ADM Final Group Project

Setting default values to get a clean output

```
knitr::opts_chunk$set(message = FALSE)
knitr::opts_chunk$set(warning = FALSE)
```

Loading all the required packages

```
library("readr")
library("ISLR")
library("caret")
library("class")
library("e1071")
library("dplyr")
library("tidyverse")
library("ggplot2")
library("esquisse")
library("gmodels")
library("MASS")
library("broom")
library("modelr")
library("Hmisc")
library("missForest")
library("rpart")
library("rattle")
library("pROC")
library("ROCR")
library("cutpointr")
library("ROSE")
library("moments")
library("glmnet")
library("glmnetUtils")
library("VIM")
library("mice")
library("xgboost")
library("randomForest")
library("ranger")
```

Loading the Data

```
data <- read.csv("train_v3.csv")
```

Exploratory Data Analysis

Column and Row Count

```r
ncol(data)
```

```
## [1] 762
```

```r
nrow(data)
```

```
## [1] 79999
```

There are 762 columns and 79999 rows in the initial data that we loaded.

Column Names

```r
col_names <- (colnames(data[1:20]))
col_names
```

```
##  [1] "id"  "f1"  "f3"  "f4"  "f5"  "f6"  "f7"  "f8"  "f9"  "f13" "f14" "f15"
## [13] "f16" "f17" "f18" "f19" "f20" "f21" "f22" "f23"
```

Upon examining the column names, it is not readily apparent what each one represents. Additionally, the naming convention for the other columns follows the pattern of starting with the letter "f" followed by the column count. As a result, it may be challenging to discern the contents of each column based on its name alone.

Row Values

```r
row_values <- data[1:5,1:5]
row_values
```

```
##      id  f1        f3   f4 f5
## 1 78539 120 0.1462511 2200  4
## 2 61541 154 0.3493991 4200  4
## 3 76531 126 0.9759692 1500 10
## 4 22066 127 0.9513029 3100  7
## 5 45589 162 0.5184856 3500  7
```

Our analysis aimed to investigate the type of information stored in the first five attributes. However, it was not immediately apparent from a cursory examination of the data since "f1" contained integer type values, "f3" contained double type values in decimal form, while "f4" and "f5" also contained integer values. The challenge, however, was not solely the data type, but rather the inability to ascertain the meaning of each value and its significance in determining the ultimate decision, i.e., whether a loan will default or not and if it is going to default what is the % of default which can be considered as a loss to the organisation. It is therefore crucial to gain a comprehensive understanding of the nature of the information contained in each attribute to draw meaningful conclusions.

Target Attribute

```r
target <- data[1:5,762]
target
```

```
## [1]  0  0  0 11  0
```

The target attribute has values between 0 to 100, if the customer takes 100,000 Loan and repays 90,000 then we say that the customer defaulted 10% of the total loan taken and the loss value in the target attribute would be 10.

Data Cleaning and Feature Reduction

```
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"
```

Eliminating the columns with 0 variance

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

Missing Values/NAs

Looking if there are any NAs or Missing Values in the Data

```
missing_value_cols <- colMeans(is.na(data))
missing_value_cols <- round(missing_value_cols,3)
missing_value_cols[1:20]
```

```
##    id    f1    f3    f4    f5    f6    f7    f8    f9   f13   f14   f15   f16
## 0.000 0.000 0.000 0.000 0.000 0.000 0.002 0.001 0.000 0.000 0.001 0.000 0.000
##   f17   f18   f19   f20   f21   f22   f23
## 0.002 0.000 0.000 0.004 0.017 0.016 0.007
```

Just we are looking at the initial 20 columns to know if there exists any NAs and it is likely seen that there are a greater count of columns with NAs in the data set.

Max NA % in the features

```
max(missing_value_cols)
```

```
## [1] 0.178
```

As we can see above we get to know that the maximum % of one or more than one of the features with missing values in the data set is 17.8%, we are now basing our assumptions and want to eliminate the columns with > 10% of missing values in them.

Sub-Setting NAs > 10 % Missing Values into a Data Frame

```
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"
```

Here, we are trying to subset the columns where the mean value of the missing values in that column is > 10 % of the entire observations, so that we can use these columns to drop them in the next step.

Before dropping the columns let's check the correlation of the above printed attributes with the target variable - loss

Correlation - Independent & Dependent

```
# 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)
```

