PROJECT REPORT

Data Modelling to Predict Loan Defaulters for XYZ Corporation

Submitted towards the partial fulfilment of the criteria for an award of Genpact Data Science Prodegree by Imarticus

Submitted By:

Pratik Aphale
Balaso Kumbhar
Suniti Kaul

Course and Batch: DSP30 - June 2020

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 a defaulter or not using the data collected when the loan has been given.

Acknowledgements

We are using this opportunity to express my 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 several issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guide 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 DSP program.

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

Date: March 5th, 2021

Place: Pune

Pratik Aphale

Balaso Kumbhar

Suniti Kaul

Certificate of Completion

I hereby certify that the project titled “Data Modelling to Predict Loan Defaulters for XYZ Corporation” was undertaken and completed under my supervision by Pratik Aphale, Balaso Kumbhar and Suniti Kaul from the batch of DSP30 (June 2020)

Date: March 2, 2021

Place – Pune

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 on 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 The need for 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 the manual investing feature provided by the lending club. This, however, does not necessarily lead to the 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 in the future. 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 understudy:

XYZ Corporation Lending Data is under 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 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 information about the status of the loan defaulter. The dataset contains 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: Google colab, Jupyter Notebook

Techniques: Logistic Regression, Decision Tree, Random Forest, AdaBoost, K-Nearest Neighbour, Artificial Neural Networks

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 is elaborated below. After importing libraries, a sequence of steps was followed to perform data preprocessing

2.1 Phase I – Data Extraction and Cleaning:

- **Missing Value Analysis and Treatment**

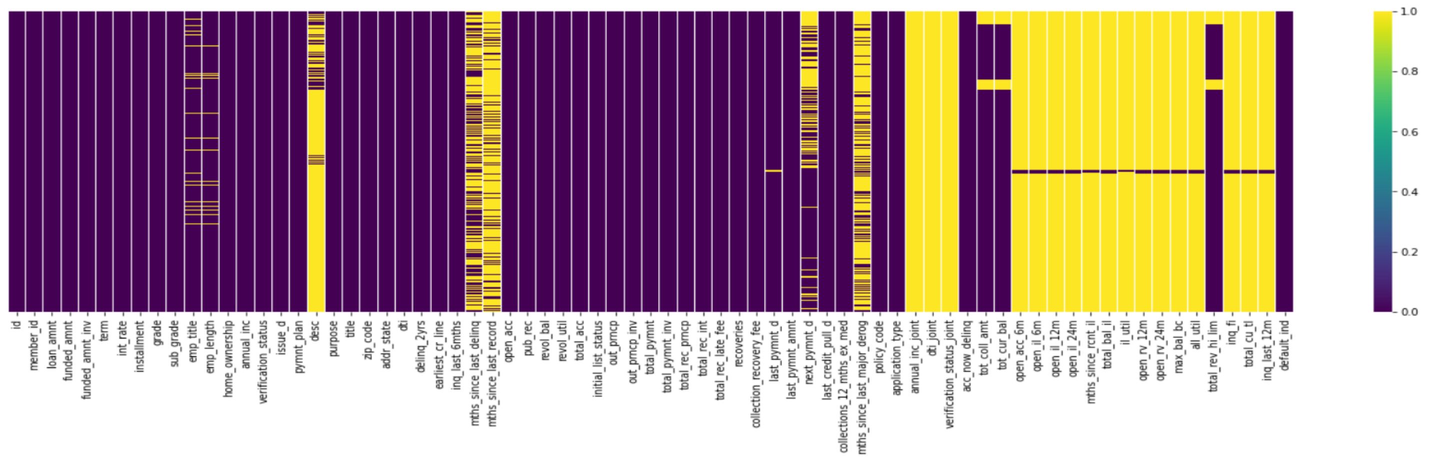
Dataset consists of 855969 rows and 73 columns. To understand Null values we created a dataframe consisting of only null values & their total percentage using for loop & list comprehension as follows and a heatmap to visualize missing data :

```
bl_null = pd.DataFrame({'Features': [var for var in bl.columns if bl[var].isnull().sum() > 1],
                        'Null Values' : [bl[var].isnull().sum() for var in bl.columns if bl[var].isnull().sum() > 1],
                        '% missing' : [bl[var].isnull().sum() / bl.shape[0] *100 for var in bl.columns if bl[var].isnull().sum() > 1 ] })
```

	Features	Null Values	% missing
0	dti_joint	855529	99.948596
1	annual_inc_joint	855527	99.948363
2	verification_status_joint	855527	99.948363
3	il_util	844360	98.643759
4	mths_since_rcnt_il	843035	98.488964
5	inq_last_12m	842681	98.447607
6	open_il_24m	842681	98.447607
7	open_il_12m	842681	98.447607
8	open_il_6m	842681	98.447607
9	open_acc_6m	842681	98.447607
10	open_rv_12m	842681	98.447607
11	open_rv_24m	842681	98.447607
12	total_bal_il	842681	98.447607
13	max_bal_bc	842681	98.447607
14	all_util	842681	98.447607
15	inq_fi	842681	98.447607
16	total_cu_tl	842681	98.447607
17	desc	734157	85.769111
18	mths_since_last_record	724785	84.674211
19	mths_since_last_major_derog	642830	75.099682
20	mths_since_last_delinq	439812	51.381767
21	next_pymnt_d	252971	29.553757
22	total_rev_hi_lim	67313	7.863953
23	tot_cur_bal	67313	7.863953
24	tot_coll_amt	67313	7.863953
25	emp_title	49443	5.776261
26	emp_length	43061	5.030673
27	last_pymnt_d	8862	1.035318

e

```
plt.figure(figsize=(25,6))
sns.heatmap(bl.isnull(),yticklabels=False,cmap="viridis")
```



We have 20 columns with more than 75% of missing values. Above heatmap shows the intensity of values that are missing in every columns. All the yellow coloured columns represents the amount of missing values present in that specific column.

We have set threshold of 30% tolerance of missing values, hence we will delete columns having more than 30% of missing values. Dropping that will leave us with 52 columns.

```
bl.dropna('columns', 'any', thresh=(0.70*len(bl)), inplace=True )
print(bl.shape)
```

```
(855969, 52)
```

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

```
bl.drop(['id','member_id','emp_title','zip_code','policy_code','title'],axis=1, inplace=True)
bl.shape
```

```
del bl['pymnt_plan']
del bl['next_pymnt_d']
del bl['last_pymnt_d']
del bl['last_credit_pull_d']
del bl['earliest_cr_line']
```

- **Id, member_id, zip_code –** unique identifier variables.
- **Policy_code, payment plan –** These variables have same value for all observations.
- **Emp_title –** Categorical variable with 290912 levels, also since it is user input field, it does not have any relation whether person will default or not.
- **Last_credit_pull_d –** because it's a date variable with 102 levels.
- **Title –** loan title is user input field, hence is related to default status of person, also this is a categorical variable with 61000 levels.
- **Next_payment_d -** Date variable with having 29% missing values, also, next payment is not a correct predictor of whether person will default or not.
- **Last_paymnt_d –** date varible.
- **Earliest_cr_line –** Date variable with 697 levels.

Hence we are left with finally 41 columns.

Remaining missing values were imputed by median & mode. Numerical missing values were imputed using median since, we know there are a lot of outliers present in dataset which will affect mean, thus eliminatiing possibility of imputing missing values by mean.

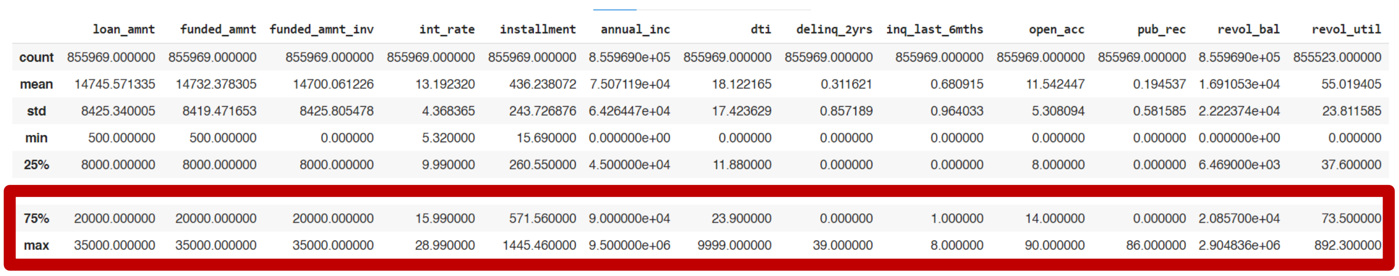
Imputing missing values for numerical variables using Median:

- Tot_coll_amnt
- Tot_cur_bal
- Total_rev_hi_lim
- Revol_util
- Collections_12_mths_ex_med

Imputing missing values for categorical variables using mode:

