**Bank Loan Underwriting: Machine Learning Model**

Project Goal

In the bank loan application, underwriting is the process of analyzing an applicant’s “credit history, employment history, assets, debts, and other factors”, in order to evaluate the individual’s ability to pay back the loan and therefore determine whether the loan should be approved (Nelson, 2014). Traditionally, human underwriters take on this task, which has its disadvantages. For example, evaluation time can be long when application volume is high and can, therefore, lead to lower customer satisfaction. In addition, there might be very complex underlying patterns that are difficult for a human underwriter to pick up. In contrast, a machine learning model can process an individual evaluation in a very short amount of time once the model is built, with the ability to pick up very complex underlying patterns that may not be obvious at all to human analysts.

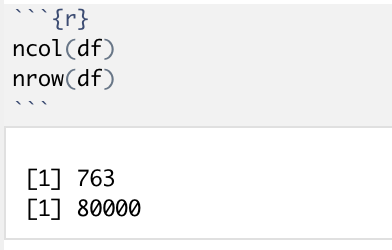
The goal of the current project is to develop a machine learning model to help the underwriting department of our bank to make loan approval decisions for customers. More specifically, we seek to predict both the default and the degree of losses, in order to make joint decisions that maximize profit. This model will help overcome some of the disadvantages of the current manual underwriting process.

Overview & Data Exploration Analysis

Three datasets were used in this analysis. The training dataset, **train\_v3.csv,** and two test datasets, **test\_scenario1\_2.csv** and **test\_scenario3.csv.** Before using the training data to build our model, we explored the data in order to examine the data quality and transform data as necessary. Below are the specific steps we went through:

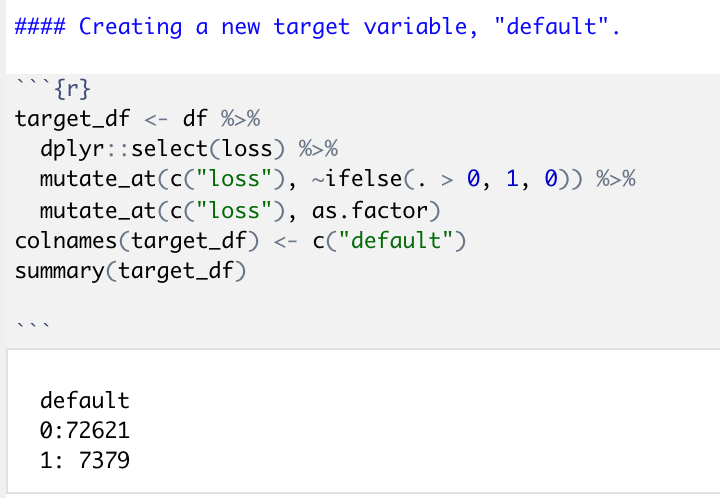
1. Summary statistics

First, we attempted to have a high-level view of what the training dataset looks like. Because the dataset is very large, and because the variable names are arbitrary, we decided to run ncol(df) and nrow(df) to just see how big the dataset is, instead of running summary(df). The result shows that we have a dataset with 763 variables and 80000 observations.



1. Creating the target variable

In the original training dataset, we have a variable named “loss”, indicating the loss percentage given default. However, we would like to predict both the loss and the default. To indicate whether a case defaults, we created a new variable “default”, where default = 0 if loss = 0; otherwise, default = 1.