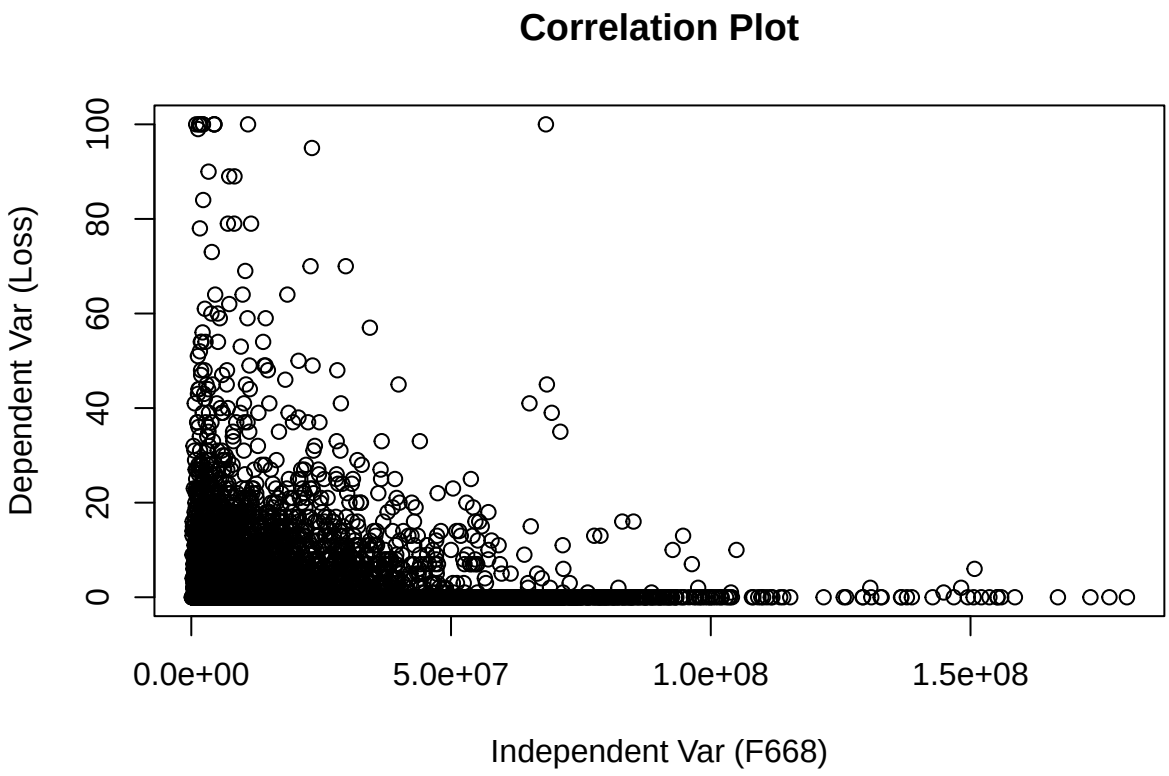
```
##         [,1]
## f159  0.0008
## f160 -0.0004
## f169 -0.0008
## f170 -0.0007
## f179 -0.0046
## f180 -0.0043
## f189  0.0028
## f190 -0.0020
## f330  0.0063
## f331  0.0028
## f340  0.0010
## f341  0.0021
## f422 -0.0102
## f618 -0.0082
## f619 -0.0057
## f653 -0.0028
## f662 -0.0015
## f663 -0.0026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

It can be noted that the columns with >10% of NAs are not having a significant relationship with the target variable - loss, the correlation values are close to 0 which is more likely a case of no existence of correlation between these attributes and the target variable - loss

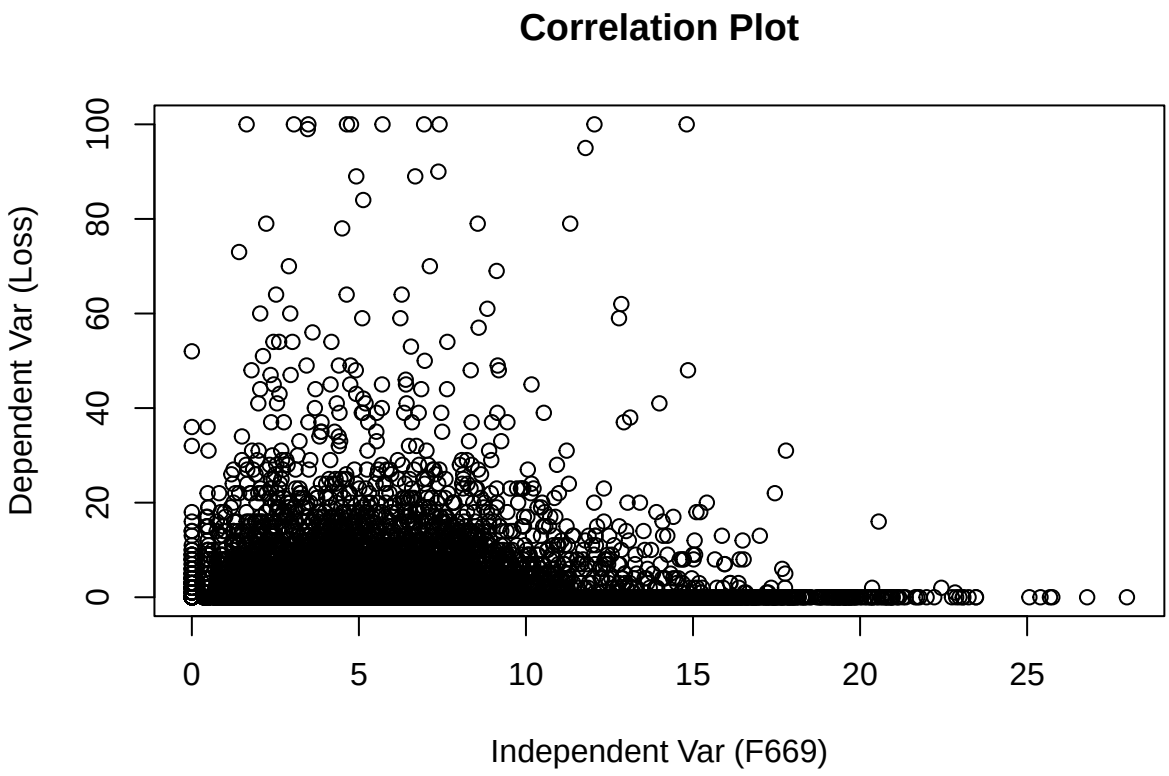
Let's plot few independent variables with the dependent variable loss

Scatter Plot