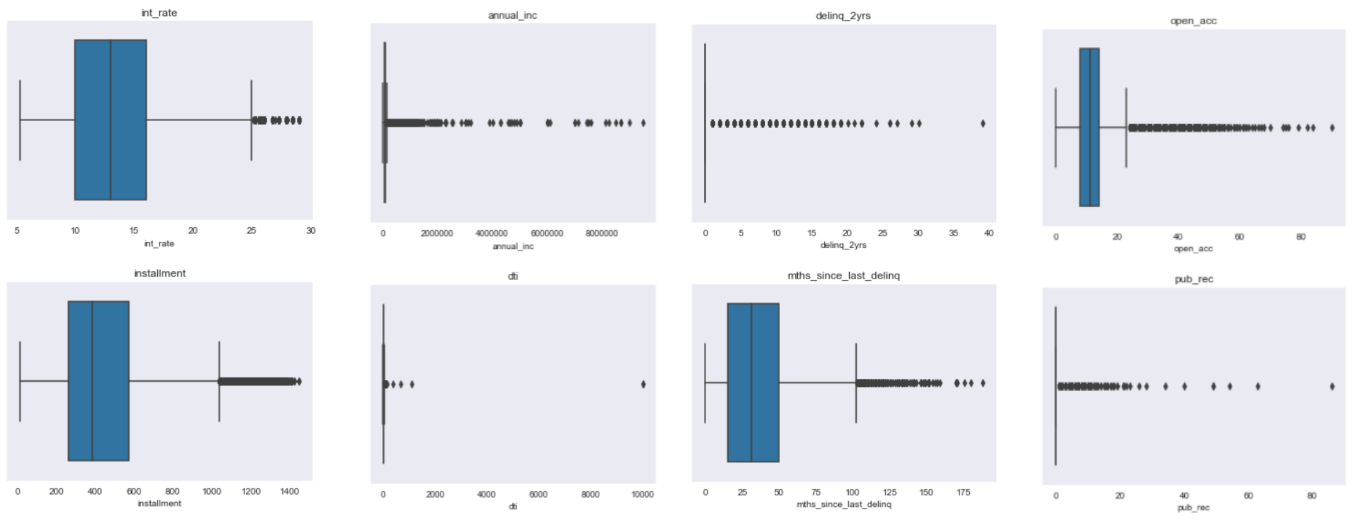
- Emp_length

- <u>**Handling Outliers**</u>



	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installment	annual_inc	dti	delinq_2yrs	inq_last_6mths	open_acc	pub_rec	revol_bal	revol_util
count	855969.000000	855969.000000	855969.000000	855969.000000	855969.000000	8.559690e+05	855969.000000	855969.000000	855969.000000	855969.000000	855969.000000	8.559690e+05	855523.000000
mean	14745.571335	14732.378305	14700.061226	13.192320	436.238072	7.507119e+04	18.122165	0.311621	0.680915	11.542447	0.194537	1.691053e+04	55.019405
std	8425.340005	8419.471653	8425.805478	4.368365	243.726876	6.426447e+04	17.423629	0.857189	0.964033	5.308094	0.581585	2.222374e+04	23.811585
min	500.000000	500.000000	0.000000	5.320000	15.690000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000
25%	8000.000000	8000.000000	8000.000000	9.990000	260.550000	4.500000e+04	11.880000	0.000000	0.000000	8.000000	0.000000	6.469000e+03	37.600000
75%	20000.000000	20000.000000	20000.000000	15.990000	571.560000	9.000000e+04	23.900000	0.000000	1.000000	14.000000	0.000000	2.085700e+04	73.500000
max	35000.000000	35000.000000	35000.000000	28.990000	1445.460000	9.500000e+06	9999.000000	39.000000	8.000000	90.000000	86.000000	2.904836e+06	892.300000

There is huge difference between 75th percentile value & max values, which indicate that there outliers present in the dataset



Outlier Treatment was not done because of following reasons:

- Lack of domain knowledge to decide whether to delete those, or assign some other values to it
- Presence of clusters in outliers.
- Possibility of extreme values containing important information.

2.2 Phase II - Feature Engineering and Feature Selection:

Encoding Categorical to numeric values:

- First we will convert ‘issue_d’ variable to datetime category so that we do not encode it with numeric values. Also issue_d will be useful to split data according to mentioned dates.
- Encoding Categorical variables with label encoding – We are not using One hot encoding, because will introduce lot of variables, which will in turn increase computation time and introduce curse of dimensionality.
- We have done categorical encoding on following variables:
    - Term
    - Grade, Subgrade
    - Emp_Length
    - Home Ownership
    - Verification status
    - Purpose
    - Addr_state
    - Earliest_cr_line
    - Initial_list_status
    - Application Type.

```
# label endcoding for the object datatypes except the column "issue_d"
for var in bl.columns:
    if bl[var].dtypes == 'object':
        le = preprocessing.LabelEncoder()
        le = le.fit(bl[var])
        bl[var] = le.transform(bl[var])
        print('Completed Label encoding on',var)
```

```
Completed Label encoding on term
Completed Label encoding on grade
Completed Label encoding on sub_grade
Completed Label encoding on emp_length
Completed Label encoding on home_ownership
Completed Label encoding on verification_status
Completed Label encoding on purpose
Completed Label encoding on addr_state
Completed Label encoding on earliest_cr_line
Completed Label encoding on initial_list_status
Completed Label encoding on application_type
```

Feature Selection:

- There are different methods for feature selection, like Select K Best with chi-sq test of association, Boruta etc…
- We have used Boruta for feature selection.

Boruta Feature Selection(BorutaPy):

Python implementations of the Boruta R package. This implementation tries to mimic the scikit-learn interface, so use fit, transform or fit_transform, to run the feature selection.

Boruta is an all relevant feature selection method, while most other are minimal optimal; this means it tries to find all features carrying information usable for prediction, rather than finding a possibly compact subset of features on which some classifier has a minimal error.

1. Features compete with a randomized version of them based on random forest classifier which is capable of capturing non-linear relationships and interactions.
2. Model is fit on X_shadow(randomized dataset) & y
3. A feature is useful only if it’s capable of doing better than the best randomized feature.

```
import boruta
from boruta import BorutaPy
from sklearn.ensemble import RandomForestClassifier
x = np.array(x)
y = np.array(y)
```

```
rf = RandomForestClassifier()
Borutaa = BorutaPy(rf, n_estimators='auto', verbose=2, max_iter=15)
```

```
Borutaa.fit(x,y)
```

```
BorutaPy finished running.

Iteration:      9 / 15
Confirmed:      25
Tentative:      0
Rejected:       14
BorutaPy(estimator=RandomForestClassifier(n_estimators=88,
                                          random_state=RandomState(MT19937) at 0x2AE38930BF8),
         max_iter=15, n_estimators='auto',
         random_state=RandomState(MT19937) at 0x2AE38930BF8, verbose=2)
```

2.3 Data Dictionary:

LoanStatNew	**Description**
annual_inc	The self-reported annual income provided by the borrower during registration.
dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested loan, divided by the borrower's self-reported monthly income.
funded_amnt	The total amount committed to that loan at that point in time.
funded_amnt_inv	The total amount committed by investors for that loan at that point in time.
grade	XYZ corp. assigned loan grade
initial_list_status	The initial listing status of the loan. Possible values are – W, F
installment	Installemnts of loan paid
int_rate	Interest Rate on the loan
last_pymnt_amnt	Last total payment amount received
loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
out_prncp	Remaining outstanding principal for total amount funded
out_prncp_inv	Remaining outstanding principal for portion of total amount funded by investors
recoveries	post charge off gross recovery
revol_bal	Total credit revolving balance
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
sub_grade	XYZ assigned assigned loan subgrade

term	The number of payments on the loan. Values are in months and can be either 36 or 60.
total_pymnt	Payments received to date for total amount funded
total_pymnt_inv	Payments received to date for portion of total amount funded by investors
total_rec_int	Interest received to date
total_rec_late_fee	Late fees received to date
total_rec_prncp	Principal received to date
total_rev_hi_lim	Total revolving high credit/credit limit
tot_cur_bal	Total current balance of all accounts

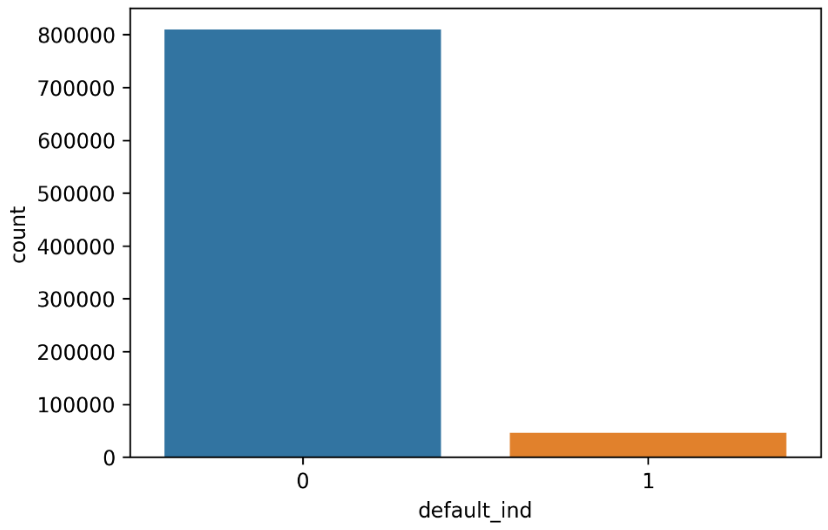
2.4 Exploratory Data Analysis:

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 ('0')

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

```
0    809502
1     46467
```