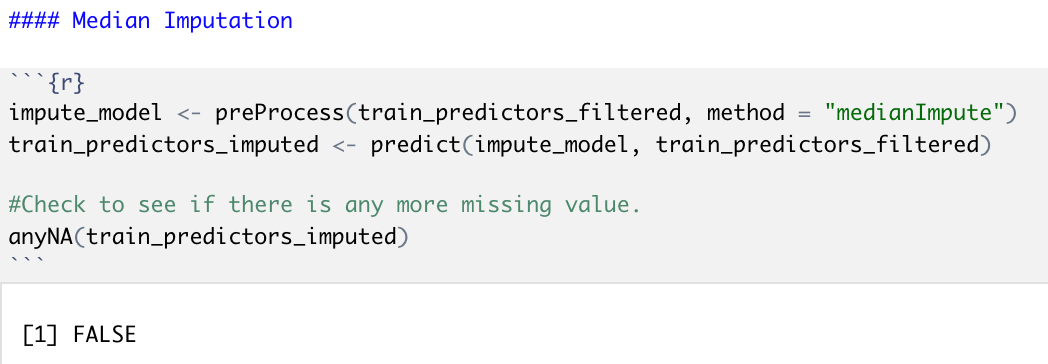
1. Variable selection

For variable selection, because we do not have any metadata to inform us the exact meaning of the variables in our datasets (all variables in the training dataset are given arbitrary names such as v1, v2, v3, etc., except for the outcome variable, “loss” ), we were not able to examine the relevance of the variables manually. However, we ran a variance analysis on all variables and identified several variables that have near-zero variance. We removed these variables as they would not have a significant contribution to the explanatory power of the model.

We also ran a lasso regression, which automatically eliminated 136 variables from the model due to low explanatory power. In our eventual model development, we have 600 predictor variables.

1. Imputing missing values

The data set had 593281 missing values. We feel that simply removing cases with missing values will create too big of a decrease in the size of our training data. Therefore, we imputed missing data points with column medians.



Modeling Strategy

The team considered the glmnet package in r as our foundation model. We have applied lasso regression on our training data to build the model. The team considered a straightforward approach i.e, a 10-fold cross-validation approach in training the model. The metric that we used is Area under the curve.

Reasons for choosing cv.glmnet with Lasso regression:

1. Lasso regression is what is called the Penalized regression method, often used in machine learning to select the subset of variables. Briefly, shrinkage decreases model variance in exchange for some increased model bias and thus limiting overfitting of a model.
2. This constraint causes regression coefficients for some variables to shrink towards zero. This is the shrinkage process. The shrinkage process allows for better interpretation of the model and identifies the variables most strongly associated with the target corresponds variable
3. It provides greater prediction accuracy when compared with the ordinary least square regression.
4. With Lasso Regression, the regression coefficients for unimportant variables are reduced to zero which effectively removes them from the model and produces a simpler model that selects only the most important predictors.
5. Since Lasso nullifies the unimportant, multicollinearity features and prevents the model from overfitting by a regularization parameter, The model will have substantially less variance and bias.
6. We choose the train() function from caret package to build a model for Loss given default to find good predictors from noise using R-squared metric. Then, we built a generalized linear model with good predictors that contribute to the model’s explanatory power.

Estimation Of The Model’s Performance

For model evaluation performance metrics are used, i.e. the r-squared and AUC for this model. As a result, we get 10 test set metrics, which we subsequently average after 10 iterations yielding the final model performance metric. In lasso regression, it is a set of proposed regularization values λ. For each proposed hyperparameter, the metric is calculated using the test set, resp. hold-out fold, as described above. The hyperparameter supplying the best metric, i.e. Highest R-Squared and AUC is used to create the final model.

Call: cv.glmnet(x = x.train, y = y.train, type.measure = "auc", parallel = T, family = "binomial", alpha = 1)