```
plot(data_cor$f668, data_cor$loss, main="Correlation Plot",
     xlab="Independent Var (F668)", ylab="Dependent Var (Loss)")
```




```
plot(data_cor$f669, data_cor$loss, main="Correlation Plot",
     xlab="Independent Var (F669)", ylab="Dependent Var (Loss)")
```



As we can see from the above plot there exists no solid correlation between the independent and the dependent attributes. We can now go ahead and eliminate the columns with >10% of missing values.

Dropping Columns with NAs > 10 %

```
data_less_nas <- subset(data_no_var, select = -c(f159, f160, f169, f170, f179, f180, f189, f190, f330, f
```

Instead of removing all the attributes where we have NAs we are setting a threshold i.e. if a column has > 10 % of missing values we are dropping them. By dropping all the attributes with missing values we might lose a greater percentage of important attributes and if we do this by row instances then we would be left with nearly 50 % less data, so we took this approach.

Updated Column Count

```
ncol(data_less_nas)
```

```
## [1] 726
```

Earlier it was 762 attributes, after removing 25 attributes which had >10 % missing values (NAs) and the columns with zero variance we now have the updated count of columns to be 726.

But still there are attributes with < 10 % of missing values and it's important for us to treat them and still we need to reduce the features count which will help us build models with better efficiency.

Imputting Missing Values

```
# 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_nas %>% mutate_all(na_median)
```

For the imputation we have tried many methods such as missForest, multiple imputation using mice and knn imputation, but all of these methods were computationally very expensive so we had to work around with median imputation since it was computationally efficient and robust to any presence of outliers in the data.

Let's check if there are any NAs

```
anyNA(data_clean)
```

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

Data Transformation

```
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 the Classes with the help of Synthetic Data

```
data_balanced <- ovun.sample(PD ~ ., data = data_clean, method = "both", p=0.5, N=100000, seed = 123)$da
```

Balancing the data set is quite important, if not dealt this may cause the model to predict the values only of the class which has greater number of observations, in our case "not default (0)" has greater number of observations, but we want accurate predictions for "default (1)".

Row Count for PD Values - Before Balancing

```
table(data_clean$PD)
```

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

Row Count for PD Values - After Balancing

```
table(data_balanced$PD)
```

```
##
##     0     1
## 50259 49741
```

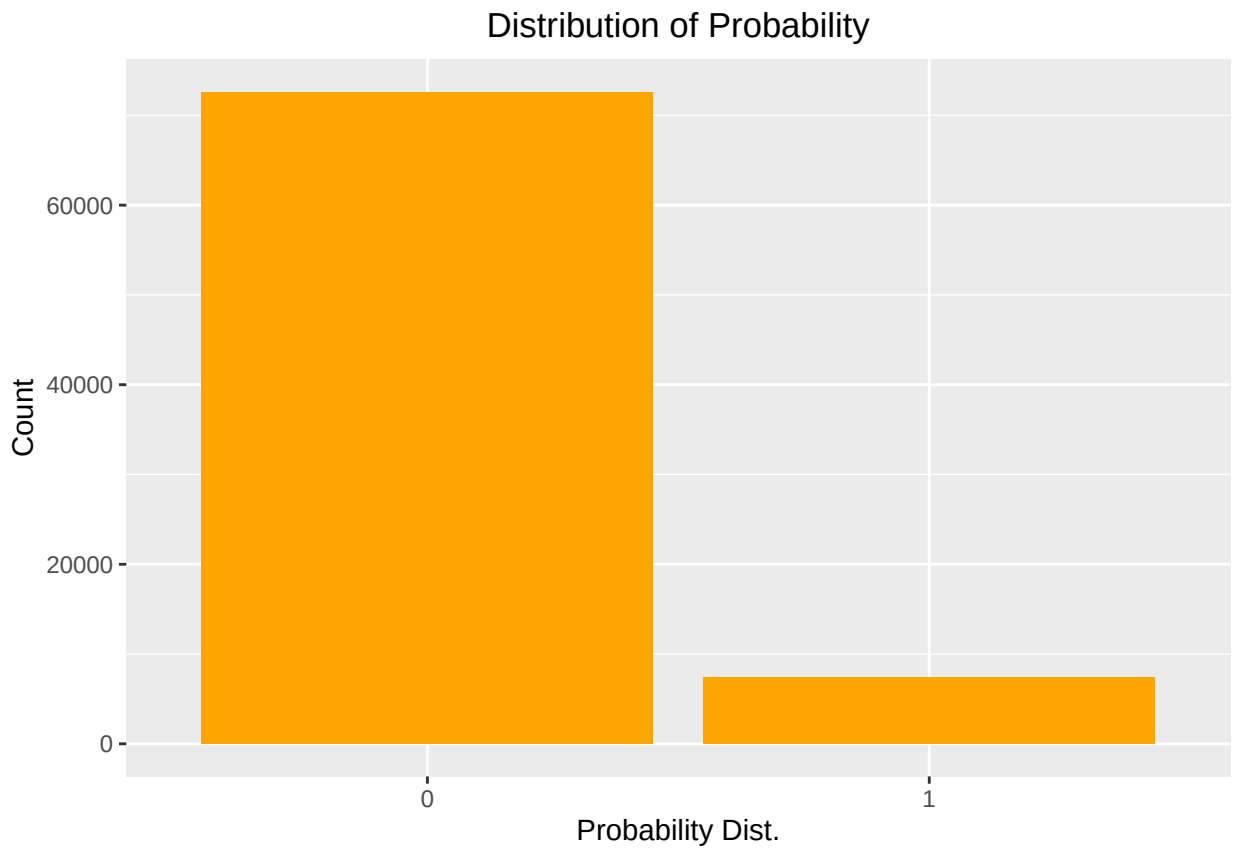
We can see the difference between the classes, imbalanced data sets can cause a serious problem while modelling.

