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:

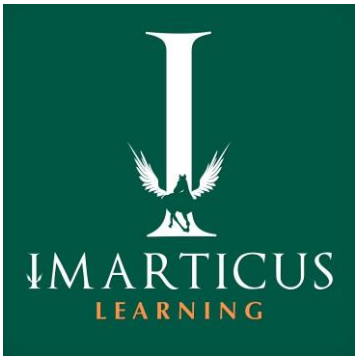
Utkarsha Desale

Manish Madhavi

Shalaka Patil

Jyothirmai Palle

Course and Batch: PGA-06 Feb. 2021



Abstract

Investors (lenders) provide loans to borrowers in exchange with an agreement of repayment with interest. That means the lender only makes profit (interest) if the borrower pays off the loan. However, if he/she doesn't repay the loan, then the lender loses money.

XYZ Corp. is looking towrads building a model to predict default in the future, which will help the them in deciding whether sanction a loan or not.

Acknowledgement

We are using this opportunity to express our gratitude to everyone who supported us throughout the course. We are thankful for their aspiring guidance, friendly advices 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.

Furthermore we are fortunate to have great trainers who readily shared their immense knowledge in Data Analytics and guided us in a manner that the outcome resulted in enhancing our data skills.
Also, extremely thankful to Prof. Nikita Tandel who seamlessly trained us in Python and guided us to give our best performance.

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

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

Date: 25 Feb., 2021

Place: Mumbai

Utkarsha Desale
Manish Madhavi
Shalaka Patil
Jyothirmai Palle

Certificate of Completion

I hereby certify that the Project titled "**XYZ Corporation Lending Data**" was undertaken and completed under the supervision of Prof. Nikita Tandel. By Utkarsha Desale, Manish Madhavi, Shalaka Patil and Jyothirmai Palle from the batch of PGA-6 (Feb 2021)

Date: Feb 25, 2021

Place – Mumbai

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Title & Objective of the Study

'XYZ Corporation Lending Data' is the project we are working upon which falls under the BFSI domain (Banking Financial services and Insurance sector).
The text file contains complete loan data for all loans issued by XYZ Corp. through 2007-2015. It is basically like Stepping in the shoe of the loan issuer and manage credit risk by using the past data and deciding whom to give loan in the future.
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. 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 have predicted the 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, 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 model 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, Random Forest Classifier, Gradient Boosting Classifier, Artificial Neural Networks.

CHAPTER 2: DATA PREPARATION AND UNDERSTANDING

Firstly imported the required Libraries and then the following sequence of steps were followed to perform data preprocessing.

2.1 Data Extraction and Cleaning:

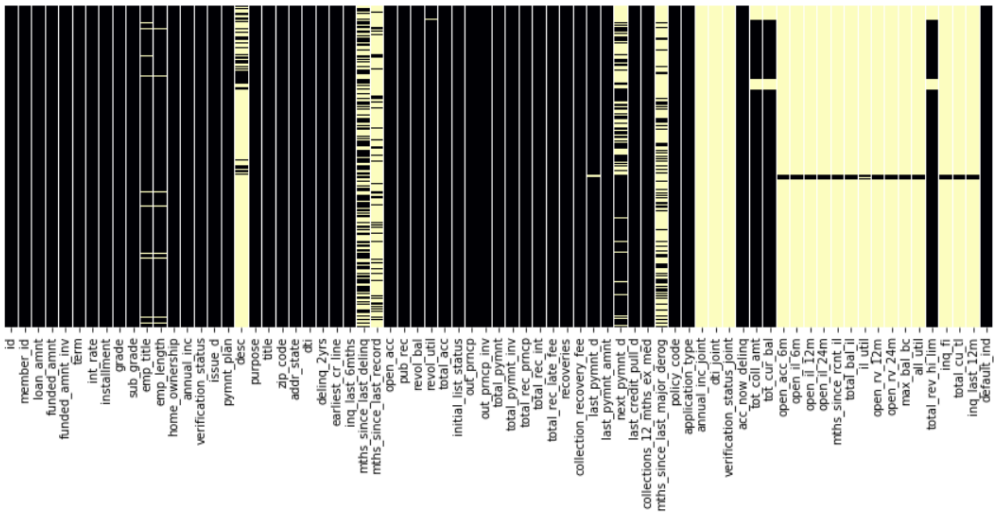
2.1.1 Missing Value Analysis

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.

Visualising the missing values in each column

```
In [11]: plt.figure(figsize=(15,5))
         sns.heatmap(data1.isnull(), cbar = False, yticklabels=False, cmap="magma" )

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0xbfd5518>
```



