Project Report on Bank Loan Default and Loss

Nikhil Kumar Sampath, Snehitha Anpur, Kiran Kour, Shivani Pitla

MIS64037-003: Advanced Data Mining And Predictive Analytics

Professor Dr Rouzbeh Razavi

Kent State University

Project Goal:

This project addresses the challenge of loan default prediction by predicting the likelihood of default and estimating the potential loss that may occur in the event of a default. Unlike traditional finance-based approaches that categorize counterparties as either good or bad in a binary way, this approach considers both the probability of default and the potential severity of losses.

This project aims to bridge the gap between traditional banking and asset management by optimizing the risk to the financial investor. Rather than solely focusing on reducing the consumption of economic capital, this approach considers the impact of default on the investor's portfolio and seeks to minimize this risk.

By incorporating both the probability of default and the potential loss, this project provides a more nuanced and comprehensive risk assessment for the financial investor. This information can help inform lending decisions and ultimately lead to better outcomes for both the investor and the borrower.

Overview of Data:

Upon receiving the banking dataset, our group was initially overwhelmed by the sheer volume of data and the absence of identifiable client names. Instead, clients were represented by ID numbers and various variables. The dataset's columns could have been better labelled, hindering our efforts to anticipate default rates and loss severity. Nevertheless, we aimed to bridge traditional banking with asset management by optimizing risk for financial investors while minimizing the consumption of economic capital. However, the anonymous variable titles limited our ability to apply domain knowledge, leaving us to rely primarily on data preparation and correlation analysis. This led us to focus on preparing the dataset, which required considerable effort and attention to detail.

To prepare the data, we undertook an extensive data-cleaning process. Initially, we analysed the data structure to determine the columns and row count, and the type of data variables. We then examined the data set for missing data, as indicated in the data structure section. Next, we investigated the variables with zero or near-zero variance and those that exhibited high correlations with each other.

Upon further examination of the dataset, we discovered a "loss" column that contained valuable information. Specifically, clients with zero value in the "loss" column had paid off their loans without defaulting. Conversely, those with numerical values greater than zero had defaulted on their loans at some point. For instance, if Mrs Alice had a value of 10 in the default column, this would indicate that she had paid off 90% of his loan before defaulting.

The Data Structure:

The primary dataset consists of rows and columns, comprising 79999 rows corresponding to individual clients, and 762 columns representing various anonymous variables describing each client's profile. All variables in the dataset are numeric, and there are 79999 observations/customers and 762 variables, with one variable representing the "loss" percentage of each customer's loan. However, the generic nature of the variables made it challenging to ascertain their true meanings.

Exploratory Data Analysis: Probability Default Model

Data Cleaning and Feature Reduction:

```r
for (i in seq_along(data)) {
  tmp <- data[[i]]
  if (class(tmp) %in% c("numeric", "integer")) {
    if (any(is.na(tmp))) {
      tmp <- tmp[!is.na(tmp)]
      if (length(tmp) == 0 || var(tmp) == 0) {
        print(paste("no variance in column", names(data)[i]))
      }
    } else {
      if (var(tmp) == 0) {
        print(paste("no variance in column", names(data)[i]))
      }
    }
  }
}
```

```
## [1] "no variance in column f33"
## [1] "no variance in column f34"
## [1] "no variance in column f35"
## [1] "no variance in column f37"
## [1] "no variance in column f38"
## [1] "no variance in column f678"
## [1] "no variance in column f700"
## [1] "no variance in column f701"
## [1] "no variance in column f702"
## [1] "no variance in column f736"
## [1] "no variance in column f764"
```

To reduce the feature count and consider only the important variables, we did feature reduction by finding the features with no variance and eliminating them.

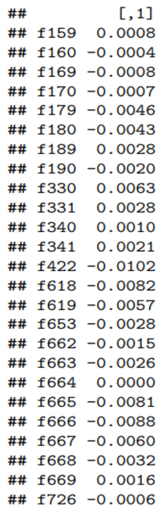
Eliminating the columns with 0 variance

```r
data_no_var <- subset(data, select = -c(f33, f34, f35, f37, f38, f678, f700, f701, f702, f736, f764))
```

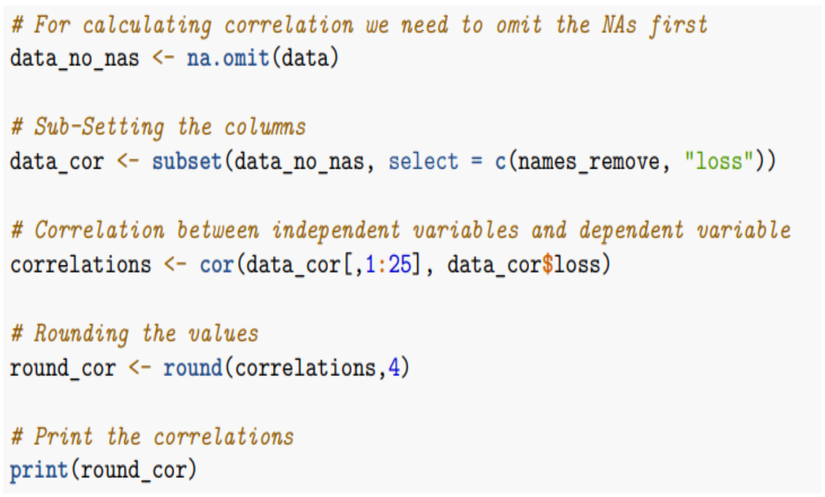
Reviewing the Missing Values:

Upon reviewing the dataset for missing values, we discovered numerous missing values throughout the dataset. We examined the first 20 columns of the dataset to identify the presence of any missing values (NAs). It was evident that many columns in the dataset contained NAs. After calculating the percentage of NAs in features, we discovered that the maximum percentage of features with missing values in the dataset was 17.8%. We eliminated columns with more than 10% missing values using this information.

Before dropping any columns, we checked the correlation between the attributes with missing values and the target variable, i.e., "loss."



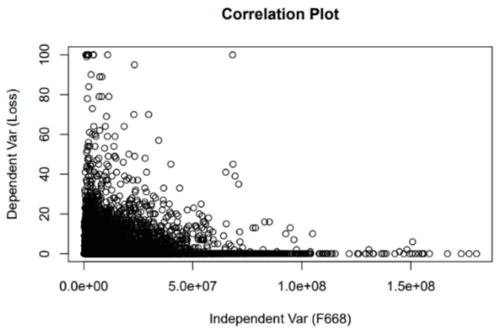
```
##          [,1]
## f159  0.0008
## f160 -0.0004
## f169 -0.0008
## f170 -0.0007
## f179 -0.0046
## f180 -0.0043
## f189  0.0028
## f190 -0.0020
## f330  0.0063
## f331  0.0028
## f340  0.0010
## f341  0.0021
## f422 -0.0102
## f618 -0.0082
## f619 -0.0057
## f653 -0.0028
## f662 -0.0015
## f663 -0.0026
## f664  0.0000
## f665 -0.0081
## f666 -0.0088
## f667 -0.0060
## f668 -0.0032
## f669  0.0016
## f726 -0.0006
```

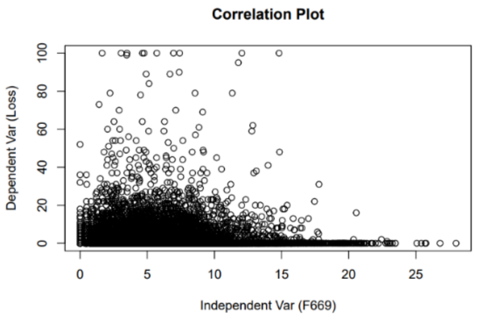


```
# For calculating correlation we need to omit the NAs first
data_no_nas <- na.omit(data)

# Sub-Setting the columns
data_cor <- subset(data_no_nas, select = c(names_remove, "loss"))

# Correlation between independent variables and dependent variable
correlations <- cor(data_cor[,1:25], data_cor$loss)

# Rounding the values
round_cor <- round(correlations,4)

# Print the correlations
print(round_cor)
```

As seen in the figure above, upon examining the correlation values between the attributes with missing values and the target variable, we found no significant relationship between them. The correlation values were close to 0, indicating the absence of any correlation between these attributes and the target variable, i.e., "loss."