Decision Variable -> Factor

```
data_clean$PD <- as.factor(data_clean$PD)
data_balanced$PD <- as.factor(data_balanced$PD)
```

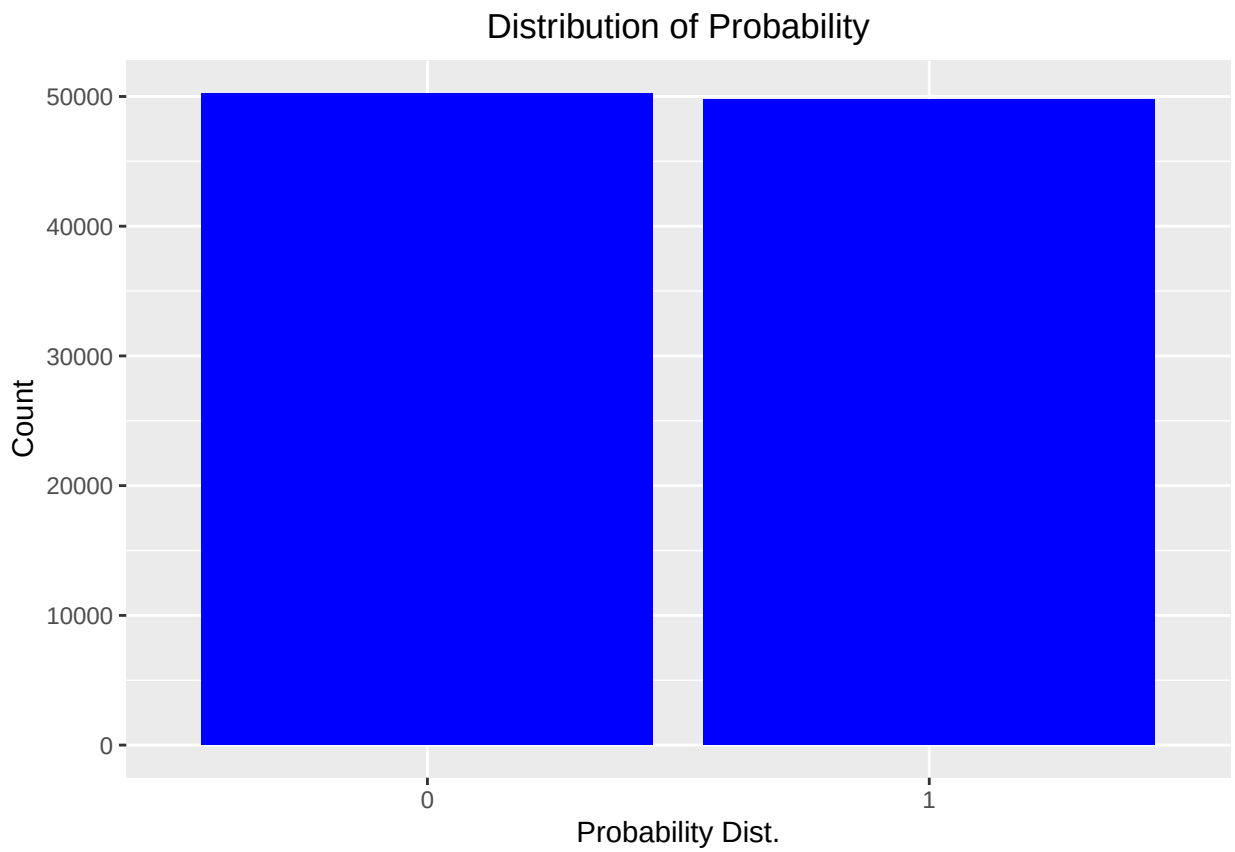
Visualizing the target variable - Before Balancing

```
ggplot(data_clean, aes(x = PD)) +
geom_bar(fill="orange") +
scale_x_discrete(labels = c("0", "1"),drop=F) +
labs(x="Probability Dist.", y="Count", title="Distribution of Probability") +
theme(plot.title = element_text(hjust = 0.5))
```