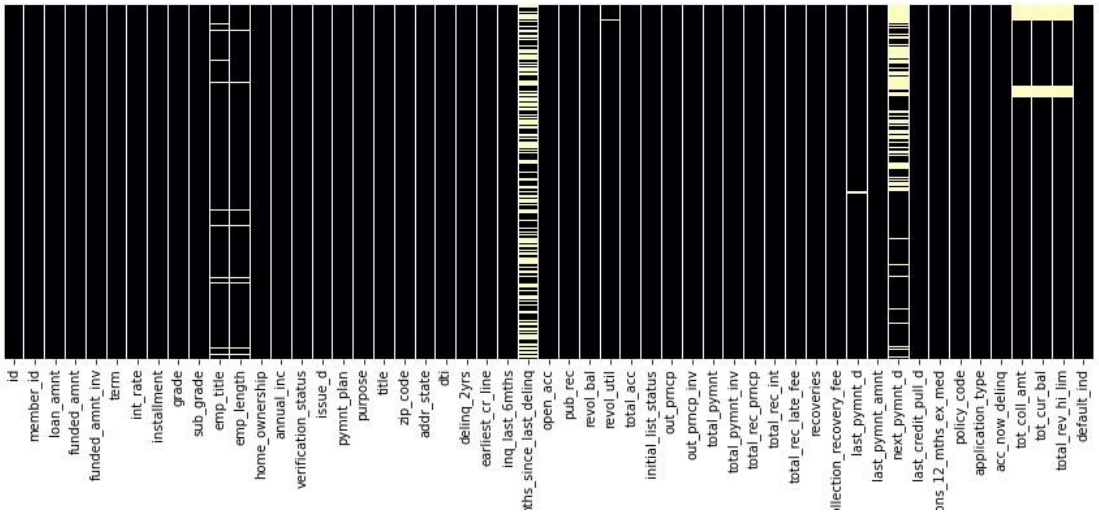
From the above heatmap it is clear that our dataset contains alot of missing values and we cannot use use such variables.

Above heatmap shows the intensity of values that are missing in each columns. All the light colored columns represent the amount of missing values present in that specific column. Firstly, setting a threshold of 75%, i.e. dropping the columns which have more than or equal to 75% missing values. We are then left with 53 variables.

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

```
In [12]:
# Visualizing the missing value in each column in after dropping the variables
plt.figure(figsize=(15,5))
sns.heatmap(data1.isnull(), cbar = False, yticklabels=False, cmap="magma" )
```

```
Out[12]: <AxesSubplot:>
```



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 and member_id** ==> id and member_id are all unique which is a bit misleading. So every record is a unique customer
- **application_type** ==> application_type is 'INDIVIDUAL' having 855527 records which is 99.94% of the records
- **acc_now_delinq** ==> acc_now_delinq is 0 having 852039 records which is 99.5% of the records
- **emp_title** ==> emp_title not needed here
- **zip_code** ==> zip_code not needed for this level of analysis
- **title** ==> title variable because it's a categorical varibale with (60991 level)
- **earliest_cr_line** ==> earliest_cr_line variable because it's a date varibale with (697 level)
- **next_pymnt_d** ==> as per the domain knowledge it is also not imp variable
- other few variables we deleted according to domain knowledge

```
In [21]:
data1 = data1.drop(['id' ,'member_id', 'sub_grade' ,
                    'emp_title' , 'title' , 'zip_code' ,
                    'addr_state' ,
                    'earliest_cr_line',
                   'inq_last_6mths' , 'pub_rec' ,
                    'initial_list_status' ,
                    'recoveries' ,'next_pymnt_d',
                    'collection_recovery_fee' ,
                    'acc_now_delinq',
                   'policy_code' , 'tot_coll_amt' ,
                    'collections_12_mths_ex_med']
                   ,axis = 1)
```

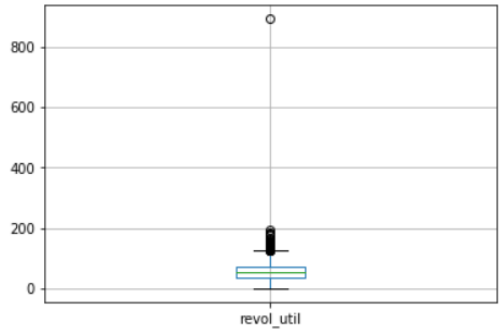
2.1.2 Variable Transformation:

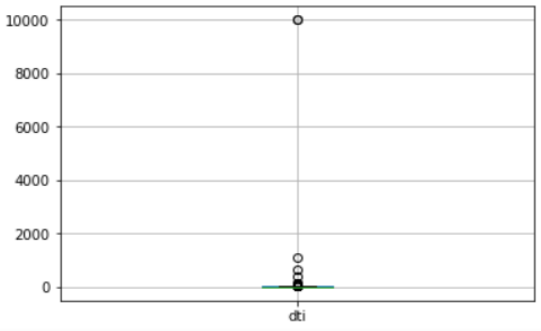
Here we have extracted numbers from emp_length. If emp_length==10+ years then we leave it as 10 and similarly for other observations too.

```
def emplen(row):
    return str(row.replace('10+ years' , '10').replace('9 years' , '9').replace('8 years' , '8').replace('7 years' , '7')
          .replace('6 years' , '6').replace('5 years' , '5').replace('4 years' , '4').replace('3 years' , '3')
          .replace('2 years' , '2').replace('1 year' , '1').replace('< 1 year' , '< 1'))

data1['emp_length'] = data1['emp_length'].astype(str).apply(emplen)
data1['emp_length'] = data1['emp_length'].replace('< 1' , '0' , regex = True)

data1['emp_length'] = data1['emp_length'].replace('nan' , '10' , regex = True) #replacing with mode value.
```

2.1.3 Detecting the Outliers:

Here we generated the Boxplot for all the numeric variables where we found that two variables revol_util , dti have single observations that are too far away from the remaining data.





revol_util has only one variable above 193 and that is 892 and dti has only one variable above 1093 which is 9999. So, we removed those 2 observations.