To visualize this information, we plotted scattered plots to see the absence of correlation between these attributes and the target variable.





Instead of removing all attributes with missing values, we set a threshold of dropping any column with more than 10% missing values. We chose this approach because removing all attributes with missing values could result in losing a significant percentage of essential attributes. Additionally, dropping rows with missing values could result in a loss of nearly 50% of the data, which was undesirable.

Previously, the dataset contained 762 attributes. However, after dropping 25 attributes with more than 10% missing values and columns with zero variance, the updated count of columns is 726. Despite this reduction, some attributes with less than 10% missing values still require treatment. Additionally, reducing the number of features is crucial for building more efficient models.

```r
data_nas <- data[, colMeans(is.na(data)) > 0.1]

names_remove <- colnames(data_nas)
names_remove
```

```
##  [1] "f159" "f160" "f169" "f170" "f179" "f180" "f189" "f190" "f330" "f331"
## [11] "f340" "f341" "f422" "f618" "f619" "f653" "f662" "f663" "f664" "f665"
## [21] "f666" "f667" "f668" "f669" "f726"
```

Imputing Missing Values:

We explored several imputation methods, including missForest, multiple imputations using mice, and knn imputation, but found that they were computationally expensive. Here is some information on how these methods work:

- missForest- MissForest is an imputation method used to fill in missing values in a dataset. It is a non-parametric imputation method based on a random forest algorithm; it trains a random forest model on the non-missing data and then uses this model to impute missing values in the same dataset.

- Mice- The basic idea of MICE is to use a series of regression models to impute missing values where each variable with missing values is imputed using a separate regression model, with the imputed values updated in a chained process.

- Knn imputation- K-nearest neighbour (KNN) imputation is a non-parametric method used to impute missing values in a dataset. In this method, the missing values are imputed by finding the K-nearest neighbours of the sample with the missing value and using their average or weighted average as the imputed value.

The reason for not using the above-mentioned techniques is that they are computationally very intensive and complex.

Therefore, we opted for median imputation, which was both computationally efficient and resistant to outliers in the data.

```
# Performing median imputation on data_less_nas
set.seed(123)

na_median <- function(x) {
  ifelse(is.na(x), median(x, na.rm=TRUE), x)
}

data_clean <- data_less_features %>% mutate_all(na_median)
```

After the imputation of missing values, we checked for NAs and found no NA values.

The next step was to perform a data transformation wherein we introduced a new column called PD, a binary column with 0 or 1. PD will be our target variable for the Probability of loan default model, letting us know the potential customers who will be defaulting and not defaulting.

```
data_clean["PD"] <- ifelse(data_clean$loss>0,1,0)
```

Now we have a new column called PD which is a binary column having the values as 0 or 1.

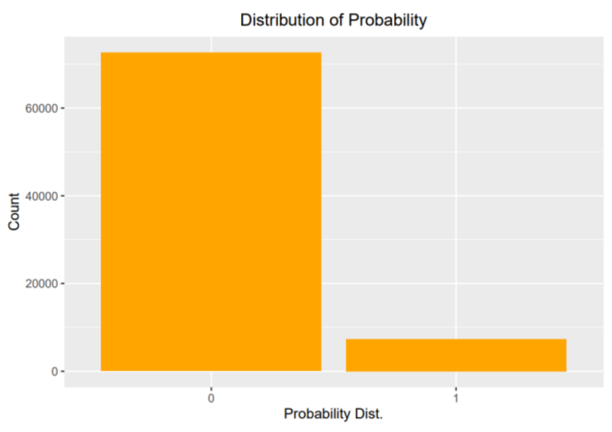
Balancing both classes with the help of Synthetic Data

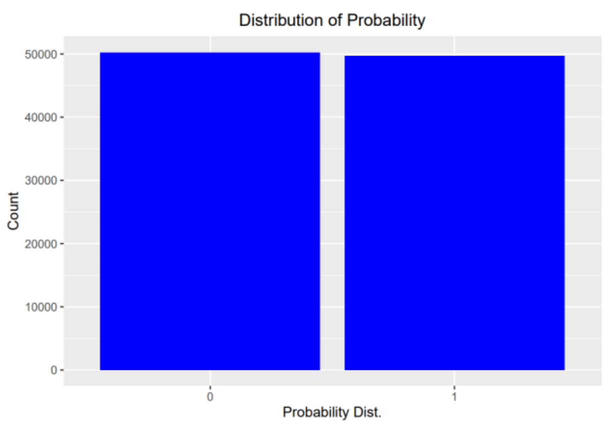
The next step was balancing both classes with the help of Synthetic Data. Balancing the dataset is crucial to avoid bias towards the class with more observations. Failing to balance the dataset may result in the model only predicting values for the class with a higher number of observations; in this case, "not default (0)" has a more significant number of observations compared to "default (1)". However, we aim to predict the "default (1)" class accurately.

Before balancing the classes, the "not default (0)" category had a row count of 72620, meaning there were 72620 observations in that category. Similarly, the "default (1)" category had a row count of 7379, indicating 7379 observations in that category.

After balancing the classes, the row count for "not default (0)" became 50259, indicating that the number of observations in this category has decreased. The row count for "default (1)" became 49741, indicating that the number of observations in this category has increased.

Balancing the classes in this way can be helpful in situations with an imbalance in the dataset, as it can help prevent the model from being biased towards the majority class. Balancing the data is crucial because the train set learns the data (both defaulted and non-defaulted classes). Balancing data makes our model learn more about the attributes and features of both classes. With this, the model will see a vast amount of data, and when applied to the test set, it will know the nature of the customers (0 and 1) who will default and not default.





The above figure shows the difference between the row count for “no default (0)” and “default (1)” before and after balancing the classes.

Feature Selection:

Feature selection is a process in machine learning and data mining that involves selecting a subset of relevant and informative features (or variables) from a larger set of possible features to improve the performance and efficiency of a machine learning model. This is done by removing irrelevant, redundant, or noisy features that contribute little to the accuracy or generalization of the model.

We tried various feature selection techniques, such as:

- **Boruta Technique-** Boruta is a wrapper algorithm based on Random Forest, which identifies relevant features by comparing the importance of original attributes and randomized versions.
  Disadvantage- Boruta Technique can be computationally expensive and time-consuming, especially for large datasets.

- **Step-Wise Regression-** Stepwise regression involves sequentially adding or removing features based on their statistical significance.
  Disadvantage- Step-wise regression suffers from overfitting if not performed carefully. Also, it assumes that the relationship between the predictors and the response variable is linear, which may only sometimes be the case.

- **Principal Component Analysis (PCA) -** PCA stands for Principal Component Analysis, a statistical method for data dimensionality reduction. It is a technique that transforms the data into a new set of variables, known as principal components, that are linear combinations of the original variables.

The principal components are ranked in order of their ability to explain the maximum variance in the data.

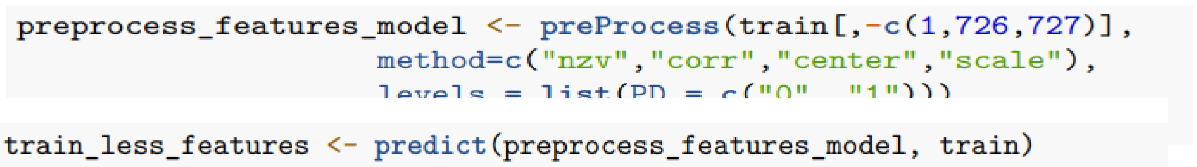
Disadvantage- The principal components may only sometimes be easily interpretable, especially if many variables exist. Also, it assumes a linear relationship between the variables, which may only sometimes be appropriate.

- **Partial Least Squares (PLS)-** PLS stands for Partial Squares, a multivariate statistical analysis technique used to model the relationship between two sets of variables - predictor variables and response variables. PLS aims to identify the latent variables (latent factors or components) in the predictor variables that explain the maximum variance in the response variables.

