Data Wizards (Group 4) Project 2

Di Chen Mai Castellano Tyler Kussee Spencer (Hutchison) Yang

```
#install.packages("htmltools")
```

Introduction to dataset

In our pursuit of statistical inquiry, we have chosen to explore the Vehicle Loan Default dataset, comprising approximately 41 columns, with one designated as the response variable. Encompassing diverse information, the dataset delves into loan details, including date of birth, employment type, and credit score, alongside loan-related specifics such as disbursal details and loan-to-value ratios. The dataset presents challenges, notably in the form of odd date and time length columns, requiring standardization and transformation into comprehensible formats conducive to model development.

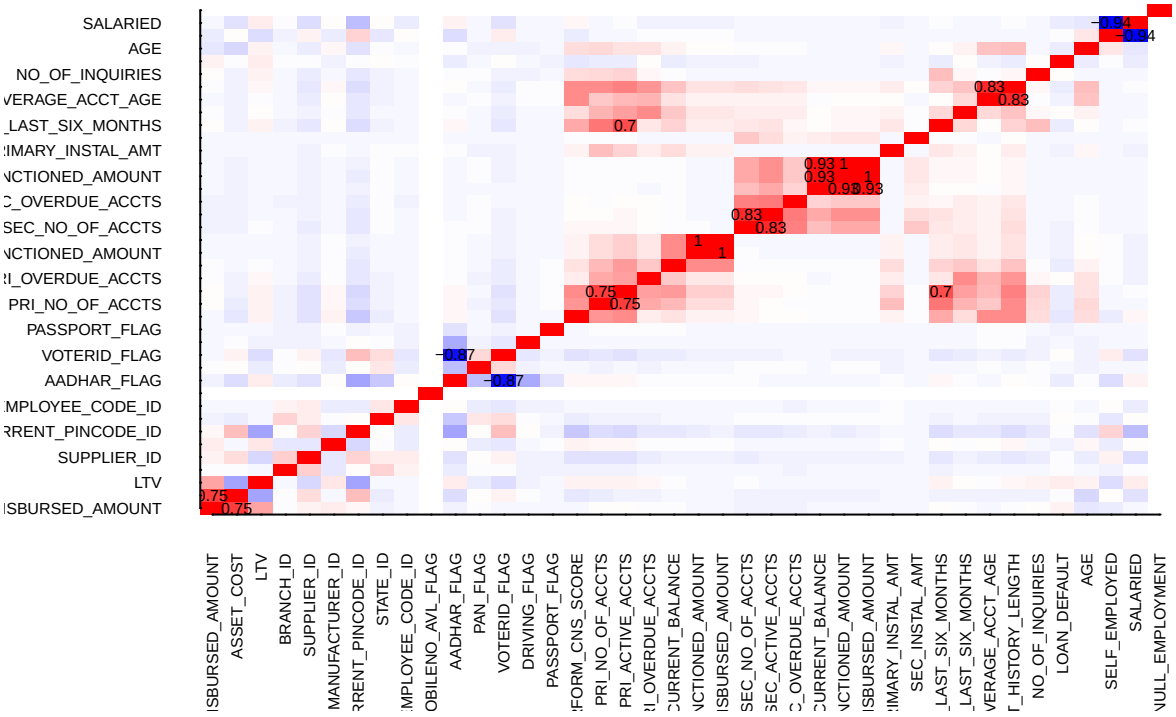
We want to discover the most influential explanatory variables driving loan default, and their impact within the dataset. We also want to find the optimal modeling approach for harnessing the training data, evaluating various methodologies to identify the most effective. Ultimately, our investigation extends to which among them best identifies the underlying dynamics of vehicle loan default prediction.

The data was pulled from the Vehicle Loan Default Prediction datasets available on Kaggle. As mentioned earlier, we have approximately 41 columns, with one of the columns designated as the response variable.

Feature extraction and description

In the case of this dataset, we need to find the features that have the biggest impact on our "LOAN_DEFAULT" variable.

First, we want to check the correlations between different variables.