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

2.1.4 Imputing Missing values:

Missing values treatment with Mean:

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

- tot_cur_bal
- total_rev_hi_lim
- revol_util
- mths_since_last_delinq

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

- emp_length

And the missing values of following variables are treated by ffill() function:

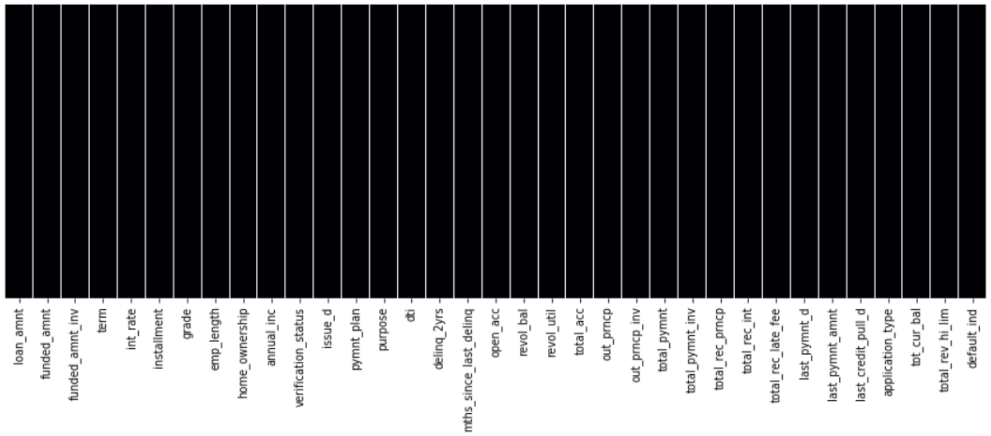
- last_pymnt_d
- last_credit_pull_d

(Since these are dates we will fill the null values with previous values using ffill() function
Pandas dataframe.ffill() function is used to fill the missing value in the dataframe. 'ffill' stands for 'forward fill' and will propagate last valid observation forward)

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

```
# Visualizing the missing value in each column after imputing missing values
plt.figure(figsize=(15,5))
sns.heatmap(data1.isnull(), cbar = False, yticklabels=False, cmap="magma" )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x5ea43208>
```

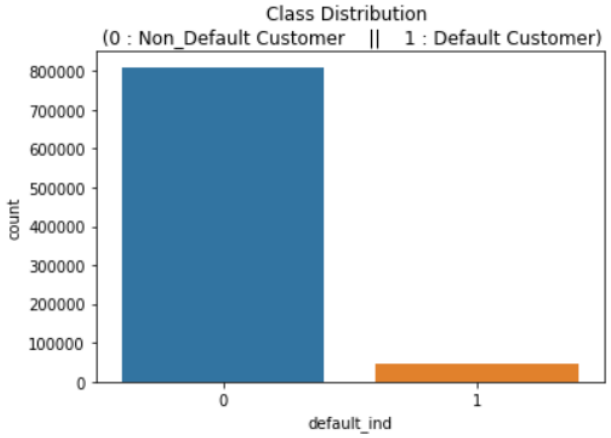


2.2 Data Visualization:

2.2.1 Data Visualization for Categorical Variables:

The response variable in this data is '**default_ind**' which indicates that the customer will Default ('1') or Non-Default ('0')

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

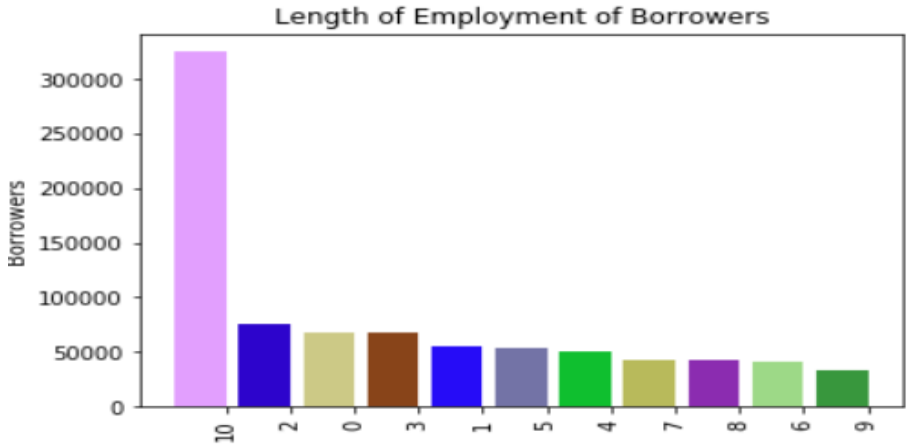


Non-Default Customer: 94.57 % of the dataset.

