CAPSTONE PROJECT

Loan Analysis: Finding Defaulters

Mentor: Nikita Tandel

Rachit Agarwal
Vishwesh Deore
Shreyash Upadhyaya

Language/ Tool: Python, Tableau

Loan Analysis Overview

Identifying the Business problem

Banks may face huge losses due to the **defaults** made by their clients which increases the rejection rate of the loan applicants. To overcome this situation, we need a better credit risk scoring model which can correctly identify which

applicant will be a **defaulter** in the future. This will be done by analysing the historic data and identifying the patterns. Such a model would **minimize credit risk** and will prevent the clients which are capable of repayment from getting rejected.

Data Description

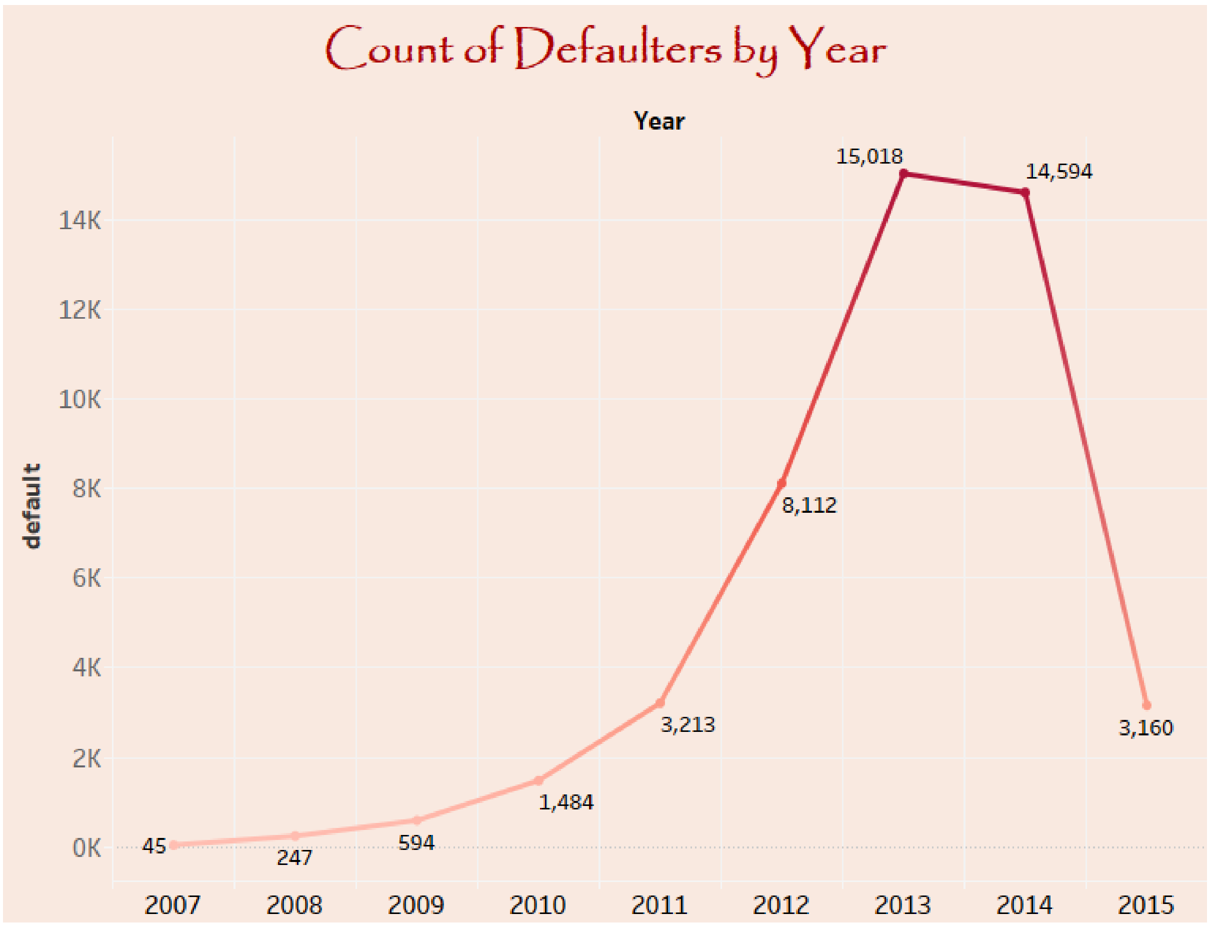
The dataset contains complete loan data for all loans issued through the year **2007 - 2015**, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. Our dataset contains total of **8,55,969** records with **73 features** including target variable. Moreover, the dataset is very **unbalanced**, with approximately **6 %** of loans considered as **defaulted**. This dataset has different types of features such as categorical, numeric & date.

Some important features:

- loan_amnt - Amount of money requested by the borrower.
- int_rate - Interest rate of the loan.
- grade - Loan grade with categories A, B, C, D, E, F, G.
- annual_inc - Borrowers annual income.
- purpose - The primary purpose of borrowing.
- installments – Monthly amount payments for opted loan.
- term – duration of the loan until it’s paid off

A default can occur when a borrower is unable to make timely payments, misses payments, avoids or stops making payments.

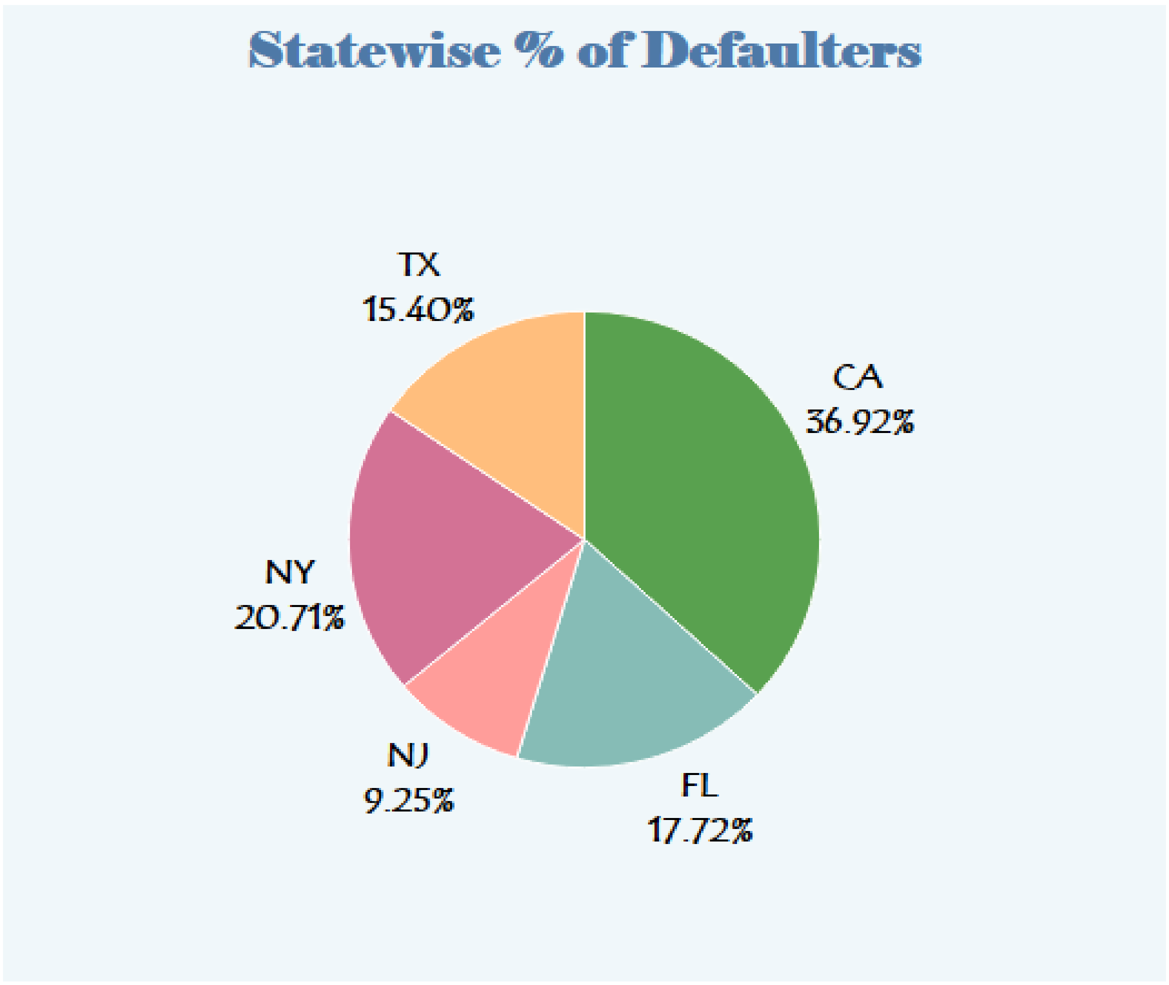
Data Visualisation



The above graph shows total no. of defaulter's year wise.

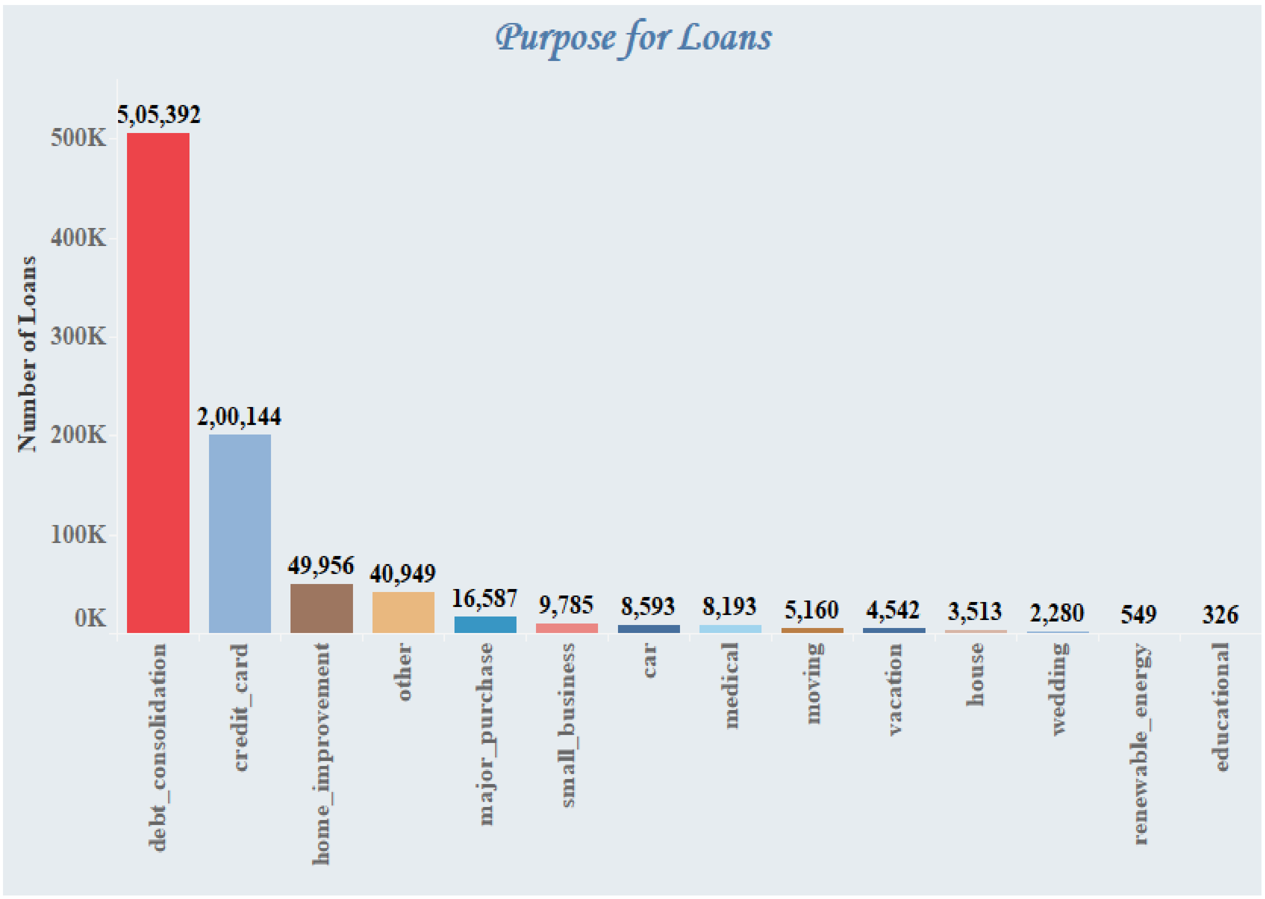
The number of defaulters show a significant **rise** during the years from **2012** to **2014**, some of the generalized reasons affecting this might be

- LIBOR Scandal
- Hurricane Sandy
- Hostess Files for Bankruptcy



These are the major **states** with highest percent of defaulters.

State	State Code	Percentage of Defaulters
California	CA	36.92
New York	NY	20.71
Florida	FL	17.72
Texas	TX	15.40
New Jersey	NJ	9.25



Loan Purpose is the primary reason a borrower requesting a loan.