Visualizing the target variable - After Balancing

```
ggplot(data_balanced, aes(x = PD)) +
geom_bar(fill="blue") +
scale_x_discrete(labels = c("0", "1"),drop=F) +
labs(x="Probability Dist.", y="Count", title="Distribution of Probability") +
theme(plot.title = element_text(hjust = 0.5))
```



Now, we have a data set which is balanced in a better way. Both, the classes are having almost equal distribution.

Now, we have to begin with feature reduction, in order to do feature selection it's important for us to replicate the same features onto the test set, so we are partitioning the data into train and validate

Data Partition

```
set.seed(123)
split_data <- createDataPartition(data_balanced$PD, p=0.75, list=F)
train <- data_balanced[split_data,]
validate <- data_balanced[-split_data,]
```

Feature Reduction along with Normalization

```
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)
```

Time to reduce the features, previously we have removed features with greater than 10% NAs and the features which have exactly 0 variance. But still there might be few columns with variance almost closer to 0 and having those features aren't going to add any value to the modelling task so now we are going to remove the attributes that have variance closing to zero (0) and also the attributes that are highly correlated with each other, because multi-collinearlity is going to be a problem for the modelling task.

Updated Row Count

```
ncol(train_less_features[,-c(1,255,256)])
```

```
## [1] 253
```

```
ncol(validate_less_features[,-c(1,255,256)])
```

```
## [1] 253
```

Now the features have been reduced to 253 which is a great reduction from 726 to 253 excluding (id, PD and loss).

Lasso for Variable Selection

In Lasso Model the coefficients are shrunk towards zero and towards each other. But when this happens if the independent variables do not have the same scale, the shrinking is not fair. Two independent variables with different scales will have different contributions, so it is important to normalize the attributes.

Lasso model requires the y variable to numeric, so converting the factor to numeric values

```
train_less_features$PD <- ifelse(train_less_features$PD == "0", 0, 1)
```

Transforming the data into input and output for lasso model

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

Lasso for Variable Selection

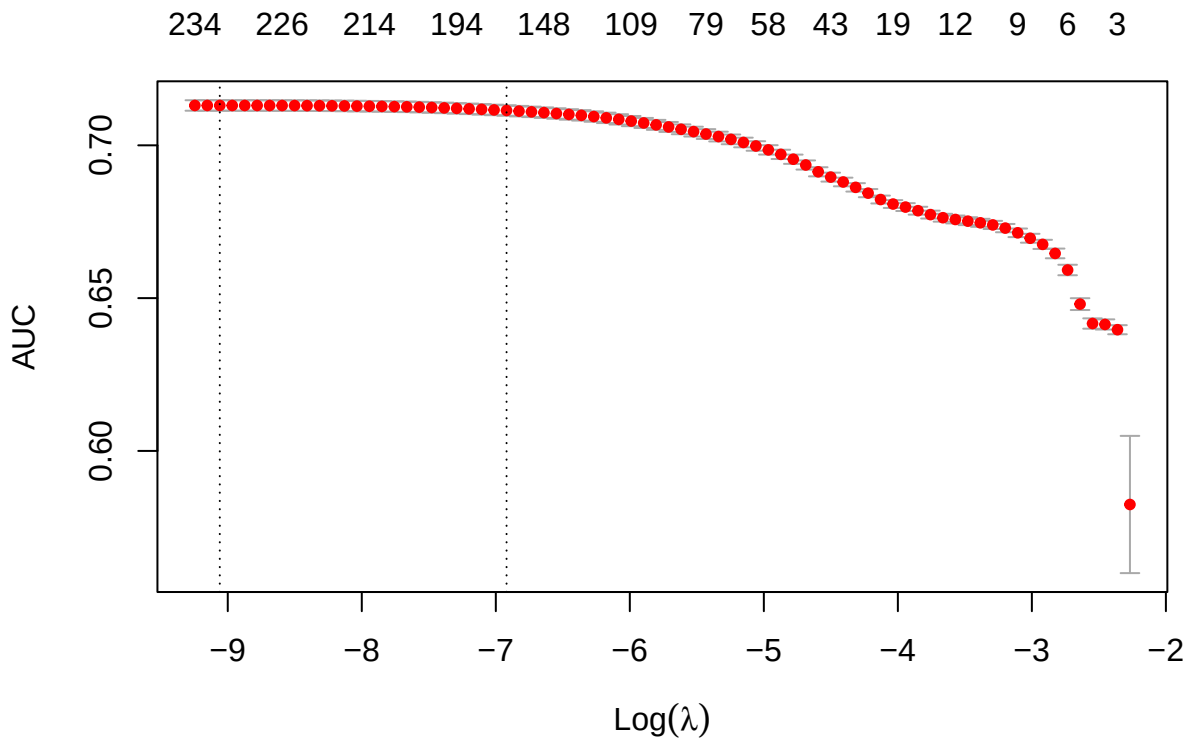
```
set.seed(123)
glm_model <- cv.glmnet(x=input, y=output, data=train_less_features, family="binomial", type.measure= "au

glm_model
```

```
##
## Call:  glmnet::cv.glmnet(x = input, y = output, data = train_less_features,      family = "binomial"
##
## 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
```

```r
plot(glm_model)
```