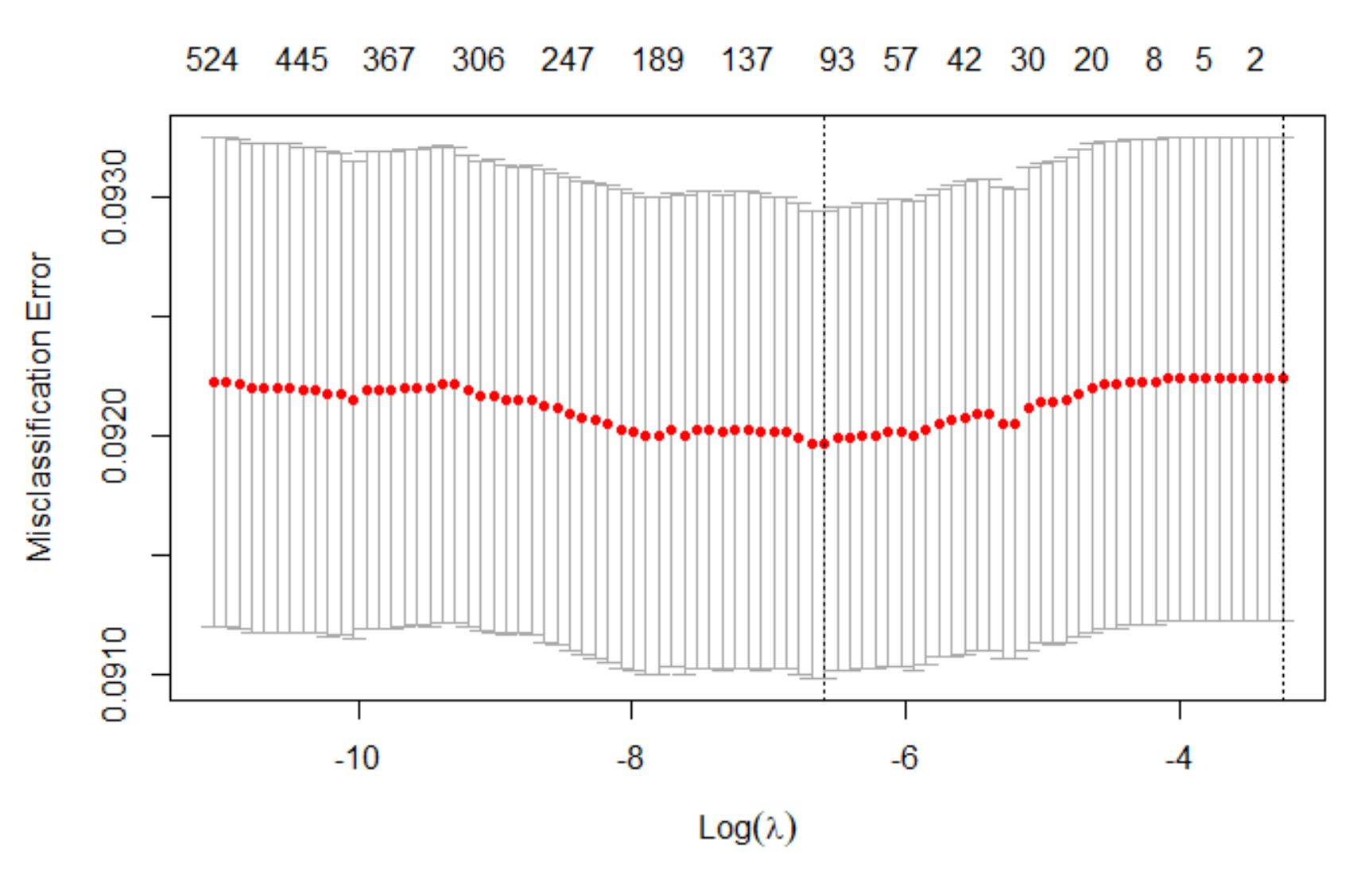
Measure: AUC

Lambda Measure SE Nonzero

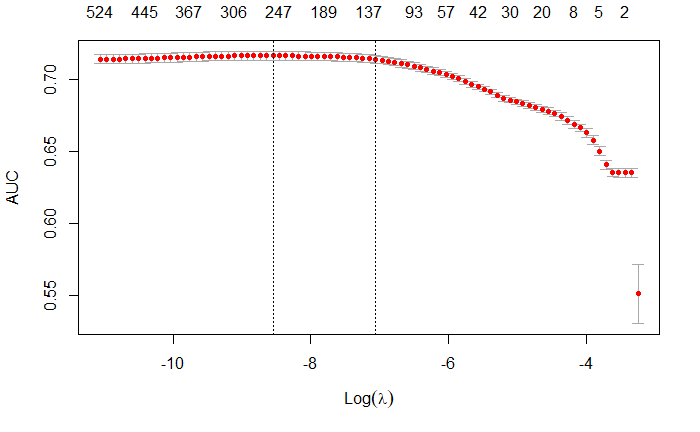
min 0.0001943 0.7168 0.003063 258

1se 0.0008609 0.7142 0.002721 128

The optimal Lambda is suggested by the model on the basis of the misclassification error.