Different Purpose of Loans: -

- **Debt consolidation -** Debt Consolidation is a term that indicates a person takes a loan to clear the previous loan.
- House improvement – Loan taken for renovation of flat/house.
- House – House or Home Loan is generally taken either to buy flat/house.
- Car – Loan purchase to buy a car
- Educational – A loan taken for education

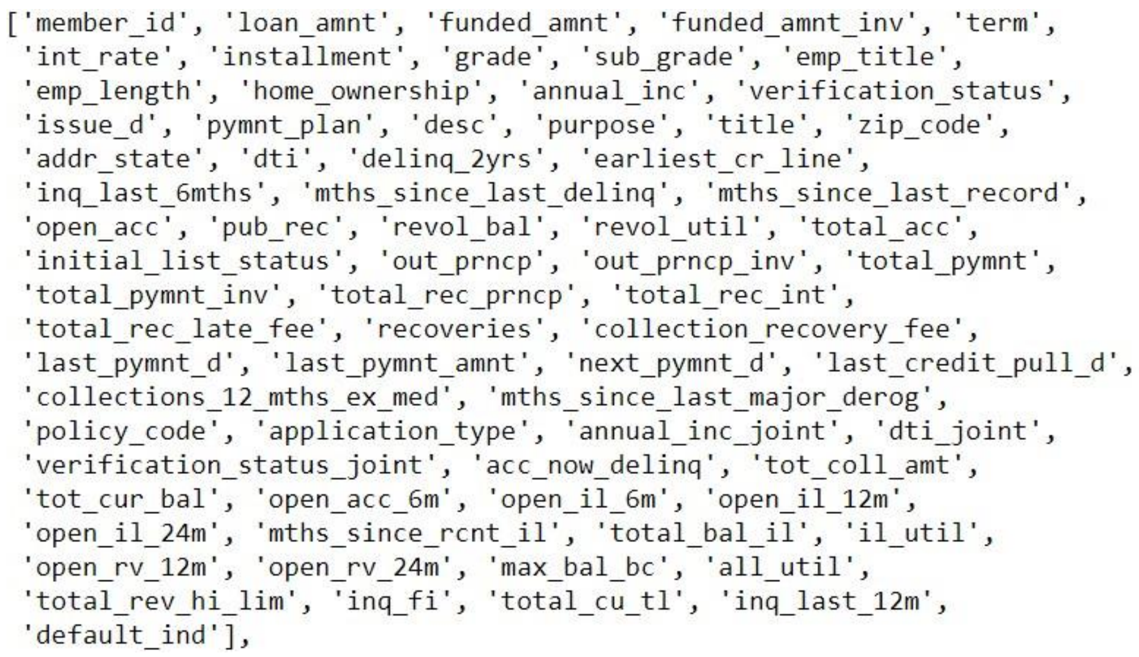
Exploratory Data Analysis

Prior to data mining model analysis, the data was **reviewed**, **cleaned** and **prepared** as follows:

1. Data Cleaning

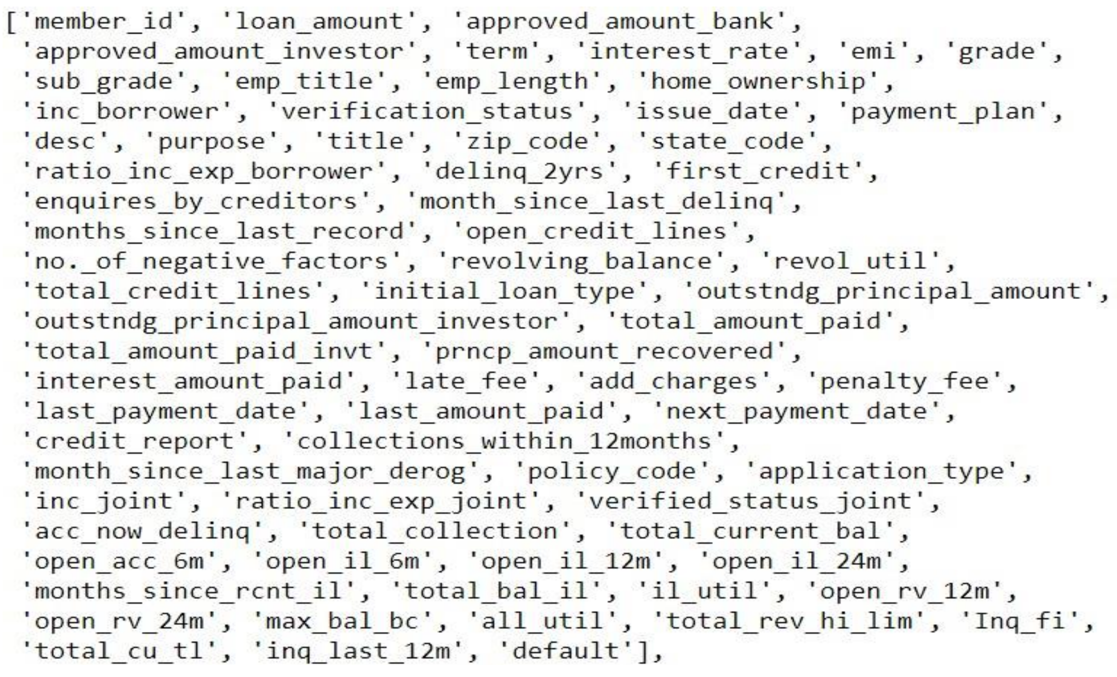
Renaming the variables

These are the **variables** given as per the dataset –



```
['member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term',
 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title',
 'emp_length', 'home_ownership', 'annual_inc', 'verification_status',
 'issue_d', 'pymnt_plan', 'desc', 'purpose', 'title', 'zip_code',
 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line',
 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since_last_record',
 '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_d', 'last_pymnt_amnt', 'next_pymnt_d', 'last_credit_pull_d',
 'collections_12_mths_ex_med', 'mths_since_last_major_derog',
 'policy_code', 'application_type', 'annual_inc_joint', 'dti_joint',
 'verification_status_joint', 'acc_now_delinq', 'tot_coll_amt',
 'tot_cur_bal', 'open_acc_6m', 'open_il_6m', 'open_il_12m',
 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il', 'il_util',
 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util',
 'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m',
 'default_ind'],
```

We renamed the **variables** as per our convenience –



```
['member_id', 'loan_amount', 'approved_amount_bank',
 'approved_amount_investor', 'term', 'interest_rate', 'emi', 'grade',
 'sub_grade', 'emp_title', 'emp_length', 'home_ownership',
 'inc_borrower', 'verification_status', 'issue_date', 'payment_plan',
 'desc', 'purpose', 'title', 'zip_code', 'state_code',
 'ratio_inc_exp_borrower', 'delinq_2yrs', 'first_credit',
 'enquires_by_creditors', 'month_since_last_delinq',
 'months_since_last_record', 'open_credit_lines',
 'no._of_negative_factors', 'revolving_balance', 'revol_util',
 'total_credit_lines', 'initial_loan_type', 'outstndg_principal_amount',
 'outstndg_principal_amount_investor', 'total_amount_paid',
 'total_amount_paid_invt', 'prncp_amount_recovered',
 'interest_amount_paid', 'late_fee', 'add_charges', 'penalty_fee',
 'last_payment_date', 'last_amount_paid', 'next_payment_date',
 'credit_report', 'collections_within_12months',
 'month_since_last_major_derog', 'policy_code', 'application_type',
 'inc_joint', 'ratio_inc_exp_joint', 'verified_status_joint',
 'acc_now_delinq', 'total_collection', 'total_current_bal',
 'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m',
 'months_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m',
 'open_rv_24m', 'max_bal_bc', 'all_util', 'total_rev_hi_lim', 'Inq_fi',
 'total_cu_tl', 'inq_last_12m', 'default'],
```

Extracting the Values

Extracted the values of variables like **emp_length**, **term**.

emp_length had values in string where we have extracted the numeric part.

E.g.: - '10+ years' -> 10

'2 years' -> 2

'1 year' -> 1

'<1 year' -> 0

Did the same for the term where values were in string.

E.g.: - 36 months -> 36

60 months -> 60

Replacing the values from one variable to another variable

Replacing the **income value** of **borrower** with **co-borrower** on basis of **application type.**

If application_type is **INDIVIDUAL** then income of borrower will remain as it is and if application_type is **JOINT** then the existing value of income of borrower is replaced with income of joint. whereas income of joint is the combined income value of borrower and co-borrower. The same procedure is for **dti/ratio_inc_exp_borrower** and **dti_joint/ratio_inc_exp_joint.**

Replacing the values in Home Ownership variable

Replacing 'OTHER', 'ANY' & "NONE' values with "**OTHER**’

2. Pre-Processing

Dropping Variables

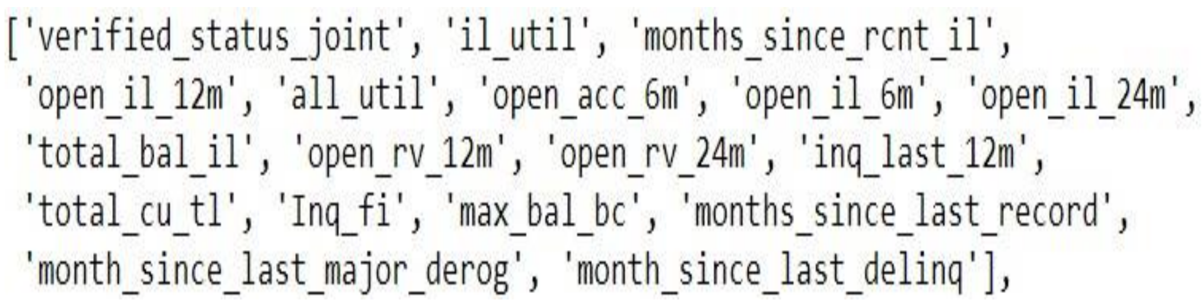
Dropped the variables which were not needed for building models

member_id -> every value is unique

emp_title, desc, zip_code, title, sub_grade

policy_code -> every value is 1

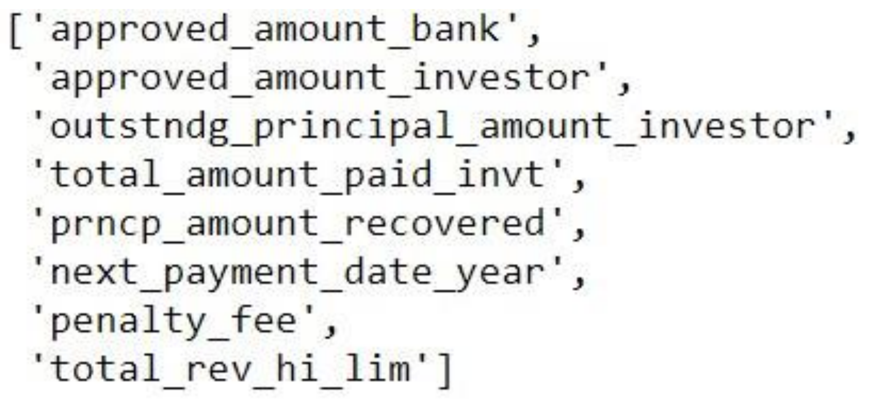
Dropped those variables which contain more than 50% of null values



```
['verified_status_joint', 'il_util', 'months_since_rcnt_il',
 'open_il_12m', 'all_util', 'open_acc_6m', 'open_il_6m', 'open_il_24m',
 'total_bal_il', 'open_rv_12m', 'open_rv_24m', 'inq_last_12m',
 'total_cu_tl', 'Inq_fi', 'max_bal_bc', 'months_since_last_record',
 'month_since_last_major_derog', 'month_since_last_delinq'],
```

Dropped those variables which having multicollinearity greater than 0.7

Multicollinearity – linear relationship between independent variables

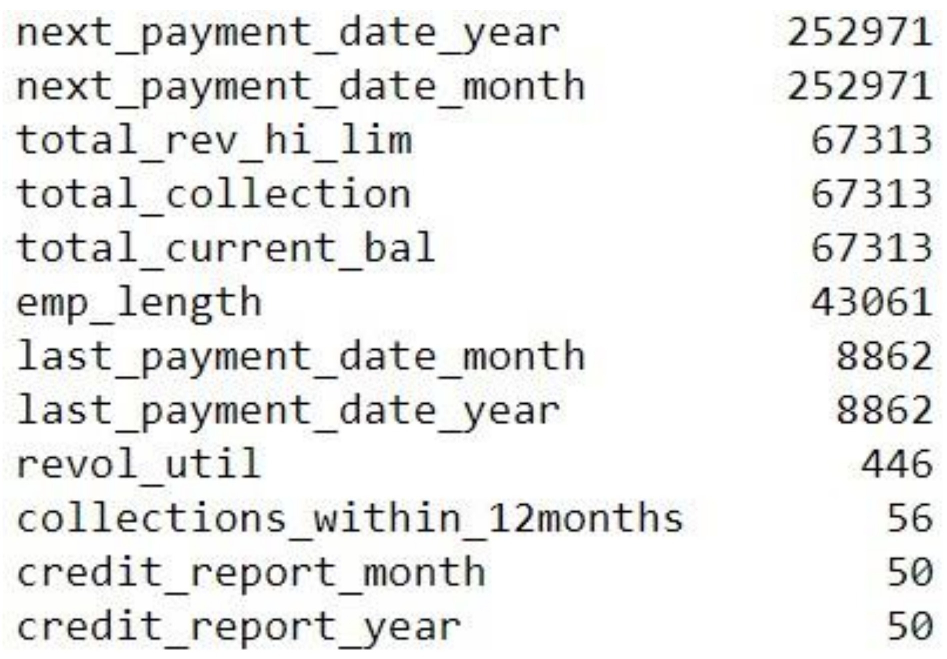


```
['approved_amount_bank',
 'approved_amount_investor',
 'outstndg_principal_amount_investor',
 'total_amount_paid_invt',
 'prncp_amount_recovered',
 'next_payment_date_year',
 'penalty_fee',
 'total_rev_hi_lim']
```

Emi, loan_amount, approved_amount_bank, approved_amount_investor these were the variables which were **highly corelated** to each other.

