Determing Key Characteristics of Loan Default

Joe Broton

5/5/2020

Contents

Introduction

There are many reasons why you would take out a loan. A simple reason is that you need money that you do not currently have. I took out a loan for college as I could not afford to pay tuition costs right up front.I am sure there was a background check on myself, and my mother who had to cosign for the loan as well. At some point, I was approved for the loan and after 4 years I am in my last week of school ready to graduate. However, sometimes students, as well as anyone else who applied for a loan could get denied. Whoever is in charge of approving the loan, has a data-driven decision based on your likely-ness to pay it back. Your likely-ness to pay the loan back in full (plus interest) is determined by them somehow. In this paper, I attempt to figure out what characteristics determine whether or not someone defaults on their loan. I analyze the lending club dataset which contains loans issued from 2007 through 2015. The dataset provies all types of information such as the loan amount, interest rates, monthly payments, length of the loan, and other numeric information. The lending club also provides characteristics of the people who receieved the loan such as their employment, how long they worked there, their yearly income, along with geographic location of the issued loan. Typically Banks would prefer to only give our to those who are able to pay their loan in full without having to default on their payments. By the end of this paper, I will try to report what variables I found to be the most important in predicting loan default.

Summary Statistics

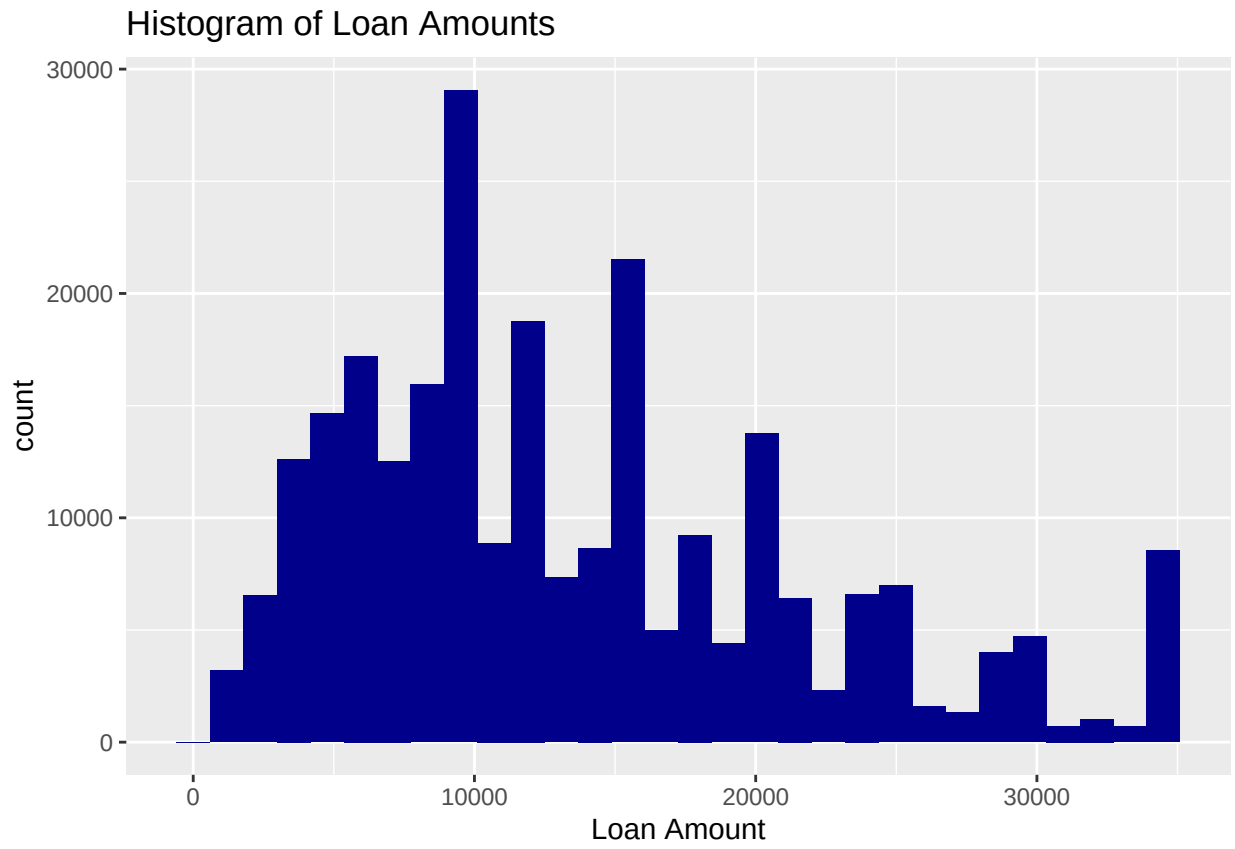
```
library(tidyverse)
loan_data = readRDS("loan_data.Rds")

loan_data = loan_data %>%
  mutate(loan_status = replace(loan_status, loan_status == "Charged Off", "Default")) %>%
  filter(loan_status == "Default" | loan_status == "Fully Paid")
```

```
summary(select(loan_data, loan_amnt, int_rate, installment, annual_inc, dti))
```

```
##    loan_amnt        int_rate      installment        annual_inc
##  Min.   :  500   Min.   : 5.32   Min.   :  15.69   Min.   :   3000
##  1st Qu.: 7250   1st Qu.:10.74   1st Qu.: 239.56   1st Qu.:  45000
##  Median :12000   Median :13.53   Median : 365.23   Median :  62000
##  Mean   :13571   Mean   :13.78   Mean   : 418.27   Mean   :  72511
##  3rd Qu.:18250   3rd Qu.:16.55   3rd Qu.: 547.55   3rd Qu.:  87000
##  Max.   :35000   Max.   :28.99   Max.   :1424.57   Max.   :8706582
##       dti
##  Min.   : 0.00
##  1st Qu.:10.77
##  Median :16.22
##  Mean   :16.56
##  3rd Qu.:22.01
##  Max.   :57.14
```