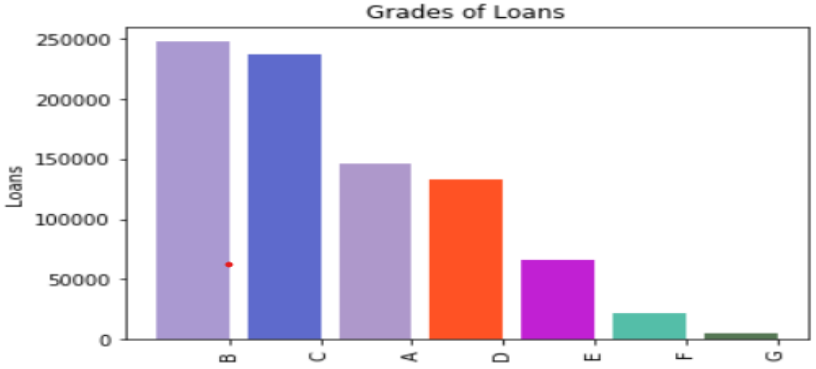
Default Customer: 5.43 % of the dataset

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

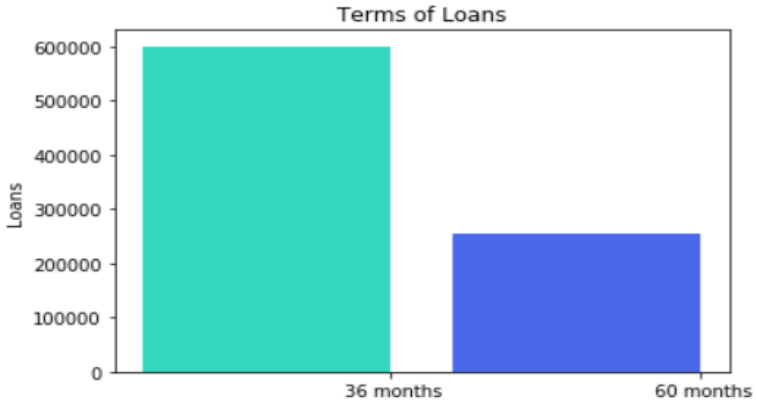
Plot showing the distribution of 'length of employment of borrowers' variable:



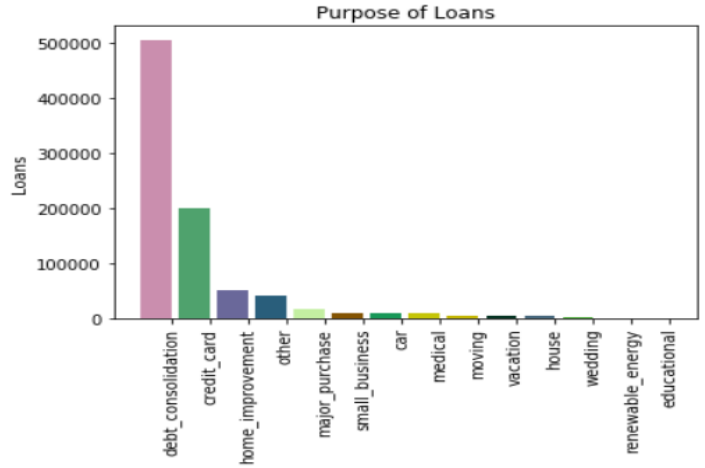
Plot showing the distribution of 'Grades of loans' variable:



Plot showing the distribution of 'Term of loans' variable:



Plot showing the distribution of 'Purpose of loans' variable:

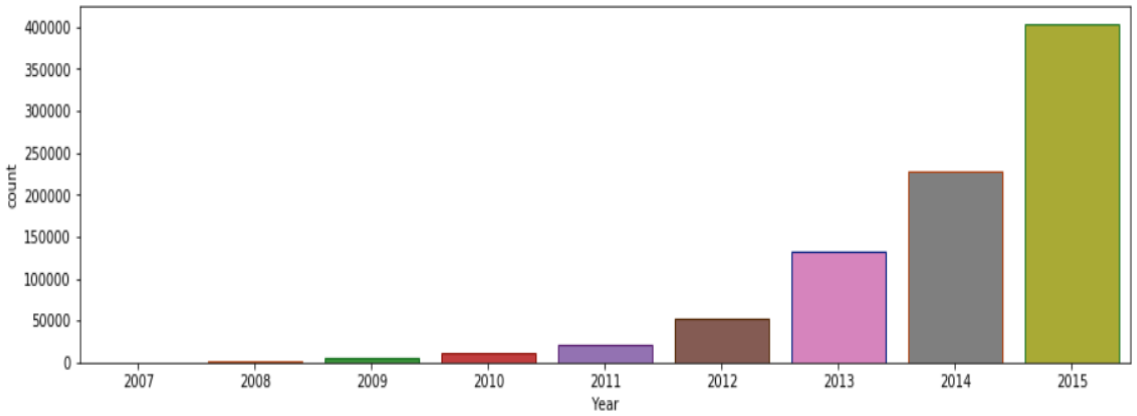


Plot showing Issue date of the loan amount

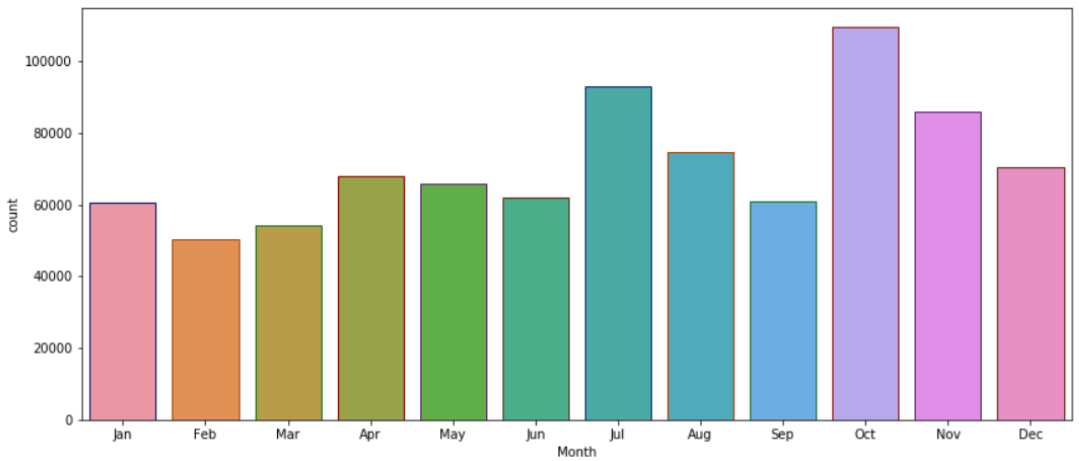
A function is created that will split the '**issue_d**' variable which is nothing but the year & month in which the loan was funded.

```
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)
```

