else 'd'
            thresh = cm.max() /2.

            for i , j in itertools.product(range(cm.shape[0]), range(cm.shape[0])):
                plt.text(j, i, format(cm[i,j], fmt),
                         horizontalalignment='center',
                         color='white' if cm[i, j] > thresh else 'black')

            plt.ylabel('True label')
            plt.xlabel('Predicted label')
            plt.tight_layout()
```

4.1.1 Logistic Regression

```
In [14]: from sklearn.linear_model import LogisticRegression

         classifier = LogisticRegression()

         # fitting training data to the model
         classifier.fit(X_train, Y_train)

Out[14]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)

In [15]: Y_pred = classifier.predict(X_test)

In [17]: from sklearn.metrics import confusion_matrix, accuracy_score, \
         classification_report

         conf_matrix = confusion_matrix(Y_test,Y_pred)
         plot_confusion_metrix(conf_matrix,classes=['Non-Default   :  0','Default  :1'])
         plt.show()

         print('Classification report')

         print(classification_report(Y_test,Y_pred))

         acc= accuracy_score(Y_test,Y_pred)
         print("Accuracy of the model:", acc)
```

From the sklearn library using the 'LogisticRegression' class, we created a logistic regression model and following results were interpreted:




```
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.83      0.80      0.81       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.91      0.90      0.91    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9995525135121464
```

Referring to the above confusion matrix, we can clearly see that the **Type I** error is **52** while the **Type II** error is **63**.

Since the data is unbalanced, we would not focus on the accuracy of the model but instead tune the model for less Type I and Type II errors.

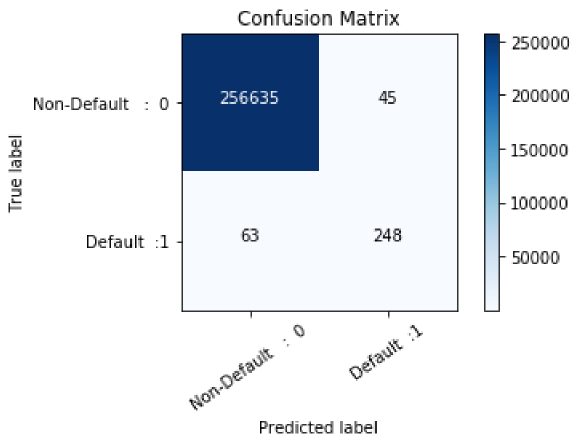
TUNING THE MODEL

Adjusting the threshold level of the probabilities to 0.60:

```
In [19]: y_pred_class=[]
         for value in y_pred_prob[:,1]:
             if value > 0.60:
                 y_pred_class.append(1)
             else:
                 y_pred_class.append(0)
```

```
In [20]: from sklearn.metrics import confusion_matrix, accuracy_score, \
         classification_report

         conf_matrix = confusion_matrix(Y_test,y_pred_class)
         plot_confusion_metrix(conf_matrix,classes=['Non-Default    :  0','Default   :1'])
         plt.show()

         print('Classification report')

         print(classification_report(Y_test,y_pred_class))

         acc= accuracy_score(Y_test,y_pred_class)
         print("Accuracy of the model:", acc)
```

After tuning the model, we get the following results:




```
Classification report
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    256680
          1       0.85      0.80      0.82       311

  micro avg       1.00      1.00      1.00    256991
  macro avg       0.92      0.90      0.91    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9995797518201026
```

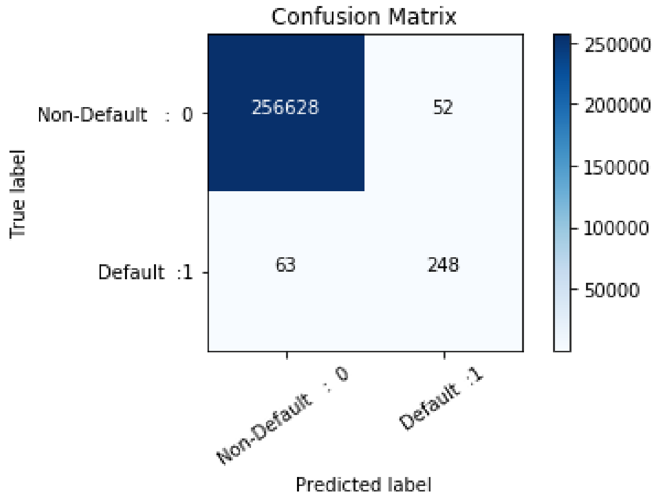
Now the **Type I** error has decreased to **45** after tuning while the Type II error is still the same.

USING CROSS VALIDATION:

```
In [22]: #Using cross validation

         classifier=(LogisticRegression())

         #performing kfold_cross_validation
         from sklearn.model_selection import KFold
         kfold_cv=KFold(n_splits=10)
         print(kfold_cv)

         from sklearn.model_selection import cross_val_score
         #running the model using scoring metric as accuracy
         kfold_cv_result=cross_val_score(estimator=classifier,X=X_train,
         y=Y_train, cv=kfold_cv)
         print(kfold_cv_result)
         #finding the mean
         print(kfold_cv_result.mean())

         KFold(n_splits=10, random_state=None, shuffle=False)
         [0.98636015 0.99410665 0.99727871 0.9979632  0.9966443  0.99659421
          0.99722862 0.99701159 0.99734544 0.99791308]
         0.9958445945749894
```

```
In [23]: from sklearn.metrics import confusion_matrix, accuracy_score, \
         classification_report

         conf_matrix = confusion_matrix(Y_test,Y_pred)
         plot_confusion_metrix(conf_matrix,classes=['Non-Default   :  0','Default  :1'])
         plt.show()

         print('Classification report')

         print(classification_report(Y_test,Y_pred))

         acc= accuracy_score(Y_test,Y_pred)
         print("Accuracy of the model:", acc)
```

By using the k-fold cross validation, we get the following Confusion Matrix:



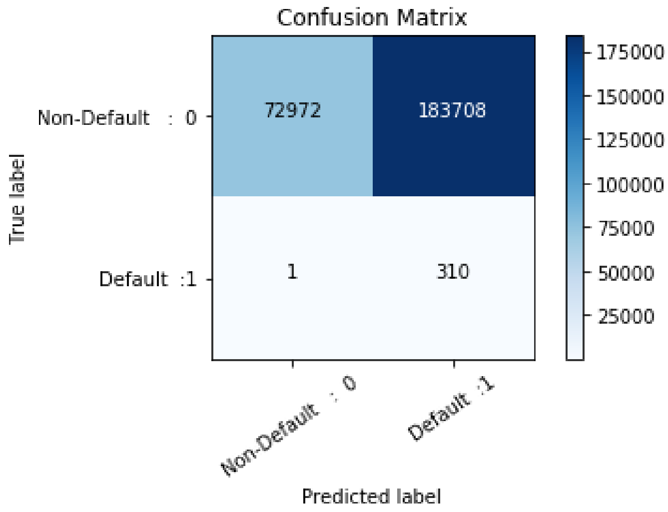

```
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.83      0.80      0.81       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.91      0.90      0.91    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9995525135121464
```

After implementing cross validation, we get the same Type I error as compared to the Logistic regression model without tuning which is 52.

4.1.2 Decision Tree Classification

Training the model on the train set and then predicting on the test set using ‘**Entropy**’ for splitter selection and using the ‘DecisionTreeClassifier’ class.

In [29]:
```
###
#Running Decision Tree Model
from sklearn.tree import DecisionTreeClassifier

model_DecisionTree = DecisionTreeClassifier(criterion = 'entropy',max_features=8,random_state=0,)
model_DecisionTree.fit(x_train_scale,y_train)

#fit the model on the data and predict the values

y_pred = model_DecisionTree.predict(x_test_scale)
```




```
Classification report
              precision    recall  f1-score   support

         0.0       1.00      0.28      0.44    256680
         1.0       0.00      1.00      0.00       311

   micro avg       0.29      0.29      0.29    256991
   macro avg       0.50      0.64      0.22    256991
weighted avg       1.00      0.29      0.44    256991

Accuracy of the model: 0.2851539548077559
```

In this model, the Type II error is low but the Type I error is extremely high which is not acceptable.

4.1.3 Artificial Neural Networks (ANN)

- **Deep learning on Unbalanced Dataset**

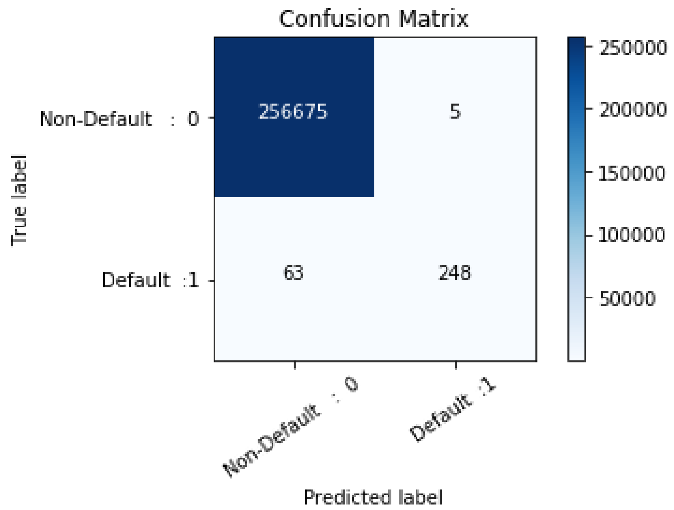
After importing all the keras libraries and packages for deep learning, we create the following layers:

```
In [37]: # Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 19, kernel_initializer = 'uniform',
                     activation = 'relu', input_dim = 35))


# Adding the second hidden layer
classifier.add(Dense(units=19, kernel_initializer='uniform',
                     activation='relu'))

# dropout for second layer
# classifier.add(Dropout(p = 0.1))

# Adding the third hidden layer
classifier.add(Dense(units=19, kernel_initializer='uniform',
                     activation='relu'))

'''# Adding the fourth hidden layer
classifier.add(Dense(units=19, kernel_initializer='uniform',
                     activation='relu'))'''


# Adding the output layer
classifier.add(Dense(units=1, kernel_initializer='uniform',
                     activation='sigmoid'))
```

Then compiling and fitting the ANN to the training set with batch size of 100 and 10 epochs.

```
In [38]: # Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                   metrics = ['accuracy'])

In [39]: # Fitting the ANN to the Training set

classifier.fit(x_train, y_train, batch_size = 100, epochs = 10)

WARNING:tensorflow:From C:\Users\LENOVO\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/10
598978/598978 [==============================] - 9s 15us/step - loss: 0.0377 - acc: 0.9923
Epoch 2/10
598978/598978 [==============================] - 9s 15us/step - loss: 0.0250 - acc: 0.9960: 1s - loss:
Epoch 3/10
598978/598978 [==============================] - 8s 13us/step - loss: 0.0214 - acc: 0.9966
Epoch 4/10
598978/598978 [==============================] - 9s 15us/step - loss: 0.0202 - acc: 0.9967
Epoch 5/10
598978/598978 [==============================] - 9s 15us/step - loss: 0.0187 - acc: 0.9970
Epoch 6/10
598978/598978 [==============================] - 9s 15us/step - loss: 0.0188 - acc: 0.9969
Epoch 7/10
598978/598978 [==============================] - 8s 13us/step - loss: 0.0195 - acc: 0.9967
Epoch 8/10
598978/598978 [==============================] - 8s 14us/step - loss: 0.0187 - acc: 0.9970
Epoch 9/10
598978/598978 [==============================] - 8s 14us/step - loss: 0.0185 - acc: 0.9971
Epoch 10/10
598978/598978 [==============================] - 8s 14us/step - loss: 0.0191 - acc: 0.9967

Out[39]: <keras.callbacks.History at 0x25604090a20>
```

After predicting the test results, we the following Confusion Matrix:




```
Classification report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.98      0.80      0.88       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.99      0.90      0.94    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9997353992941387
```

The Type I error is 5 which is very less as compared to the previous models while the Type II error is still the same.

- **Deep learning on Balanced Dataset**

The dataset is unbalanced with almost 95% of non-defaulters and 5% of defaulters.

To overcome this, we use SMOTE function to make the data balanced from the imblearn.over_sampling library and importing the SMOTE class.

```
In [45]: from imblearn.over_sampling import SMOTE
```

```
In [49]: x_resample, y_resample = SMOTE().fit_sample(x_train_scale,y_train)
```

```
In [47]: x_resample
```

```
In [50]: x_cols=[['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
                 'installment', 'grade', 'emp_length', 'home_ownership', 'annual_inc',
                 'verification_status', 'purpose', 'dti', 'delinq_2yrs',
                 'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
                 'total_acc', 'initial_list_status', 'out_prncp', 'out_prncp_inv',
                 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int',
                 'total_rec_late_fee', 'recoveries', 'collection_recovery_fee',
                 'last_pymnt_amnt', 'collections_12_mths_ex_med', 'tot_coll_amt',
                 'tot_cur_bal', 'total_rev_hi_lim']]