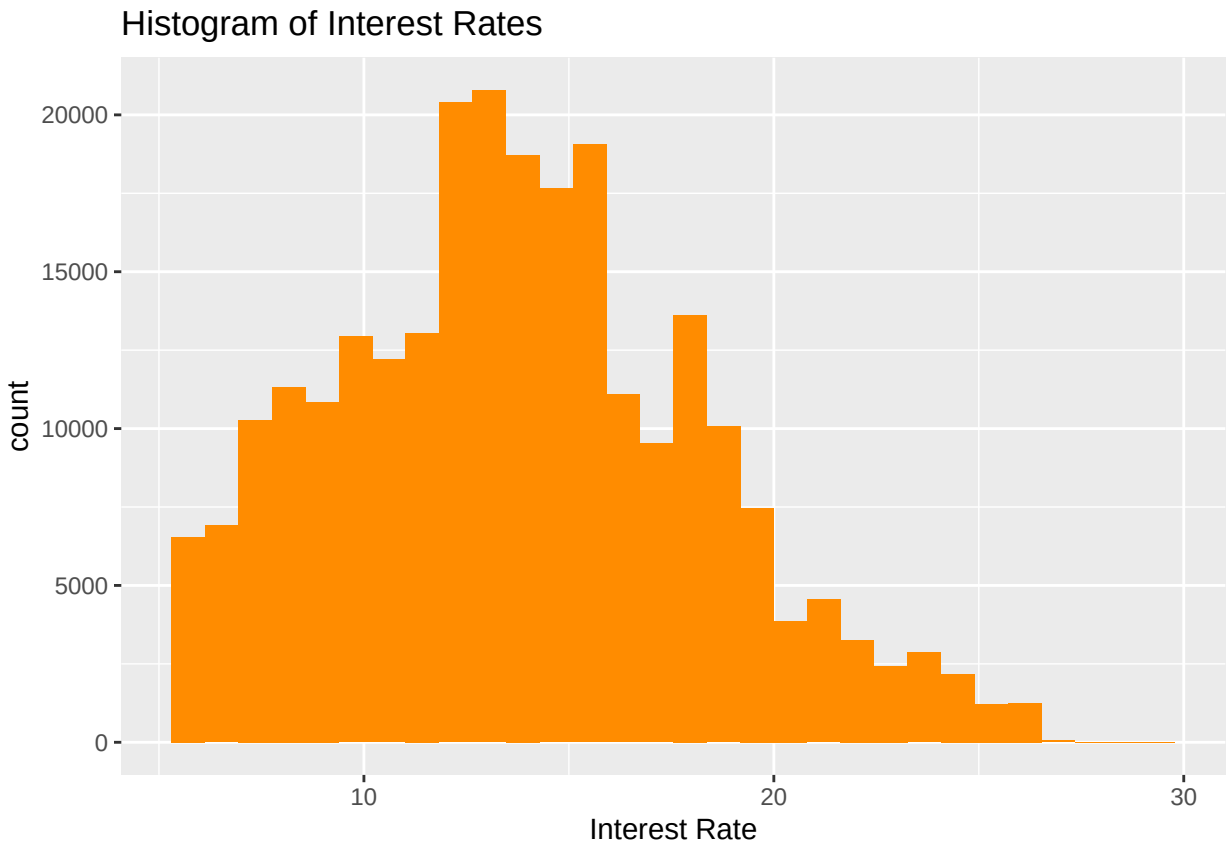
Above is some of the numeric variables being summarized. The Annual income of the participants range from $3,000 all the way to $8.7 million. The typical income of somone requesting a loan is more likely to be around $65,000. Loan amounts can be as little as $500 and as big as $35,000. As a result of the loan amounts, interest rates vary from 5% to 29% and installments can be from $15 a month to $1,424.

```r
# Histogram of Loan Amounts
ggplot(loan_data, aes(loan_amnt)) +
  geom_histogram(bins = 30 , fill = "darkblue") +
  labs(x = "Loan Amount", title = "Histogram of Loan Amounts")
```