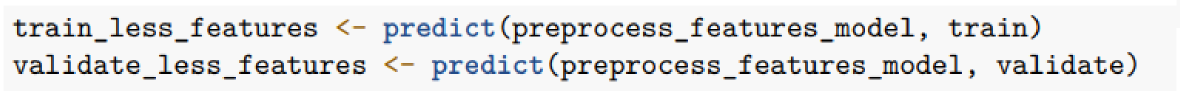
  Disadvantage- Requires careful selection of the number of components to use, which can be difficult to determine. It also assumes a linear relationship between the variables, which may not always be appropriate.

Feature Selection along with Normalization:

Now, we have to begin with feature reduction. We need to replicate the same features onto the test set to do feature selection, so we are partitioning the data into train and validate.



```
preprocess_features_model <- preProcess(train[,-c(1,726,727)],
                            method=c("nzv","corr","center","scale"),
                            levels = list(PD = c("0", "1")))
```



```
train_less_features <- predict(preprocess_features_model, train)
validate_less_features <- predict(preprocess_features_model, validate)
```

Reducing the number of features is a crucial step in any modelling task, as it helps improve the model's efficiency and accuracy. In this project, we have already removed features with more than 10% missing values and those with zero variance. However, there may still be some columns with a very low variance that will not add any value to the model. Hence, we will remove those attributes with a variance close to zero.

Furthermore, we will also check for highly correlated features and remove them as well. A high correlation between features can lead to multicollinearity, which can cause problems in the modelling process. Multicollinearity can make interpreting the model's results challenging, leading to unreliable coefficients, standard errors, and p-values.

After removing the features with variance close to zero and high correlation with each other, the features reduced from 726 to 253, excluding ID and loss.

Lasso for Variable Selection:

The Lasso regression analysis method is a type of linear regression that incorporates regularization to perform variable selection and avoid overfitting. The Lasso method uses the L1-norm penalty to penalize the absolute size of the regression coefficients, which results in some coefficients being shrunk to zero, effectively performing variable selection and reducing the number of variables included in the final model.

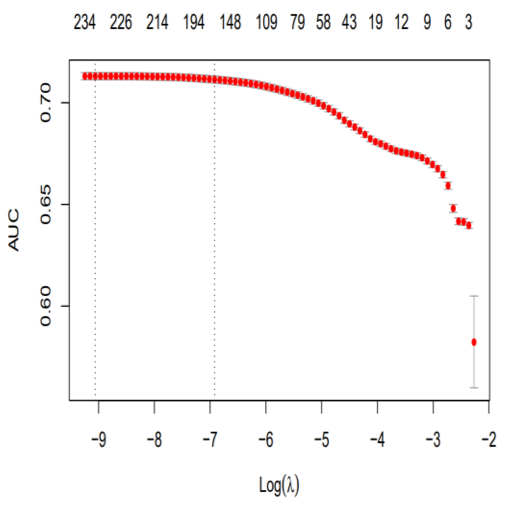
In this project, the Lasso regression method was used to analyze 253 variables and select the most important variables for predicting whether a customer is approved or rejected based on their default history. By selecting only the most relevant variables, the Lasso method increases the prediction accuracy and interpretability of the model, making it easier for the bank to understand and act upon the results. Now we will transform the data into input-output for lasso regression.

```
input <- as.matrix(train_less_features[,-c(1,255,256)])
output <- as.vector(train_less_features[,256])
```

```
set.seed(123)
glm_model <- cv.glmnet(x=input, y=output, data=train_less_features, family="binomial", type.measure= "auc", nfold
s=10, alpha=1)

glm_model
```

```
Call:  glmnet::cv.glmnet(x = input, y = output, data = train_less_features,      family = "binomial", type.measur
e = "auc", nfolds = 10, alpha = 1)

Measure: AUC

       Lambda Index Measure       SE Nonzero
min 0.0001163    74  0.7130 0.001717     233
1se 0.0009880    51  0.7114 0.001715     175
```

In the Figure below, the graph shows the Optimum Lambda value for the feature selection that was reduced from 253 variables to 233 variables. The first vertical dashed line indicates the lambda. Min value, while the second dash line shows the lambda value within one standard deviation to reduce the variables further. We decided to go with the lambda. Min value.



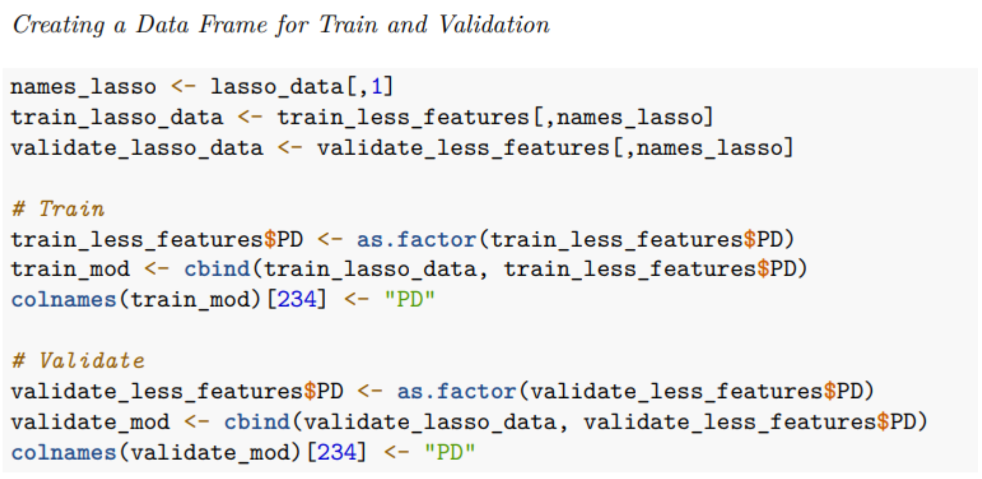
Storing the variables which have been retained when lambda=min

```
# Coefficients of columns at lambda.min
coef_lasso <- as.matrix(coef(glm_model, s="lambda.min"))

# Getting the column names of non zero attributes at lambda.min
var_lasso <- rownames(coef_lasso)[coef_lasso != 0]

# Removed Intercept
var_lasso <- var_lasso[-1]

# Sub-Setting the Non-Zero Columns along with their Coefficient values
lasso_coefs <- as.data.frame(coef_lasso[var_lasso,])

# Converting Row to Column
lasso_data <- rownames_to_column(lasso_coefs, var = "name")

# Changing the column name
colnames(lasso_data)[2] <- "coefficients"
```

Model Building

We will focus on two modelling approaches to achieve these two main objectives. For the first step, we will build a classification model that predicts the probability of loan default (PD). The model will output a probability score between 0 and 1, where values closer to 1 indicate a higher probability of default. For the second step, we will build a regression model that predicts the loss-given default (LGD). The model will estimate the percentage of the loan the customer will not repay in the event of a default, ranging from 0 to 1.

The Probability Default Model will be built by using all the 233 attributes selected while using the lasso regression model; we want to assess the model's performance using just the lasso-selected variables.



Creating a Data Frame for Train and Validation

```r
names_lasso <- lasso_data[,1]
train_lasso_data <- train_less_features[,names_lasso]
validate_lasso_data <- validate_less_features[,names_lasso]

# Train
train_less_features$PD <- as.factor(train_less_features$PD)
train_mod <- cbind(train_lasso_data, train_less_features$PD)
colnames(train_mod)[234] <- "PD"

# Validate
validate_less_features$PD <- as.factor(validate_less_features$PD)
validate_mod <- cbind(validate_lasso_data, validate_less_features$PD)
colnames(validate_mod)[234] <- "PD"
```

Modelling Strategies

After cleaning the data, the next step in the modelling strategy was to create the models for predicting PD and LGD separately. This can be done using various machine learning algorithms, such as logistic regression, decision trees, random forests, or neural networks.

To create the PD model, we used a classification algorithm that can predict the probability of default. We trained the model on a labelled dataset, where each observation has a binary label indicating whether the borrower defaulted or not. We selected the features most predictive of default, such as credit score, loan amount, income, and other relevant variables.

To create the LGD model, we used a regression algorithm that can predict the expected loss given default. We trained the model on a labelled dataset, where each observation had a continuous label indicating the amount of loss in case of default. We selected the relevant features that were most predictive of loss, such as loan-to-value ratio, collateral value, recovery rate, and other relevant variables.

Once the models were trained, we evaluated their performance on a validation dataset and fine-tuned their parameters to improve their accuracy and generalization performance on the test set.