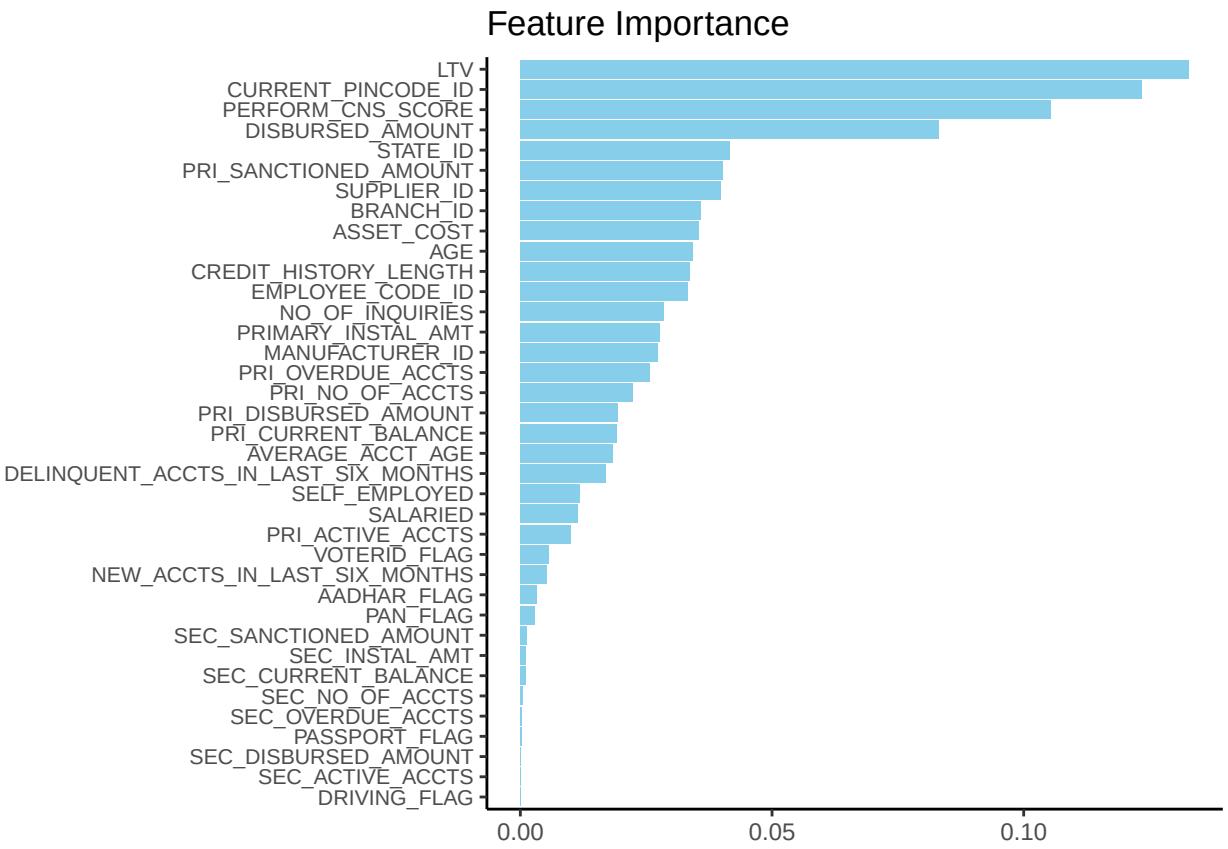


From the correlation matrix, we observe that darker shades of red signify robust positive correlations, while deeper shades of blue denote strong negative correlations.