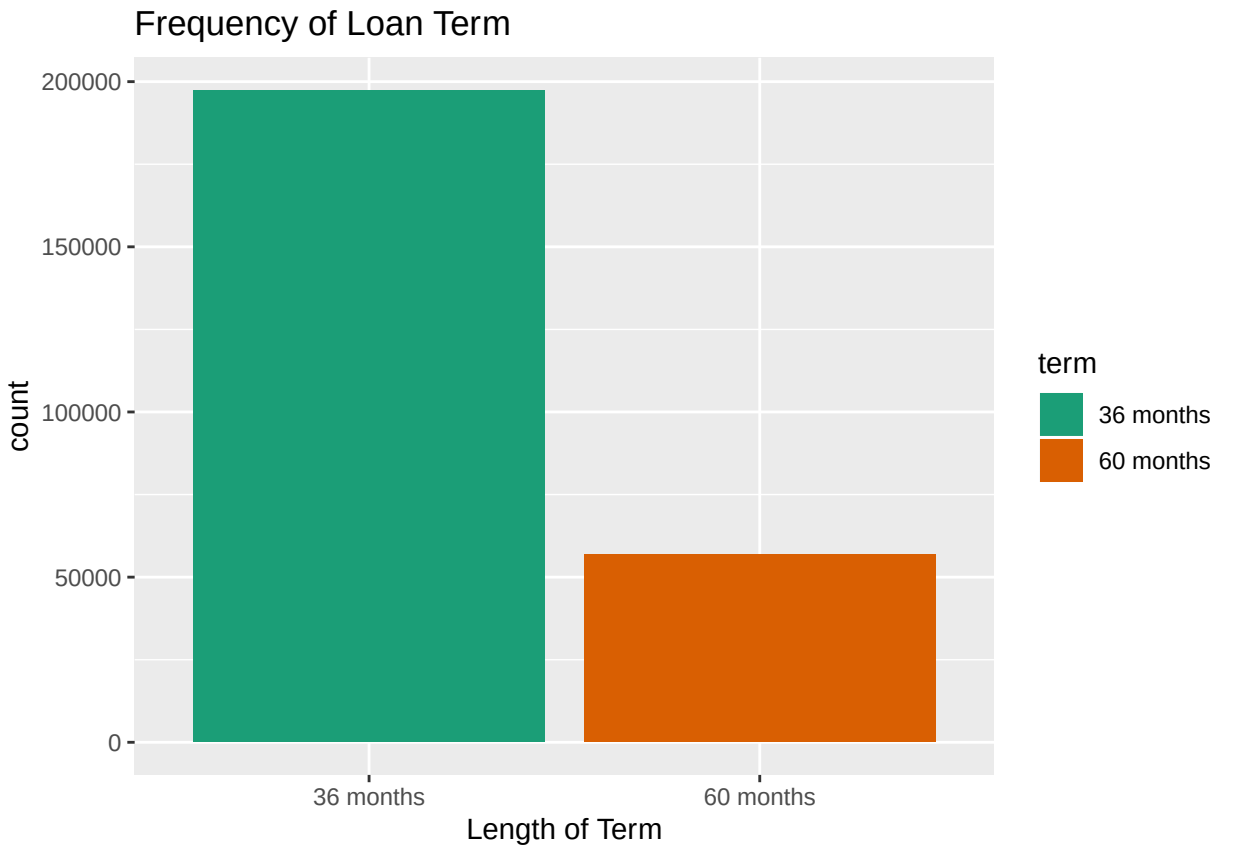
The Loan Amount appears to be right skewed. The mean is at the $13,000 shown earlier. There is quite a few loans at the max which was $35,000 around 9,000 of them.

```r
# Histogram of Interest Rates
ggplot(loan_data, aes(x = int_rate)) +
geom_histogram(bins = 30, fill = "darkorange") +
labs(x = "Interest Rate", title = "Histogram of Interest Rates")
```