         y_cols = [['default_ind']]
```

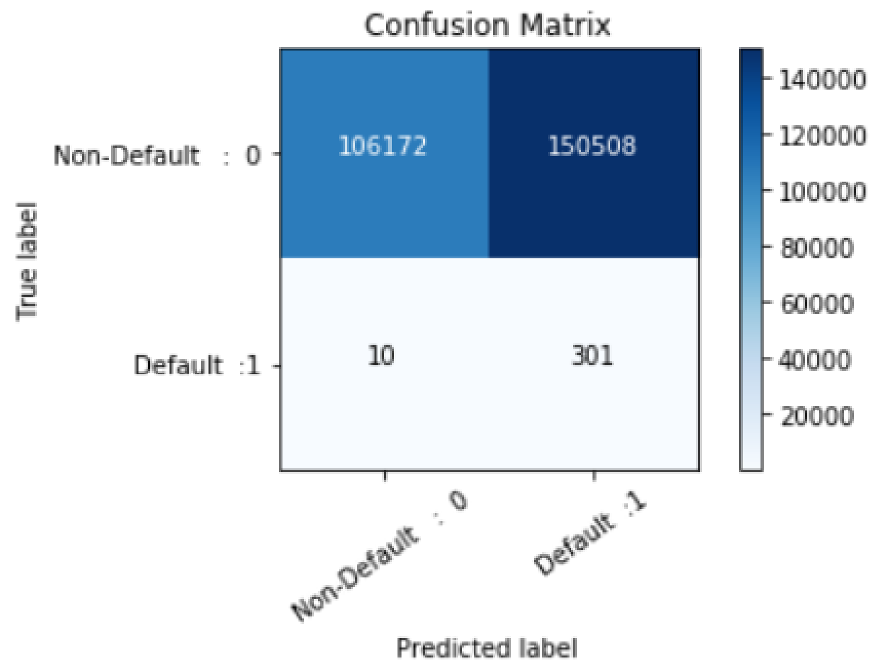
```
In [51]: x = pd.DataFrame(x_resample, columns=x_cols)
         y = pd.DataFrame(y_resample, columns=y_cols)
```

```
In [52]: data = pd.concat([x, y], axis = 1)
```

Then further printing the shape of the dataset, we can clearly see that the number of observations has increased i.e. oversampling.

```
In [54]: data.shape

Out[54]: (1105644, 36)
```

Therefore, after using SMOTE and then followed by layers creation, compiling and fitting the ANN to the training set, we obtain the following Confusion Matrix:

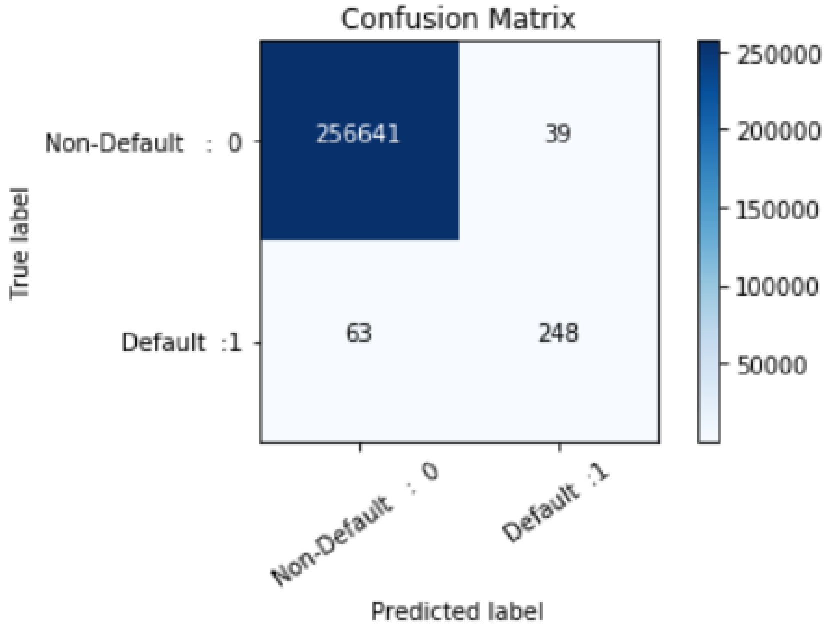



```
Classification report
              precision    recall  f1-score   support

         0.0       1.00      0.41      0.59    256680
         1.0       0.00      0.97      0.00       311

   micro avg       0.41      0.41      0.41    256991
   macro avg       0.50      0.69      0.29    256991
weighted avg       1.00      0.41      0.58    256991