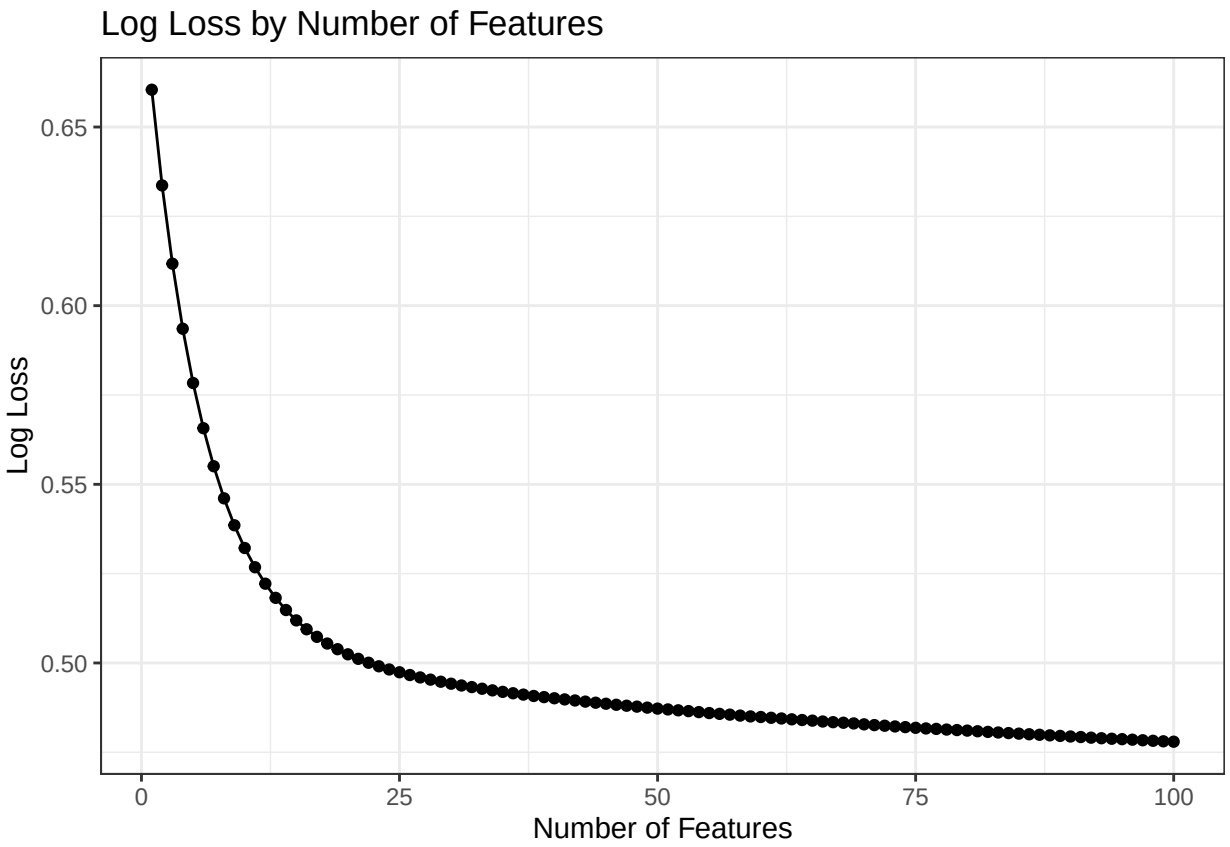
The size and intensity of the squares within the heatmap correspond to the magnitude of the correlation coefficients between variables. Notably, variables such as PRI.DISBURSED.AMOUNT, LTV, SEC.NO.OF.ACCTS, SEC.ACTIVE.ACCTS, PRI.CURRENT.BALANCE, and AVERAGE.ACCT.AGE exhibit notable correlations with other independent variables. However, to derive conclusive insights, further in-depth analysis and interpretation are warranted.

In order to find the features that have the biggest impact, we will be using the XGBoost machine learning algorithm package to train our matrix and label vector from the training data, setting the parameters, and performing cross-validation to find the optimal number of rounds for training the model.

Once the parameters and optimal number of rounds were found, we trained the model and then proceeded to calculate the feature importance scores.



Each bar in the plot represents a feature from the dataset used in the machine learning model. The length of each bar corresponds to the importance of the feature, with longer bars indicating more important features. The features are sorted in descending order of importance, with the most important features appearing at the top. The importance score, displayed on the x-axis, quantifies the contribution of each feature to the model's predictions. Higher scores indicate greater importance, thus allowing us to identify which features have the most significant impact on the model's performance.