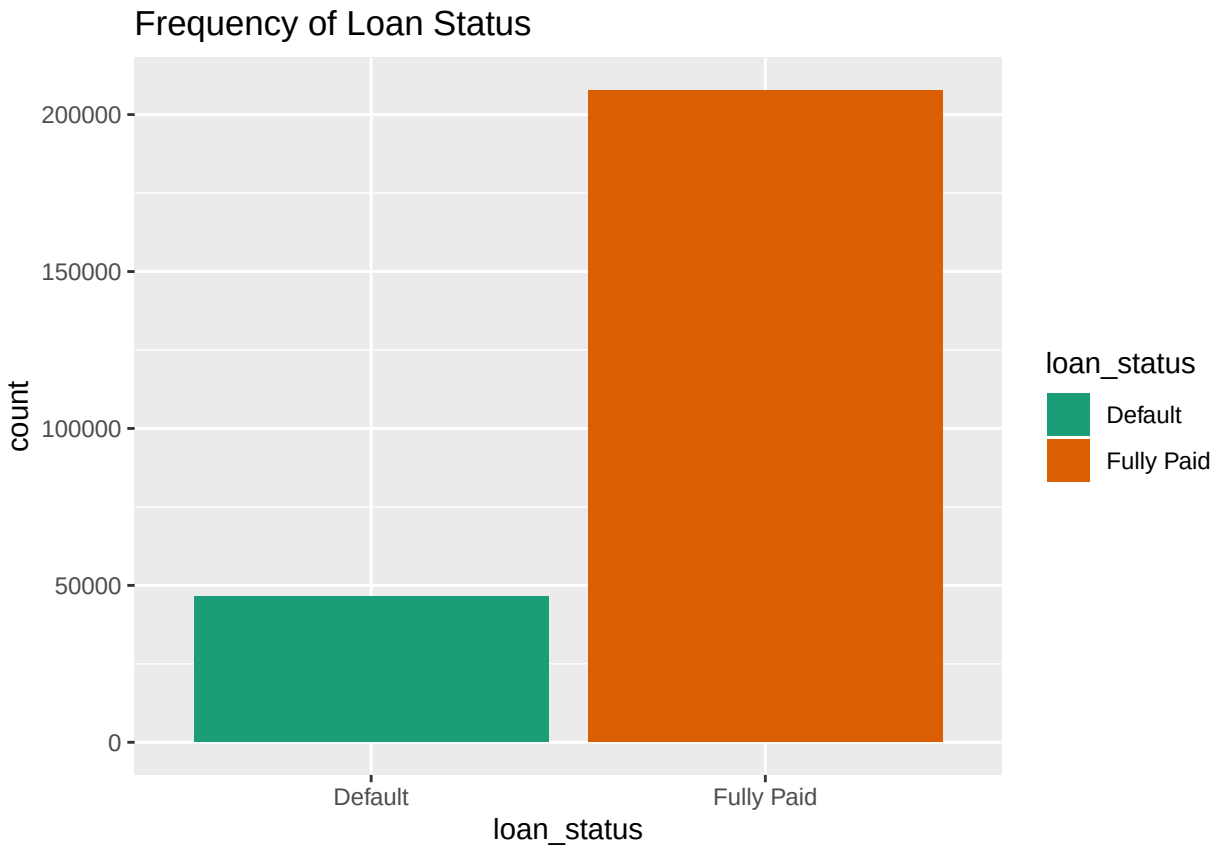
Interest Rates appear a lot more normal than the Loan amount but still a little right skewed, meaning there are few fairly high interest rates in the ranges of 20-27ish. The mean is roughly around 13-14%. The highest interest rate appears to be was 28.99% shown earlier and it appears there's only a few of them as opposed to the max loan amount histogram.

```
# Term Bar Plot
ggplot(loan_data, aes(x= term, fill = term)) +
  geom_bar() +
  scale_fill_brewer(palette = "Dark2") +
  labs(x = "Length of Term", title = "Frequency of Loan Term")
```



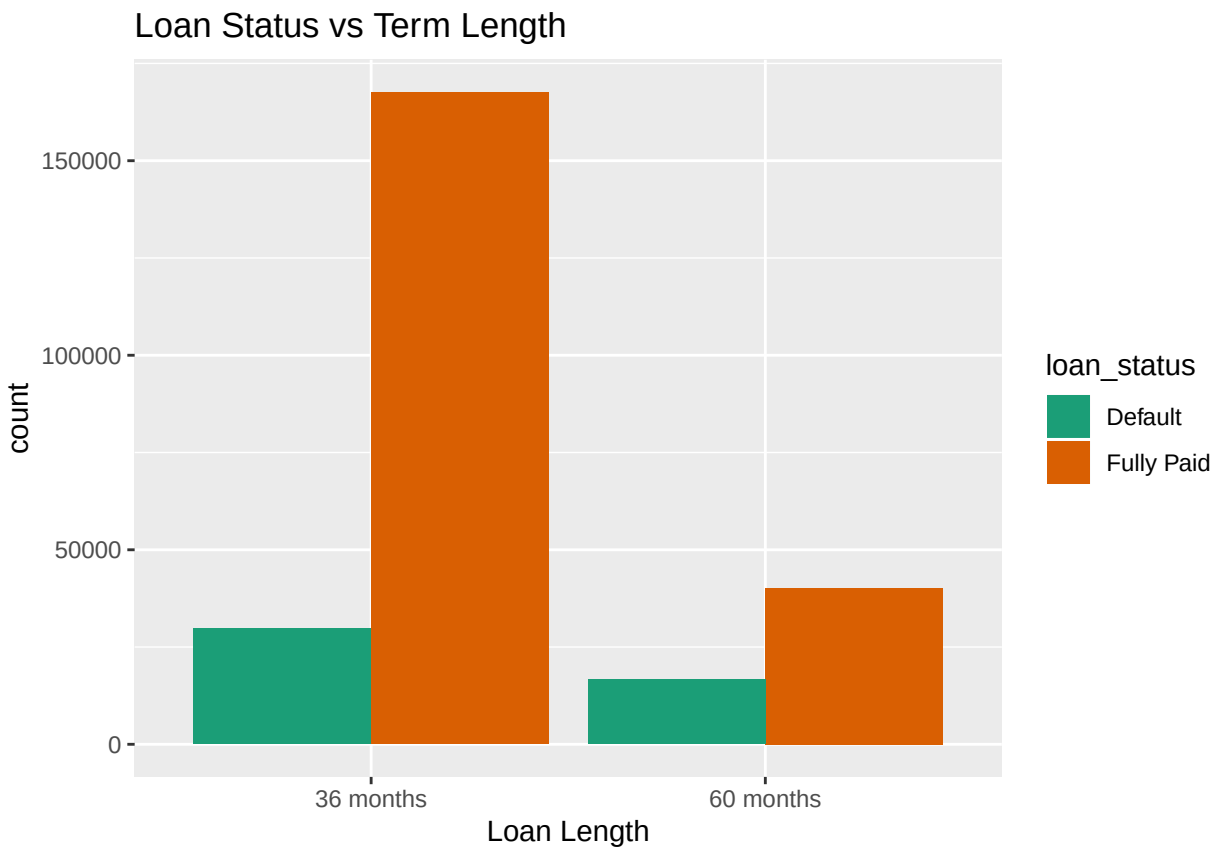
There are way more loans that were given 36 month term length than the 60 months. It appears to be a little more than 3 times the amount than the 60 months.

```
# Loan Status Bar Plot
ggplot(loan_data, aes(x = loan_status, fill = loan_status)) +
  geom_bar() +
  scale_fill_brewer(palette = "Dark2") +
  labs(title = "Frequency of Loan Status")
```