Overall, the modelling strategy for predicting PD and LGD involved several steps, including data cleaning, feature selection, model creation, model evaluation, and parameter tuning. The goal was to create accurate and robust models to help lenders make informed decisions about loan approvals and risk management.

Random Forest for Probability of Default (PD) Model:

Before applying any modelling strategies to the data, splitting the data into train vs. validation is always advised. We have already split the data for pre-processing in a 3:1 ratio, i.e., 75% into the **"Training Set"** and 25% into the "**Validation Set."**

Random Forest is a robust machine learning algorithm that can be used for classification tasks, and it has several advantages over other algorithms:

Model Building

```
set.seed(123)
rf_model <- randomForest(PD~.,data=train_mod, ntree=100, mtry=5)
rf_model
```

```
##
## Call:
##  randomForest(formula = PD ~ ., data = train_mod, ntree = 100,      mtry = 5)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 0.95%
## Confusion matrix:
##       0     1 class.error
## 0 37215   480 0.012733784
## 1   229 37077 0.006138423
```

- High Accuracy: Random Forest has a higher accuracy rate than other algorithms, especially when the data is noisy and has many features. It can handle high-dimensional data well and can model complex relationships between variables.

- Variable Importance: Random Forest measures variable importance, which helps identify the most important features for classification. This information can be used to gain insights into the underlying data and to optimize feature selection.

- Flexibility: Random Forest can be used for both classification and regression tasks, and it can handle binary, categorical, and continuous data. It is also easy to use and requires minimal parameter tuning.

Overall, Random Forest is a versatile and reliable algorithm for classification tasks. We then predicted the training set of the random forest model on the validation set, to see the model's performance on the unseen dataset.

```r
pred_val <- predict(rf_model, validate_mod[,-234], type="prob")
validation_testing <- as.data.frame(cbind(validate_mod[,234],pred_val))
```

Cut-off Threshold

We utilized the ROC (Receiver Operating Characteristic) measure to evaluate the performance of each model we built. The ROC curve is a valuable tool for assessing binary classification models' performance by plotting the relationship between True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds. The optimal point on the ROC curve represents the point closest to the top left corner, where TPR is high, and FPR is low.

Overall, the ROC measure and the optimal threshold are essential in evaluating binary classification models' performance and making informed decisions based on their predictions. By utilizing this measure, we could identify the optimal point for each model and determine which customers are likely to default and which are not.

Cut-Off for Default and Not Default

```r
ROC_pred_rf_test <- prediction(pred_val[,2],validation_testing$V1)
ROCR_perf_rf_test <- performance(ROC_pred_rf_test,'tpr','fpr')
acc_rf_perf <- performance(ROC_pred_rf_test,"acc")
ROC_pred_rf_test@cutoffs[[1]][which.max(acc_rf_perf@y.values[[1]])]
```

```
## 84759
##   0.6
```

If the predicted probability class of 1 is greater than 0.6 then it is going to be considered as defaulted, else not defaulted.

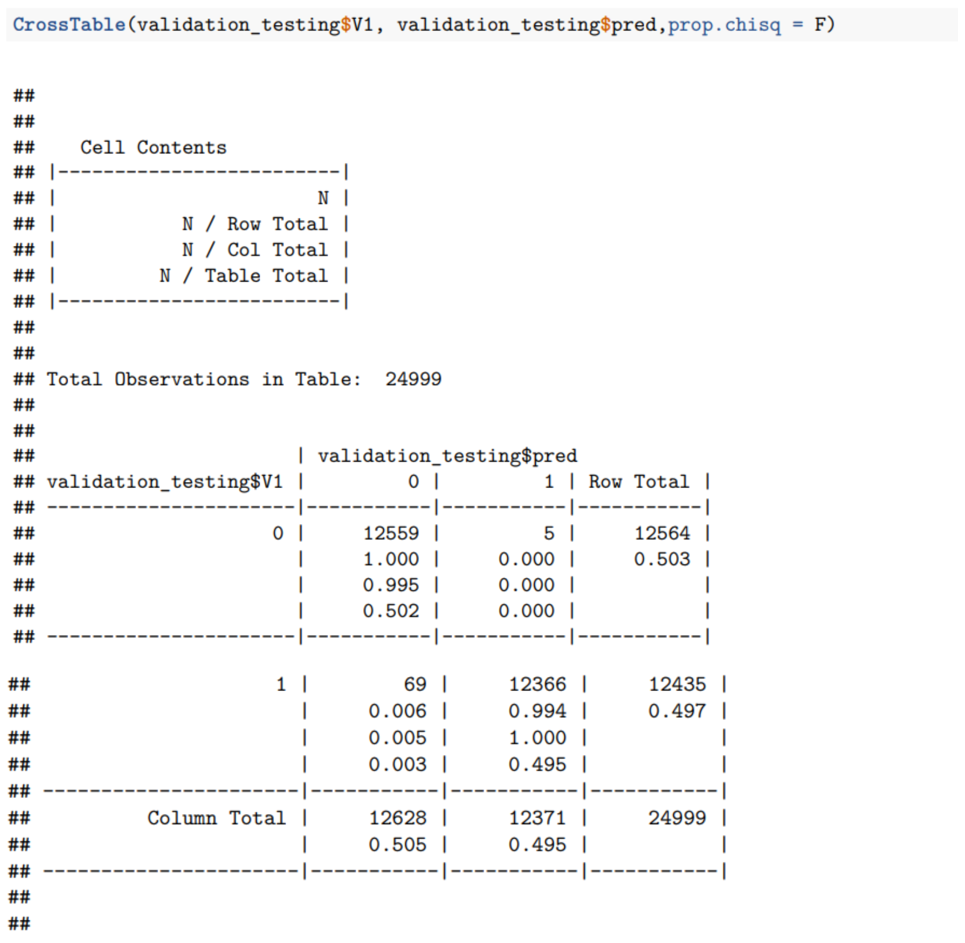
Setting the Cut-Off

```r
validation_testing['pred'] <- as.factor(ifelse(validation_testing$`1`>0.6,1,0))
validation_testing$V1 <- as.factor(validation_testing$V1)

# 0 and 1 are being changed to 1 and 2 so arranging it back to the original levels
validation_testing$V1 <- ifelse(validation_testing$V1=="1","0","1")
```

The cut-off threshold set for rf_model was 0.6, meaning if the predicted probability class of 1 is greater than 0.6, it will be considered defaulted or else not defaulted.

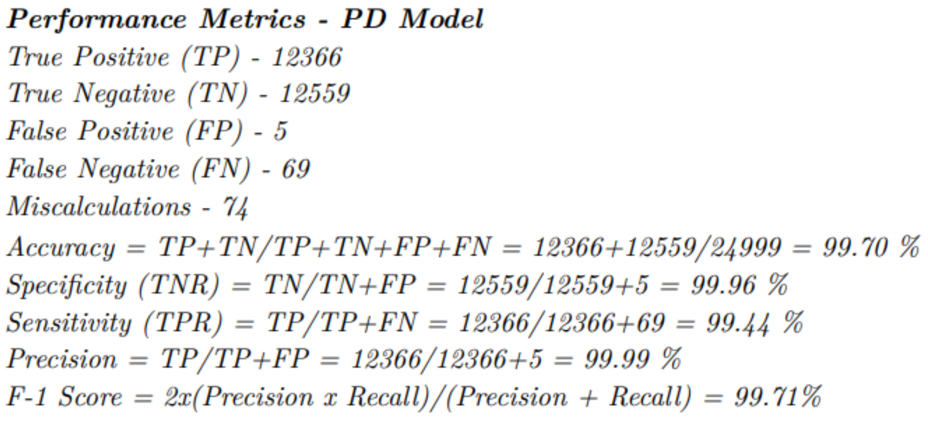
Evaluation of Model’s Performance:

The performance of the Random Forest model was assessed by predicting them over the validation set.



```
CrossTable(validation_testing$V1, validation_testing$pred,prop.chisq = F)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  24999
##
##
##                      | validation_testing$pred
## validation_testing$V1 |         0 |         1 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     0 |     12559 |         5 |     12564 |
##                       |     1.000 |     0.000 |     0.503 |
##                       |     0.995 |     0.000 |           |
##                       |     0.502 |     0.000 |           |
## ----------------------|-----------|-----------|-----------|
##                     1 |        69 |     12366 |     12435 |
##                       |     0.006 |     0.994 |     0.497 |
##                       |     0.005 |     1.000 |           |
##                       |     0.003 |     0.495 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |     12628 |     12371 |     24999 |
##                       |     0.505 |     0.495 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```



Performance Metrics - PD Model

True Positive (TP) - 12366

True Negative (TN) - 12559

False Positive (FP) - 5

False Negative (FN) - 69

Miscalculations - 74

Accuracy = TP+TN/TP+TN+FP+FN = 12366+12559/24999 = 99.70 %

Specificity (TNR) = TN/TN+FP = 12559/12559+5 = 99.96 %

Sensitivity (TPR) = TP/TP+FN = 12366/12366+69 = 99.44 %

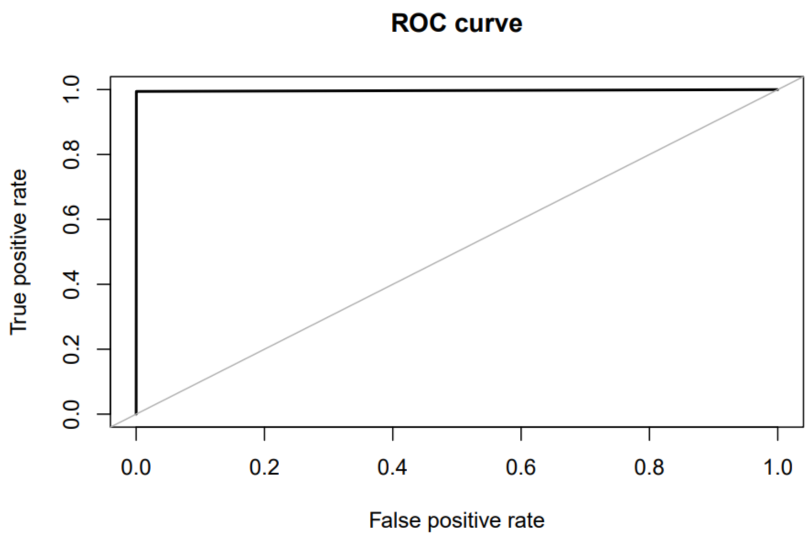
Precision = TP/TP+FP = 12366/12366+5 = 99.99 %

F-1 Score = 2x(Precision x Recall)/(Precision + Recall) = 99.71%

The main aim for us while building the PD model is to capture the default values without any wrong predictions, the cost of False Negative > False Positive, but here when we look at the TPR, we are doing great at 99.43%. Also, the F-1 Score and Precision are above nearly 99%.

It was observed that the accuracy of the Random Forest Model was 99.70%, and Miscalculations/Errors were 74. The Specificity of the model is 99.96%, and the Sensitivity of the model is 99.44%.

To assess the performance of a classification model, we used the AUC (Area Under the Curve) metric, which takes into account both True Positive Rate (TPR) and False Positive Rate (FPR) across different probability thresholds. We used the "RandomForest" package to build the model and hyper-tuned the "ntree" and "mtry" variables to improve its performance. "ntree" represents the number of trees in the random forest, and "mtry" represents the number of variables randomly sampled at each split of a tree. After hyper-tuning, we determined that the optimal values for "ntree" and "mtry" were 100 and 5, respectively, which resulted in an AUC value of 0.997.



```
## Area under the curve (AUC): 0.997
```

The Lasso attribute model for feature selection and the random forest for building the model have shown promising results in generalizing well on the validation data, an essential factor when considering a model's performance. One of the key advantages of using the Lasso model to select features is that it identifies a subset of attributes that are most relevant to predicting the target variable. These attributes are contextually more interpretable and provide a clearer understanding of the business problem. This helps in better decision-making and makes it easier to explain the model to stakeholders who may need a technical background.

After building the random forest model, we can store it for future use in predicting new data. This model has been trained using 233 features and has shown outstanding performance in precision, AUC, sensitivity, and F-1 score metrics. Therefore, we have decided to save the rf_model as the final model for making predictions on the test set. This will enable us to use the best-performing model to predict unseen data.

Loss Given Default (LGD) Exploratory Data Analysis

For the Loss Given Default model we will be supplying only the data related to default, the combined output i.e., PD will give the default and no default and LGD will give the extent of the loss incurred by the customers who have defaulted.

Data Transformation:

In the dataset, we are performing Data transformation to calculate the counts of defaults using the “loss” variable whenever the loss attribute value is greater than zero.

```
data["PD"] <- ifelse(data$loss>0,1,0)
```

Count of Observations - Default and Not Default

```
table(data$PD)
```

```
##
##     0     1
## 72620  7379
```

When we observe the loss attribute value, we see that it has a large range of values. Due to this it can cause numerical instability and make the algorithm difficult to converge to a solution. Hence, we must normalize the decision variable "loss", to reduce its range and make it the algorithm easier to find the optimal solution

```
# Normalizing the decision variable
data$loss <- (data$loss/100)
```

Defaulted Observations

```
# Sub-Set
data_lgd <- subset(data, data$PD == 1)

# Eliminating id and pd column created since we now have defaulted values
data_lgd <- data_lgd[,-c(1,763)]
```

Data Partition:

The data partition is performed on the data set into 75% Training data and 25% Test Data

Data Partition

```
set.seed(123)
data_split <- createDataPartition(data_lgd$loss, p=0.75, list=F)
train_data <- data_lgd[data_split,]
validate_data <- data_lgd[-data_split,]
```

Feature Selection and Normalization for LGD

We are trying to eliminate the attributes with near to zero variance, which is highly correlated among each other and finally imputing the missing values with the median. Note: We used directly the pre-processing technique to do the feature selection instead of using iterative steps.

```
preprocess_lgd <- preProcess(train_data[,-761], method = c("nzv", "corr",
                  "medianImpute","center","scale"))

train_lgd <- predict(preprocess_lgd, train_data)
validate_lgd <- predict(preprocess_lgd, validate_data)
```

Updated Column Count

```
ncol(train_lgd[,-250])
```

```
## [1] 249
```

Therefore, the feature reduction technique has resulted in 249 attributes in which none of the attributes is highly correlated, have high collinearity, and have no missing values. But the count of attributes is still huge, let us try to reduce the features by using a feature selection algorithm i.e., Lasso Model.

The input data for the Lasso model must be in the form of a Matrix and the output as a vector. Transforming the data into input and output as required for the lasso model

Transforming the data into input and output for lasso model

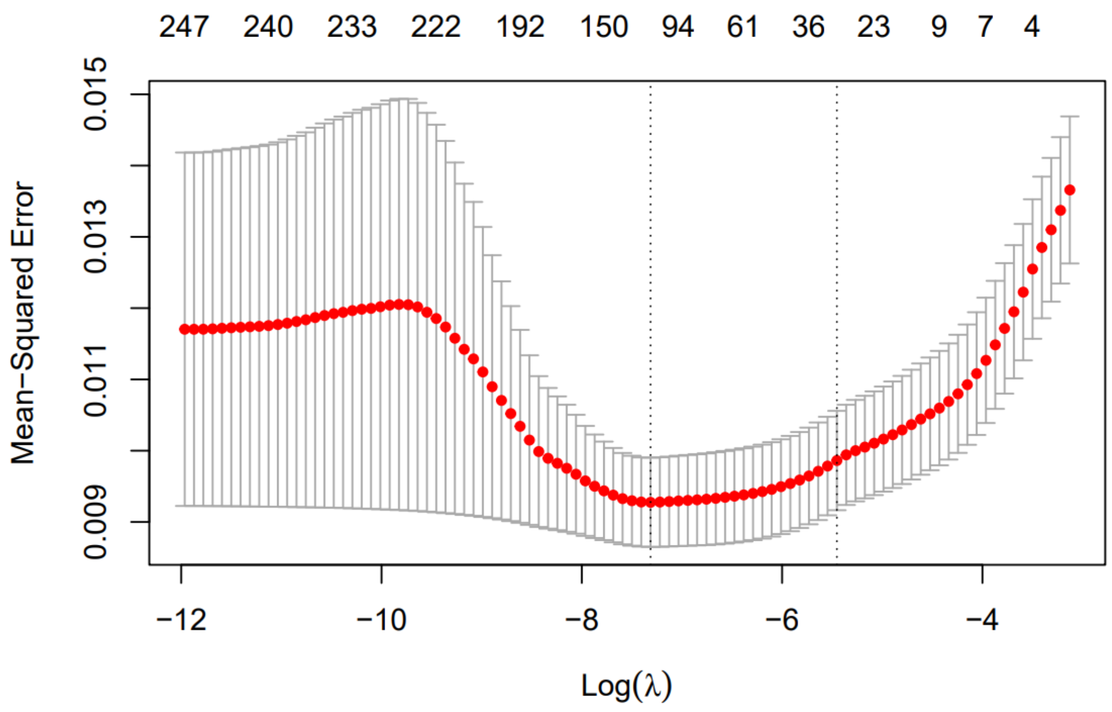
```
input <- as.matrix(train_lgd[,-250])
output <- as.vector(train_lgd[,250])
```

After the data normalization and transformation, we built the lasso model to further reduce the independent variables which are not significant in the LGD model.