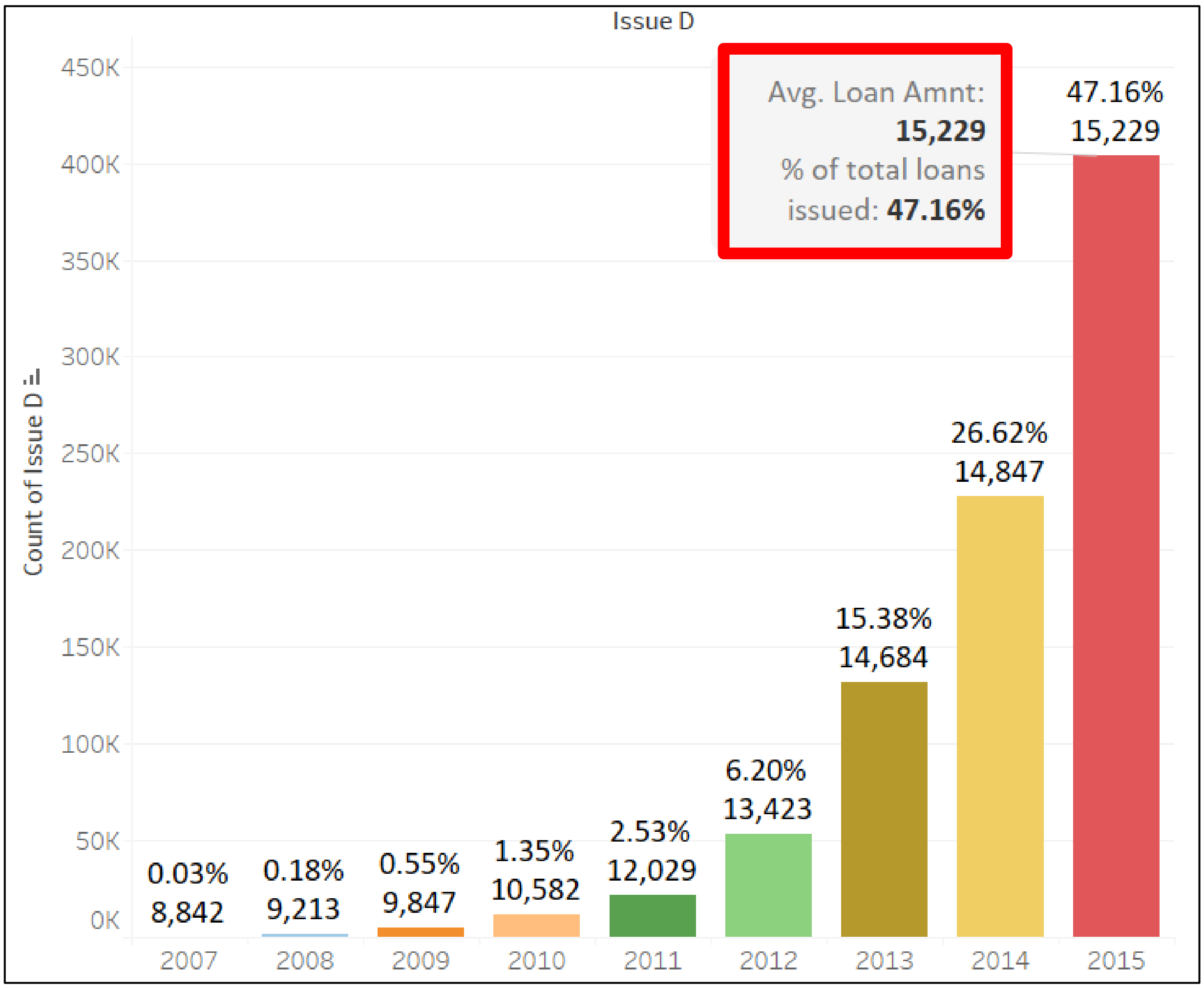


Non-Default Customer: 94.57 % of the dataset.

Default Customer: 5.43 % of the dataset.

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

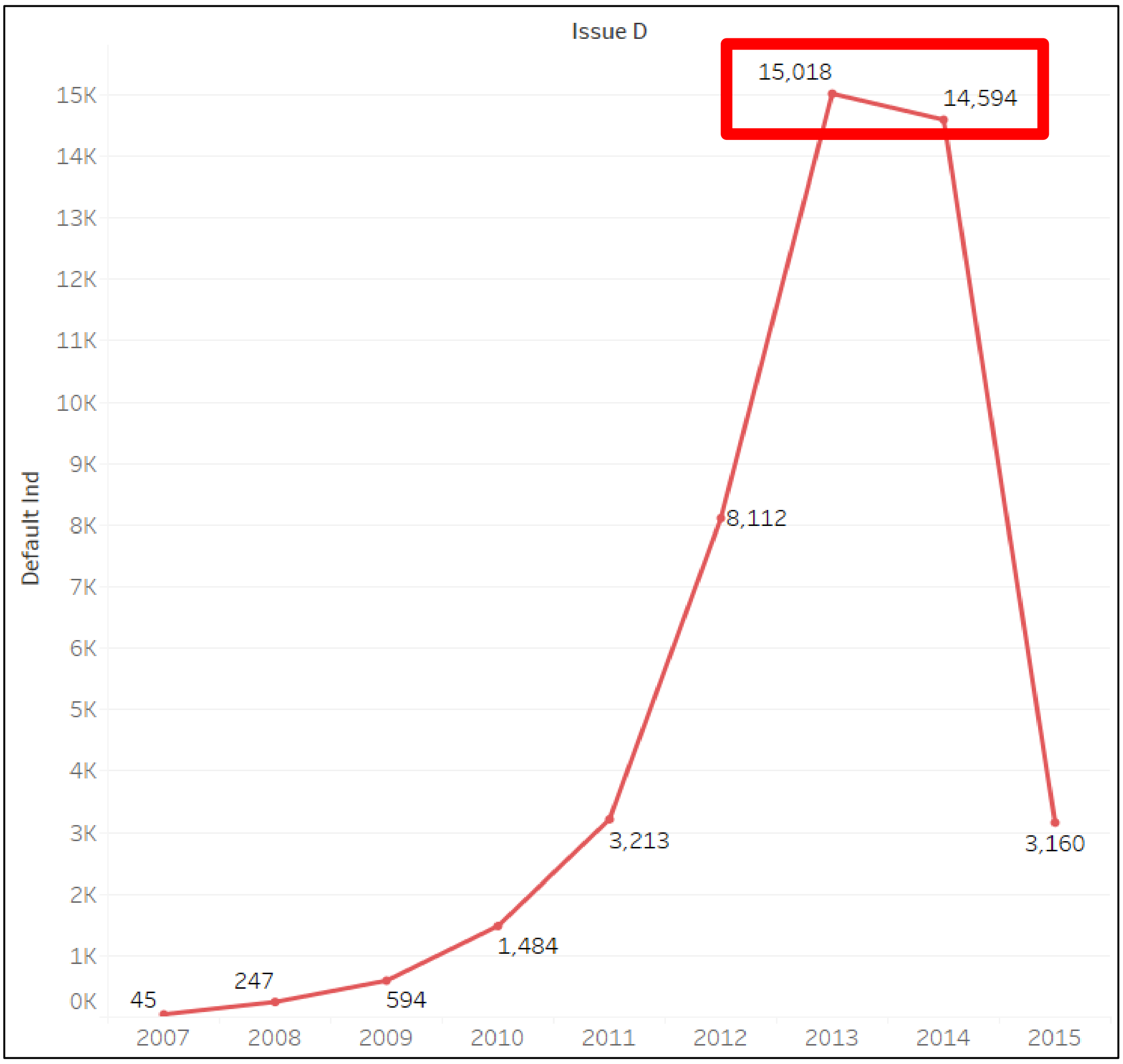
- Plot of Year Vs Loans issued along with average loan amount



1. We can see that number of loans issued has increased exponentially from 2007 to 2015.
2. Average loan amount has also increased accordingly every year, so much so that average loan amount in 2015

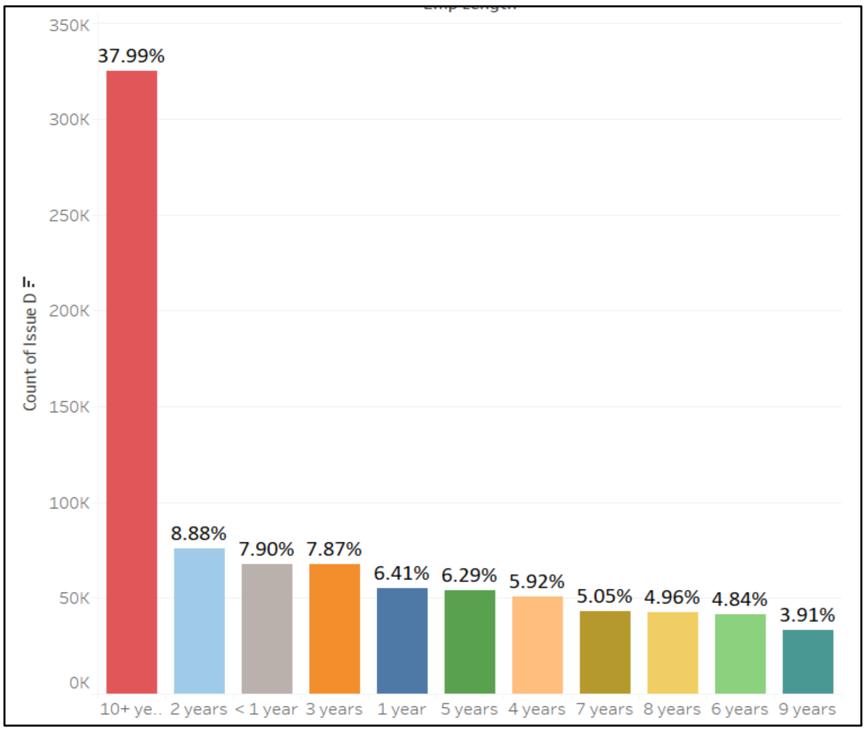
3. We can see that there has been drastic rise in loan issued from 2013. This can be due to increased adoption of digital technology in all the sectors. This helped automate complicated tasks like credit analysing & due diligence for passing loans.

- Plot of loan defaulters per year.