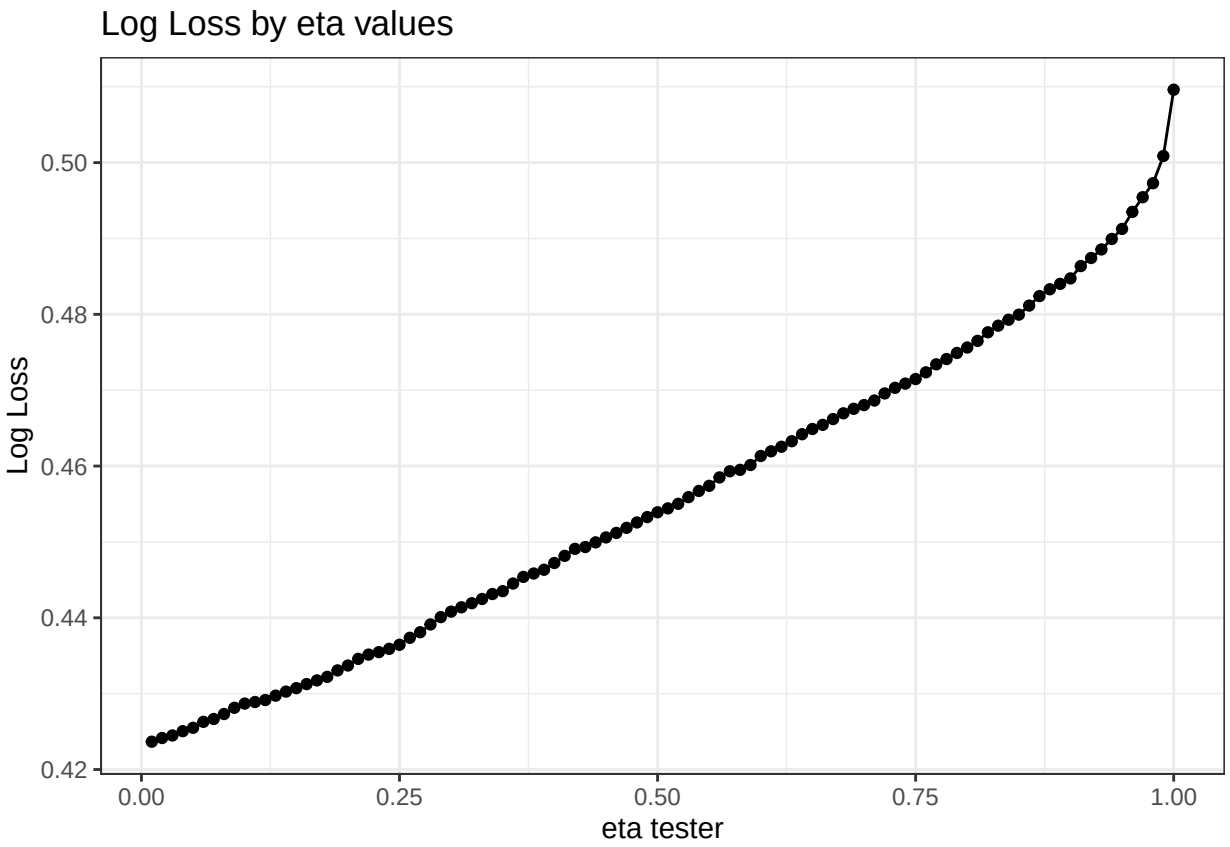
The optimal number of features for a machine learning model was determined by analyzing log loss metrics obtained from cross-validation. Based on the plot, it was observed that the optimal model should include 15 variables, which represents a balance between complexity and log loss minimization. These features, identified through the importance scores from the XGBoost model are, LTV, CURRENT_PINCODE_ID, PERFORM_CNS_SCORE, DISBURSED_AMOUNT, STATE_ID, ASSET_COST, SUPPLIER_ID, PRI_SANCTIONED_AMOUNT, EMPLOYEE_CODE_ID, BRANCH_ID, CREDIT_HISTORY_LENGTH, AGE, NO_OF_INQUIRIES, PRI_OVERDUE_ACCTS, and PRIMARY_INSTAL_AMT.

Building classifiers and results

The dataset, containing 15 explanatory variables as identified and 1 response variable, was divided into training and testing sets, with an 80-20 split. A logistic regression model was then trained on the training data. The confusion matrix was created and accuracy was calculated. The logistic regression model on the training data correctly predicted the outcome approximately 75.02% of the time when using a threshold of 0.3, However, with only 33.96% in precision and 16.1% in recall, further improvement is needed for model training.