Plot showing the distribution of 'Exponential rise in the num. of applications over a period of year' variable:

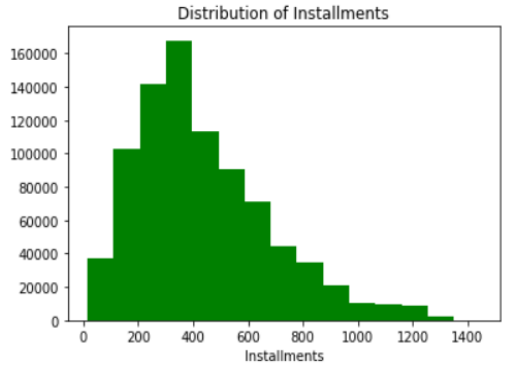


Plot showing the distribution of 'Exponential rise in the num. of applications over a period of month' variable:

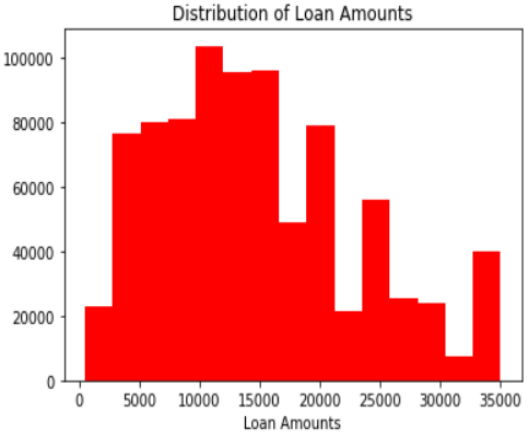


2.2.2 Data Visualization for Numeric Variables:

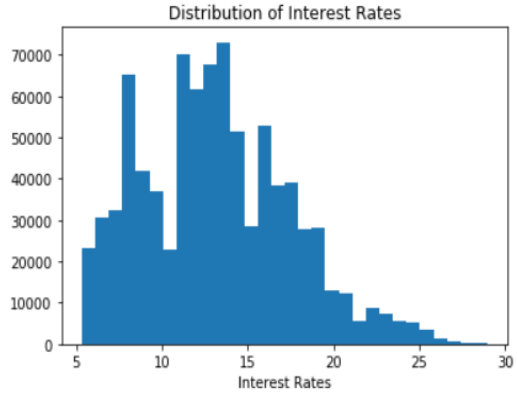
Plot showing the distribution of 'Installments' of variable:



Plot showing the distribution of 'Loan Amounts' variable:



Plot showing the distribution of 'Interest Rates' variable:



2.3 Feature Engineering and Variable Selection:

Keeping the loan_amount, and creating a metric which indicates that the total amount committed by investors for that loan at that point in time (funded_amnt_inv) is less than what the borrower requested.

Then we computed the ratio of the number of open credit lines in the borrower's credit file divided by the total number of credit lines currently in the borrower's credit file.

```
data1['amt_difference'] = 'eq'
data1.loc[ ( data1['funded_amnt'] - data1['funded_amnt_inv']) > 0, 'amt_difference' ] = 'less'

#--------------------

# Create new metric
data1['acc_ratio'] = data1.open_acc / data1.total_acc
```

As we converted funded_amnt and funded_amnt_inv into a single variable **amt_difference** similarly we converted open_acc and total_acc into a single variable **acc_ratio**. Then we removed those 4 variables.

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.

```
colname=[]
for x in data1.columns[:]:
    if data1[x].dtype=='object':
        colname.append(x)   #append(x) for X increament
colname
```

```
['amt_difference',
 'term',
 'grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'pymnt_plan',
 'purpose',
 'last_pymnt_d',
 'last_credit_pull_d',
 'application_type']
```

The above code represents the variables of datatype object. We converted these variables into numbers using following code.

```
from sklearn import preprocessing

le=preprocessing.LabelEncoder()

for x in colname:
    data1[x]=le.fit_transform(data1[x])
```

2.5 Splitting the Dataset into train and test data:

The model training dataset should be used to train and build models and the final performance should be on Test data

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 in the Testing data.**

```
to_date=pd.to_datetime('06/01/2015')
train_data=data1.loc[(data1.issue_d < to_date)]
train_data.shape
```

```
(598978, 33)
```

```
test_data=data1.loc[(data1.issue_d >= to_date)]
test_data.shape
```

```
(256991, 33)
```

Creating X_train, X_test, Y_train, Y_test:

```
X_train= train_data.values[:,:-1]
Y_train= train_data.values[:,-1]
```

```
X_test= test_data.values[:,:-1]
Y_test= test_data.values[:,-1]
```

2.6 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.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform (X_test)
```

We scale the data to avoid bias that can arise from large discrepancy in numerical variables. While Normalization forces to return the values from 0 to 1. Standardization forces to return the values from -3 to 3.

Chapter 3: 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()
```