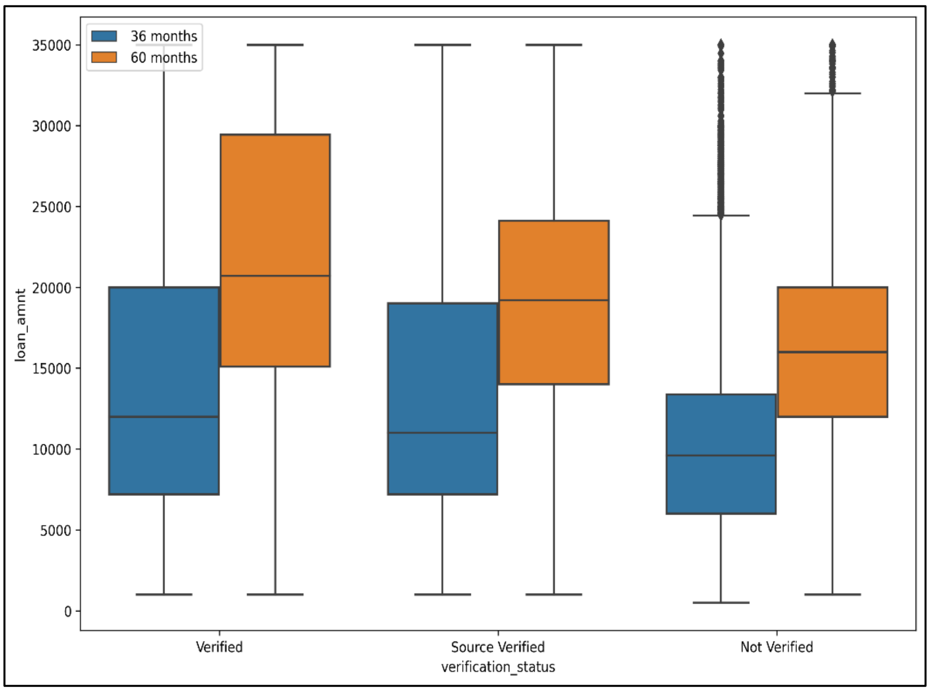
1. Since there was increase in loan issuing, number of defaulters also are increasing accordingly.
2. But observe that in 2013 & 2014 have highest number of defaulters & then in 2015 number reduced drastically.
3. This can be accounted to rise on adoption of digital technologies & companies adopting AI &ML to automate process of loan approvals.
4. Since there was early adoption, all this automation did not perform well in starting years, but then started yielding best results from 2015.

- Plot of number of loans issued Vs Employment length



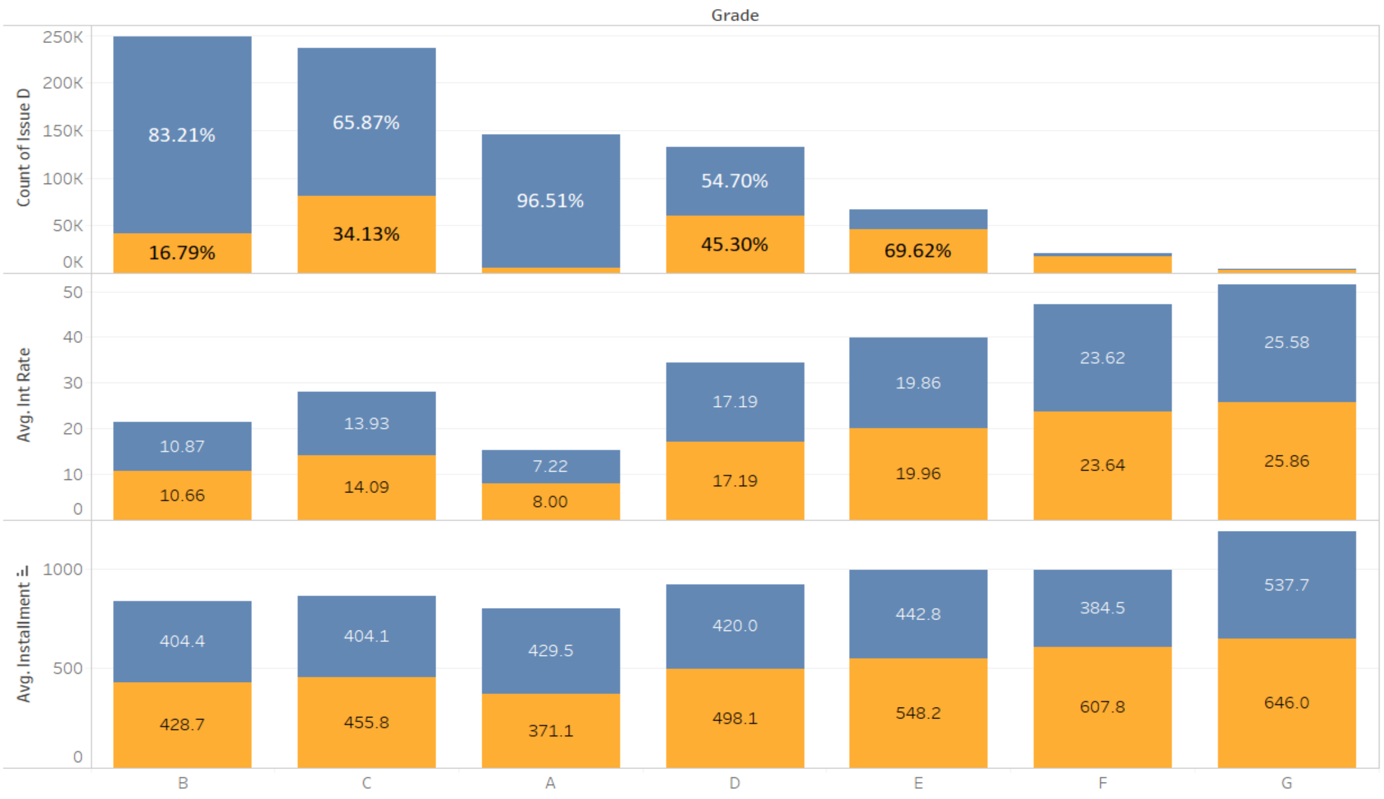
1. 38% of total loans passed we issued to people with 10+ years of employment.

- Boxplot of Loan Amount by Verification Status



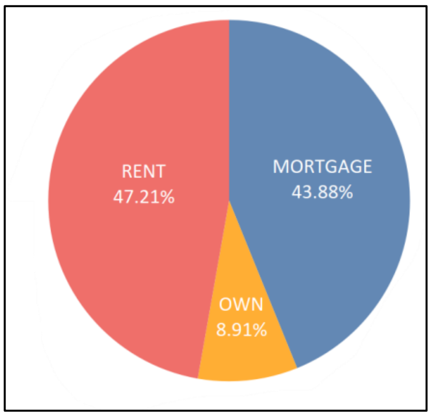
1. Loan Amount for Not Verified loans is less irrespective of loan term.
2. Loan term with 60 months generally has higher loan amount.
3. Verified loans are issued more loan amount than any other category.

- Analysis of Grade with respect to term for:
    1. Number of loans issued.
    2. Average interest rate.
    3. Average installment.



1. For Grade A most number of loans (96%) are of term 36 months. Hence it has least interest rate (7-8%).
2. Most number of loans are of Grade B & C.
3. Number of loans issued start decreasing after Grade D drastically.
4. Average interest rate increases with increase in Grade
5. As the grade increases, contribution of 60 months loans start increasing.
6. As the grade increases average interest rate & Average instalment also increase.

- **Pie Chart of loan defaulter based on home ownership**

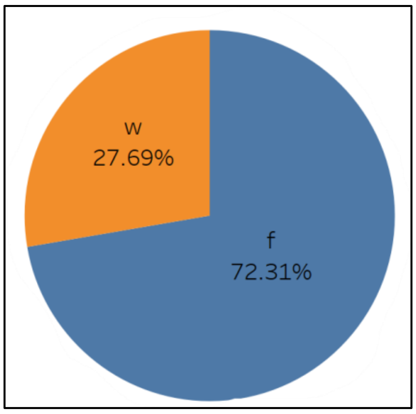


1. Highest number of defaulters can be seen for people leaving on rent (47%).

2. People having mortgage also have large number of defaulters. (44%)

3. Least number of defaulters can be seen for people having their own house.

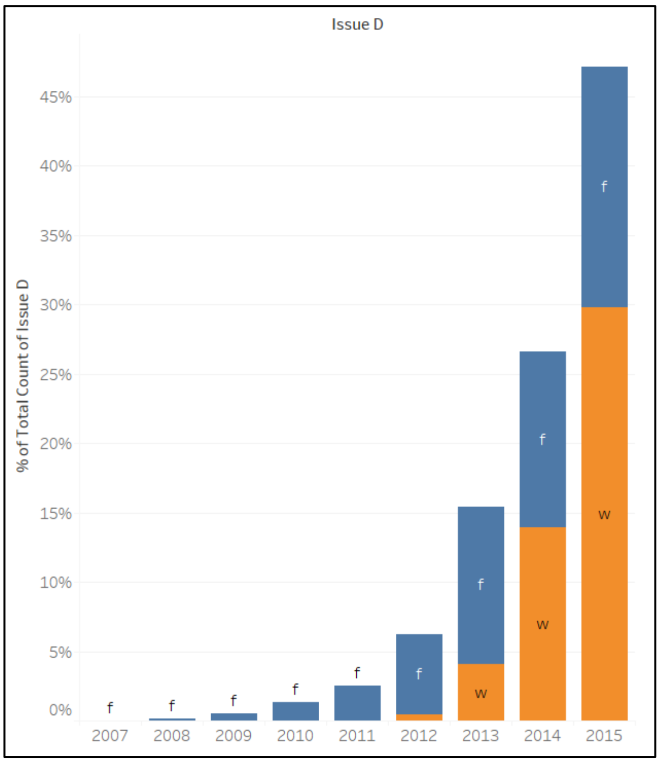
- **Pie Chart of loan defaulter based on initial list status**