```
set.seed(123)
glm_model_lgd <- cv.glmnet(x=input, y=output, data=train_lgd, family="gaussian", type.measure= "mse", nfolds=10,
alpha=1)

glm_model_lgd
```

```
Call:  glmnet::cv.glmnet(x = input, y = output, data = train_lgd, family = "gaussian",      type.measure = "mse",
nfolds = 10, alpha = 1)

Measure: Mean-Squared Error

      Lambda Index  Measure        SE Nonzero
min 0.000662    46 0.009237 0.0007284     119
1se 0.005127    24 0.009960 0.0007985      25
```

The mean square error resulted as shown below:



The variables with the minimum Lambda have been retained when lambda= min.

```r
# Coefficients of columns at lambda.min
coef_lasso_lgd <- as.matrix(coef(glm_model_lgd, s="lambda.min"))

# Getting the column names of non zero attributes at lambda.min
var_lasso_lgd <- rownames(coef_lasso_lgd)[coef_lasso_lgd != 0]

# Removed Intercept
var_lasso_lgd <- var_lasso_lgd[-1]

# Sub-Setting the Non-Zero Columns along with their Coefficient values
lasso_coefs_lgd <- as.data.frame(coef_lasso_lgd[var_lasso_lgd,])

# Converting Row to Column
lasso_data_lgd <- rownames_to_column(lasso_coefs_lgd, var = "name")

# Changing the column name
colnames(lasso_data_lgd)[2] <- "coefficients"
```

By using the Lasso model, the number of dependent variables has decreased to 119. This number of columns is better to build a model. Hence, subsetting the required columns from the model.

```r
names_lasso_lgd <- lasso_data_lgd[,1]
train_lasso_data_lgd <- train_lgd[,names_lasso_lgd]
validate_lasso_data_lgd <- validate_lgd[,names_lasso_lgd]

# Train
train_mod_lgd <- cbind(train_lasso_data_lgd, train_lgd$loss)
colnames(train_mod_lgd)[120] <- "loss"

# Validate
validate_mod_lgd <- cbind(validate_lasso_data_lgd, validate_lgd$loss)
colnames(validate_mod_lgd)[120] <- "loss"
```

Since we have the selected features that are important, let us build a ridge regression model.

The input data for the ridge model must be in the form of a Matrix and the output as a vector. Transforming the data into input and output as required for the ridge model.

Data Preparation

```
x_input <- as.matrix(train_mod_lgd[,-120])
y_output <- as.vector(train_mod_lgd[,120])
```

Ridge Regression for Loss Given Default (LGD) Model:

Ridge regression is a linear method to model the relationship between a dependent variable and one or more independent variables. It is particularly useful when the number of independent variables is large or multicollinearity among the independent variables.

```
set.seed(123)
ridge_model <- cv.glmnet(x=x_input, y=y_output, data=train_mod_lgd, alpha=0, family = "gaussian", nfold

ridge_model
```

```
##
## Call:  glmnet::cv.glmnet(x = x_input, y = y_output, data = train_mod_lgd,      alpha = 0, family = "
##
## Measure: Mean Absolute Error
##
##      Lambda Index Measure       SE Nonzero
## min 0.03373    78 0.05476 0.001211     119
## 1se 0.23799    57 0.05588 0.001156     119
```

- Handles multicollinearity: Ridge regression is particularly useful when multicollinearity exists among the independent variables. Multicollinearity occurs when two or more highly correlated predictor variables can lead to unstable and unreliable coefficient estimates in a linear regression model. Ridge regression helps to mitigate this issue by shrinking the coefficients towards zero, reducing their magnitudes, and increasing the model's stability.

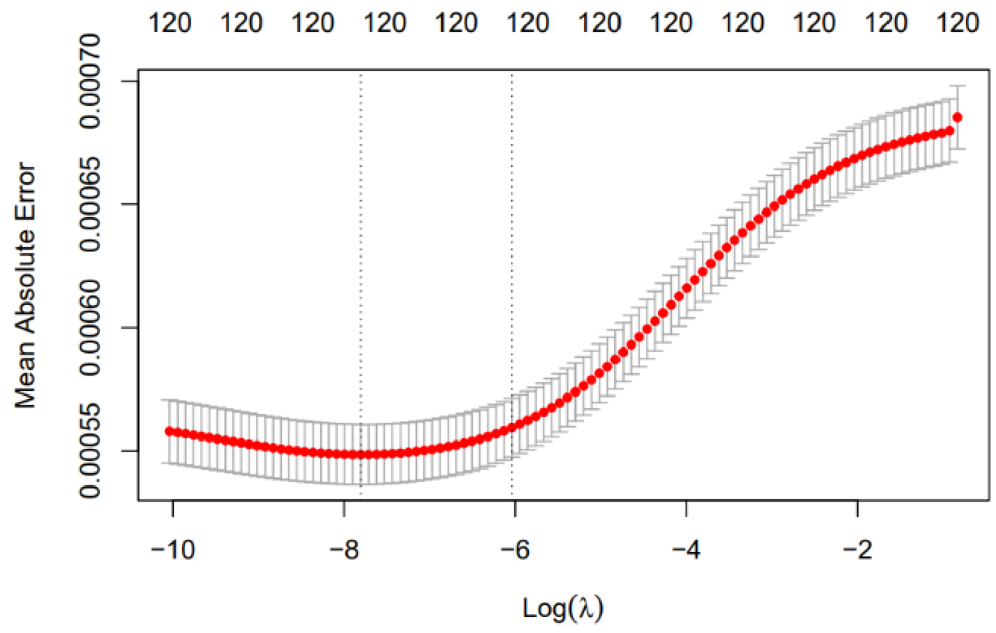
- Prevents overfitting: Overfitting occurs when a model is too complex and fits the training data too closely, resulting in poor performance on new, unseen data. Ridge regression helps prevent overfitting by introducing a regularization term that penalizes large coefficient values. This reduces the model's complexity and generalizes better to new data.

- Produces stable coefficient estimates: Ridge regression produces more stable coefficient estimates than ordinary least squares (OLS) regression, mainly when there are high levels of multicollinearity. This stability ensures that the model's predictions are more reliable and trustworthy.

Evaluation of the Ridge Regression used for Loss Given Default Model.

The objective function was to minimize the mean absolute error (MAE). A lower MAE (Mean Absolute Error) represents a better model performance in terms of its ability to predict the target variable. MAE measures the average absolute difference between the predicted and actual values of the target variable. So, a lower MAE indicates that the model's predictions are closer to the actual values, which suggests that the model has better accuracy and precision in predicting the target variable. In contrast, a higher MAE indicates that the model's predictions are further away from the actual values, which suggests that the model performs poorly in predicting the target variable.

We used ridge regression to calculate LGD and MAE (Mean Absolute Error) as a metric to measure the model's performance. MAE for the LGD Model is 0.05, which was a very low value, and it may indicate that the model's predictions are closer to the actual values.