From these 4 variables, we kept **emi** and **loan_amount**.

Imputation of Missing values

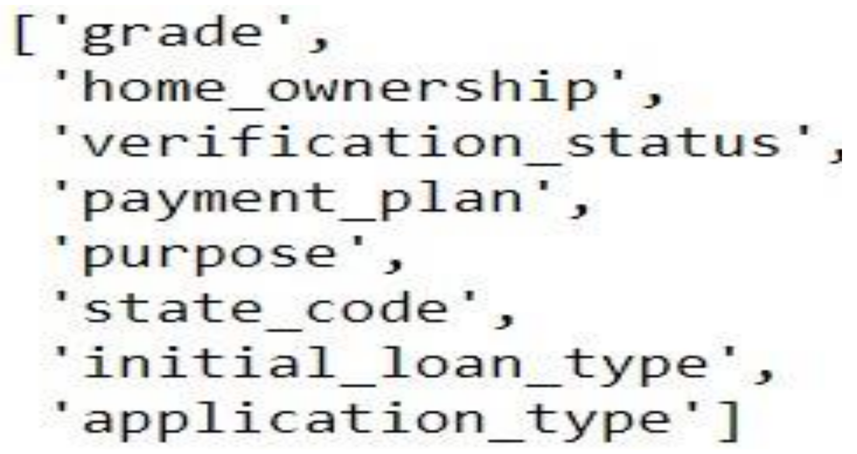


```
next_payment_date_year        252971
next_payment_date_month       252971
total_rev_hi_lim               67313
total_collection               67313
total_current_bal              67313
emp_length                     43061
last_payment_date_month         8862
last_payment_date_year          8862
revol_util                       446
collections_within_12months       56
credit_report_month               50
credit_report_year                50
```

Imputed the above **missing** data with **zero** values.

Label Encoder

Label Encoder is a technique used to convert the **categorical** variables to **numeric** variables.



```
['grade',
 'home_ownership',
 'verification_status',
 'payment_plan',
 'purpose',
 'state_code',
 'initial_loan_type',
 'application_type']
```

The above variables are categorical and we have applied Label encoder technique. In this technique, we have two function **fit** and **transform**.

Steps of Label Encoder:

1. Fetching the unique values
2. Arranging them in ascending order
3. Mapping the values starting with 0,1,2 and so on.

Fit function will perform all the above three steps.

Transform function will actually replace the data values in the dataframe.

Train Test Split

Splitting the data into train and test on **issue_date** variable.

The **Training** part contains data from **June 2007** to **May 2015**.

- Train Size **–** **598978** records with **38** features

Testing part is from **June 2015** to **December 2015.**

- Test Size **–** **256991** records with **38** features

Scaling the data

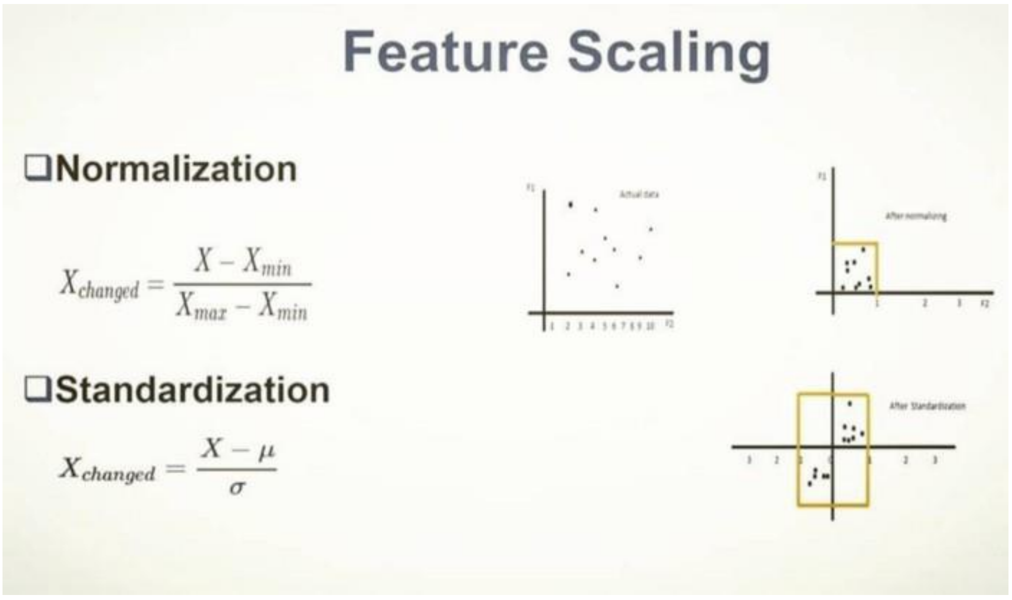
Feature **Scaling** is a technique to **standardize**/**normalize** the independent features present in the **data** in an approximate range.

Scaling is used to ensure uniformity across the data.

There are two methods of Scaling:

- Standardization: In standardization, approximate range is **-3 to +3**
- Normalization: In normalization, fixed range is **0 to 1**

In this dataset, we have used standardization technique to scale the data.



Feature Scaling

❑Normalization

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

❑Standardization

$$X_{changed} = \frac{X - \mu}{\sigma}$$

Predictive Modelling and Evaluation

Since the **target/response** variable output is in Binary form i.e. 0 and 1 so this is a **classification** problem and we have used the below mentioned techniques:

- **Logistic Regression**
- **Decision Tree**
- **Bagging**
- **AdaBoost**
- **ANN (Artificial Neural Network)**
- **Voting Classifier**

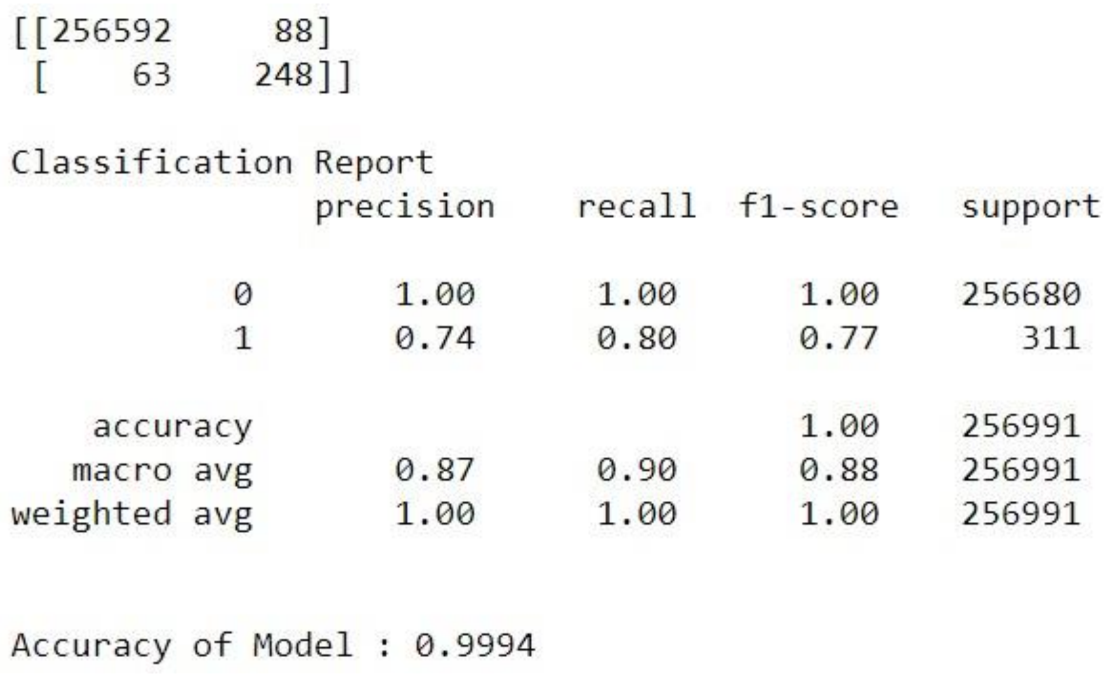
Metrics used for Evaluation of Models:

- **Confusion matrix**
- **Recall Score** – Accuracy of Individual classes
- **Precision Score** – How relevant are the results?
- **F1 Score** – Harmonic mean of precision & recall value
- **AUC – ROC Curve**
- **Accuracy Score**

LOGISTIC REGRESSION

Logistic regression is a statistical model that in its basic form uses a **logit function** to model a binary dependent variable.

<u>**Prediction on Testing Data**</u>



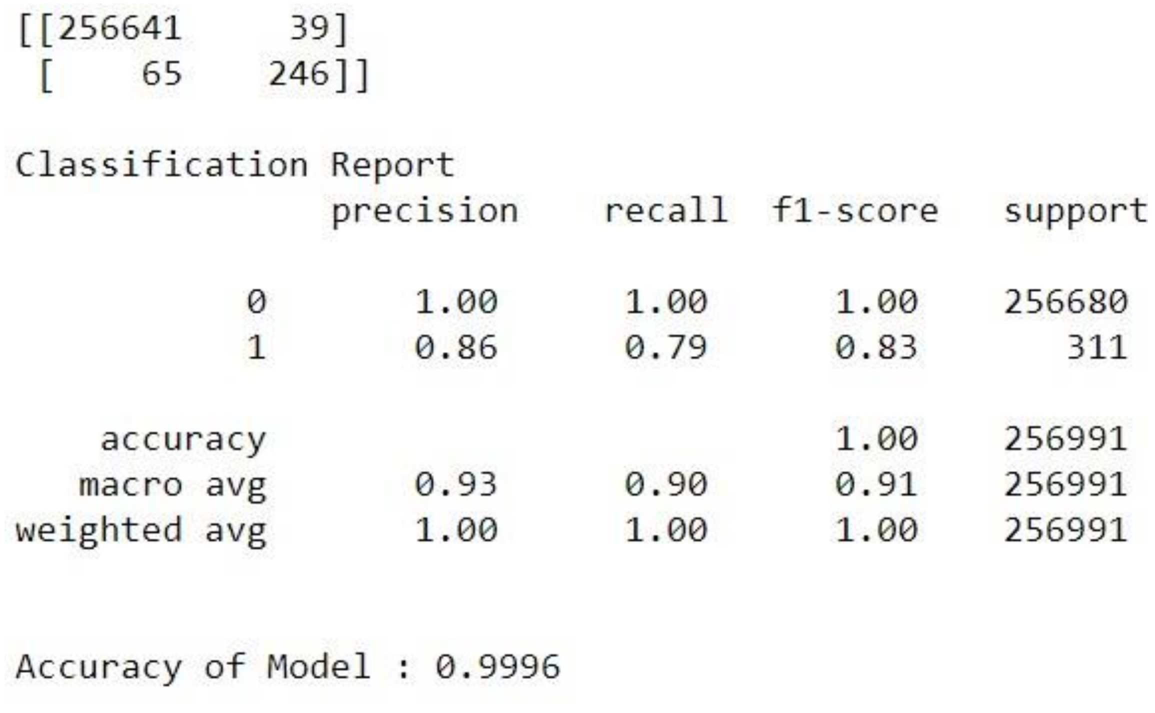
```
[[256592     88]
 [    63    248]]

Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.74      0.80      0.77       311

    accuracy                           1.00    256991
   macro avg       0.87      0.90      0.88    256991
weighted avg       1.00      1.00      1.00    256991


Accuracy of Model : 0.9994
```

In Logistic Regression, threshold value is 50% i.e. **Equal Weightage** to both the classes.

Type I error is 88 and Type II error is 63.

After **adjusting** the threshold value to 70% i.e. 70% weightage to class 0 and 30% weightage to class 1.

<u>Prediction on Testing Data</u>



```
[[256641     39]
 [    65    246]]

Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.86      0.79      0.83       311

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


Accuracy of Model : 0.9996
```

We can see drastic change in Type I error which is **decreased** to 39 from 88,

Precision Score has increase from 0.74 to 0.88

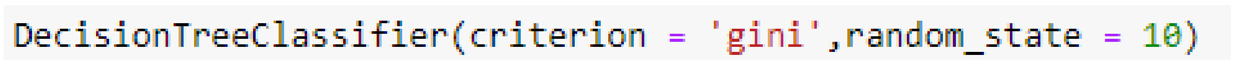
F1- Score has also been increased from 0.77 to 0.83

DECISION TREE

Decision Tree is a Supervised learning technique that can be used for both **Classification** and **Regression** problems, but mostly it is preferred for solving **Classification** problems. There are three types of nodes: **Root** Node, **Decision** Node & **Leaf** Node.