1. Highest number of defaulters can be seen for initial list status 'f' (fractional) (72%)

2. We'll explore that in the next bar graph

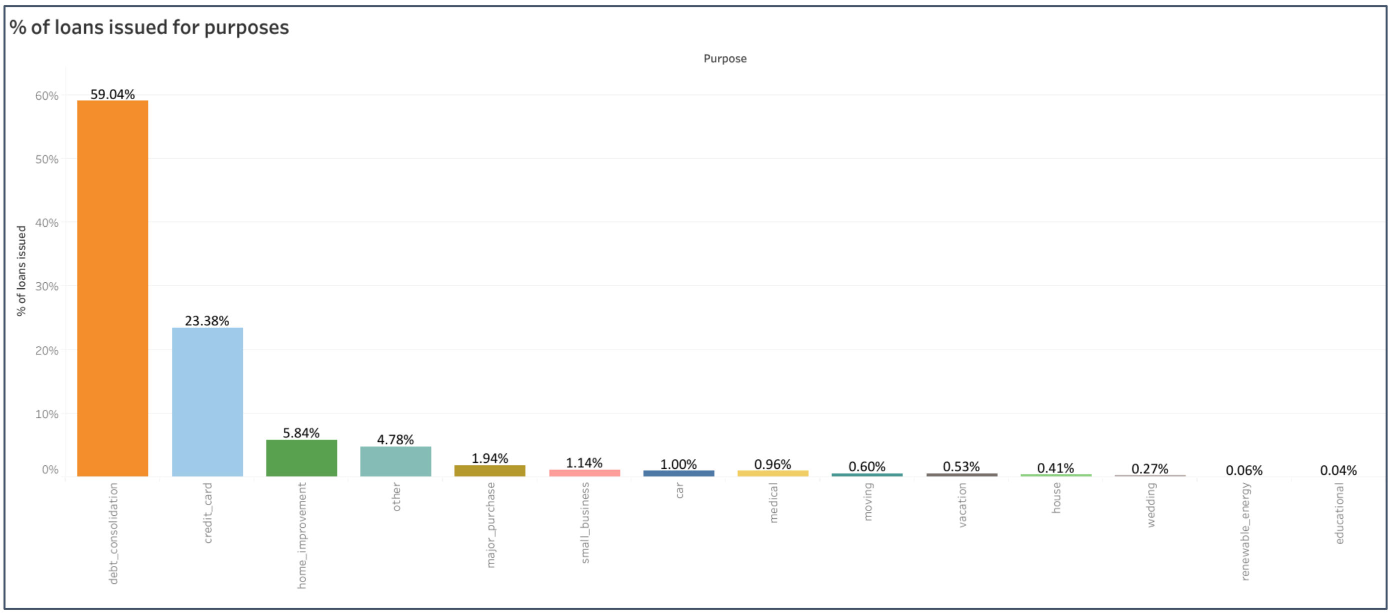
- Bar chart of Number of loans issued per year by listing status



1. There are lower number of defaulter in 'w' (whole) initial list status is because these loans were introduced from 2012.

2. From 2013 we can see that 'w' initial list status loans started increasing rapidly, so much that in 2015 total loans issued, maximum number of loans were of 'w'

- **Percentage of loans issued for purpose**



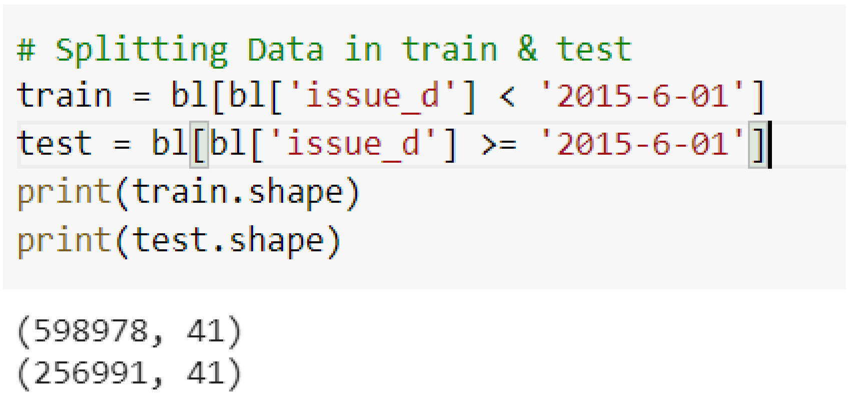
1. Maximum number of loans (~60%) were issued for debt consolidation purpose.
2. Credit card related loans are on 2nd number for purpose.

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.

We already have divide **‘issue_d’** to datetime format, we will split data by selecting dates before 01-06-2015 for train & after 01-06-2015 for test.

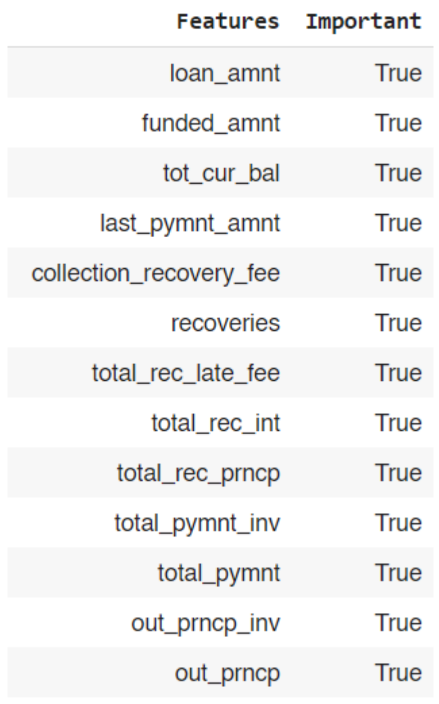
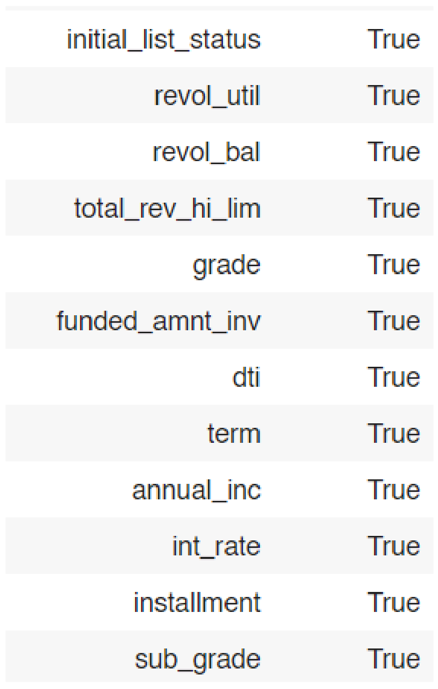


```
# Splitting Data in train & test
train = bl[bl['issue_d'] < '2015-6-01']
test = bl[bl['issue_d'] >= '2015-6-01']
print(train.shape)
print(test.shape)
```

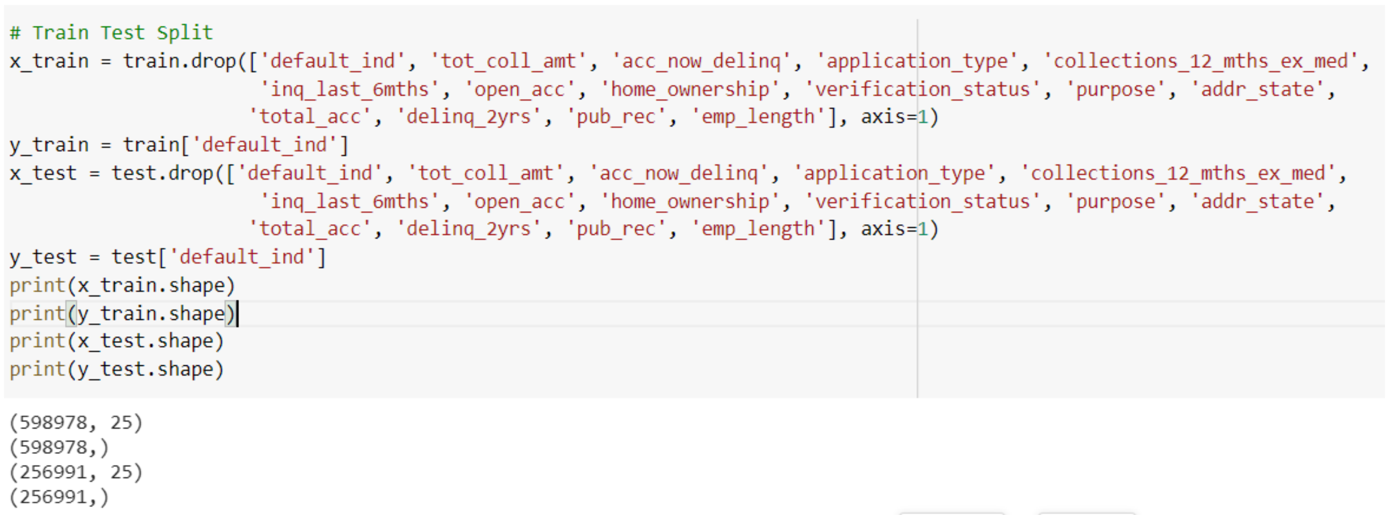
```
(598978, 41)
(256991, 41)
```

- Creating x_train, y_train, x_test, y_test dataframes.