Accuracy of the model: 0.4143063375760241
```

After balancing the dataset, the Type I error has increased drastically to 1,50,508 while the Type II error has decreased to 10.

Therefore, after tuning and oversampling the models and still not gaining the required result, we then applied feature engineering as explained in 2.3 and hence, building the models on the new dataset obtained after selecting new variables and rerunning the same models on the new dataset.

4.1.4 Logistic Regression on the new dataset



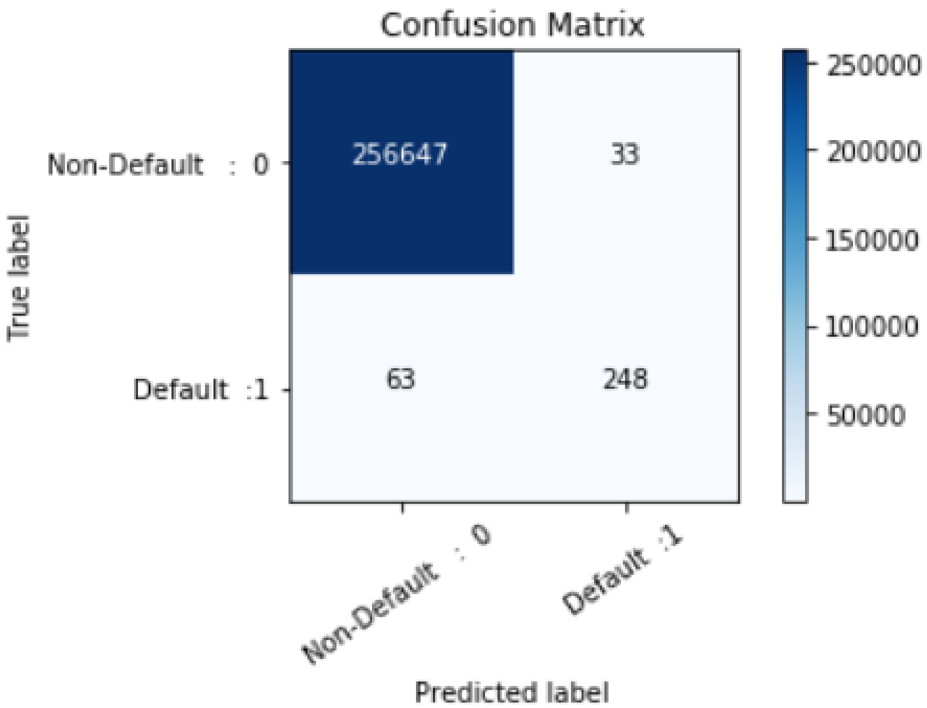
Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	256680
1	0.86	0.80	0.83	311
micro avg	1.00	1.00	1.00	256991
macro avg	0.93	0.90	0.91	256991
weighted avg	1.00	1.00	1.00	256991

Accuracy of the model: 0.999603098941208

Here the Type I error has reduced from 52 to 39 as compared to the logistic regression model on the previous dataset.

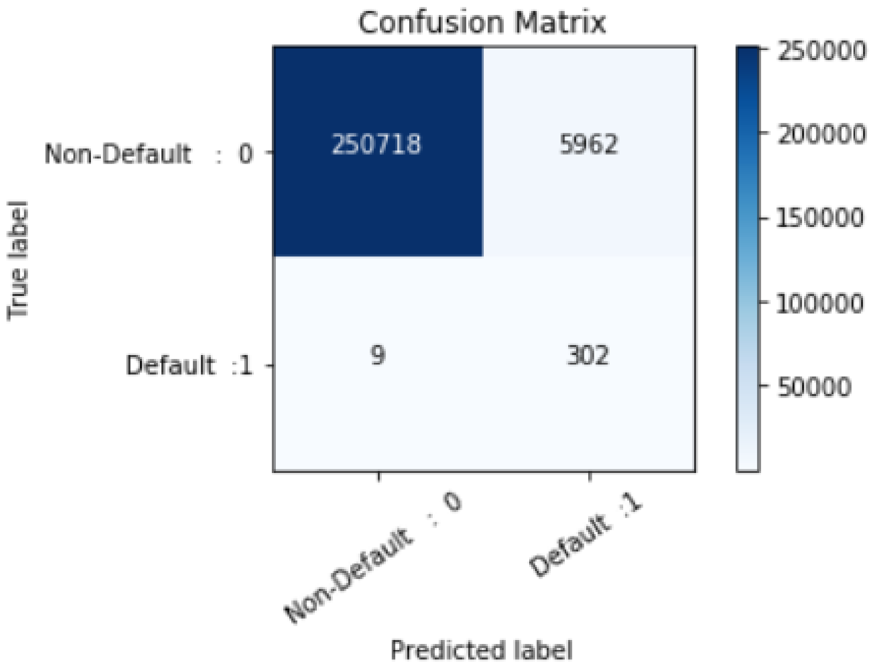
After tuning the model:




```
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.88      0.80      0.84       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.94      0.90      0.92    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9996264460623134
```

The Type I error has decreased from 45 to 33 as compared to the previously tuned model.

4.1.5 Decision Tree Classification on the new dataset




```
Classification report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    256680
           1       0.05      0.97      0.09       311

   micro avg       0.98      0.98      0.98    256991
   macro avg       0.52      0.97      0.54    256991
weighted avg       1.00      0.98      0.99    256991

Accuracy of the model: 0.9767657233132678
```

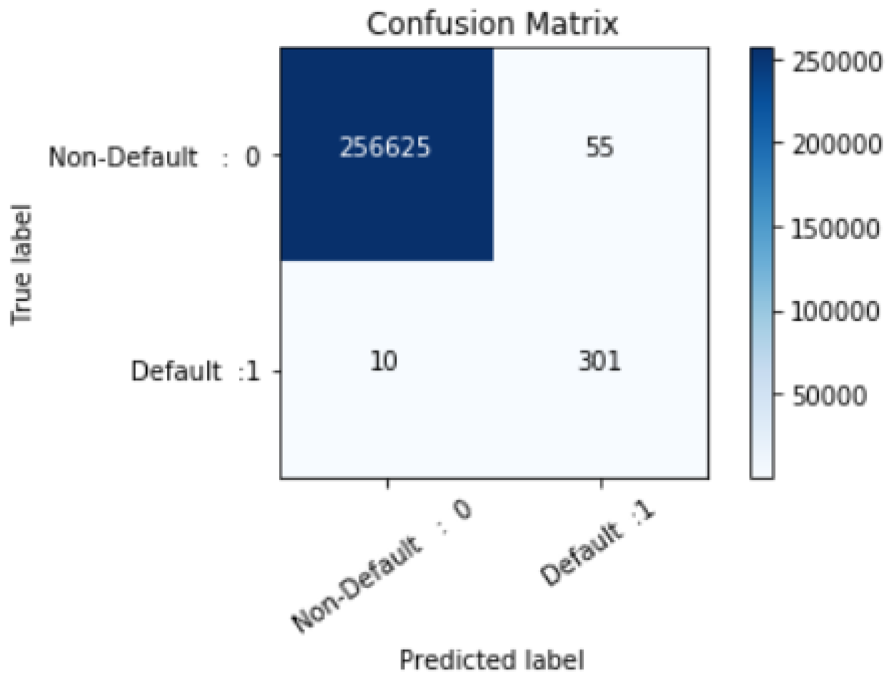
Observing the previous Decision Tree model and the current one, the Type I error has drastically reduced from 183708 to 5962, while the Type II error has increased from 1 to 9.

4.1.6 Gradient Boosting Classifier

Using the 'sklearn.ensemble' library and importing 'GradientBoostingClassifier', we build a model as shown:

```
In [90]: #predicting using the
         from sklearn.ensemble import GradientBoostingClassifier

         model_GradientBoosting=GradientBoostingClassifier()
         #model_GradientBoosting=DecisionTreeClassifier()

         #fit the model on the data and predict the values
         model_GradientBoosting.fit(X_train,Y_train)

         Y_pred=model_GradientBoosting.predict(X_test)
```

```
In [91]: #checking result
         from sklearn.metrics import confusion_matrix, accuracy_score, \
         classification_report

         conf_matrix = confusion_matrix(Y_test,Y_pred)
         plot_confusion_metrix(conf_matrix,classes=['Non-Default   :  0','Default  :1'])
         plt.show()

         print('Classification report')

         print(classification_report(Y_test,Y_pred))