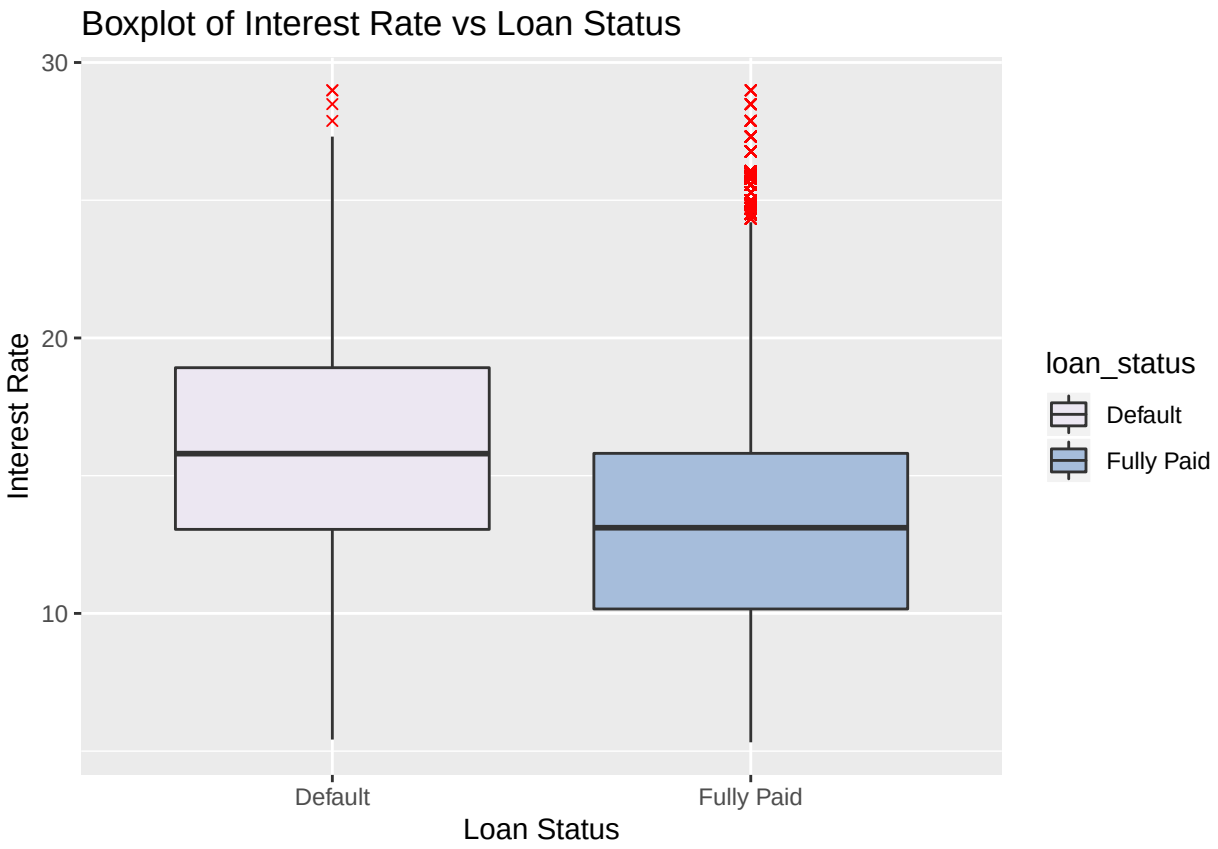
There are way more loans that are full paid than defaulted. Defaulted only appears roughly 50,000 times in the dataset. That's only around 20% of our dataset.

```
# Loan Status vs Term Length
ggplot(loan_data, aes(x = term, fill = loan_status)) +
  geom_bar(position = "dodge") +
  labs(title = "Loan Status vs Term Length", x = "Loan Length") +
  scale_fill_brewer(palette = "Dark2")
```