After using Logistic Regression and even adjusting the threshold value the Type I error is 39 and Type II error is 65.

To improve on both the errors, we decided to implement Decision Tree.



```
DecisionTreeClassifier(criterion = 'gini',random_state = 10)
```

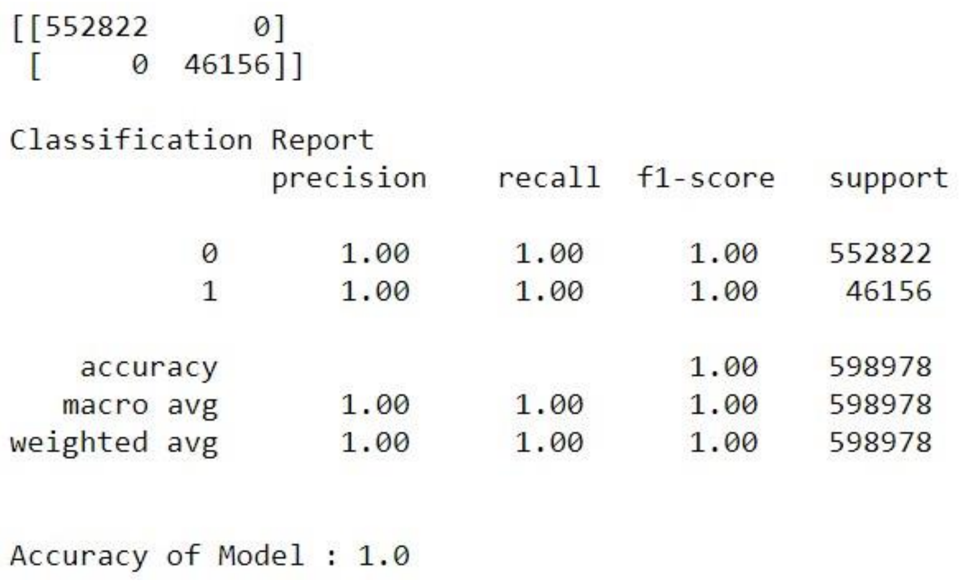
The hyperparameters used for building Decision Tree are criterion and random_state.

criterion: {"gini", "entropy"}, default="gini". The function to measure the quality of a split.

Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

random_state: use to set the seed value.

Prediction on Training Data

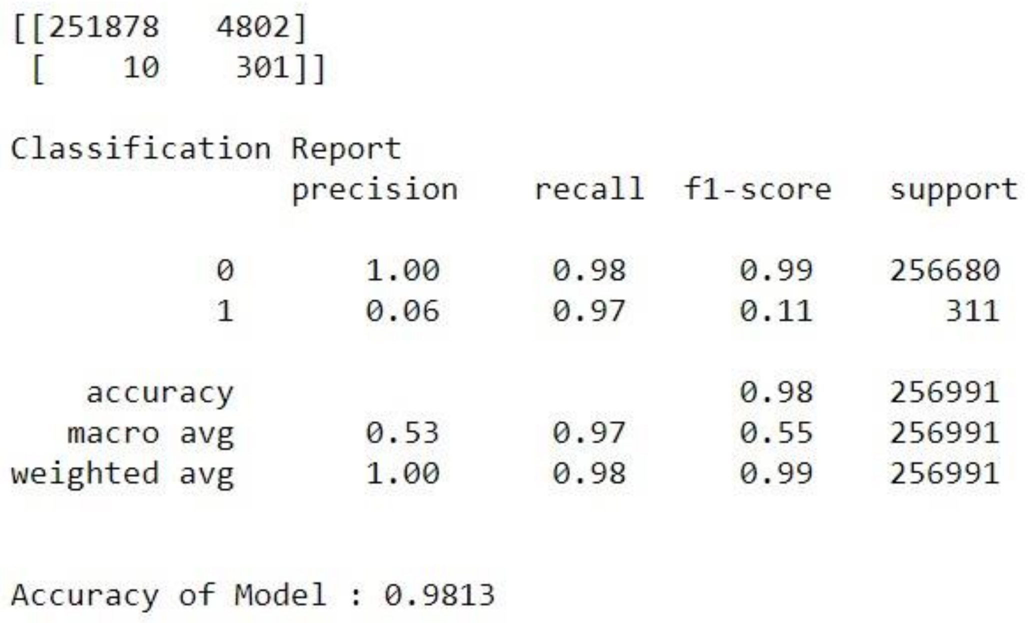


```
[[552822      0]
 [     0  46156]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    552822
           1       1.00      1.00      1.00     46156

    accuracy                           1.00    598978
   macro avg       1.00      1.00      1.00    598978
weighted avg       1.00      1.00      1.00    598978


Accuracy of Model : 1.0
```

While predicting on Training data, Type I error & Type II error is 0.

The Precision score, Recall score & F1-Score for each class is 1.00 (100%).

Prediction on Testing Data



```
[[251878   4802]
 [    10    301]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    256680
           1       0.06      0.97      0.11       311

    accuracy                           0.98    256991
   macro avg       0.53      0.97      0.55    256991
weighted avg       1.00      0.98      0.99    256991


Accuracy of Model : 0.9813
```

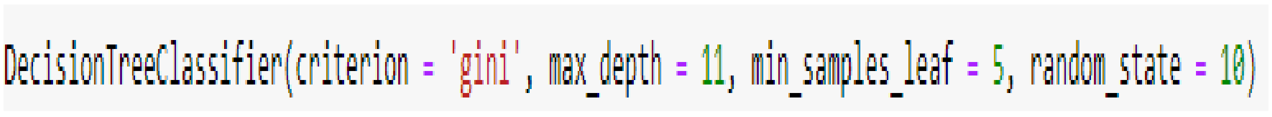
Above Confusion matrix shows Type I error is 4802 & Type II error is 10.

The Precision score is 0.06, Recall Score is 0.97 & F1 – Score is 0.11 for class 1.

Looking at Prediction on Training and Testing data, we can clearly say that the model is **overfitted**. (based on Precision and F1- Score of class 1)

TUNED DECISION TREE

Decision Tree with some additional **hyperparameters** used for tuning.

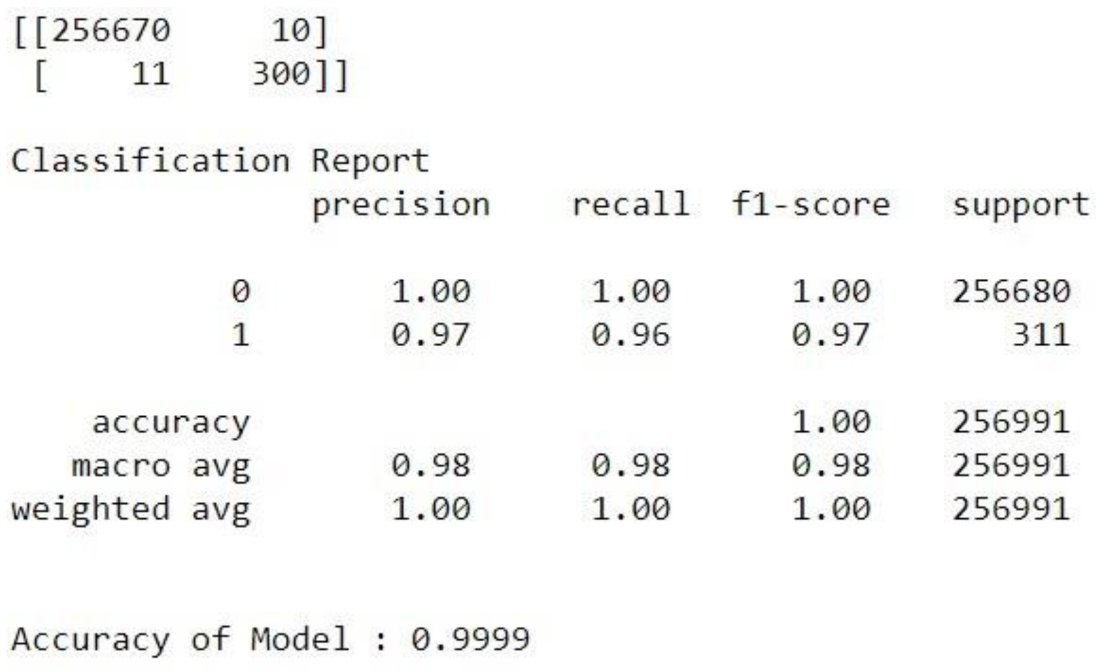


```
DecisionTreeClassifier(criterion = 'gini', max_depth = 11, min_samples_leaf = 5, random_state = 10)
```

max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.

Prediction on Testing Data



```
[[256670     10]
 [    11    300]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       0.97      0.96      0.97       311

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


Accuracy of Model : 0.9999
```

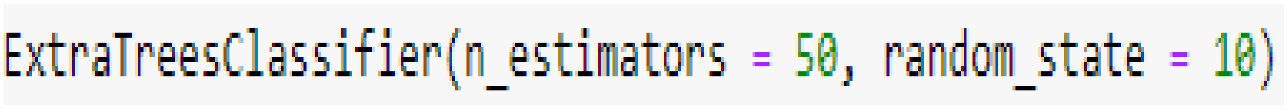
After **tunning** the Decision Tree on Testing data, Type I error has shown drastic change from 4802 to 10.

Precision Score has increase to 0.97 from 0.06.

F1 Score has increase to 0.97 from 0.11.

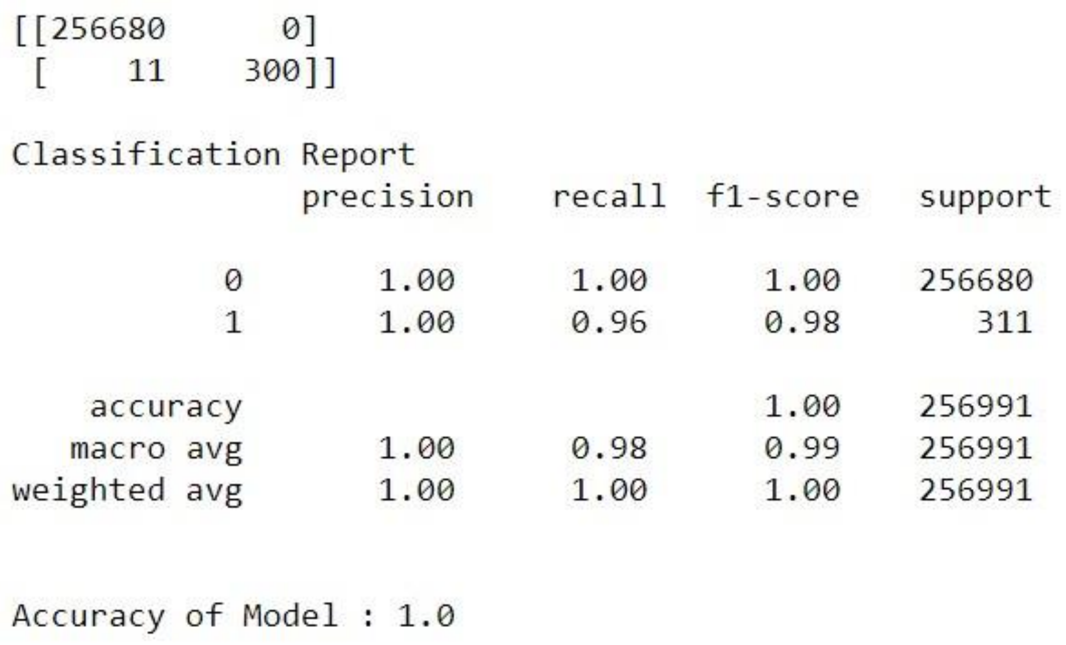
BAGGING

Bootstrap aggregating also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also **reduces variance** and helps to **avoid overfitting**.



```
ExtraTreesClassifier(n_estimators = 50, random_state = 10)
```

n_estimators: The number of trees in the forest.



```
[[256680      0]
 [    11    300]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       1.00      0.96      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 Model : 1.0
```

From the above confusion matrix, we can say there is no Type I error.

Precision Score for class 1 is 1.00 (100%)

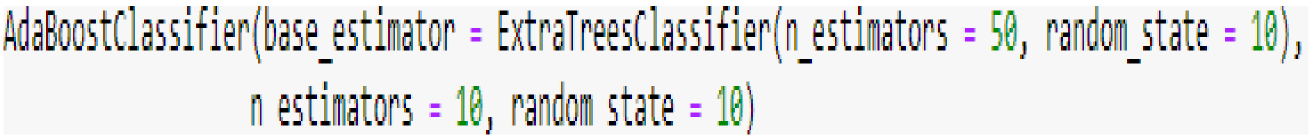
Recall Score for class 1 is 0.96 (96%)

ADABOOST (ADAPTIVE BOOSTING)

AdaBoost is an **ensemble** learning method (also known as “meta-learning”) which was initially created to increase the efficiency of binary classifiers. AdaBoost uses an iterative approach to learn from the mistakes of **weak classifiers**, and turn them into **strong ones**.

After using Logistic Regression, Decision Tree, Tunned Decision Tree, Bagging we have reached to the stage where there is no Type I error and Type II error is 11.