```
# Validating the LGD model.
x_input_val <- as.matrix(validate_mod_lgd[,-120])
y_output_val <- as.vector(validate_mod_lgd[,120])

# Prediction
predicted_loss <- predict(ridge_model, s = ridge_model$lambda.min, newx = x_input_val)
predicted_loss <- abs(round(predicted_loss,2))

# Evaluating Performance on Validation
Error_lgd = mean(abs((predicted_loss - y_output_val)))
print(Error_lgd)
```

```
## [1] 0.05277808
```

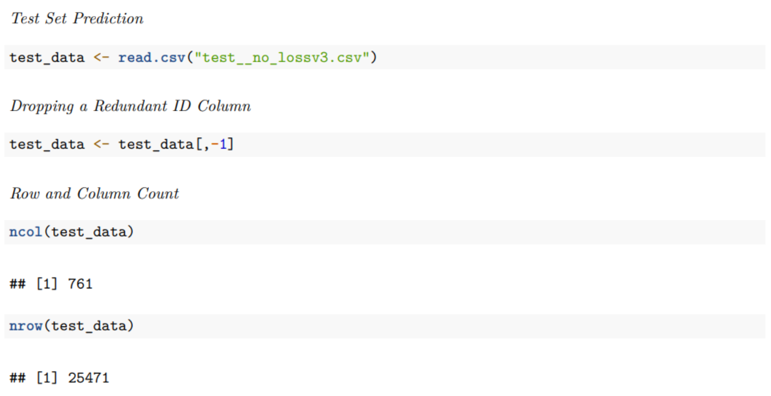
```
head(check_perf,n=10)
```

```
    Actual  Expected
36    0.04      0.08
67    0.20      0.18
150   0.04      0.10
190   0.01      0.03
211   0.11      0.10
269   0.10      0.06
334   0.02      0.00
407   0.22      0.14
423   0.05      0.04
434   0.03      0.05
```

The above table gives us a glance look of the actual loss values and the expected loss values. As we can see there is no major difference in the values we can think of the model carrying a good predictive power towards the target attribute i.e. loss.

Test Set Prediction

To finally evaluate the performance of the models, we first predicted the PD model on the test to get the predictions of the number of customers who defaulted and those who did not.



Test Set Prediction

```
test_data <- read.csv("test__no_lossv3.csv")
```

Dropping a Redundant ID Column

```
test_data <- test_data[,-1]
```

Row and Column Count

```
ncol(test_data)
```

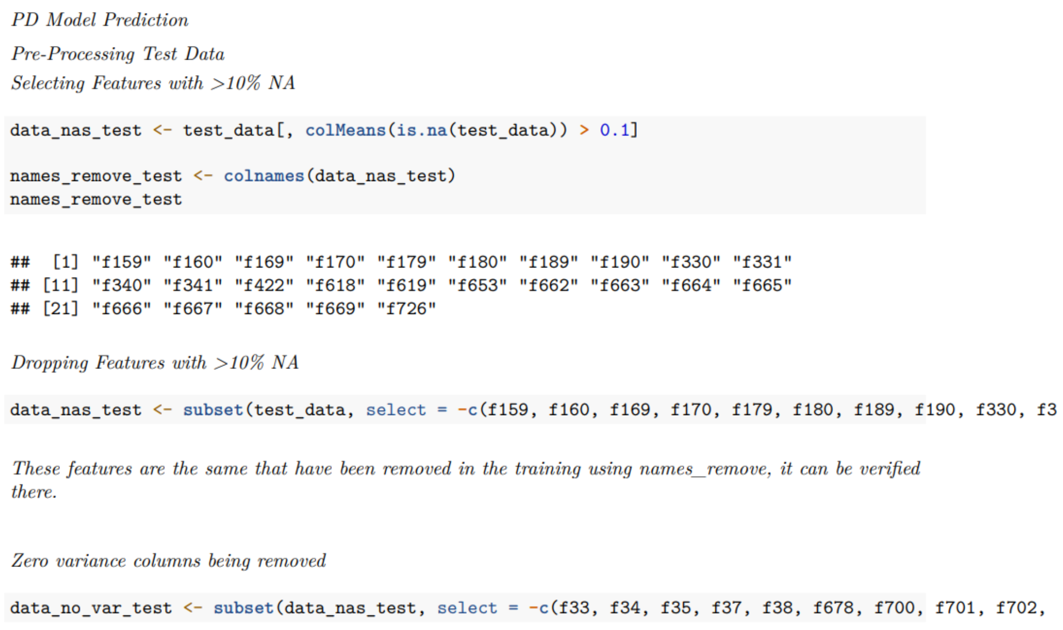
```
## [1] 761
```

```
nrow(test_data)
```

```
## [1] 25471
```

We loaded the test set data, dropped the redundant ID column got the count of rows and columns.

First, we used the PD model to get the default and no default customers and then eliminated the non-default customers for the final LGD Model.



PD Model Prediction

Pre-Processing Test Data

Selecting Features with >10% NA

```
data_nas_test <- test_data[, colMeans(is.na(test_data)) > 0.1]

names_remove_test <- colnames(data_nas_test)
names_remove_test
```

```
##  [1] "f159" "f160" "f169" "f170" "f179" "f180" "f189" "f190" "f330" "f331"
## [11] "f340" "f341" "f422" "f618" "f619" "f653" "f662" "f663" "f664" "f665"
## [21] "f666" "f667" "f668" "f669" "f726"
```

Dropping Features with >10% NA

```
data_nas_test <- subset(test_data, select = -c(f159, f160, f169, f170, f179, f180, f189, f190, f330, f3
```

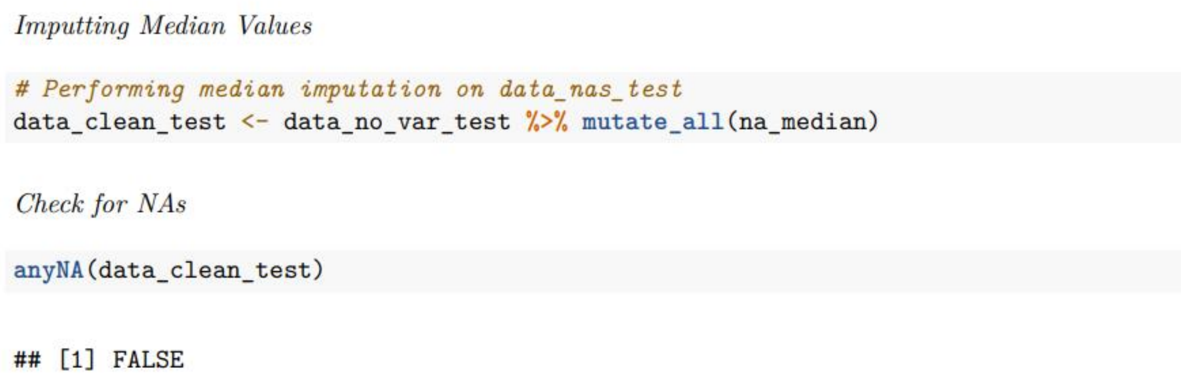
These features are the same that have been removed in the training using names_remove, it can be verified there.

Zero variance columns being removed

```
data_no_var_test <- subset(data_nas_test, select = -c(f33, f34, f35, f37, f38, f678, f700, f701, f702,
```

We again performed feature selection for the test set. We dropped the features having more than 10% of the missing data. These are the same features that have been removed from the training process. Then the features with zero variance columns were also removed.

In the preprocess_feature_model step, we will remove attributes with low variance, starting with those with zero variance since they add no value to the model. This is done before applying the nzv method, removing features with very low variance.



Imputting Median Values

```
# Performing median imputation on data_nas_test
data_clean_test <- data_no_var_test %>% mutate_all(na_median)
```

Check for NAs

```
anyNA(data_clean_test)
```

```
## [1] FALSE
```

We again opted for median imputation, which was both computationally efficient and resistant to outliers in the data. After that, we checked for any NA values and found no such values.

Feature Selection using LASSO regression:

Once the ID attribute has been excluded, the number of records in the test data is equivalent to that of the training data. We will proceed to further subset the data using only the columns selected during the lasso model feature selection process.