Storing the variables which have been retained when lambda=min

```r
# 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"
```

Now, we have 233 attributes that deemed to be important for the probability distribution task.

Creating a Data Frame for Train and Validation

```
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"
```

This model is going to be built by using all the 233 attributes which were selected while using the lasso regression model, we want to assess the performance of the model using just the lasso selected variables.

There are two steps which is going to be the main focus area from now on

1. Probability Default Model - We would want to know the probability of a customer defaulting the loan i.e. if a customer defaults the entry will be returned as 1 and if he doesn't default then it is going to be 0.

2. Loss Given Default Model- If the customer has defaulted the loan what is the loss incurred to the bank by the customer, let's say customer has paid 90% of the loan and defaulted 10% then the LGD is going to be 0.1 or 10.

Probability Default Model

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
```

rf_model prediction on validate_mod

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

Cut-Off for Default and Not Default

```
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

```
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")
```

Performance Metrics Evaluation

```
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 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```

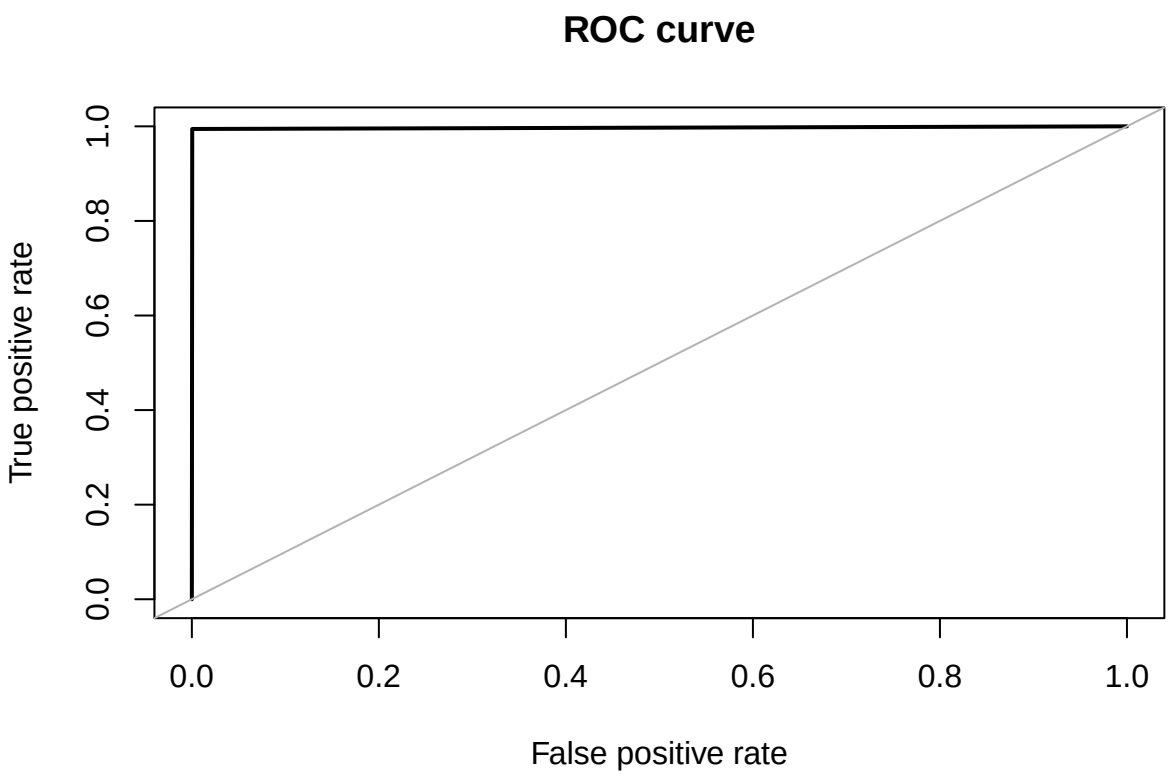
Performance Metrics - PD Model

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%.

Area Under Curve Value and Plot