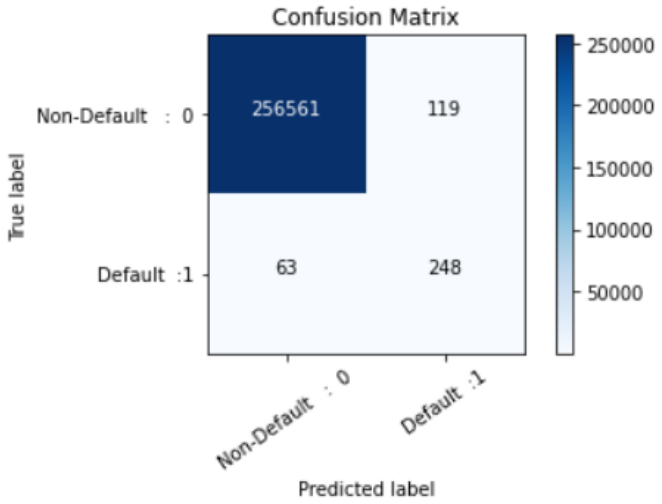
3.1.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
#create a model
classifier = LogisticRegression()
#fitting training data to the model
classifier.fit(X_train,Y_train)

Y_pred=classifier.predict(X_test)
```

```
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.linear_model library using the 'LogisticRegression' , we created a logistic regression model and following results were interpreted

```
Confusion Matrix, Without Normalisation
[[256561    119]
 [    63    248]]
```




```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.68      0.80      0.73       311

    accuracy                           1.00    256991
   macro avg       0.84      0.90      0.87    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model:  0.999291803993136
```

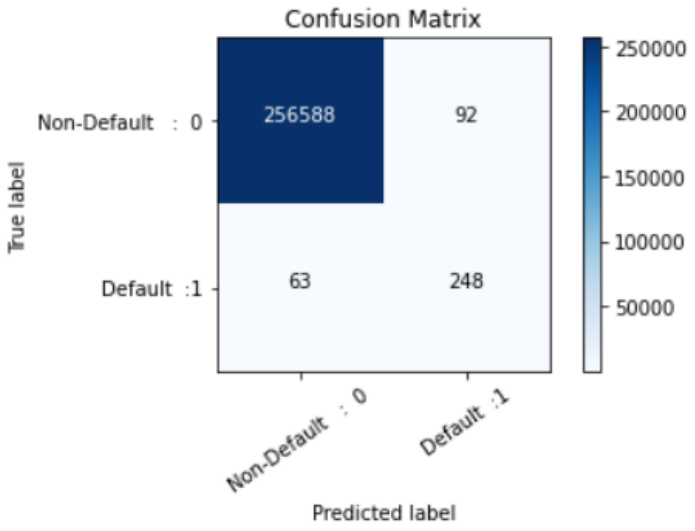
From the confusion matrix we can conclude that **Type II error value is 63**, **Type I error is 119**, **F1 score** for class 1 is **0.73** and the **accuracy of the model is 0.99**

3.1.2 Tunning the Logistic Regression model

Adjusting the threshold level of the probabilities to 0.56

```
# we select the 0.56 as a threshold because total error is also less and type II error is also less
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)
#print(y_pred_class)
```

```
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)
```




```
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.73      0.80      0.76       311

    accuracy                           1.00    256991
   macro avg       0.86      0.90      0.88    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model : 0.9993968660381103
```

After running the threshold check, we came to the conclusion that there is no change in **Type II error and for 0.56** threshold value the total error is less and the **Type I error is reduced to 92**.

After adjusting the threshold value we observed the **F1 score** for class 1 which previously was **0.73 and now is increased to 0.76**.

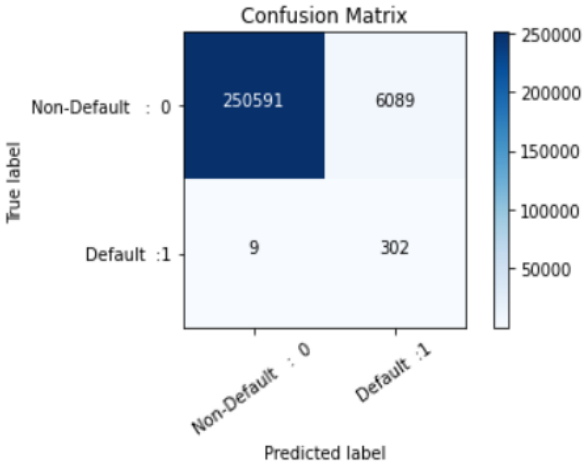
3.2.1 DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier

model_DecisionTree=DecisionTreeClassifier(random_state=10)

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


Y_pred = model_DecisionTree.predict(X_test)
print(Y_pred)
```

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

```
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.tree library using the 'DecisionTreeClassifier' , we created a DecisionTree model and following results were interpreted




```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99    256680
         1.0       0.05      0.97      0.09       311

    accuracy                           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:  0.9762715425832033
```

From the confusion matrix we can conclude that **Type II error is 9 & Type I error is 6098**, **F1 score** for class 1 is very low which is **0.09** and the **accuracy of the model is 0.9762**