Dropping columns which returned **'False'** after Boruta Feature Selection & selecting only columns which returned **'True'** as follows

Features	Important
loan_amnt	True
funded_amnt	True
tot_cur_bal	True
last_pymnt_amnt	True
collection_recovery_fee	True
recoveries	True
total_rec_late_fee	True
total_rec_int	True
total_rec_prncp	True
total_pymnt_inv	True
total_pymnt	True
out_prncp_inv	True
out_prncp	True
initial_list_status	True
revol_util	True
revol_bal	True
total_rev_hi_lim	True
grade	True
funded_amnt_inv	True
dti	True
term	True
annual_inc	True
int_rate	True
installment	True
sub_grade	True



```
# Train Test Split
x_train = train.drop(['default_ind', 'tot_coll_amt', 'acc_now_delinq', 'application_type', 'collections_12_mths_ex_med',
                      'inq_last_6mths', 'open_acc', 'home_ownership', 'verification_status', 'purpose', 'addr_state',
                     'total_acc', 'delinq_2yrs', 'pub_rec', 'emp_length'], axis=1)
y_train = train['default_ind']
x_test = test.drop(['default_ind', 'tot_coll_amt', 'acc_now_delinq', 'application_type', 'collections_12_mths_ex_med',
                      'inq_last_6mths', 'open_acc', 'home_ownership', 'verification_status', 'purpose', 'addr_state',
                     'total_acc', 'delinq_2yrs', 'pub_rec', 'emp_length'], axis=1)
y_test = test['default_ind']
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(598978, 25)
(598978,)
(256991, 25)
(256991,)
```

3.2 Model Building

- **Logistic Regression**

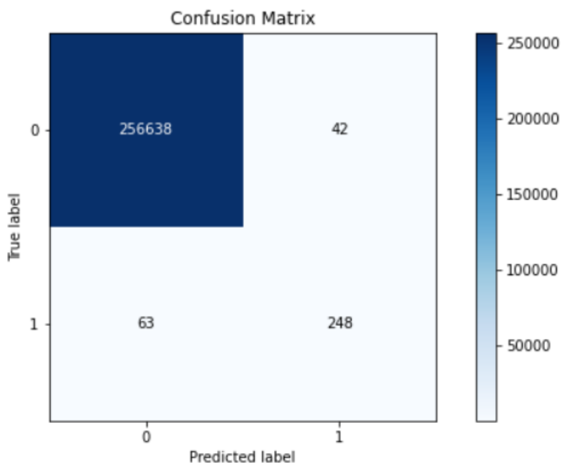
By running model with default parameters

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
lgr = LogisticRegression()
lgr.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# Prediction on test data
lgr_pred = lgr.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256701
           1       0.80      0.86      0.83       290

    accuracy                           1.00    256991
   macro avg       0.90      0.93      0.91    256991
weighted avg       1.00      1.00      1.00    256991

[[256638     42]
 [    63    248]]
```

Referring to the above confusion matrix, we can clearly see that that **Precision** is **52** while the **Recall** is **63**. Since the data is unbalanced, we would not focus on the accuracy of the model but instead tune the model for high Precision & Recall Value.

- **Logistic Regression – By Deleting Null Values**

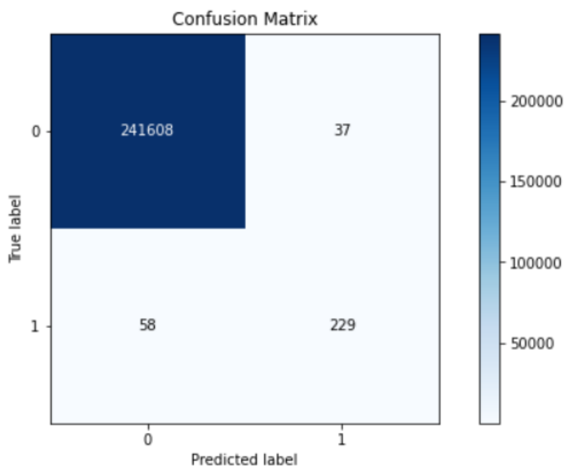
We also try to check model behaviour my deleting null values, sometimes this works better since lot of intentional null values remove unnecessary information from the dataset, thereby increasing model performance

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
lgr = LogisticRegression()
lgr.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# Prediction on test data
lgr_pred = lgr.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    241666
           1       0.80      0.86      0.83       266

    accuracy                           1.00    241932
   macro avg       0.90      0.93      0.91    241932
weighted avg       1.00      1.00      1.00    241932

[[241608     37]
 [    58    229]]
```

There is slight decrease in FPR values, improving Precision slightly. We can further tune this model by using Grid Search CV. Since we are getting better result by deleting null values, we’ll use this dataset for building all other models also.

- **<u>Logistic Regression – Handling Class Imbalance</u>**

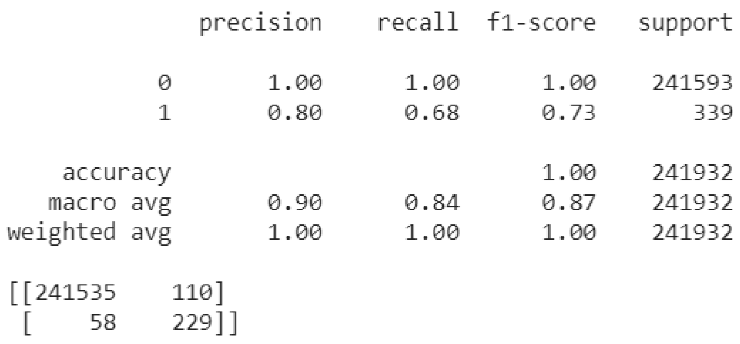
We see that there is heavy class imbalance in the target variable, due to this there will occur some problems, namely as follows:

1. Biased Predictions
2. Misleading Accuracy

SMOTE (Synthetic Minority Oversampling Technique)

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

SMOTE plots all the target variable points in 2D scatter plot. It then calculates distance between minority points based on number of neighbors (k=5). After that it calculates distance between neighbors & multiplies that value with any number between 0 & 1 and plots new synthetic point between two points thereby increasing minority class samples.



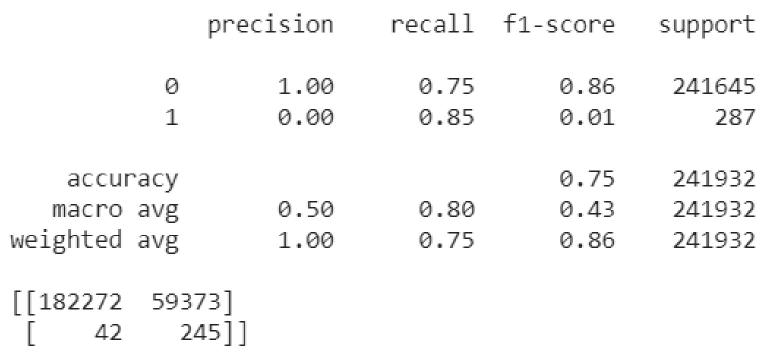
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    241593
           1       0.80      0.68      0.73       339

    accuracy                           1.00    241932
   macro avg       0.90      0.84      0.87    241932
weighted avg       1.00      1.00      1.00    241932

[[241535    110]
 [    58    229]]
```

Near Miss Undersampling Technique

Step 1: The method first finds the distances between all instances of the majority class and the instances of the minority class. Here, majority class is to be under-sampled.

Step 2: Then, **n** instances of the majority class that have the smallest distances to those in the minority class are selected.

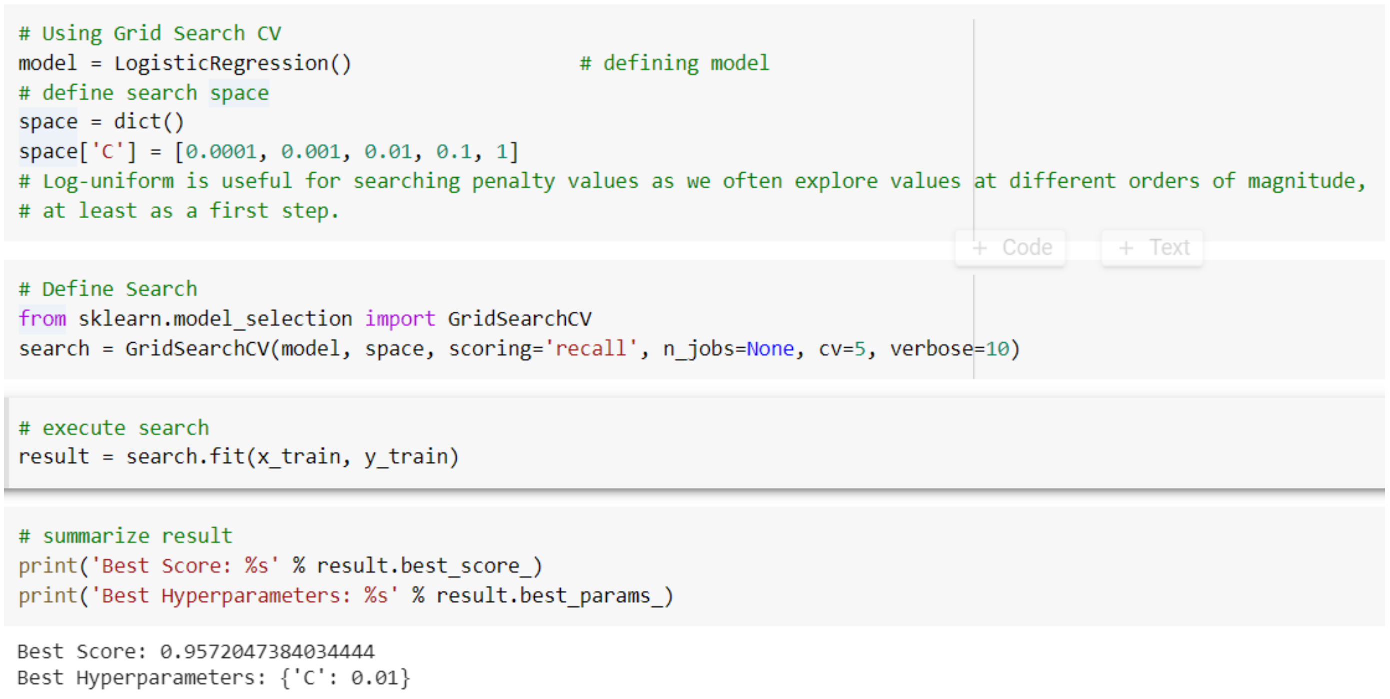
Step 3: If there are k instances in the minority class, the nearest method will result in **k*n** instances of the majority class.