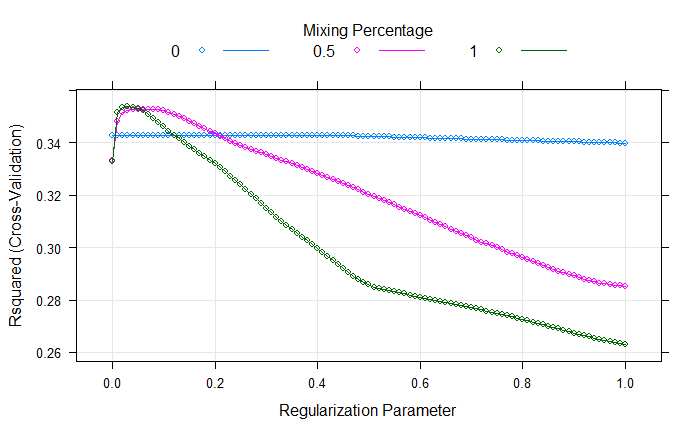


For the model training using cv.glmnet(), the standardization of all the predictors will be taken care of by this function by default and there is no need to mention explicitly. The AUC value suggested by cv model is 0.716



Following the hyperparameter tuning of lambda, the regularization parameter of the Lasso-Regression we eventually fitted the final model using the hyperparameter lambda which yielded the highest cross-validated R-Squared statistic. the value of lambda was λ= 0.00019 with a final R Squared value of 0.40072 however, we do not bother about this value, since we estimated the R-Squared statistic for unseen data using cross-validation, and we accept the model to perform as indicated by the cross-validation procedure.

For the below plot, we choose to do a brief comparison between ridge, lasso, and elastic net regression, and it turns out, Lasso performed well among the three. The regularization parameter is maximum at r squared = 0.365. Hence we choose Lasso to build the model.



**Scenario 1:**

In the first scenario, we have $1.4 billion as a company to loan. This is enough to provide loans to all as the sum of the requested amount column is below this total.

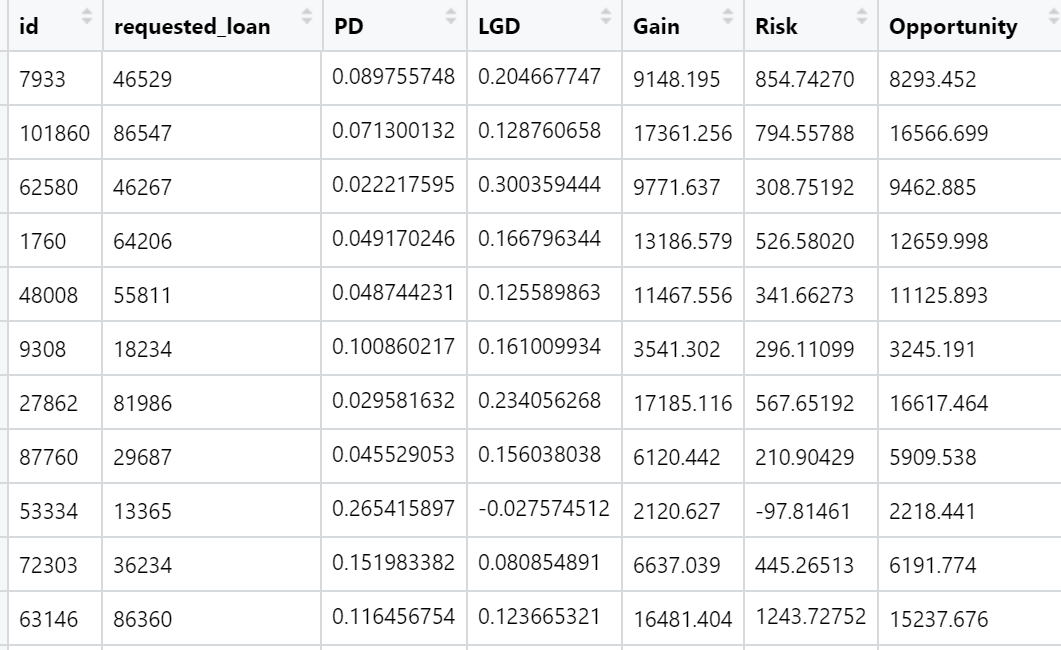
Interest Percentage: 4.32%

The probability of default and Probability of loan given default is calculated. Expected loss and Expected gain for each transaction is done by using this formula