By using Adaboost technique, we will try reducing the Type II error.



```
AdaBoostClassifier(base_estimator = ExtraTreesClassifier(n_estimators = 50, random_state = 10),
                   n estimators = 10, random state = 10)
```

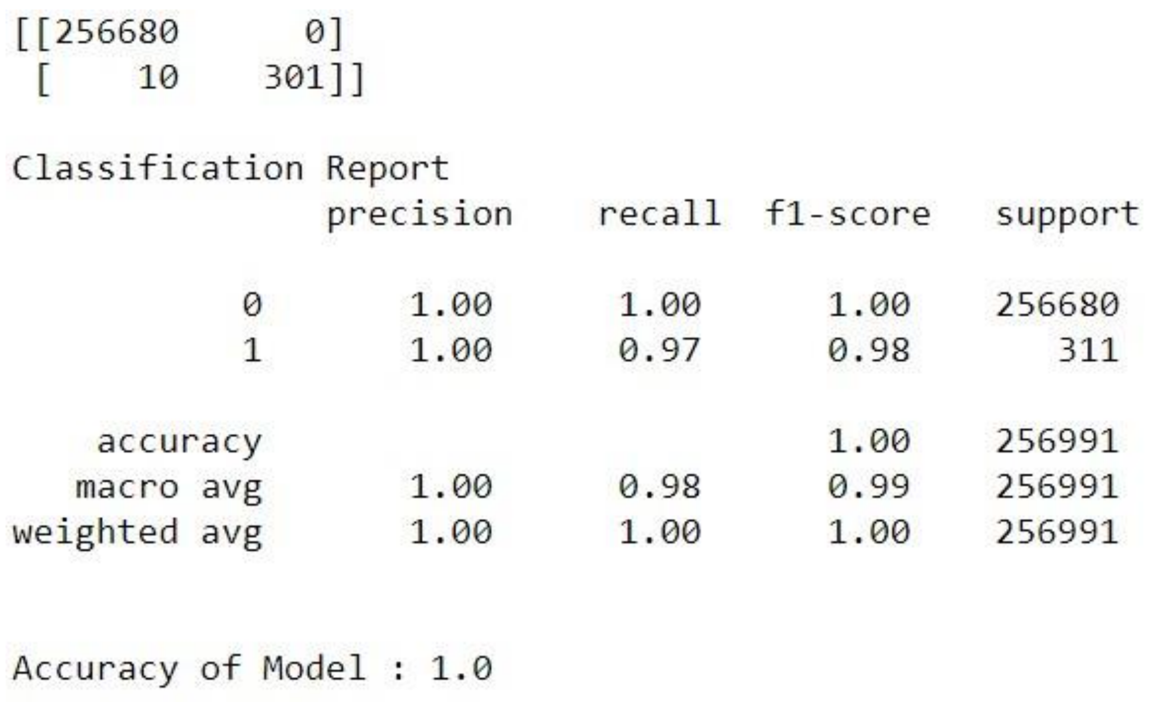
Here,

base_estimator: The base estimator from which the boosted ensemble is built.

n_estimator = 50: The number of trees in the forest. (used in bagging algorithm)

n_estimator = 10: Maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. (used for adaboost)

Prediction on Testing Data



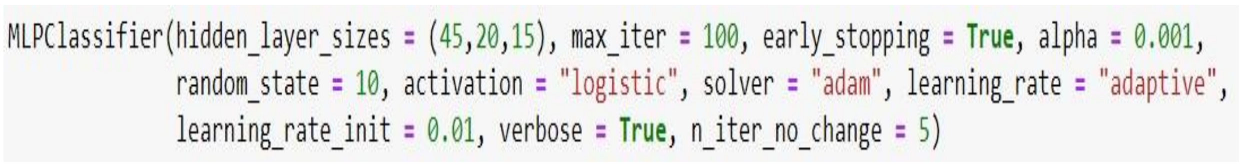
```
[[256680      0]
 [    10    301]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       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 Model : 1.0
```

Type II error is reduced by 1 compared to bagging.

Precision Score remains the same for the class 1 which is 1.00 (100%).

ARTIFICIAL NEURAL NETWORKS (ANN)

A **multilayer perceptron (MLP)** is a class of feedforward **artificial neural network (ANN)**. MLP utilizes a **supervised** learning technique called **backpropagation** for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.



```
MLPClassifier(hidden_layer_sizes = (45,20,15), max_iter = 100, early_stopping = True, alpha = 0.001,
              random_state = 10, activation = "logistic", solver = "adam", learning_rate = "adaptive",
              learning_rate_init = 0.01, verbose = True, n_iter_no_change = 5)
```

Let’s discuss the hyperparameters used for building MLP Classifier Model.

hidden_layer_sizes: The ith element represents the number of neurons in the ith hidden layer.

activation: Activation function for the hidden layer.

'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).

solver: The solver for weight optimization.

- 'adam' refers to a stochastic gradient-based optimizer proposed

alpha: L2 penalty (regularization term) parameter.

learning_rate: default='constant'. Learning rate schedule for weight updates.

'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by at least tol, or fail to increase validation score by at least tol if 'early_stopping' is on, the current learning rate is divided by 5.

max_iter: default=200 Maximum number of iterations. The solver iterates until convergence or this number of iterations

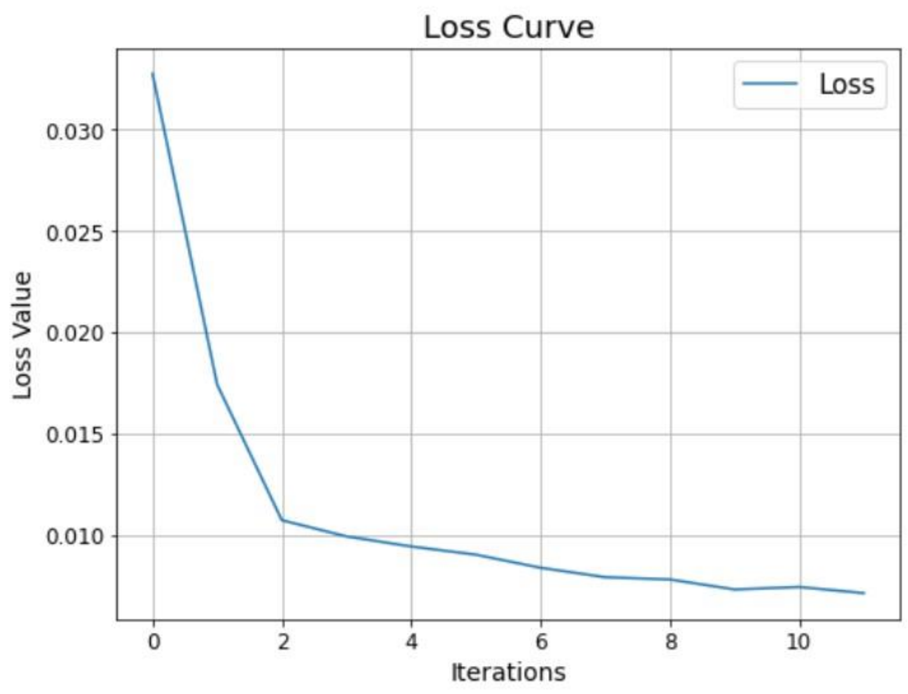
verbose: bool, default=False. Whether to print progress messages to standard output.

early_stopping: bool, default=False

Whether to use early stopping to terminate training when validation score is not improving. If set to true, it will automatically set aside 10% of training data as validation and terminate training when validation score is not improving by at least tol for `n_iter_no_change` consecutive epochs.

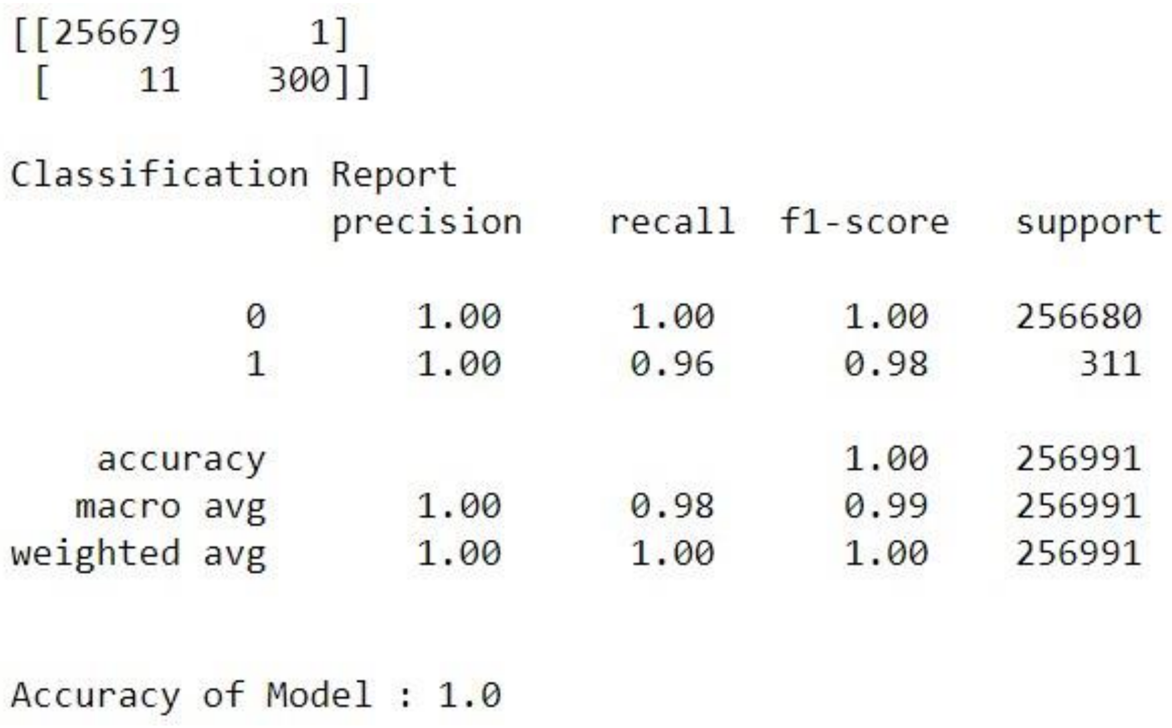
n_iter_no_change: int, default=10 Maximum number of epochs to not meet `tol` improvement. Only effective when solver='sgd' or 'adam'.

learning_rate_init: double, default=0.001. The initial learning rate used. It controls the step-size in updating the weights. Only used when solver='sgd' or 'adam'.



From the above graph, we can clearly see that from the 0th to 2nd iteration there is a steep decrease in the loss value.

Prediction on Testing Data



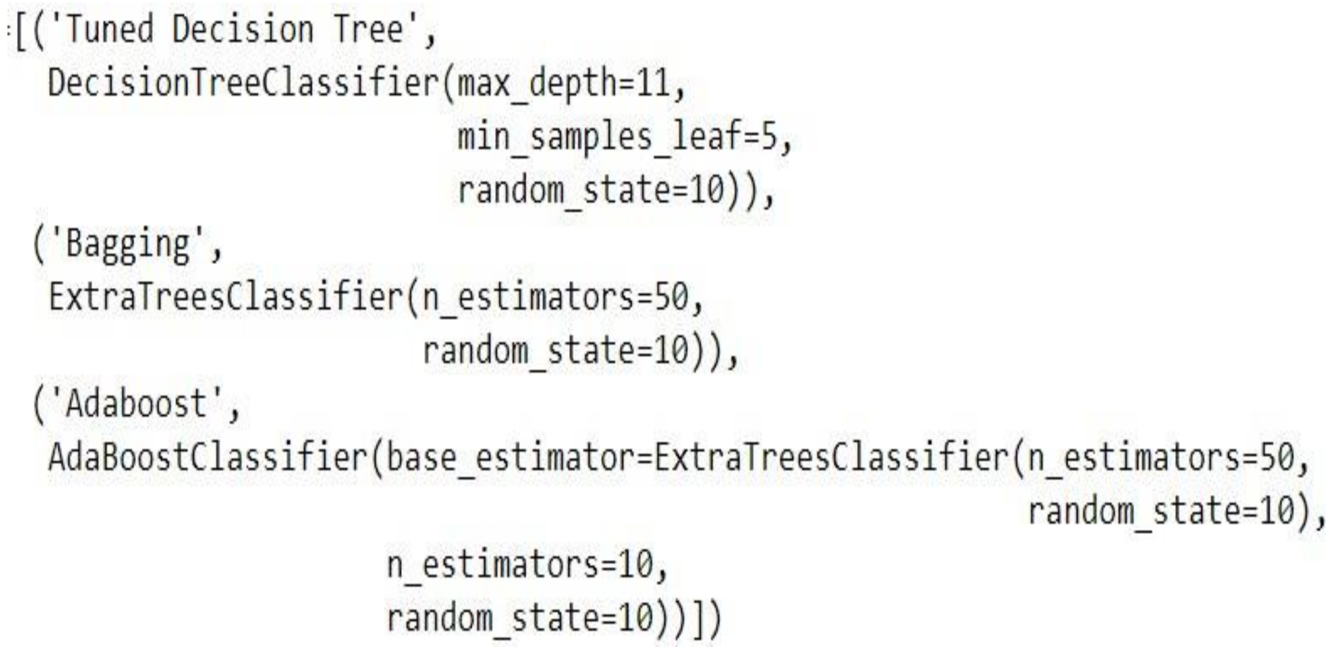
```
[[256679      1]
 [    11    300]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       1.00      0.96      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 Model : 1.0
```

VOTING CLASSIFIER

The idea is instead of creating **separate dedicated** models and finding the **accuracy** for each them, we create a single model which trains by these models and predicts output based on their **combined majority** of voting for each output class.