```
              precision    recall  f1-score   support

           0       1.00      0.75      0.86    241645
           1       0.00      0.85      0.01       287

    accuracy                           0.75    241932
   macro avg       0.50      0.80      0.43    241932
weighted avg       1.00      0.75      0.86    241932

[[182272  59373]
 [    42    245]]
```

Both SMOTE & Near Miss Techniques failed to give result better result than model without handling class imbalance. Hence we will continue without handling class imbalance for further models.

- **Logistic Regression – Tuning Model using Grid Search CV**



```
# Using Grid Search CV
model = LogisticRegression()                    # defining model
# define search space
space = dict()
space['C'] = [0.0001, 0.001, 0.01, 0.1, 1]
# Log-uniform is useful for searching penalty values as we often explore values at different orders of magnitude,
# at least as a first step.
```

```
# Define Search
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(model, space, scoring='recall', n_jobs=None, cv=5, verbose=10)
```

```
# execute search
result = search.fit(x_train, y_train)
```

```
# summarize result
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.9572047384034444
Best Hyperparameters: {'C': 0.01}
```

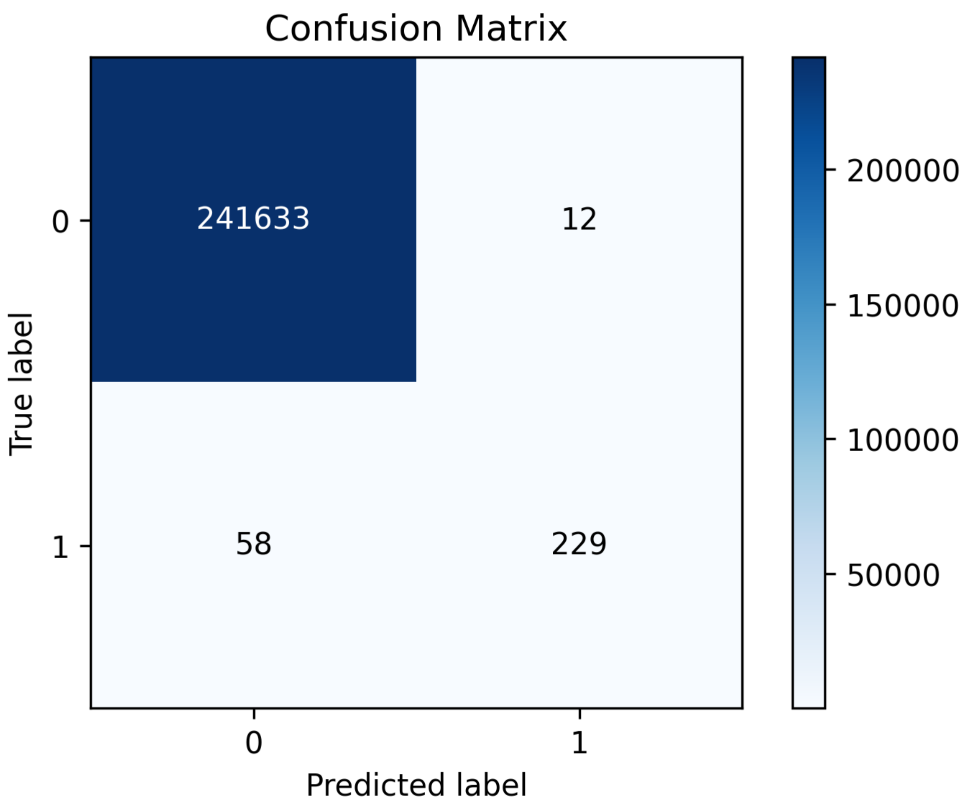
Building Model with C=0.01

```
# Running Logistic Regression using Grid searh result
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
lgr_gs = LogisticRegression(C=0.01)        # Best result for C=0.01
lgr_gs.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
lgr_gs_pred = lgr_gs.predict(x_test)
```

Results:



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    241645
           1       0.95      0.80      0.87       287

    accuracy                           1.00    241932
   macro avg       0.97      0.90      0.93    241932
weighted avg       1.00      1.00      1.00    241932
```

We can see that there is significant increase in Precision, but our Recall value is remaining same. This model should be considered our best model, since even after tuning, we could not improve recall.
But we can compensate it by improvement in Precision, since Precision is our loss of business, meaning Person is not defaulter but we predicted it as defaulter, thereby rejecting him loan. By improving Precision, we'll be able to disburse safe loans, thereby decreasing risk of Recall (Person is defaulter but we predicted it as non-defaulter thereby losing our principal & interest amount).

Decision Tree Classifier:

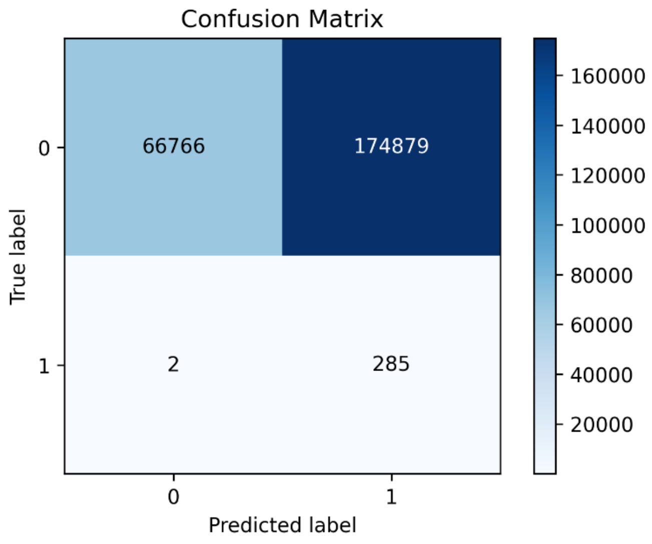
Training the model on the train set and then predicting on the test set using '**gini**' for splitter selection and using the 'DecisionTreeClassifier' class.

```
dt = DecisionTreeClassifier()
```

```
dt.fit(x_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, 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, presort='deprecated',
                       random_state=None, splitter='best')
```

```
dt_pred = dt.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      0.28      0.43    241645
           1       0.00      0.99      0.00       287

    accuracy                           0.28    241932
   macro avg       0.50      0.63      0.22    241932
weighted avg       1.00      0.28      0.43    241932
```

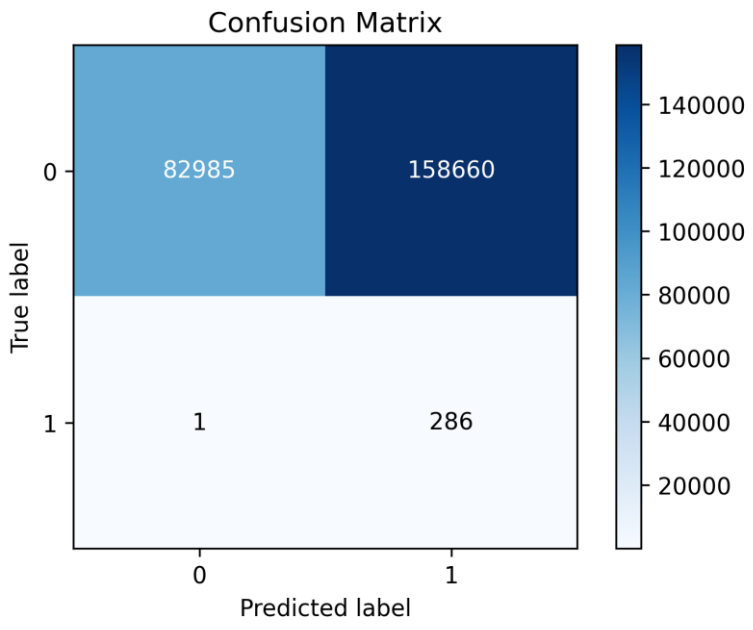
In this model, Recall value is very high which is desired, but Precision value is 0, which is totally not acceptable, since maximum applications will be rejected, thereby losing all business.

Random Forest Classifier:

```
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
rf_pred = rf.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      0.34      0.51    241645
           1       0.00      1.00      0.00       287

    accuracy                           0.34    241932
   macro avg       0.50      0.67      0.26    241932
weighted avg       1.00      0.34      0.51    241932
```

In this model, Recall value is very high which is desired, but Precision value is 0, which is totally not acceptable, since maximum applications will be rejected, thereby losing all business.