3.2.2 Tunning the Decision Tree model:

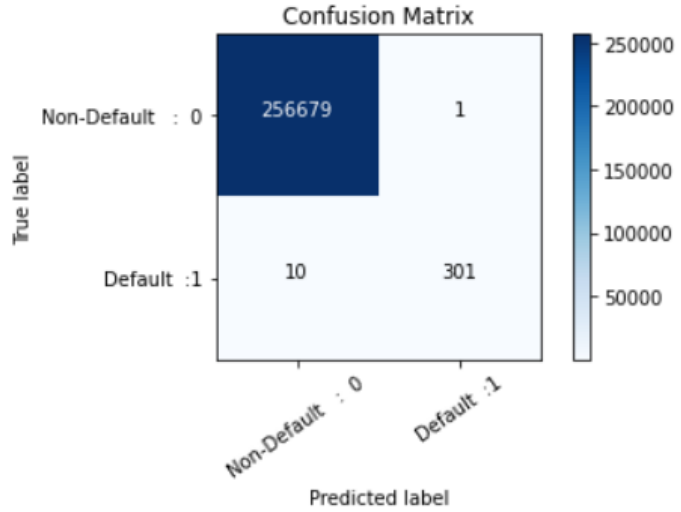
We Perform the Grid Search CV on Decision Tree model.

```
1 grid_search_cv_dt.best_estimator_
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=40,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=10, splitter='best')
```

After performing the GridSearchCV on Decision Tree Model , we found the best estimators mentioned above.

```
Confusion Matrix, Without Normalisation
[[256679      1]
 [    10    301]]
```




```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       1.00      0.97      0.98       311

    accuracy                           1.00    256991
   macro avg       1.00      0.98      0.99    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model:  0.9999571969446401
```

After running the Grid Search CV on Decision Tree model, we came to the conclusion that type II error is increased by 1 but there is drastic decrease in type I error which is 1.

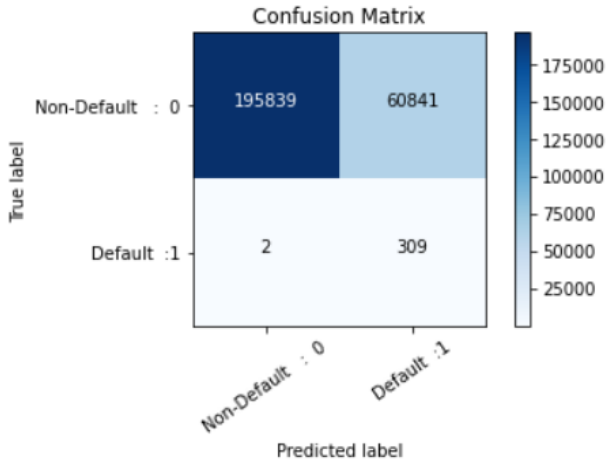
F1 score for class 1 is 0.98 which previously was very low and also there is a significant increase in accuracy of the model which is 0.99.

3.3.1 RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

model_Random = RandomForestClassifier(random_state= 10)

model_Random = model_Random.fit(X_train , Y_train)

Y_pred = model_Random.predict(X_test)
```

```

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.ensemble library using the 'RandomForestClassifier' , we created a Random Forest model and following results were interpreted:




```
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      0.76      0.87    256680
         1.0       0.01      0.99      0.01       311

    accuracy                           0.76    256991
   macro avg       0.50      0.88      0.44    256991
weighted avg       1.00      0.76      0.86    256991

Accuracy of the model: 0.7632485184306065
```

From the confusion matrix we can conclude that type II error is 2 type I error is 60841, F1 score for class 1 is very low which is 0.01 and the accuracy of the model is 0.7624.

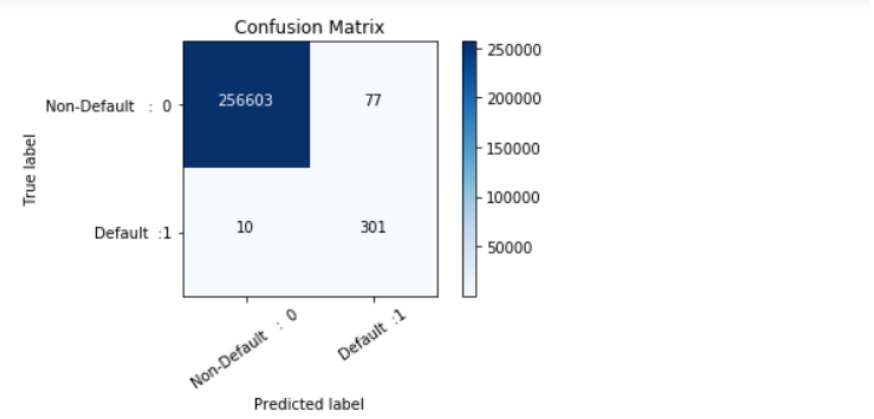
3.4.1 GradientBoostingClassifier

```

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)

Y_pred=model_GradientBoosting.predict(X_test)
```

```
'\nfrom sklearn.ensemble import GradientBoostingClassifier\n\nmodel_GradientBoosting=GradientBoostingClassifier(n_estimators=13
0,)\n\n#fit the model on the data and predict the values\nmodel_GradientBoosting.fit(X_train,Y_train)\n\nY_pred=model_GradientB
oosting.predict(X_test)'
```

```

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.ensemble library using the 'GradientBoostingClassifier' , we created a GradientBoosting model and following results were interpreted:




```
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.80      0.97      0.87       311

   micro avg       1.00      1.00      1.00    256991
   macro avg       0.90      0.98      0.94    256991
weighted avg       1.00      1.00      1.00    256991