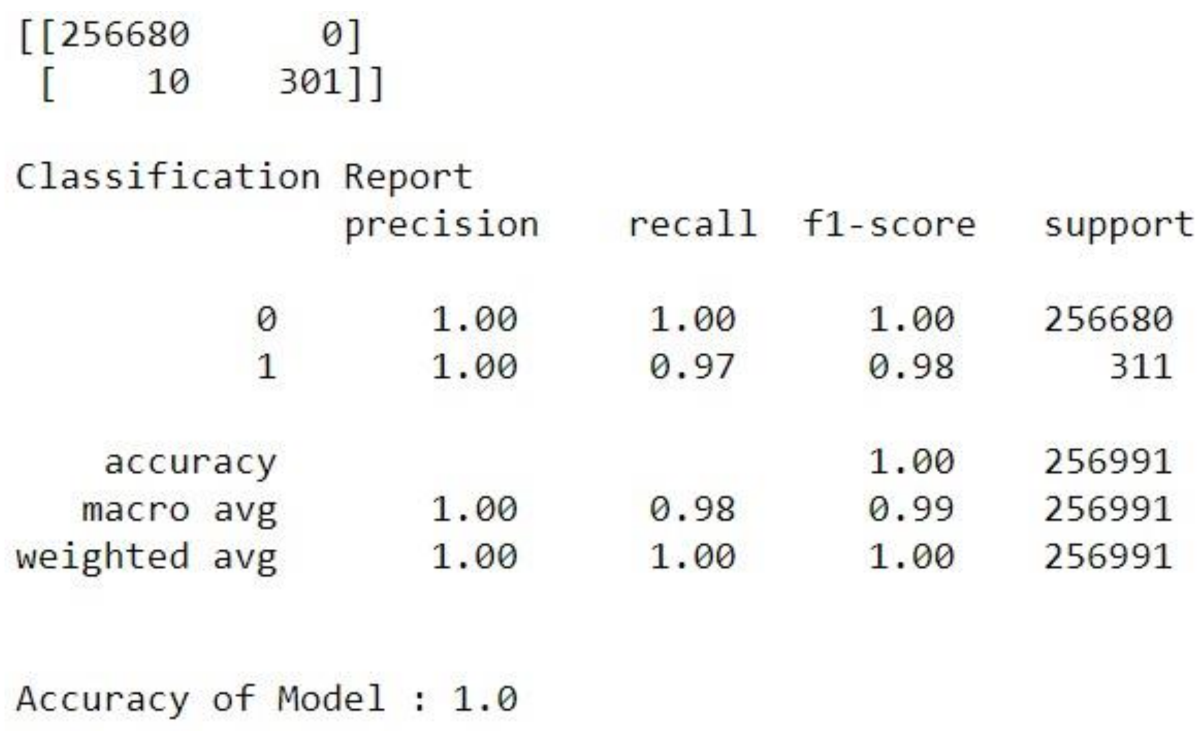
We have used 3 estimators in voting classifier.

- **Tuned Decision Tree**
- **Bagging**
- **AdaBoost**



```
[('Tuned Decision Tree',
  DecisionTreeClassifier(max_depth=11,
                         min_samples_leaf=5,
                         random_state=10)),
 ('Bagging',
  ExtraTreesClassifier(n_estimators=50,
                       random_state=10)),
 ('Adaboost',
  AdaBoostClassifier(base_estimator=ExtraTreesClassifier(n_estimators=50,
                                                         random_state=10),
                     n_estimators=10,
                     random_state=10))])
```

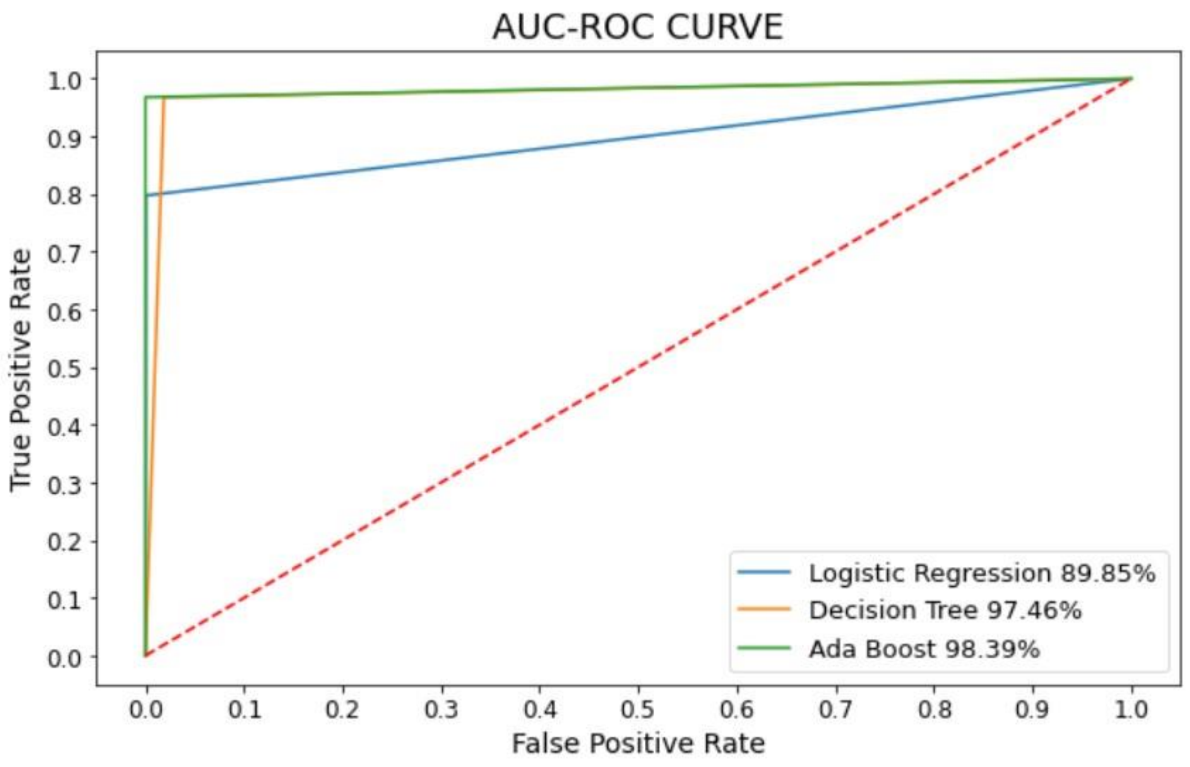
Prediction on Testing Data



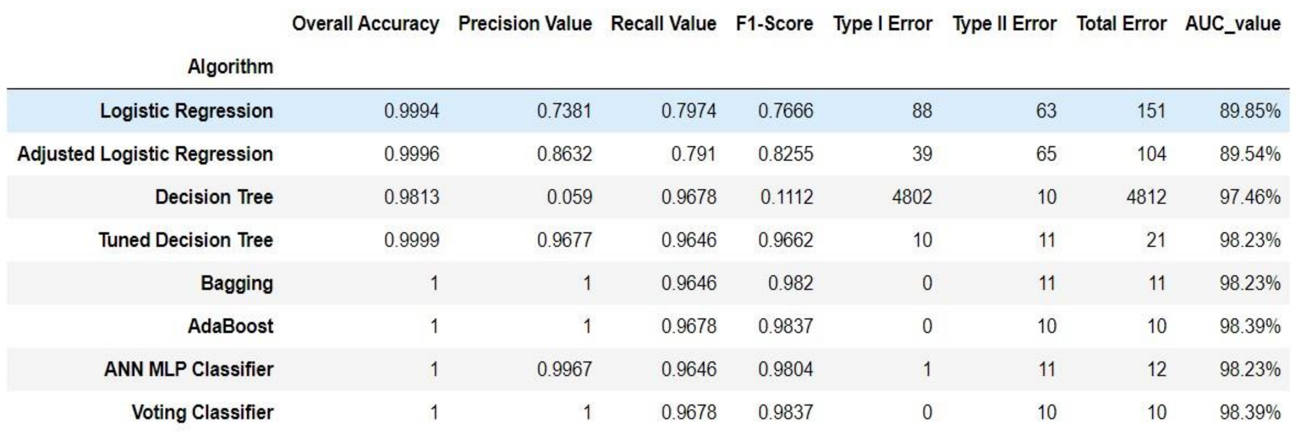
```
[[256680      0]
 [    10    301]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    256680
           1       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 Model : 1.0
```

AUC ROC CURVE

It is one of the **evaluation metrics** used for checking the classification models performance. It tells how much model is capable of **distinguishing** between classes. By analogy, **Higher** the **AUC**, better the model is at distinguishing between customers as **Defaulters** and **Not Defaulters.**



MODEL COMPARISON



Algorithm	Overall Accuracy	Precision Value	Recall Value	F1-Score	Type I Error	Type II Error	Total Error	AUC_value
Logistic Regression	0.9994	0.7381	0.7974	0.7666	88	63	151	89.85%
Adjusted Logistic Regression	0.9996	0.8632	0.791	0.8255	39	65	104	89.54%
Decision Tree	0.9813	0.059	0.9678	0.1112	4802	10	4812	97.46%
Tuned Decision Tree	0.9999	0.9677	0.9646	0.9662	10	11	21	98.23%
Bagging	1	1	0.9646	0.982	0	11	11	98.23%
AdaBoost	1	1	0.9678	0.9837	0	10	10	98.39%
ANN MLP Classifier	1	0.9967	0.9646	0.9804	1	11	12	98.23%
Voting Classifier	1	1	0.9678	0.9837	0	10	10	98.39%

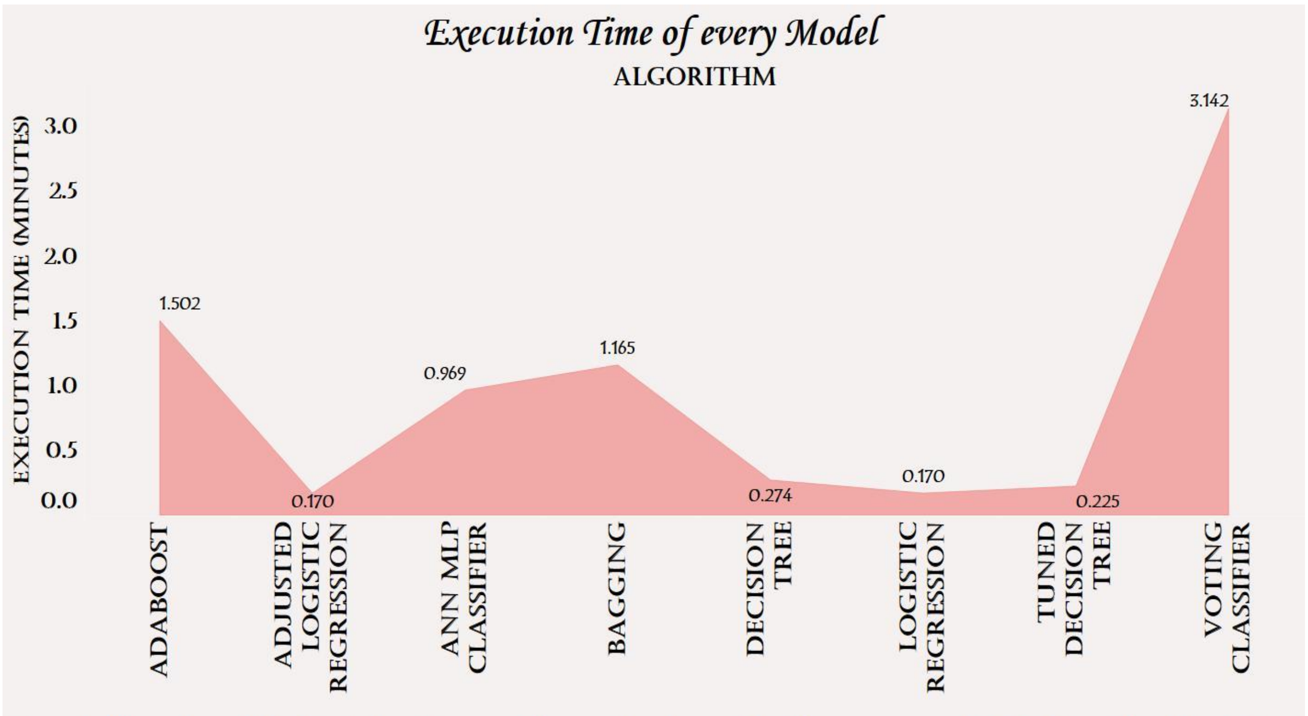
In the above table, **Precision Value, Recall Value/ Sensitivity & F1 – Score** is for the **class 1**.

To reduce the Type I and Type II errors, further we implemented various models like Tuned Decision Tree, Bagging Algorithm, AdaBoost and so on.