AdaBoost Classifier:

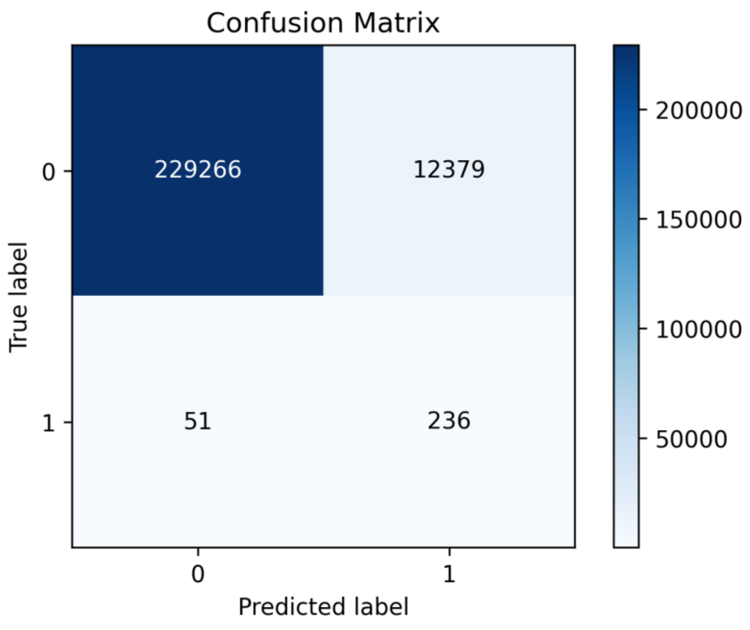
```
dt = DecisionTreeClassifier()    # adaptive boosting will run top of a classifier
```

```
adb = AdaBoostClassifier()
```

```
adb.fit(x_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
```

```
adb_pred = adb.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97    241645
           1       0.02      0.82      0.04       287

    accuracy                           0.95    241932
   macro avg       0.51      0.89      0.51    241932
weighted avg       1.00      0.95      0.97    241932
```

This model has failed to improve both Precision & Recall values.

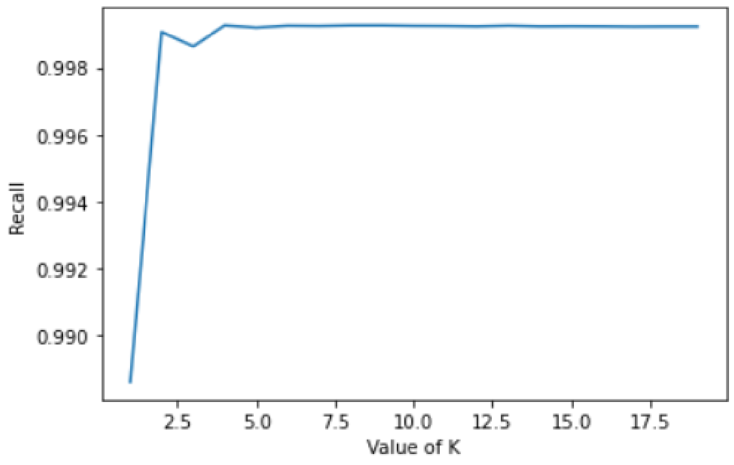
KNN Classifier:

```
knn = KNeighborsClassifier()
```

```
k = range(1,20)
dict = {}
knn_acc_list = []
for i in k:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    knn_pred = knn.predict(x_test)
    from sklearn import metrics
    dict[i] = metrics.recall_score(y_test,knn_pred)
    knn_acc_list.append(metrics.accuracy_score(y_test,knn_pred))
```

```
# Plot the accuracy and value of k

plt.plot(k,knn_acc_list)
plt.xlabel("Value of K")
plt.ylabel("Recall")
```

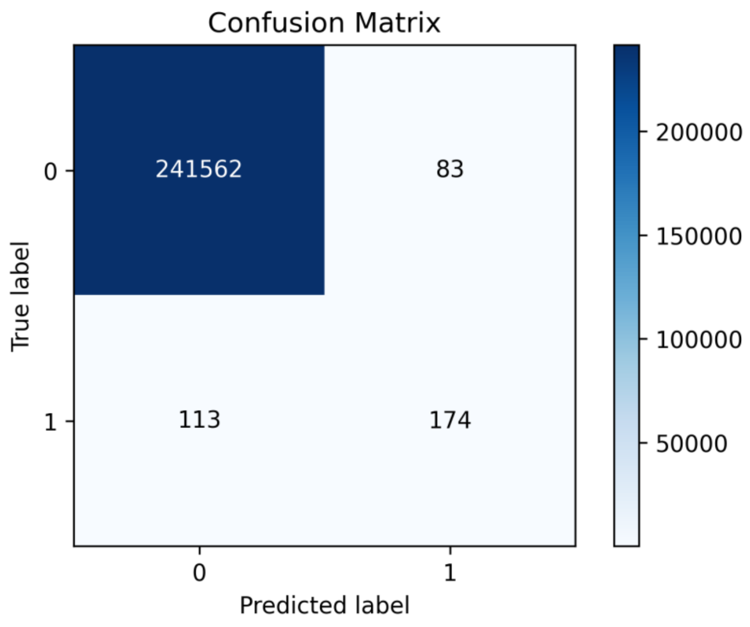
```
Text(0, 0.5, 'Recall')
```



```
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
knn_pred = knn.predict(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    241645
           1       0.68      0.61      0.64       287

    accuracy                           1.00    241932
   macro avg       0.84      0.80      0.82    241932
weighted avg       1.00      1.00      1.00    241932
```

This model also failed on both Precision & Recall performance metrics.

Artificial Neural Networks Classifier:

```
# Converting train datasets into numpy array since Neural Networks take arrays as inputs.
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
```

```
# Normalizing Train datasets to improve accuracy
x_train = tf.keras.utils.normalize(x_train)
x_test = tf.keras.utils.normalize(x_test)
```

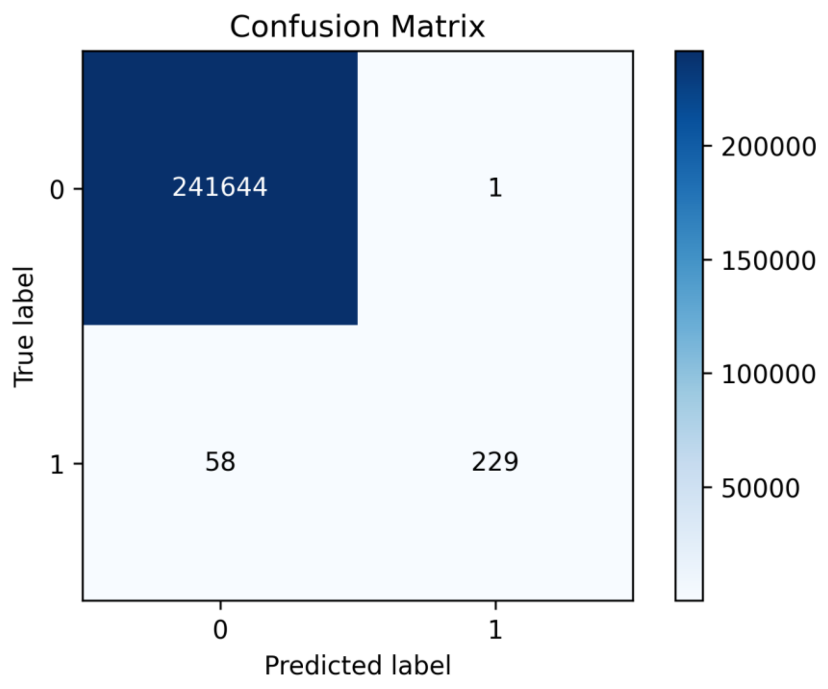
```
model = tf.keras.Sequential()    # initialize the model
model.add(tf.keras.layers.Dense(256, activation = tf.nn.relu))   # 1st layer
model.add(tf.keras.layers.Dense(256, activation = tf.nn.relu))   # 2nd layer
model.add(tf.keras.layers.Dense(512, activation = tf.nn.relu))   # 3rd layer
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))   # 4th layer
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))   # 5th layer
model.add(tf.keras.layers.Dense(10, activation = tf.nn.sigmoid)) # last layer
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy')
```

Fitting the ANN to the training set with batch size of 100 and 10 epochs.

```
model.fit(x_train, y_train, epochs=10, validation_split=0.2, batch_size=100)
```

```
Epoch 1/10
4043/4043 [==============================] - 23s 5ms/step - loss: 0.1187 - val_loss: 0.0173
Epoch 2/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0205 - val_loss: 0.0198
Epoch 3/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0182 - val_loss: 0.0180
Epoch 4/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0188 - val_loss: 0.0180
Epoch 5/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0178 - val_loss: 0.0167
Epoch 6/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0170 - val_loss: 0.0164
Epoch 7/10
4043/4043 [==============================] - 21s 5ms/step - loss: 0.0167 - val_loss: 0.0159
Epoch 8/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0168 - val_loss: 0.0162
Epoch 9/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0162 - val_loss: 0.0162
Epoch 10/10
4043/4043 [==============================] - 20s 5ms/step - loss: 0.0161 - val_loss: 0.0159
<tensorflow.python.keras.callbacks.History at 0x7fd58002a410>
```

```
ann_pred = model.predict_classes(x_test)
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    241645
           1       1.00      0.80      0.89       287

    accuracy                           1.00    241932
   macro avg       1.00      0.90      0.94    241932
weighted avg       1.00      1.00      1.00    241932
```

ANN worked better than Logistic Regression in Precision aspect.

CHAPTER 4 - CONCLUSION

We can consider ANN model as a final model, but we should go with Logistic Regression with Grid Search for final model.

This is because ANN will require substantial investment in state of art hardware (GPUs) but still improving model tab bit better.

Hence conclude that our Logistic Regression with Grid Search CV is the best model for Predicting Loan Defaulters.

CHAPTER 5 - REFERENCES

- Imarticus Class Recordings/Notes
- stackoverflow.com
- https://scikit-learn.org/stable/ - preprocessing, model selection
- Krish Naik Youtube channel
- https://pandas.pydata.org/
- https://seaborn.pydata.org/