Gain : PD \* LGD \* requested\_loan

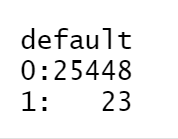
Risk : (1 - PD) \* 5 \* .0432 \* requested\_loan.

Opportunity (Delta): Gain - Risk



Case 1:

The algorithm is going to approve the loan to different customers when the **threshold of probability of default is greater than 0.5**. In this case number of customer default will be as follow:

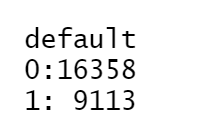


And total gain to the bank is:

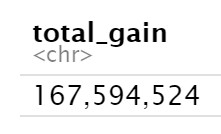


Case 2:

The algorithm is going to approve the loan to different customers when the threshold of **probability of default is greater than 0.1**. In this case number of customer default will be as follow:

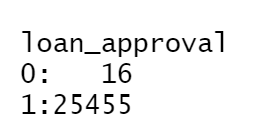


And total gain to the bank is:



Case 3:

The algorithm is going to approve the loan to different customers when **Gain >= Risk**. In this case number of customers whose loan will approve will be as follow:



And total gain to the bank is:

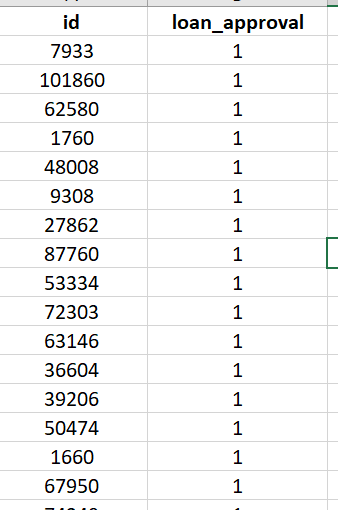


Total requested loan where Gain >= Risk:



Final Decision:

Customer Loan Approval in Scenario 1 when Gain >= Risk

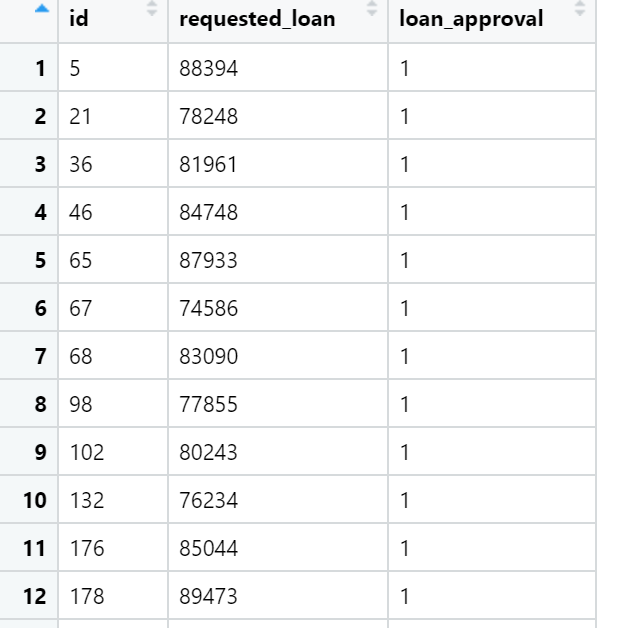


**Scenario 2:**

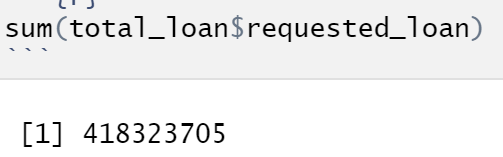
The second scenario is like the first but now we have a budget of $450 million.