         acc= accuracy_score(Y_test,Y_pred)
         print("Accuracy of the model:", acc)
```



Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	256680
1	0.85	0.97	0.90	311
micro avg	1.00	1.00	1.00	256991
macro avg	0.92	0.98	0.95	256991
weighted avg	1.00	1.00	1.00	256991

Accuracy of the model: 0.9997470728546914

Type I error - 55

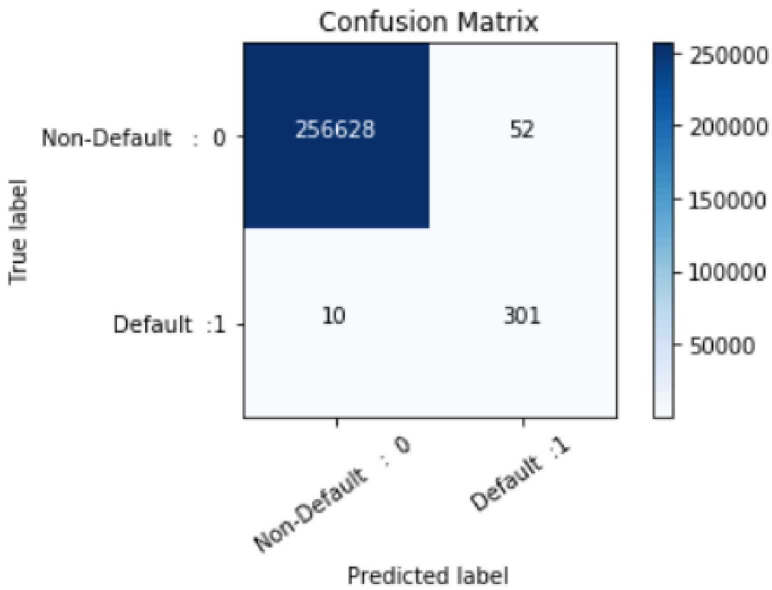
Type II error – 10

Comparing the gradient boosting model with the other models, we observe that this is the most accurate model with minimum errors.

Now trying to tune the model with different n_estimators such as 80, 120, 200 and so on, we found our best model on the value of 130.

After Tuning the Gradient Boosting model with different parameters, and obtaining the best parameter as n_estimators=130, we obtain the following confusion matrix.

```
model_GradientBoosting=GradientBoostingClassifier(n_estimators=130,)
```

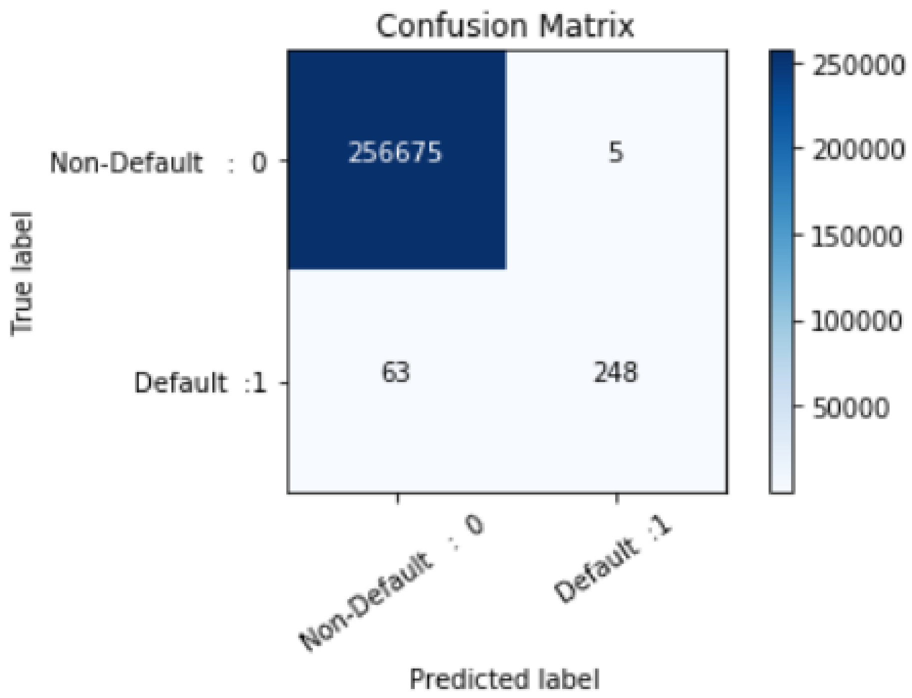