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.750075071854489"
```



To ensures that the model's learning rate is finely tuned to achieve the best possible performance in terms of minimizing log loss, a sequence of a learning rate eta values from 1 to 0.01 was tested. An XGBoost model was trained for each eta value using 100 boosting rounds and a maximum depth of 6. The log loss for each model was recorded. The eta value that resulted in the lowest log loss is 0.01.

Subsequently, two XGBoost models were trained. The initial model used a maximum depth of 6, an eta of 0.01, and 100 boosting rounds, with the binary:logistic objective and logloss evaluation metric. Cross-validation with early stopping (5 rounds) was employed to determine the optimal number of boosting rounds. The cross-validation with early stopping approach confirmed that using 100 boosting rounds is appropriate for the initial model configuration.

At a threshold of 0.5, our model is accurate about 78.32% of the time, meaning it predicts the right class most of the time. When it predicts a positive outcome, it's correct about 77% of the time. However, it misses a lot of actual positive cases, as shown by the low recall of 0.059%. Although we prioritize accuracy and precision to minimize false alarms and ensure correctness in our predictions, the model's ability to capture all positive cases is limited.

We also tried a different threshold at 0.4, the accuracy of this model is approximately 78.08%, which is slightly lower than the previous model. The precision is also lower at 45.99%. The recall is slightly higher, however, remains low at 6.11%, indicating that the model still misses a significant number of actual positive instances. Overall, this model does not improved accuracy nor precision, similar limitations in capturing all positive cases compared to the previous one. Furthermore, when we set the threshold at 0.6, the model did not predict any instances as belonging to defaulting class.

Training and Validation Split

Build Classifier and Verification

The training data that was split from original data is then divided into two subsets: 80% for training and 20% for validation. The second set of training data is trained using the XGBoost method with the same parameters as the initial XGBoost model. The model's performance is then evaluated on both the training and validation datasets.

The accuracy for the training dataset is 78.36%, with a precision of 78.57%. However, the recall remains consistently low, similar to the initial model. When tested on the validation dataset, the model maintains consistent overall

prediction accuracy on both seen and unseen data, with an accuracy of 78.13% and a precision of 81.82%. The recall rate remains low at 0.22%.

To optimize performance, the threshold on the validation testing is adjusted to 0.4. This adjustment leads to a decrease in both accuracy and precision, resulting in values of 77.99% and 44.34%, respectively. However, recall shows a slight improvement, reaching 5.17%. Setting the threshold to 0.6 results in the model failing to predict any instances belonging to the defaulting class, similar to the behavior observed in the initial model. Despite remodeling the data and adjusting the threshold, the challenge of capturing all positive cases persists.

Appendix

```
##An if statement for checking if a package is installed

if (!require(tidycensus)) {
  install.packages("tidycensus")
}
if (!require(tidyverse)) {
  install.packages("tidyverse")
}
if (!require(dplyr)) {
  install.packages("dplyr")
}
if (!require(ggplot2)) {
  install.packages("ggplot2")
}
if (!require(caret)) {
  install.packages("caret")
}
if (!require(xgboost)) {
  install.packages("xgboost")
}
#if (!require(DiagrammeR)) {
  #install.packages("htmltools")
#}
knitr::opts_chunk$set(echo=F, error=T, collapse=T, warning=F, results='hide', message=F)

#Load libraries
library(tidycensus)
library(tidyverse)
library(dplyr)
library(ggplot2)
library(caret)
library(xgboost)
#library(DiagrammeR)

#First, we'll import the dataset from the training CSV:
# Reading in the csv
train <- read.csv('train.csv')
train <- train %>% select(-"DISBURSAL_DATE")
train$DATE_OF_BIRTH <- as.Date(train$DATE_OF_BIRTH, format = "%d-%m-%Y")
train$AGE <- as.Date('01-01-2019', format = "%d-%m-%Y") - train$DATE_OF_BIRTH
train$AGE <- as.integer(floor(train$AGE / 365.25))
train <- train %>% select(-"DATE_OF_BIRTH", -"PERFORM_CNS_SCORE_DESCRIPTION")

#We then scrub the unknown values in our length and age fields:
# Calculate strange data fields
```

```r
train$acctyr <- as.numeric(gsub("yrs.*", "", train$AVERAGE_ACCT_AGE))
train$acctmo <- as.numeric(gsub(".*yrs|mon", "", train$AVERAGE_ACCT_AGE))
train$crdtyr <- as.numeric(gsub("yrs.*", "", train$CREDIT_HISTORY_LENGTH))
train$crdtmo <- as.numeric(gsub(".*yrs|mon", "", train$CREDIT_HISTORY_LENGTH))

#Then we do our calculations and create various other fields for our modeling usage:
# Replace strange data fields
train$AVERAGE_ACCT_AGE <- round(train$acctyr + train$acctmo / 12, 2)
train$CREDIT_HISTORY_LENGTH <- round(train$crdtyr + train$crdtmo / 12, 2)
# Remove calc fields
train <- train %>% select(-acctyr, -acctmo, -crdtyr, -crdtmo)
# Create emloyment dummy fields
train$SELF_EMPLOYED <- ifelse(train$EMPLOYMENT_TYPE == "Self employed", 1, 0)
train$SALARIED <- ifelse(train$EMPLOYMENT_TYPE == "Salaried", 1, 0)
train$NULL_EMPLOYMENT <- ifelse(is.na(train$EMPLOYMENT_TYPE), 1, 0)
# Remove employment_type & Unique ID
train <- train %>% select(-EMPLOYMENT_TYPE, -UNIQUEID)
# Pull CNS score letter grade, removed to use XGBoost for now since it only takes integer/numbers
#train$PERFORM_CNS_SCORE_DESCRIPTION <- as.factor(substr(train$PERFORM_CNS_SCORE_DESCRIPTION,