```
roc.curve(validation_testing$V1, validation_testing$pred, plotit = T)
```




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

We also see that we have a AUC of 99.7%

The model built using the lasso attributes is considerably generalizing well on the unseen data i.e. validation data, the key advantage of going with the model that selected the features based on the Lasso model is that it selects a subset of attributes that are most relevant to predicting the target variable. These attributes are contextually more interpretative and easier to explain in the context of the business problem.

We can store rf_model to be used for predicting on the test set. As we have seen that this model has been built using 233 Features and has the best performance in terms of Precision, AUC, Sensitivity as well as F-1 Score, so we are going to store rf_model as the final model to be used on the test set.

Now, we begin to build the Loss Given Default Model (LGD), once this model is built we will work on the test set first to get the PD and then to evaluate the loss of the defaulted customers.

Loss Given Default

The initial data set that has been loaded in the model has been saved with a variable called "data".

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 loss incurred by the customers who have defaulted.

Data Transformation

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

Count of Observations - Default and Not Default

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

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

Normalizing the decision variable

```
data$loss <- (data$loss/100)
```

The decision variable "loss" has a large range of values, it can cause numerical instability and make it difficult for the algorithm to converge to a solution. Normalizing the variable can help to reduce the range of values and make it easier for the algorithm to find an optimal solution.

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

```
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

```
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)
```

We are trying to get rid of the attributes which have near to zero variance, which are highly correlated among each other and finally imputting the missing values with median. We are also trying to scale the attributes

so that all the input attributes are on the same scale. Note: We used directly the preprocess technique to do the feature selection instead of using iterative steps.

Updated Column Count

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

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

The above used feature reduction technique has resulted in 249 attributes which have no attributes that are highly correlated, no attributes with high collinearity and no missing values. This count is after excluding the loss attribute.

But, this is still a huge count, let's try to feed this to the most powerful sparse feature selection algorithm i.e. Lasso Model.

Transforming the data into input and output for lasso model

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

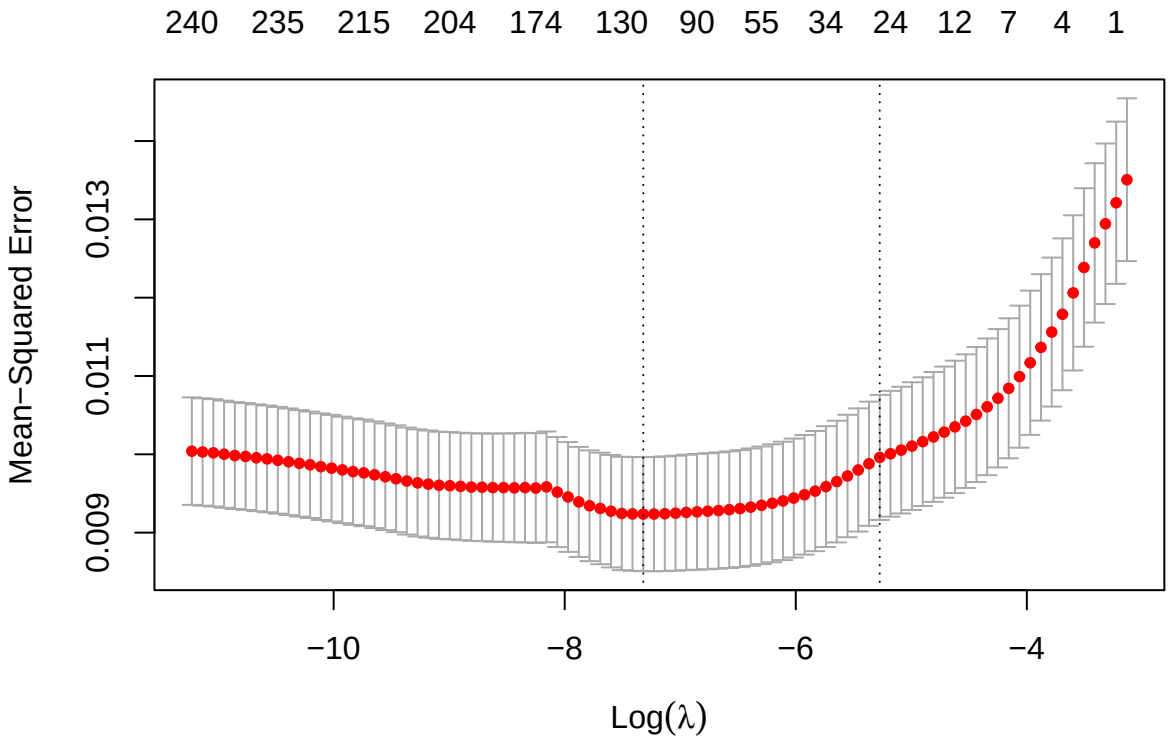
Lasso for Variable Selection

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

glm_model_lgd
```

```
##
## Call:  glmnet::cv.glmnet(x = input, y = output, data = train_lgd, family = "gaussian",      type.meas
##
## 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
```

```
plot(glm_model_lgd)
```



At lambda.min we have 119 attributes that have been selected for the loss given default task.

Storing the variables which have been retained when lambda=min

```
# 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"
```

Sub-Setting Attributes

```
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 deemed to be important let's build a ridge regression model.