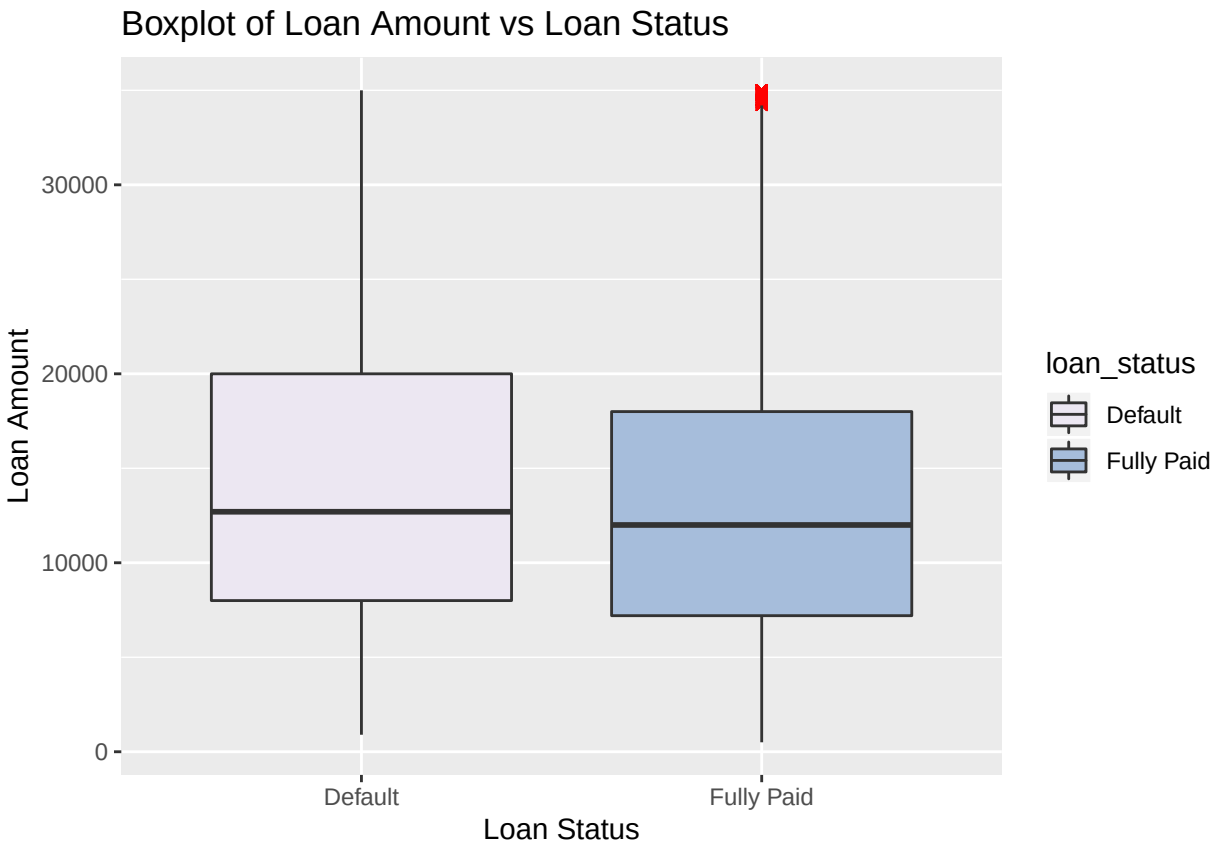
The loan status of default appears to be roughly the same for both the 36 and 60 month loan lengths. There are many more Fully paid 36 months loans.

```
# Interest Rate vs Loan Status
ggplot(loan_data, aes(x = loan_status, y = int_rate, fill = loan_status)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 4) +
  labs(x = "Loan Status", y = "Interest Rate",
       title = "Boxplot of Interest Rate vs Loan Status") +
  scale_fill_brewer(palette = 9)
```



It appears on average that Interest Rate is higher on loans that default than loans that are fully paid. There also are way more outliers on the fully paid loan status compared to the loans that defaulted.

```
# Loan Amount vs Loan Status
ggplot(loan_data, aes(x = loan_status, y = loan_amnt, fill = loan_status)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 4) +
  labs(x = "Loan Status", y = "Loan Amount",
       title = "Boxplot of Loan Amount vs Loan Status") +
  scale_fill_brewer(palette = 9)
```



This graph shows that the range is roughly the same for the loan amount for both statuses. The defaulted loan appear on average to be higher than the fully paid loans.