('Accuracy of the model:', 0.9996614667439716)
```

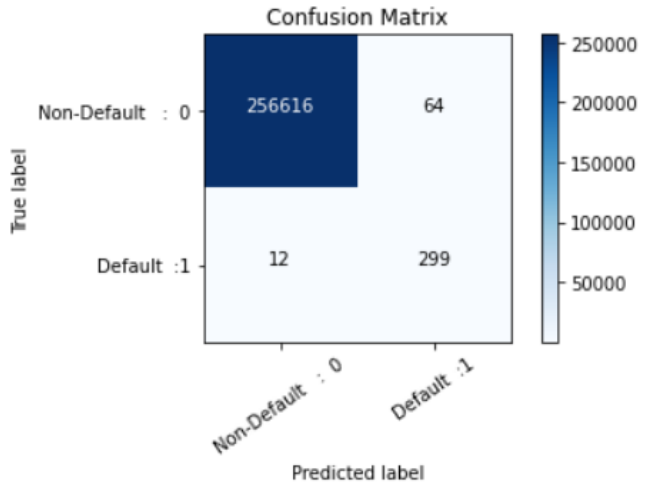
From the confusion matrix we can conclude that type II error is 10 type I error is 77, F1 score for class 0 is 1.00 and for class 1 is 0.87 and the accuracy of the model is 0.9996.

3.5.1Neural Network

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100),max_iter=100, early_stopping=True,n_iter_no_change=5,
                    random_state=10, activation="relu", solver="adam",learning_rate="constant",
                    learning_rate_init=0.01,verbose=True)
mlp.fit(X_train,Y_train)
Y_pred = mlp.predict(X_test)
```

```
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.neural_network library using the 'MLPClassifier' , we created a MLPClassifier model and following results were interpreted:




```
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.82      0.96      0.89       311

    accuracy                           1.00    256991
   macro avg       0.91      0.98      0.94    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.9997042697993315
```

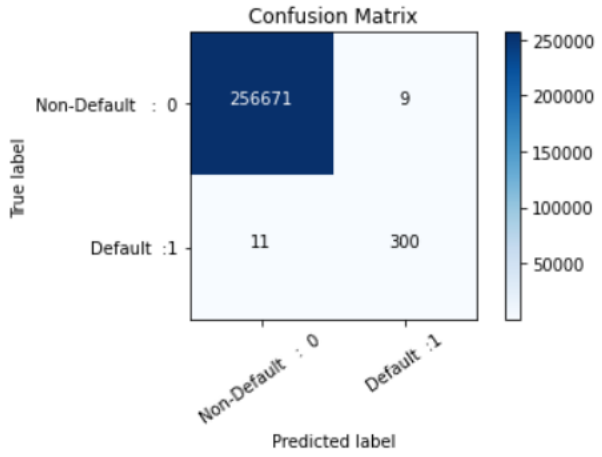
From the confusion matrix we can conclude that type II error is 12 type I error is 64, F1 score for class 1 is 0.89 and for class 0 is 1.00 and the accuracy of the model is 0.9997.

3.5.2 Tunning the MLPClassifier model:

```
1 clf.best_estimator_
```

```
MLPClassifier(activation='tanh', alpha=0.01, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=True, epsilon=1e-08,
              hidden_layer_sizes=(12, 12, 12), learning_rate='constant',
              learning_rate_init=0.01, max_fun=15000, max_iter=100,
              momentum=0.9, n_iter_no_change=5, nesterovs_momentum=True,
              power_t=0.5, random_state=10, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=True,
              warm_start=False)
```

After performing the GridSearchCV on MLPClassifier Model , we found the best estimators mentioned above.




```
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    256680
         1.0       0.97      0.96      0.97       311

    accuracy                           1.00    256991
   macro avg       0.99      0.98      0.98    256991
weighted avg       1.00      1.00      1.00    256991

Accuracy of the model: 0.999922176262982
```

After running the Grid Search CV on MLPClassifier model, we came to the conclusion that type II error is decreased by 1 and type I error is also decreased which is now 9.

F1 score for class 1 is 0.97 which previously 0.98 and accuracy of the model is 0.9999.

Chapter 4: Final Model

```
x = data1.iloc[:,:-1]
y = data1.iloc[:,-1]
```

```
from sklearn.tree import DecisionTreeClassifier

model_DecisionTree_Tuned=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=40,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=10, splitter='best')

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

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=40,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=10, splitter='best')
```

Prediction on Full data set ¶

```
Y_pred = model_DecisionTree_Tuned.predict(x)
```

```
final_df = pd.DataFrame()
```

```
final_df = pd.concat([x,y],axis=1)
```

```
final_df.head()
```

```
final_df['Predicted_class']=Y_pred
```

Decision tree gave us very high Accuracy and very low Type 1 & Type 2 error. So, we chose Decision tree model for the entire data prediction.

Above code describes the prediction on the entire data set.

Export the final data-frame to Root Directory

```
final_df.to_csv('final_data_with_prediction.csv')
```

With the above code we have exported the predicted dataframe to root directory.

Chapter 5: Conclusion

With regards the XYZ Corp. Lending

By looking at the Statistics from the above models we can conclude that all these above models are good but we would go forward with the **Decision Tree model** when we compare it with other models.
The stats of the Decision tree model are the best than other models as it has **less Type 1 & Type 2 errors and High F1 score & Accuracy.**

Therefore, we will produce the Decision Tree model to XYZ Corp. to predict future defaults.