```r
# names_lasso is having the column names selected after using lasso model
id <- test_norm_pd[,1]
test_mod <- test_norm_pd[,names_lasso]

# we would need id so binding it
mod_test <- cbind(id, test_mod)
```

rf_model prediction on mod_test

```r
pred_test <- predict(rf_model, mod_test[,-1], type="prob")
pred_test <- as.data.frame(pred_test)

PD_Test <- ifelse(pred_test$`1`>0.6,1,0)
PD_Test <- as.data.frame(PD_Test)
```

The threshold achieved for the validation set prediction is being used for the test set, as the test set is unlabelled. If the probability of a class being 1 is greater than 0.6, then it is classified as default, otherwise, it is classified as not default.

PD Model Output

```r
result_PD <- cbind(id, PD_Test)
```

Tabular Information

```r
PD_Def_Non <- table(result_PD$PD_Test)
PD_Def_Non
```

```
##
##     0     1
## 25450    21
```

21 Classes have been predicted as default, and 25450 observations have been predicted as not default.

We then separated the defaulted and non-defaulted so that we can have them for the final file submission.

Filtering 0 and 1s

```r
PD_Non_Default <- result_PD %>% filter(PD_Test==0)
colnames(PD_Non_Default) <- c("id","result")

PD_Default <- result_PD %>% filter(PD_Test==1)
```

Loss Given Default Model: Test Set Prediction

We stored the 21 defaulted records, sub-setted only the defaulted IDs for the LGD model, and then pre-processed the test data.

```r
# Storing IDs of 21 Defaulted Records
lgd_input <- PD_Default[,1]

# Sub-Setting only the defaulted IDs for LGD Model
test_data_lgd <- test_data[test_data$id %in% lgd_input, ]
```

Pre-Processing Test Data

```r
test_preprocess <- predict(preprocess_lgd, test_data_lgd[,-1])
```

Sub-Setting the lasso attributes

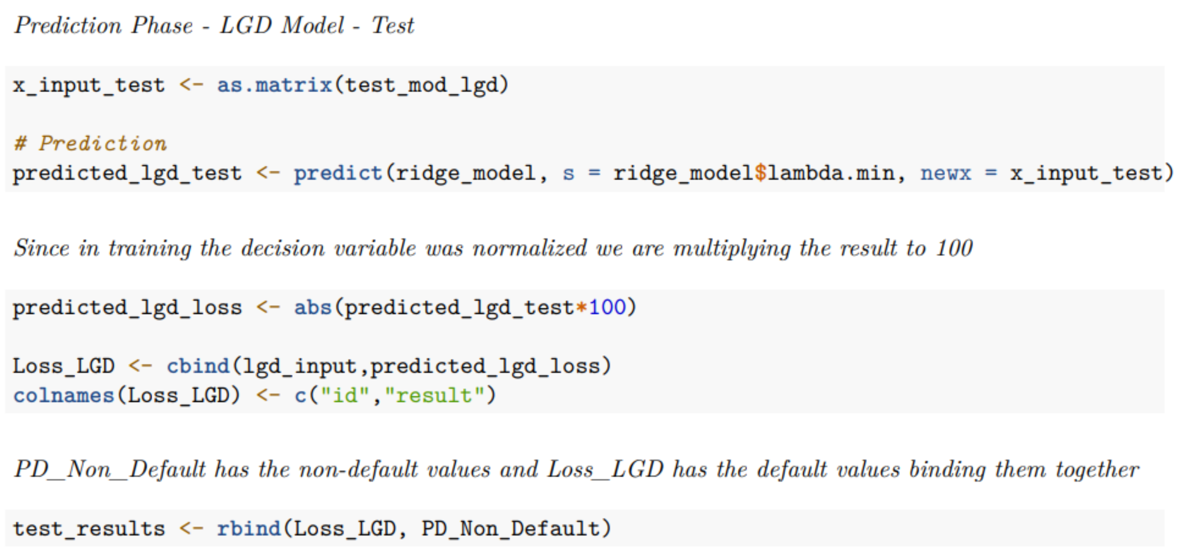
After removing the ID column, we have 249 remaining columns, which matches the count in the train and validation sets. To further refine our analysis, we will now limit our focus to only those attributes selected by the lasso model. After sub-setting the lasso attributes we now have the updated count to be 119.

```r
# names_lasso_lgd has the attributes resulted in lasso model
test_mod_lgd <- test_preprocess[,names_lasso_lgd]
ncol(test_mod_lgd)
```

```
## [1] 119
```

After applying the lasso regression, we have the same set of attributes that we received post to the application of the lasso model.

After selecting the features, now comes the prediction phase. After the ridge model has been used for prediction on the test set we will multiply the result by 100 since we normalized the target variable loss.



Prediction Phase - LGD Model - Test

```r
x_input_test <- as.matrix(test_mod_lgd)

# Prediction
predicted_lgd_test <- predict(ridge_model, s = ridge_model$lambda.min, newx = x_input_test)
```

Since in training the decision variable was normalized we are multiplying the result to 100

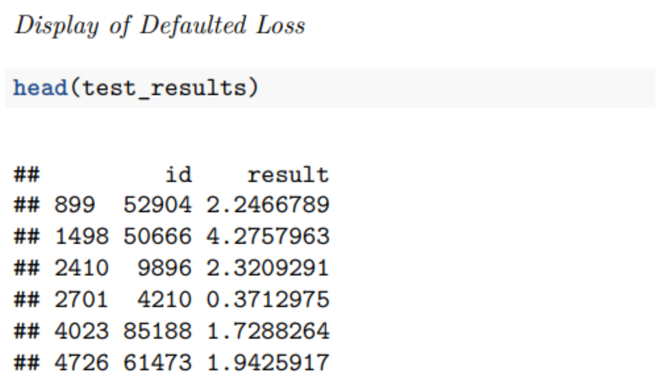
```r
predicted_lgd_loss <- abs(predicted_lgd_test*100)

Loss_LGD <- cbind(lgd_input,predicted_lgd_loss)
colnames(Loss_LGD) <- c("id","result")
```

PD_Non_Default has the non-default values and Loss_LGD has the default values binding them together

```r
test_results <- rbind(Loss_LGD, PD_Non_Default)
```

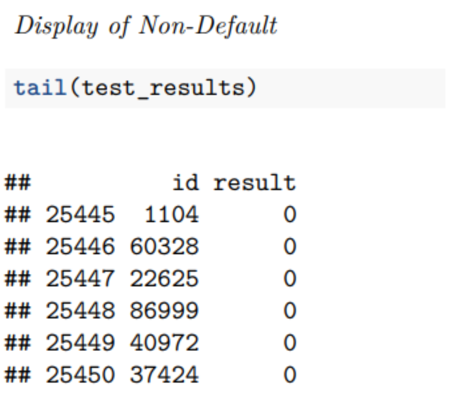
The test results were then saved in the test results file, which will be a part of our submission along with other files.

Default and Non-Default Values


Display of Defaulted Loss

```
head(test_results)
```

```
##          id    result
## 899  52904 2.2466789
## 1498 50666 4.2757963
## 2410  9896 2.3209291
## 2701  4210 0.3712975
## 4023 85188 1.7288264
## 4726 61473 1.9425917
```


Display of Non-Default

```
tail(test_results)
```

```
##          id result
## 25445  1104      0
## 25446 60328      0
## 25447 22625      0
## 25448 86999      0
## 25449 40972      0
## 25450 37424      0
```

The above two figures show the customer IDs and their losses incurred by defaulting and those who did not.

Conclusion:

In conclusion, the Probability Default and Loss Default Models are valuable tools for financial institutions to assess a customer's financial stability and likelihood of default. By utilizing these models, institutions can make informed decisions regarding lending, risk assessment, and portfolio management. The models' performance on unseen data has demonstrated stability and satisfactory results, indicating their reliability in predicting default rates. The use of various methodologies, such as feature reduction and missing value treatment, has been instrumental in constructing robust end models. Overall, the Probability Default and Loss Default Models offer a powerful approach for financial institutions to manage risk and make sound financial decisions.

Learnings:

- Imbalancement in the dataset can cause a serious problem in the modelling phase if dealt with it in the right way then we can achieve greater performance.

- Normalizing decision variables when the values are dispersed across a wide range can enhance the model's performance as the model can converge faster to an optimal solution.

- Least Absolute Shrinkage and Selection Operator can be an effective way to reduce features when the feature count is too high, due to computational inefficiency other methods couldn't be used.

- Reducing variables with no variance, with high collinearity can be useful for modelling tasks as these variables don't have any contribution towards the modelling task.