Logistic & Probit Classification

```
loan_df = select(loan_data, loan_status, loan_amnt, int_rate, installment, annual_inc, dti, term)
loan_df = loan_df %>%
  mutate(loan_status = replace(loan_status, loan_status == "Fully Paid", 0),
         loan_status = replace(loan_status, loan_status == "Default", 1),
         term = replace(term, term == "36 months", 0),
         term = replace(term, term == "60 months", 1))

loan_df = loan_df %>%
  mutate(loan_status = as.numeric(loan_status),
         term = as.numeric(term))

# Train Data
set.seed(999)
loan_train = loan_df %>%
  sample_frac(.75)

# Test Data
loan_test = loan_df %>%
  setdiff(loan_train)
```

```r
# 5 Models of Logit
model1 = glm(loan_status ~ loan_amnt + int_rate + annual_inc +
               installment + term + dti, data = loan_train,
             family = binomial(link = "logit"))

model2 = glm(loan_status ~ loan_amnt + int_rate + dti,
             data = loan_train, family = binomial(link = "logit"))

model3 = glm(loan_status ~ loan_amnt + int_rate + annual_inc,
             data = loan_train, family = binomial(link = "logit"))

model4 = glm(loan_status ~ loan_amnt + term + annual_inc,
             data = loan_train, family = binomial(link = "logit"))

model5 = glm(loan_status ~ loan_amnt + int_rate + term,
             data = loan_train, family = binomial(link = "logit"))

mod1_pred = predict(model1, loan_test, type = "response")
mod2_pred = predict(model2, loan_test, type = "response")
mod3_pred = predict(model3, loan_test, type = "response")
mod4_pred = predict(model4, loan_test, type = "response")
mod5_pred = predict(model5, loan_test, type = "response")

error_rate = function(mod_pred){
  temp_vec = mod_pred
  temp_vec[temp_vec >= .2] = 1 # Default
  temp_vec[temp_vec < .2] = 0 # Non-Default
  return(mean(temp_vec != loan_test$loan_status))
}

c(error_rate(mod1_pred),
  error_rate(mod2_pred),
  error_rate(mod3_pred),
  error_rate(mod4_pred),
  error_rate(mod5_pred))
```

```
## [1] 0.3168700 0.3235456 0.3215776 0.2761238 0.3150122
```

```r
model_4 = error_rate(mod4_pred)
model_4
```

```
## [1] 0.2761238
```

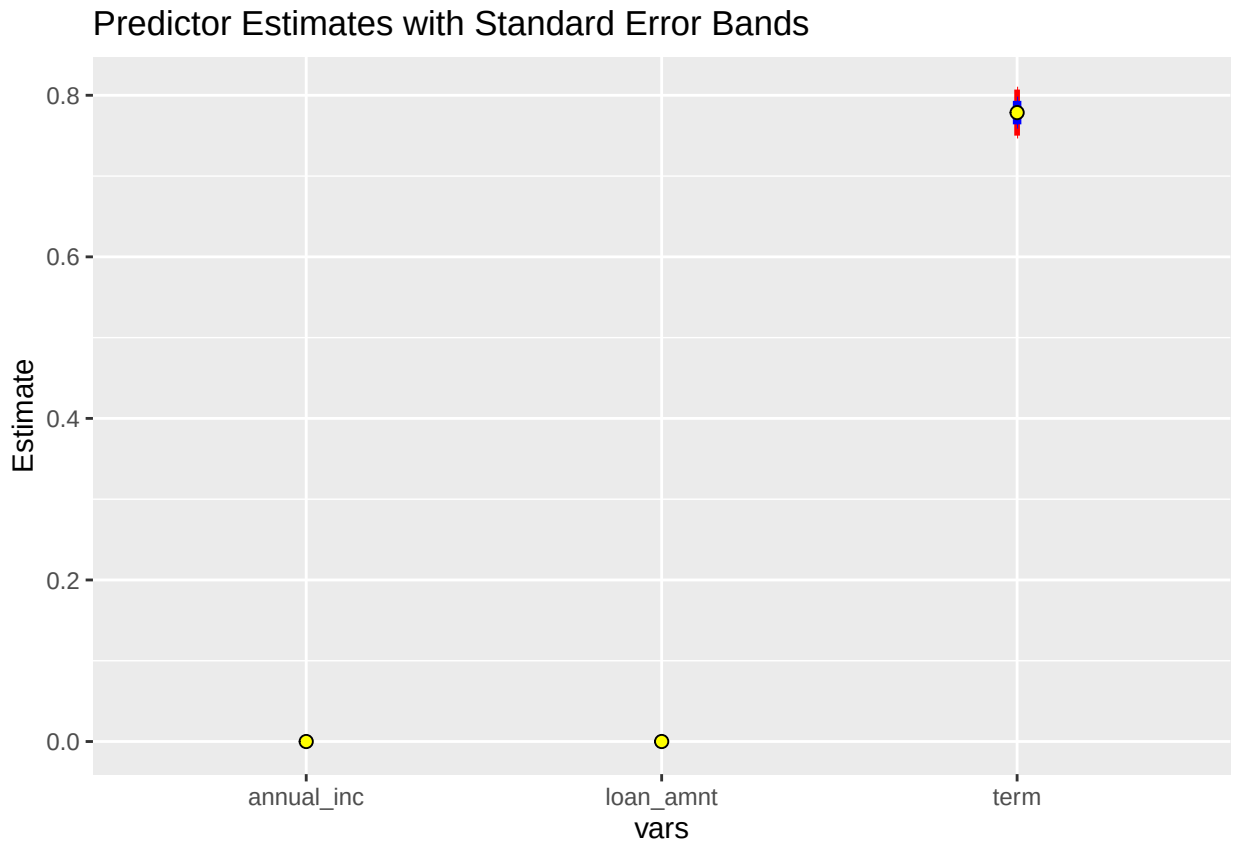
Above we test various logit models to see which model fairs best on our test dataset. I tried different predictors until I created 5 logit models. I created an error function that that takes our predictions with a treshold of .2, and returns the error rate for each model. Model 4 had the lowest error rate with 27.61%. This model only used three predictors that were loan_amnt, term, and annual_inc.

```r
mod4_pred[mod4_pred >= .2] = 1
mod4_pred[mod4_pred < .2] = 0
table("Model 4 Predictions" = mod4_pred, "True" = loan_test$loan_status)
```

```
##                      True
## Model 4 Predictions     0     1
##                   0 41546  7167
##                   1 10371  4431
```