```
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.85      0.97      0.91       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.93      0.98      0.95    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9997587464152441
```

The Type I error has reduced from 55 to 52 while the Type II error is same as the previous Gradient boosting model. Hence, Comparing with all the above models, Gradient Boosting is the best model.

4.1.7 ANN on the new dataset

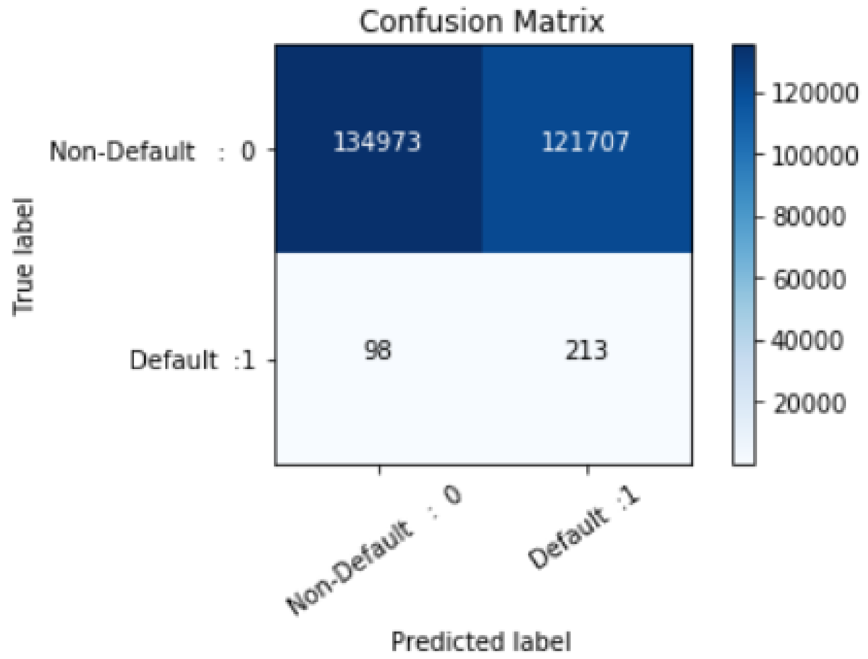
- **On Unbalanced data**




```
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.98      0.80      0.88       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.99      0.90      0.94    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9997353992941387
```

The errors are same as compared to the previous unbalanced ANN model.

- **On Balanced data**



Classification report

	precision	recall	f1-score	support
0.0	1.00	0.53	0.69	256680
1.0	0.00	0.68	0.00	311
micro avg	0.53	0.53	0.53	256991
macro avg	0.50	0.61	0.35	256991
weighted avg	1.00	0.53	0.69	256991

Accuracy of the model: 0.5260339856259558

The Type I error has decreased from 150508 to 121707 while the Type II error has increased from 10 to 98.

CHAPTER 5: FINAL MODEL

Now that we know that Gradient Boosting with tuning is our best model, we will now perform prediction on the whole dataset which consists of around 8.55 lacs observations and then concatenated the predicted variable to the dataset for final submission to the client for comparing the actual and predicted values.

In [82]:
```
y.head()
```

Out[82]:
```
0    0
1    1
2    0
3    0
4    0
Name: default_ind, dtype: int64
```

In [50]:
```
#predicting using the
from sklearn.ensemble import GradientBoostingClassifier

model_GradientBoosting=GradientBoostingClassifier(n_estimators=130,)

#fit the model on the data and predict the values
model_GradientBoosting.fit(X_train,Y_train)
```

Out[50]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

Prediction on Full data set

In [84]:
```
Y_full_pred=model_GradientBoosting.predict(x)
```

In [92]:
```
final_df.head(10)
```

Out[92]:

mnt	next_pymnt_d	last_credit_pull_d	collections_12_mths_ex_med	application_type	annual_inc_joint	acc_now_delinq	total_rev_hi_lim	default_ind	Predicted_class
1.62	0	41	0.0	0	0.0	0.0	32163.574526	0	0
9.66	0	99	0.0	0	0.0	0.0	32163.574526	1	1
9.91	0	41	0.0	0	0.0	0.0	32163.574526	0	0
7.48	0	40	0.0	0	0.0	0.0	32163.574526	0	0
7.79	0	41	0.0	0	0.0	0.0	32163.574526	0	0
1.03	0	101	0.0	0	0.0	0.0	32163.574526	0	0
0.08	0	41	0.0	0	0.0	0.0	32163.574526	0	0
1.34	0	23	0.0	0	0.0	0.0	32163.574526	0	0
2.39	0	12	0.0	0	0.0	0.0	32163.574526	1	1
1.45	0	65	0.0	0	0.0	0.0	32163.574526	1	1

CHAPTER 6: CONCLUSION

Out of all the algorithms used, Gradient Boosting Classifier gave us the least Type II error as 10 after tuning the model. Since our priority for accuracy was based on Type II error, Gradient Boosting model with tuning was preferred. But after examining all the models, ANN is the best model because despite of not tuning, it gave minimum error.

If in case of some cases where Type I error is more significant, ANN is the best suited model.