# Remove rows with any null values
train <- train[complete.cases(train), ]

# Remove duplicate rows
train <- train[!duplicated(train), ]

# converting Loan Default to a factor for binary classification
tempTrain <- train
tempTrain$LOAN_DEFAULT <- as.factor(train$LOAN_DEFAULT)

# Convert all columns to numeric
tempTrain[] <- lapply(tempTrain, as.numeric)

# Check for NaN or infinite values in the correlation matrix and replace with 0
correlation_matrix <- cor(tempTrain)
correlation_matrix <- as.matrix(correlation_matrix)
correlation_matrix[is.nan(correlation_matrix) | is.infinite(correlation_matrix)] <- 0

# Define the plot margins to accommodate all text labels
par(mar = c(5, 5, 4, 2) + 0.1)

# Plot correlation matrix directly without clustering with variable labels
image(1:nrow(correlation_matrix), 1:ncol(correlation_matrix), correlation_matrix,
      main = "Correlation Matrix Heatmap",
      xlab = "",
      ylab = "",
      col = colorRampPalette(c("blue", "white", "red"))(100),
      axes = FALSE)

# Add labels to the axes with smaller font size and remove numbers
axis(1, at = 1:ncol(correlation_matrix), labels = colnames(correlation_matrix), las = 2, cex.axis = 0.5, tcl
axis(2, at = 1:nrow(correlation_matrix), labels = rownames(correlation_matrix), las = 2, cex.axis = 0.5, tcl

# Add correlation values as text for highly correlated pairs
for(i in 1:nrow(correlation_matrix)) {
  for(j in 1:ncol(correlation_matrix)) {
    if (!is.na(correlation_matrix[i, j]) && i != j && abs(correlation_matrix[i, j]) > 0.6) {
```

```
      text(j, i, labels = round(correlation_matrix[i, j], 2), cex = 0.5)
    }
  }
}



featureMatrix <- as.matrix(train[, -which(names(train) == "LOAN_DEFAULT")])
labelVector <- as.numeric(as.character(train$LOAN_DEFAULT))

# Set the parameters
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 6,
  eta = 0.1,
  nthread = 2,
  subsample = 0.8,
  colsample_bytree = 0.8
)

# Cross-validation for rounds
cvResults <- xgb.cv(
  params = params,
  data = featureMatrix,
  label = labelVector,
  nfold = 5,
  nrounds = 100,
  early_stopping_rounds = 10,
  verbose = FALSE
)

# Train the XGBoost model
xgbModel <- xgboost(
  params = params,
  data = featureMatrix,
  label = labelVector,
  nrounds = cvResults$best_iteration
)

#Modelling the features importance
importanceMatrix <- xgb.importance(model = xgbModel)
xgb.plot.importance(importanceMatrix)

# Sort the data by importance in descending order
importance_data <- importanceMatrix[order(importanceMatrix$Importance, decreasing = TRUE), ]

# Create the bar plot
ggplot(importance_data, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Feature Importance") +
  theme_classic() +
  theme(axis.text.y = element_text(size = 8), axis.title.x=element_blank(), axis.title.y=element_blank()) +
  coord_flip()

#Extract log loss vector from Cross validation result
log_loss <- cvResults$evaluation_log$train_logloss_mean
```

```r
#Create a data frame with log loss and number of features
logLossData <- data.frame(
  NumFeatures = seq_along(log_loss),
  LogLoss = log_loss)

#Plot log loss by number of features
ggplot(logLossData, aes(x = NumFeatures, y = LogLoss)) +
  geom_line() +
  geom_point() +
  xlab("Number of Features") +
  ylab("Log Loss") +
  ggtitle("Log Loss by Number of Features") +
  theme_bw()

#It appears that the optimal model should include 15 variables
#Vector for 15 top features
topFeatures <- importanceMatrix$Feature[1:15]

# Subset the dataset with the selected features
filteredData <- train[, c(topFeatures, "LOAN_DEFAULT")]

# Training/Testing split
set.seed(123)
n <- dim(filteredData)[1]
train_indices <- sample(1:n, size = round(0.8 * n))  # 80% for training
test_indices <- setdiff(1:n, train_indices)  # Remaining 20% for testing

train.data <- (filteredData[train_indices, 1:16])
test.data <- (filteredData[test_indices, 1:16])

train.labels <- as.numeric(filteredData$LOAN_DEFAULT[train_indices])
test.labels <- as.numeric(filteredData$LOAN_DEFAULT[test_indices])