- In Tuned Decision Tree model, which reduced the Type I error to 10, but increased the Type II error to 11.
- In Bagging Algorithm, which completely eliminated Type I Error and Type II error is same i.e. 11.
- In AdaBoost, there is a further reduce in Type II error to 10.

Optimal output for **Adaboost** and **Voting Classifier** are the **same.**

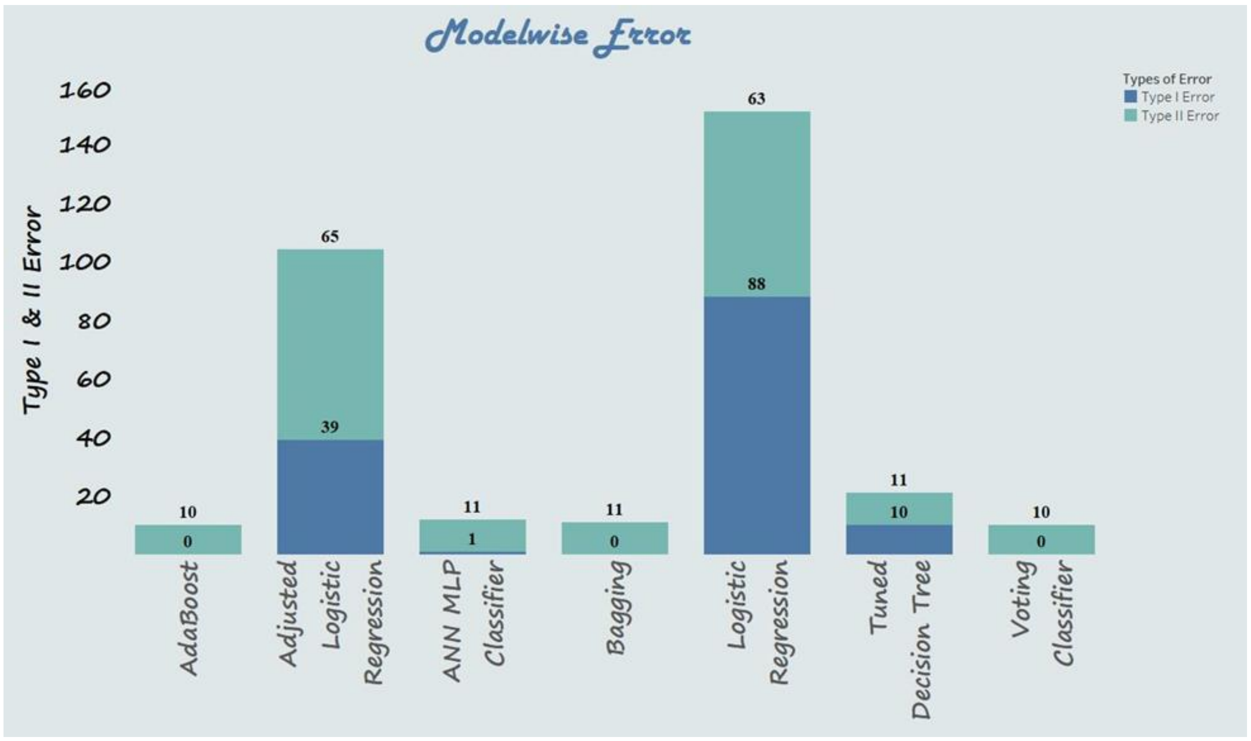
Conclusion



Since Adaboost and Voting Classifier are the optimal models, We can see that Voting Classifier takes double time to build than AdaBoost.

After using various machine learning algorithms, we have arrived on the conclusion that AdaBoost and Voting Classifier have performed well on the given dataset compared to rest of the algorithms.

Now taking **execution time** into account, we have chosen **AdaBoost** over Voting Classifier, as there is a high execution time involved in computing the Voting Classifier Model.



Using Adaboost on **Testing** data, Precision value and Recall value is 1.00.

There is no Type I error and Type II error is 10.

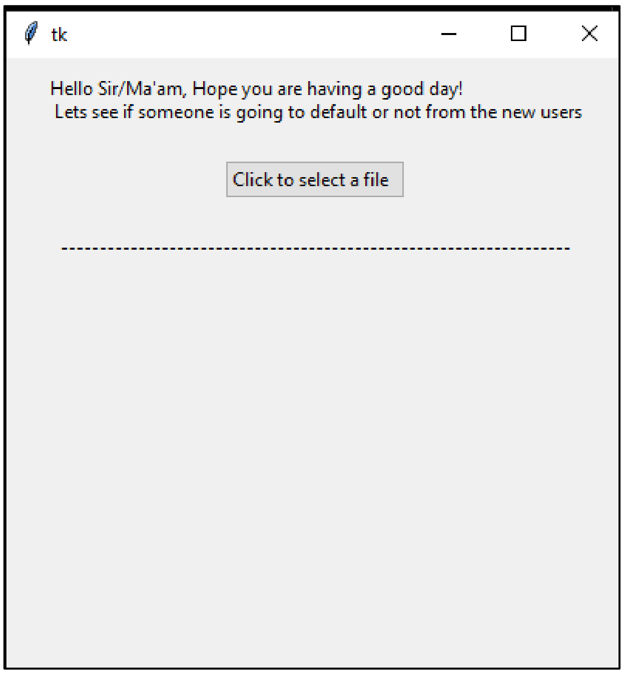
Using Adaboost on **Training** data, Precision value and Recall value is 1.00.

There is no Type I error as well as Type II error.

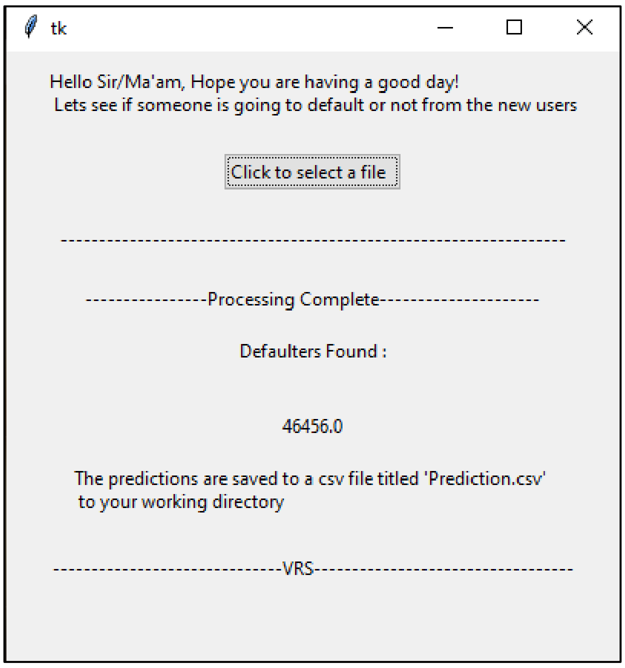
By comparing results of Testing and Training Data, we can say that model is not **overfitted**. Looking at Recall, Precision and Accuracy score, we would recommend the financial institutions to use **AdaBoost Classifier** to perform **default prediction** in future.

Basic User Interface for the Client

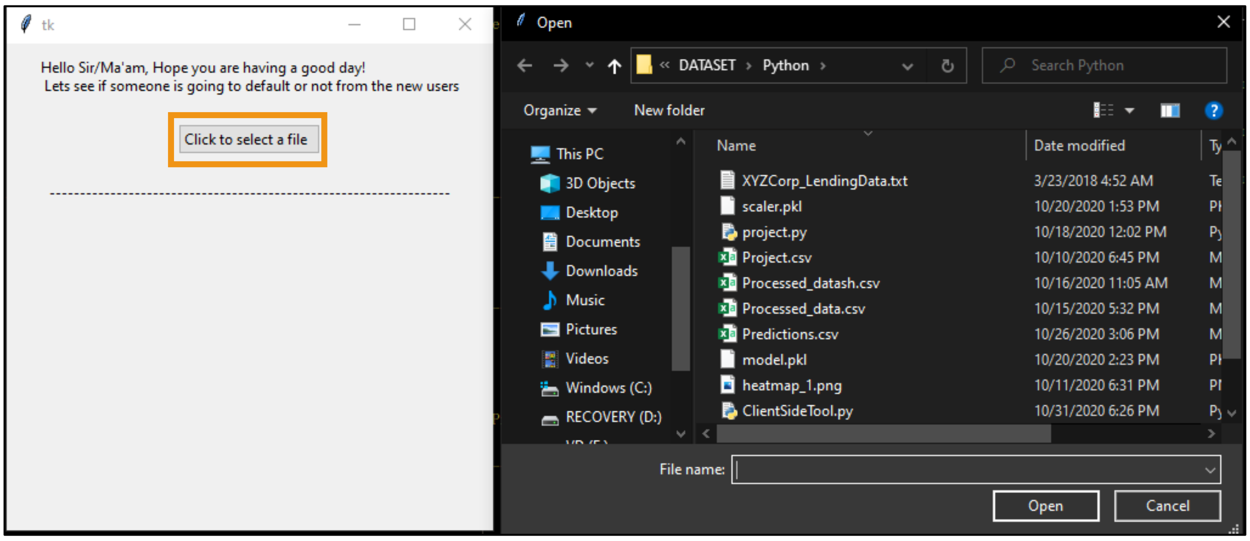
Designed a User Interface using TKinter for the client to implement the model for future predictions. The interface lets the user pass a CSV file of the data on which the predictions are to be done. The interface processes the data and returns the total number of defaulters found and saves the predictions to a CSV file in the default directory.



```
tk

Hello Sir/Ma'am, Hope you are having a good day!
 Lets see if someone is going to default or not from the new users

Click to select a file

------------------------------------------------------------------
```



```
tk

Hello Sir/Ma'am, Hope you are having a good day!
 Lets see if someone is going to default or not from the new users

Click to select a file

------------------------------------------------------------------

---------------Processing Complete--------------------

Defaulters Found :

46456.0

The predictions are saved to a csv file titled 'Prediction.csv'
 to your working directory

----------------------------VRS--------------------------------
```

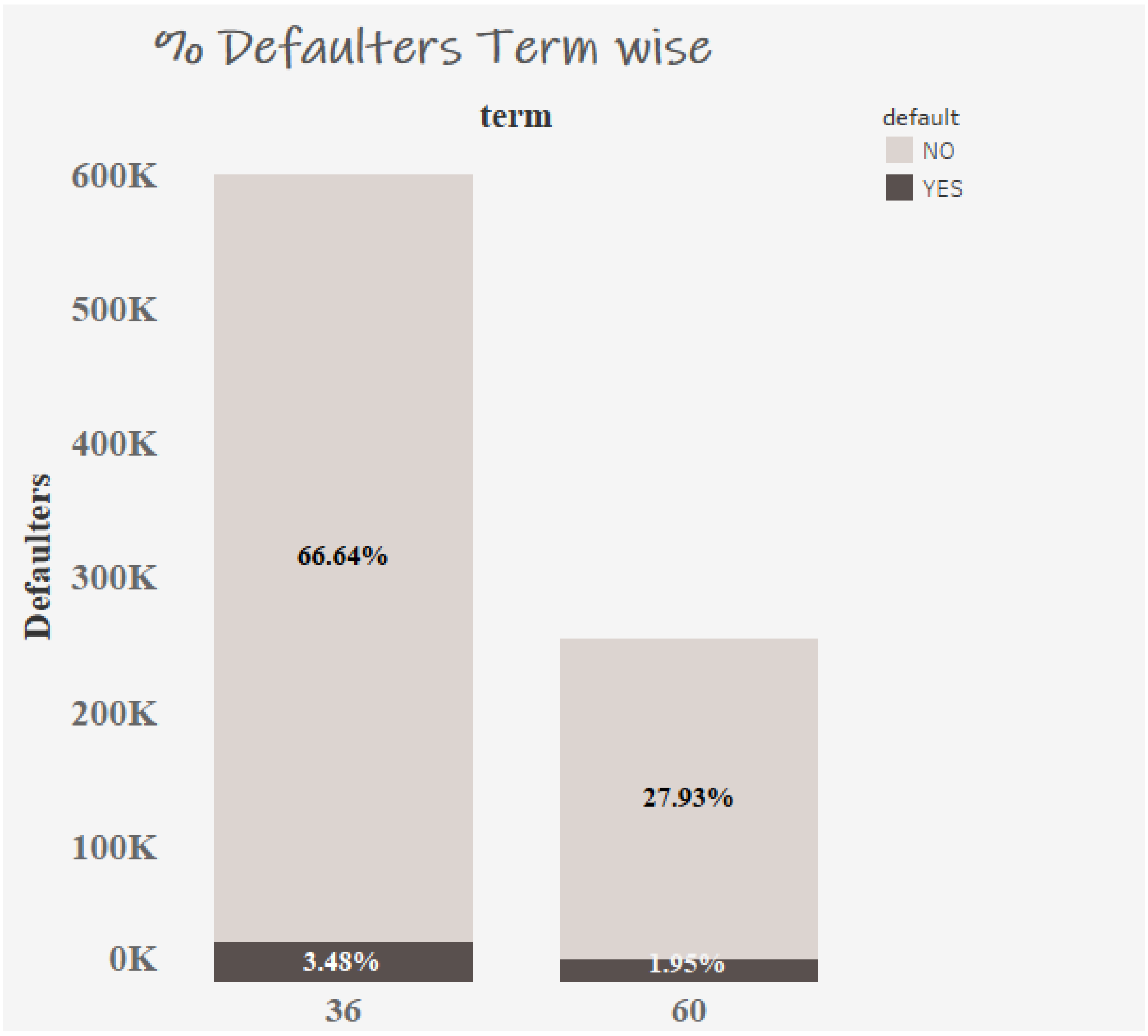


```
tk

Hello Sir/Ma'am, Hope you are having a good day!
 Lets see if someone is going to default or not from the new users

Click to select a file

----------------------------------------------------
```