Data Preparation

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

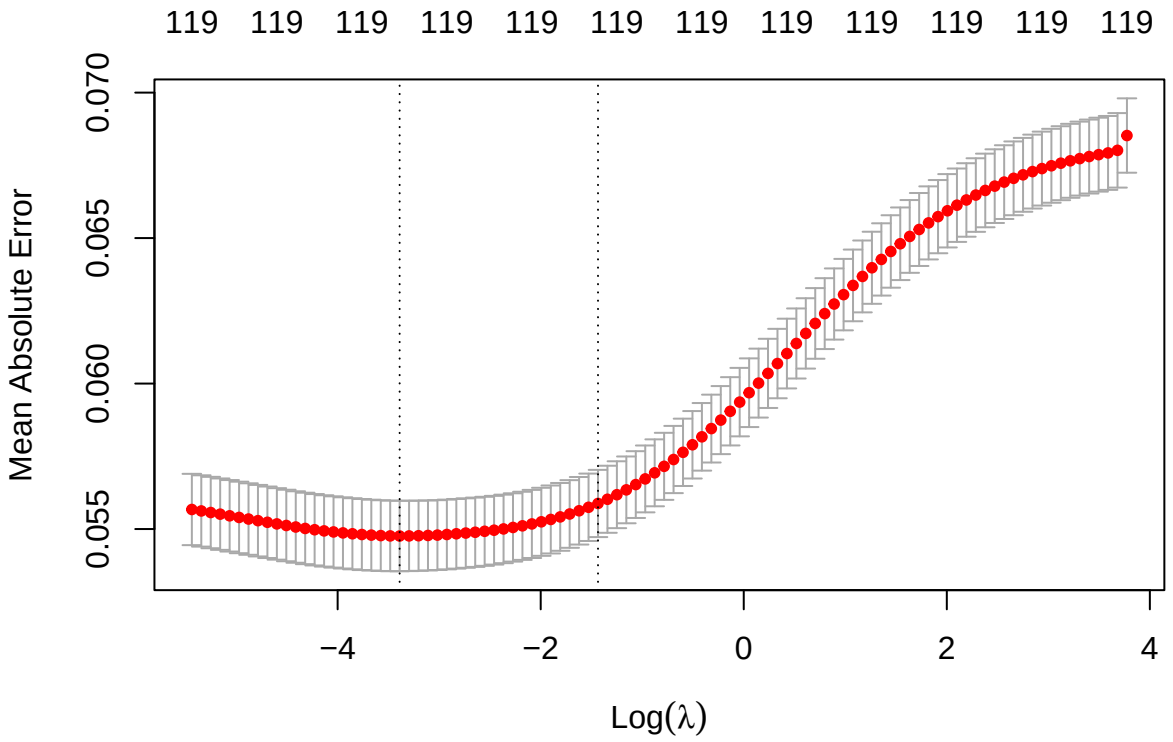
Model Building

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

ridge_model
```

```
##
## Call:  glmnet::cv.glmnet(x = x_input, y = y_output, data = train_mod_lgd,      alpha = 0, family = "g
##
## 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
```

```
plot(ridge_model)
```



Prediction Phase - LGD Model

```
# 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
```

The resulted MAE is 0.05

Appending Actuals and Predicted

```
check_perf <- cbind(y_output_val,predicted_loss)
colnames(check_perf) <- c("Actual","Expected")
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
```

Now, since we have both the models for classification and regression, let's pool the test data in and start the prediction phase

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
```

First we will be using the PD model to get the Default and No Default Customers and then eliminate 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, 
```

The attributes with less variance will be removed in the preprocess_feature_model, ideally the nzv method should remove zero variance as well but in the PD model we first eliminated attributes with zero variance so we are first removing 0 variance features.

Now, let's try to impute the missing values using median and then pass the attributes to the pre-process function.

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
```

Predicting Pre-Processed Model built on train onto the test

```
test_norm_pd <- predict(preprocess_features_model, data_clean_test)
```

Updated Column Count

```
ncol(test_norm_pd[,-1])
```

```
## [1] 253
```

After excluding the id attribute the count of test data is similar to that of the train attributes, now let's subset further using the columns that were selected during the lasso model.

Sub-Set Lasso Attributes

```
# 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

```
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)
```

We are levying the threshold which we have achieved for the validation set prediction, since the test set is not labelled we are going with the same threshold, if a class in 1 is > 0.6 we are going to classify them as default or else not default.

PD Model Output

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

Tabular Information

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

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

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

Let's separate the defaulted and non-defaulted so that we can have it for the final file submission

Filtering 0 and 1s

```
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
Sub-Setting

```
# 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

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

Column Count

```
ncol(test_preprocess)
```

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

After excluding the ID we have 249 columns which is similar to the train and validation column count, let's now subset further by using only the attributes that were selected during the lasso model.

Lasso Attribute Selection

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

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

Now, we have the same set of attributes which we received post to the application of lasso model

Prediction Phase - LGD Model - Test

```
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

```
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

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

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
```

Saving the file

```
file_test_results <- write.csv(test_results, file="test_results.csv")
```