#Verify for accuracy
head(train.data)

#Fit the Logistic regression on train data
logistic_model <- glm(LOAN_DEFAULT ~ ., data = train.data, family = binomial)

# Predictions on train data
y_pred_train <- predict(logistic_model, newdata = train.data, type = "response")
y_pred_class_train <- ifelse(y_pred_train > 0.5, 1, 0)

# Model evaluation on train data
train_accuracy <- mean(y_pred_class_train == train.labels)
train_confusion_matrix <- table(train.labels, y_pred_class_train, dnn = c("Actual", "Predicted"))

# Print accuracy and confusion matrix for train data

train_confusion_matrix
precision <- train_confusion_matrix[2, 2] / sum(train_confusion_matrix[, 2])
recall <- train_confusion_matrix[2, 2] / sum(train_confusion_matrix[2, ])

# Print adjusted accuracy, precision, and recall for train data
print(paste("Train Accuracy:", train_accuracy))
print(paste("Precision:", precision))
```

```
print(paste("Recall:", recall))

#Matrix for explanatory variable
train.matrix <- as.matrix(train.data[ , 1:15])
test.matrix <- as.matrix(test.data[ , 1:15])

#Finding optimal eta with logloss in mind
etaTester <- seq(1, 0.01, by = -0.01)
logLossVal <- numeric(length(etaTester))

#iterating through model to see which one works the best. REAL SLOW ON HARDWARE
for(i in seq_along(etaTester)) {
  xgb.loan <- xgboost(
    data = train.matrix, label = train.labels,
    max.depth = 6, eta = etaTester, nround = 100,
    objective = "binary:logistic", eval_metric = "logloss",
    verbose = 0
  )

  logLossVal[i] <- xgb.loan$evaluation_log$train_logloss[i]
}

bestIndex <- which.min(logLossVal)
bestEta <- etaTester[bestIndex]
bestLL <- logLossVal[bestIndex]

cat("Optimal eta: ", bestEta, "\n")
cat("Best logLoss: ", bestLL, "\n")

#Create a data frame with log loss and eta
eta <- data.frame(
  eta = etaTester,
  LogLoss = logLossVal)

#Plot log loss by number of features
ggplot(eta, aes(x = eta, y = LogLoss)) +
  geom_line() +
  geom_point() +
  xlab("eta tester") +
  ylab("Log Loss") +
  ggtitle("Log Loss by eta values") +
  theme_bw()


# XGBoost Classifier
xgb.loan = xgboost(data = train.matrix, label = train.labels,
                   max.depth = 6, eta = 0.01, nrounds = 100,
                   objective = "binary:logistic", eval_metric = "logloss",
                   verbose = 0)

# Early Stopping using cross-validation
xgb.loan.cv = xgb.cv(data = train.matrix, label = train.labels,
                     max.depth = 6, eta = 0.01, nrounds = 100,
                     nfold = 5, early_stopping_rounds = 5,
                     objective = "binary:logistic", eval_metric = "logloss",
                     verbose = 0)
```

```r
sel_rounds = xgb.loan.cv$best_iteration
sel_rounds
#nrounds = 100 is appropriate according to the cross-validation method, furthur training is not needed

# Fitted response labels
pred1 = predict(xgb.loan, train.matrix)
pred.xgb.loan = ifelse(pred1 > 0.5, 1, 0)

# Plotting the first two trees in the boosted classifier
xgb.plot.tree(model = xgb.loan, trees = 1:2)

# Misclassification errors in the training data
train_conf_matrix <- table(filteredData$LOAN_DEFAULT[train_indices], pred.xgb.loan)

#Accuracy
train_accuracy <- (train_conf_matrix[1,1] + train_conf_matrix[2,2]) / sum(train_conf_matrix)
precision <- train_conf_matrix[2, 2] / sum(train_conf_matrix[, 2])
recall <- train_conf_matrix[2, 2] / sum(train_conf_matrix[2, ])

train_accuracy
precision
recall

# Try different threshold
pred.xgb.loan = ifelse(pred1 > 0.4, 1, 0)

# Misclassification errors in the training data
train_conf_matrix_1 <- table(filteredData$LOAN_DEFAULT[train_indices], pred.xgb.loan)