Interest: 4.32%

In this scenario, we have sorted the customers based on Opportunity (Gain-Risk) and then approve the loan to the customer having the highest opportunity value until all the money is allocated.



Sum of total loans given out is also calculated:



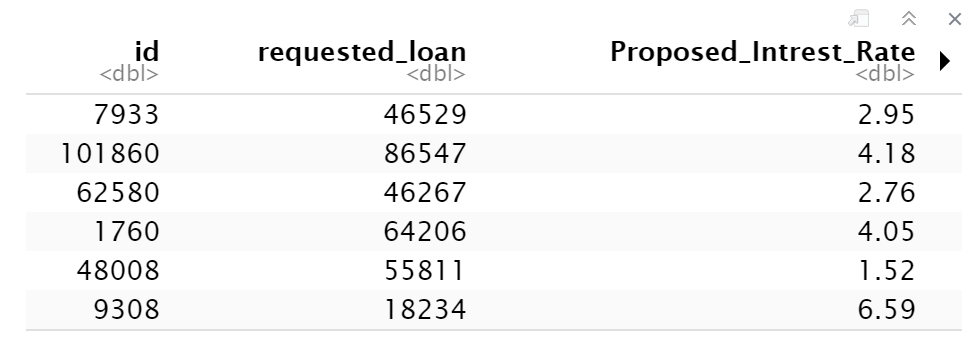
**Scenario 3:**

In the third scenario, now each customer has a proposed interest rate. This will change our gain calculation as it will vary from person to person. We will have the $1.4 billion budget.

In this case, expected loss and gain are calculated using this following formula

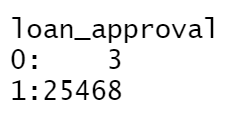
Expected Loss: PD \* LGD \* requested\_loan

Expected Gain: (1 - PD) \* 5 \* Proposed\_Intrest\_Rate \* requested\_loan



Case 1:

The algorithm is going to approve the loan to different customers when **Gain >= Risk**. In this case number of customer whose loan will approve will be as follow:

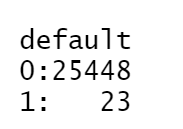


And total gain to the bank is:



Case 2:

The algorithm is going to approve the loan to different customers when the **threshold of probability of default is greater than 0.5**. In this case number of customer default will be as follow:

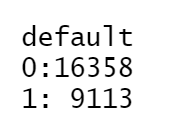


And total gain to the bank is:



Case 3:

The algorithm is going to approve the loan to different customers when the **threshold of probability of default is greater than 0.5**. In this case number of customer default will be as follow:

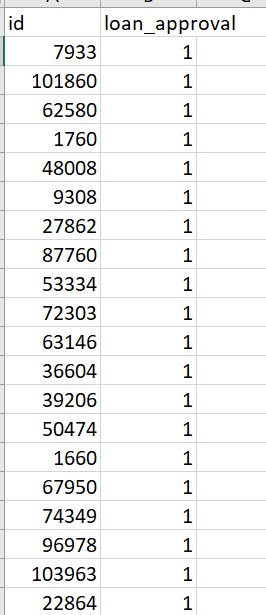


And total gain to the bank is:



Final Decision:

When the threshold of probability of default is greater than 0.5



Insights and conclusions

The probability of default model is 71% accurate in detecting the loan defaulter and the loss given default model is tuned to reduce the Mean Absolute Error and r2 value of .4072.

* Scenario1: We have taken multiple cases for loan approval. However, we concluded that maximum gain is when we consider a case where gain is greater than risk.
* Scenario2: In this scenario, we have sorted the customers based on Opportunity (Gain-Risk) and then approve the loan to the customer having the highest opportunity value until all the money is allocated.
* Scenario3: Likewise, scenario1, multiple cases for loan approval have been taken. But the final decision has been done when the threshold of probability of default is greater than 0.5.

Given that some customers may produce a negative return, the entire initial capital amount was not given out as loans in any of the scenarios.

Scenario1



Scenario2



Scenario3