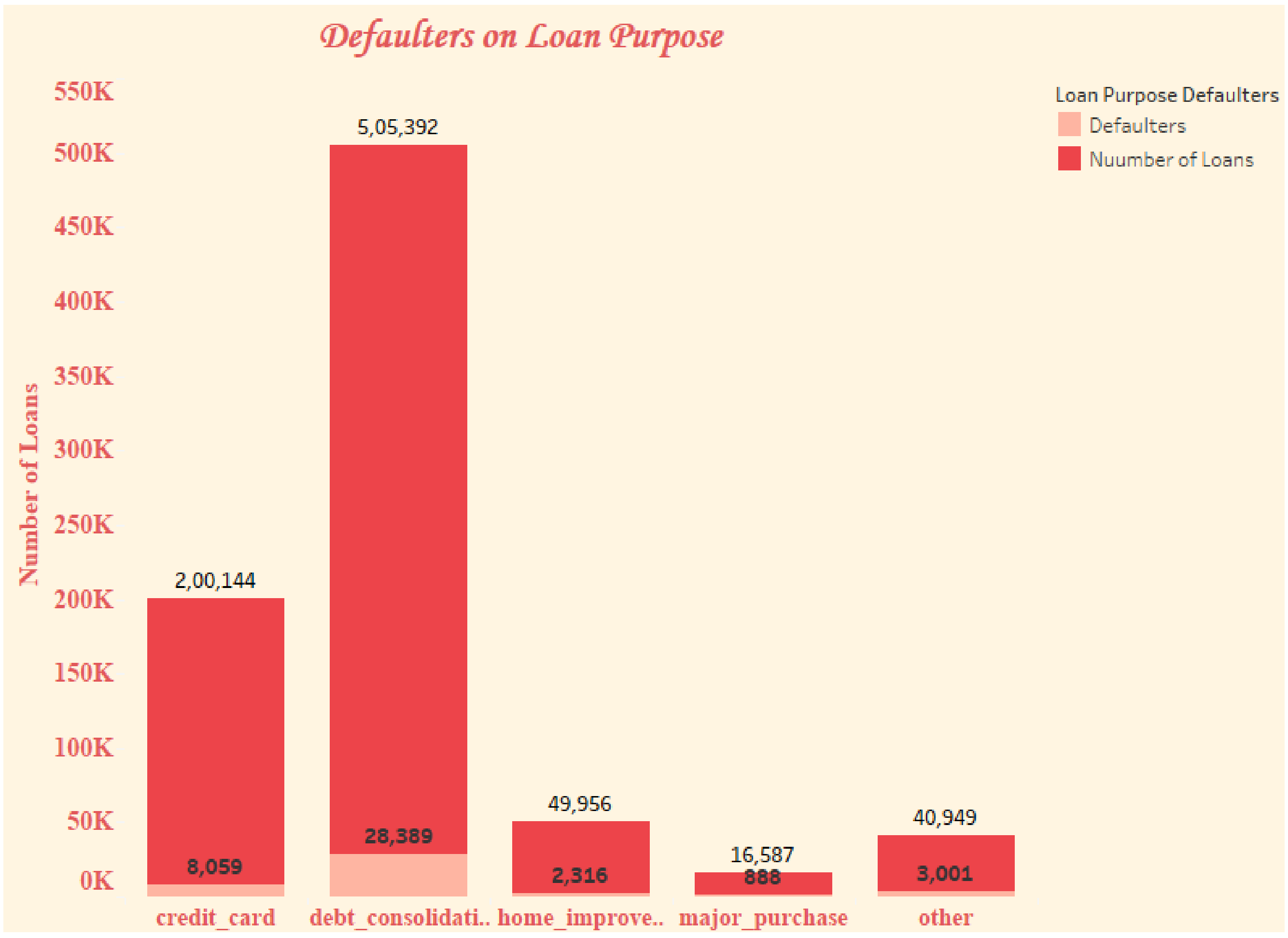
```
Open
DATASET > Python >                       Search Python
Organize ▾   New folder

This PC            Name                       Date modified         Ty
3D Objects         XYZCorp_LendingData.txt    3/23/2018 4:52 AM     Te
Desktop            scaler.pkl                 10/20/2020 1:53 PM    Pl
Documents          project.py                 10/18/2020 12:02 PM   Py
Downloads          Project.csv                10/10/2020 6:45 PM    M
Music              Processed_datash.csv       10/16/2020 11:05 AM   M
Pictures           Processed_data.csv         10/15/2020 5:32 PM    M
Videos             Predictions.csv            10/26/2020 3:06 PM    M
Windows (C:)       model.pkl                  10/20/2020 2:23 PM    Pl
RECOVERY (D:)      heatmap_1.png              10/11/2020 6:31 PM    Pl
                   ClientSideTool.py          10/31/2020 6:26 PM    Py

File name:
                                              Open       Cancel
```

Recommendations

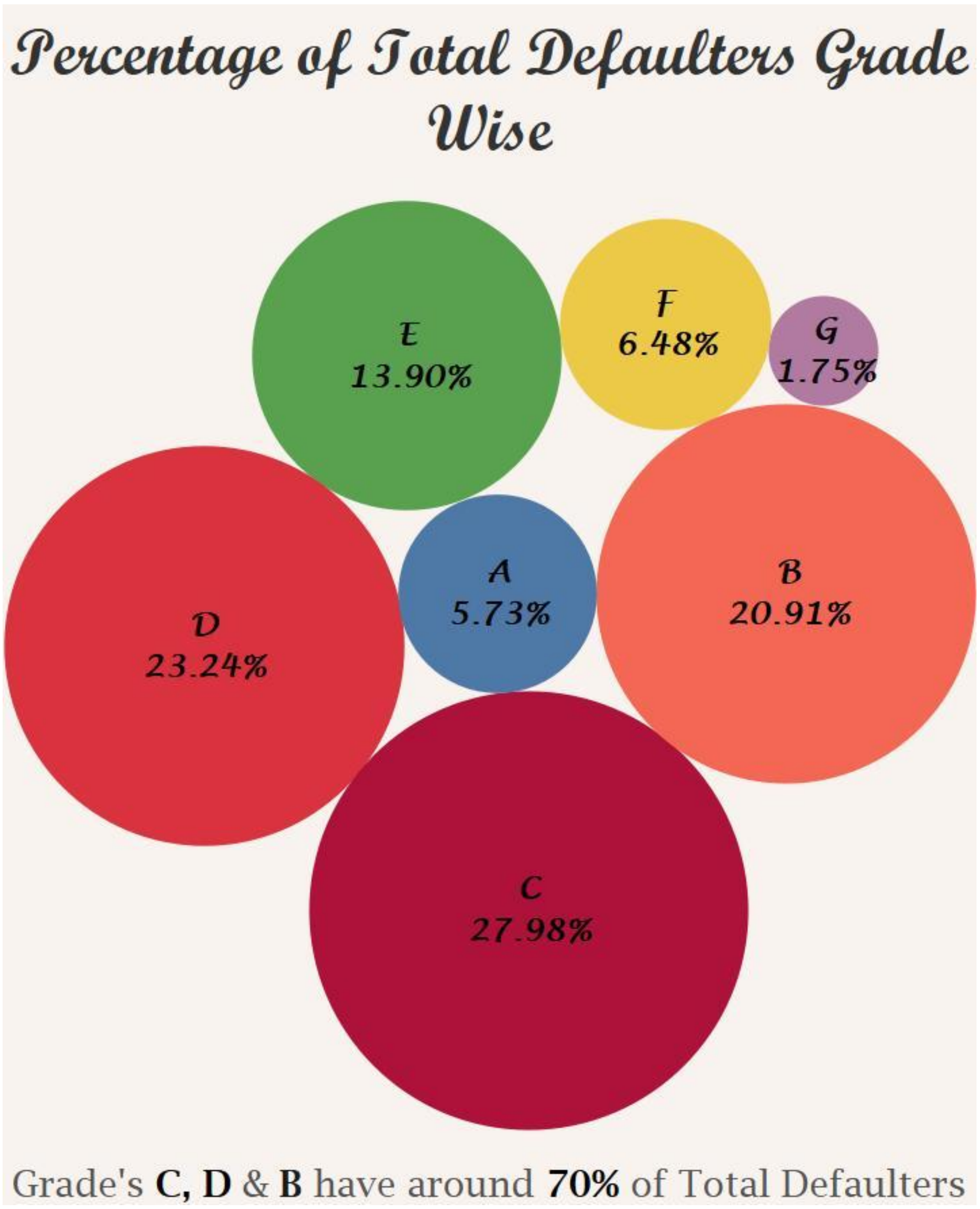
High Quality Borrowers



A borrower that chooses a short term (36 months) for the life of a loan implies that they are committed to paying off the debt as quickly as possible. However, we have found that the shorter loan term (36 months) had more defaults than the lengthier term (60 months). If a **borrowers** apply for a **loan**, suggest him/her for the **60 months** term loan rather than **36 months**.

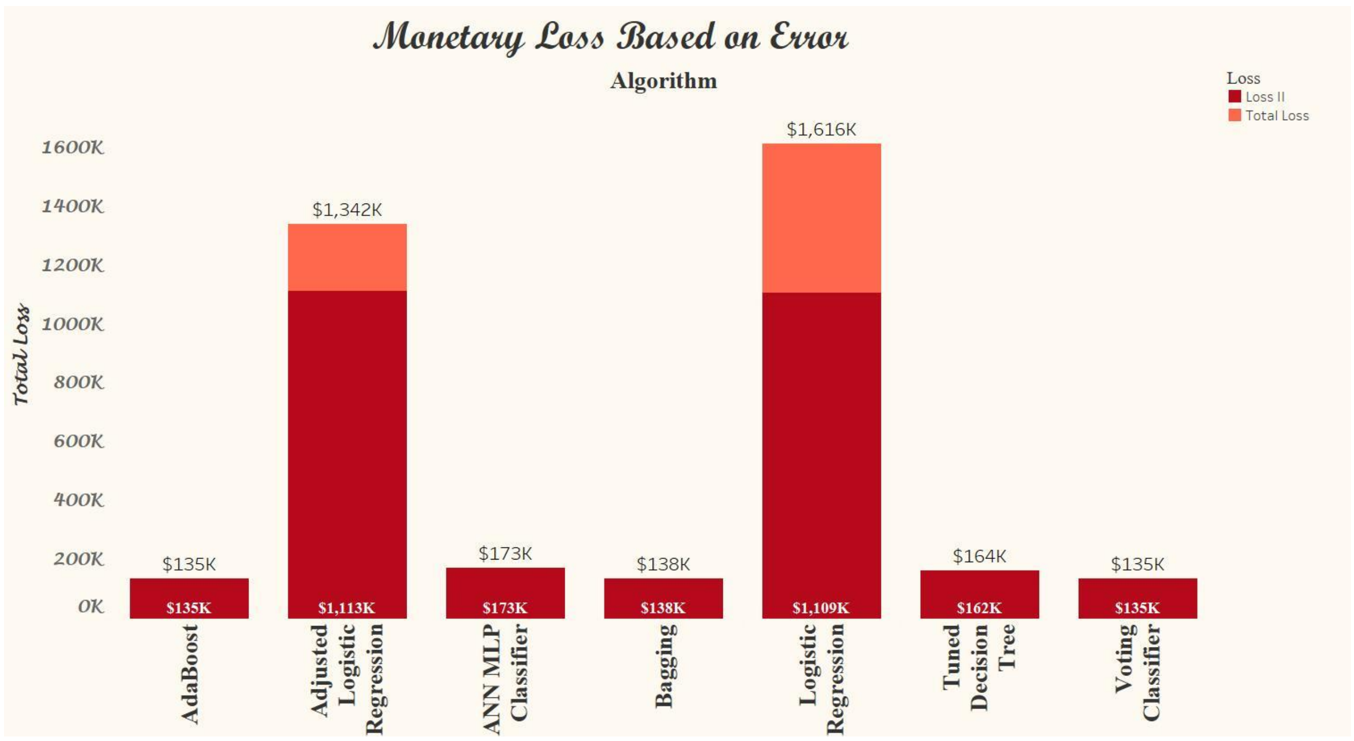


- ➢ The most common type of unsecured loans are debt consolidation, credit card, student loans, personal loans and more.
- ➢ Reduce the number of approvals where purpose is debt_consolidation followed by credit_card.



Now we can see that majority of the loan defaulters lies in grades **C, D** and **B.**

So, we suggest to reduce the loan approvals for these grades.



- Loss II – Actual value is defaulters and Predicted them as Non- defaulters.
- By taking Loss II and AdaBoost Model into consideration, we can say that the loan issuer facing a loss of $135K.