#Accuracy
train_accuracy_1 <- (train_conf_matrix_1[1,1] + train_conf_matrix_1[2,2]) / sum(train_conf_matrix_1)
precision_1 <- train_conf_matrix_1[2, 2] / sum(train_conf_matrix_1[, 2])
recall_1 <- train_conf_matrix_1[2, 2] / sum(train_conf_matrix_1[2, ])

train_accuracy_1
precision_1
recall_1

# Try different threshold
pred.xgb.loan = ifelse(pred1 > 0.6, 1, 0)

# Misclassification errors in the training data
train_conf_matrix_2 <- table(filteredData$LOAN_DEFAULT[train_indices], pred.xgb.loan)
train_conf_matrix_2
#the model did not predict any instances as belonging to class 1 when threshold is 0.6

#training and Validation split
set.seed(4566)
n <- dim(train.data)[1]
train2_indices <- sample(1:n, size = round(0.8 * n))  # 80% for training
val_indices <- setdiff(1:n, train2_indices)  # Remaining 20% for testing

train2.data <- (train.data[train2_indices, 1:16])
val.data <- (train.data[val_indices, 1:16])

train2.labels <- as.numeric(train.data$LOAN_DEFAULT[train2_indices])
val.labels <- as.numeric(train.data$LOAN_DEFAULT[val_indices])
```

```r
#Create matrix
train2.matrix <- as.matrix(train2.data[ , 1:15])
val.matrix <- as.matrix(val.data[ , 1:15])

#Fit model
xgb.loan2 <- xgboost(
  data = train2.matrix, label = train2.labels,
  max.depth = 6, eta = 0.01, nrounds = 100, #left this as eta = 0.01 from the last time.
  objective = "binary:logistic", eval_metric = "logloss",
  verbose = 0
)

trainPredict2 <- predict(xgb.loan2, train2.matrix)
trainPredictClass2 <- ifelse(trainPredict2 > 0.5, 1, 0)

# Misclassification errors in the training data
train_conf_matrix2 <- table(train.data$LOAN_DEFAULT[train2_indices], trainPredictClass2)

#Accuracy
train_accuracy2 <- (train_conf_matrix2[1,1] + train_conf_matrix2[2,2]) / sum(train_conf_matrix2)
precision2 <- train_conf_matrix2[2, 2] / sum(train_conf_matrix2[, 2])
recall2 <- train_conf_matrix2[2, 2] / sum(train_conf_matrix2[2, ])

cat("Training Accuracy:", train_accuracy2, "\n")
cat("Training Precision:", precision2, "\n")
cat("Training Recall:", recall2, "\n")

# Predictions on validation data
valPredict <- predict(xgb.loan2, val.matrix)
valPredictClass <- ifelse(valPredict > 0.5, 1, 0)

# Confusion matrix for validation data
val_conf_matrix <- table(val.labels, valPredictClass)

# Accuracy, Precision, and Recall for validation data
val_accuracy <- sum(diag(val_conf_matrix)) / sum(val_conf_matrix)
val_precision <- val_conf_matrix[2, 2] / sum(val_conf_matrix[, 2])
val_recall <- val_conf_matrix[2, 2] / sum(val_conf_matrix[2, ])

# Print metrics for validation data
cat("Validation Accuracy:", val_accuracy, "\n")
cat("Validation Precision:", val_precision, "\n")
cat("Validation Recall:", val_recall, "\n")

#Try another threashold
valPredictClass <- ifelse(valPredict > 0.4, 1, 0)

# Confusion matrix for validation data
val_conf_matrix_1 <- table(val.labels, valPredictClass)

# Accuracy, Precision, and Recall for validation data
val_accuracy_1 <- sum(diag(val_conf_matrix_1)) / sum(val_conf_matrix_1)
val_precision_1 <- val_conf_matrix_1[2, 2] / sum(val_conf_matrix_1[, 2])
val_recall_1 <- val_conf_matrix_1[2, 2] / sum(val_conf_matrix_1[2, ])

# Print metrics for validation data
```

```r
cat("Validation Accuracy:", val_accuracy_1, "\n")
cat("Validation Precision:", val_precision_1, "\n")
cat("Validation Recall:", val_recall_1, "\n")

#Try another threashold
valPredictClass <- ifelse(valPredict > 0.6, 1, 0)

# Confusion matrix for validation data
val_conf_matrix_2 <- table(val.labels, valPredictClass)
val_conf_matrix_2
```