Looking at the prediction matrix, we see that it does a good job at predicting non-default loans but misclassifies defaulted loans as non-default loans a lot. In this case it's 10,371 times.

```
mod4_summary = summary(model4)

mod4_coefs = as.data.frame(mod4_summary$coefficients[-1,1:2])
names(mod4_coefs)[2] = "se"
mod4_coefs$vars = rownames(mod4_coefs)

ggplot(mod4_coefs, aes(vars, Estimate)) +
  geom_errorbar(aes(ymin = Estimate - 1.96 * se, ymax = Estimate + 1.96 * se),
                lwd = 1, colour = "red", width = 0) +
  geom_errorbar(aes(ymin = Estimate - se, ymax = Estimate + se),
                lwd=1.5, colour="blue", width=0) +
  geom_point(size = 2, pch = 21, fill = "yellow") +
  ggtitle("Predictor Estimates with Standard Error Bands")
```



Above is the variables used in our best logit model that produced the 27.61% error rate. We see that term is the highest estimate and has a large standard error band, while the other variables are small with very tiny standard error bands.

```
mod4_probit = glm(loan_status ~ loan_amnt + term + annual_inc,
             data = loan_train, family = binomial(link = "probit"))

probit_pred = predict(mod4_probit, loan_test, type = "response")
probit_mod = error_rate(probit_pred)

error_table = rbind(model_4, probit_mod)
colnames(error_table) = "Error_Rate"
error_table
```

```
##            Error_Rate
## model_4     0.2761238
## probit_mod  0.2690860
```

After taking our best logit model, we try to fit a probit model with the same variables. After doing so, we see that the error rate goes down slightly from 27.61% to 26.91%

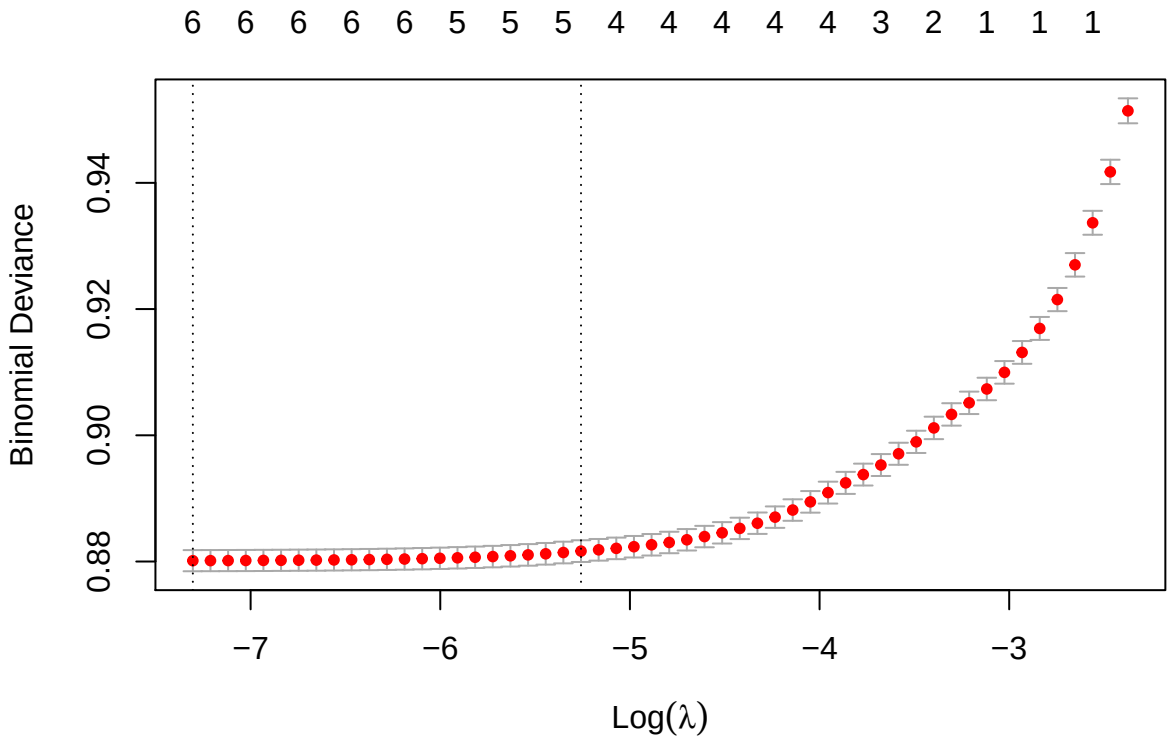
Lasso

```
library(glmnet)
X_train = as.matrix(loan_train[, -1])
Y_train = loan_train$loan_status

X_test = as.matrix(loan_test[, -1])
Y_test = loan_test$loan_status

set.seed(999)
lasso_cv.out = cv.glmnet(X_train, as.factor(Y_train), family = "binomial", alpha = 1)
lasso_bestlam = lasso_cv.out$lambda.1se
lasso_bestlam
```

```
## [1] 0.005206207
```

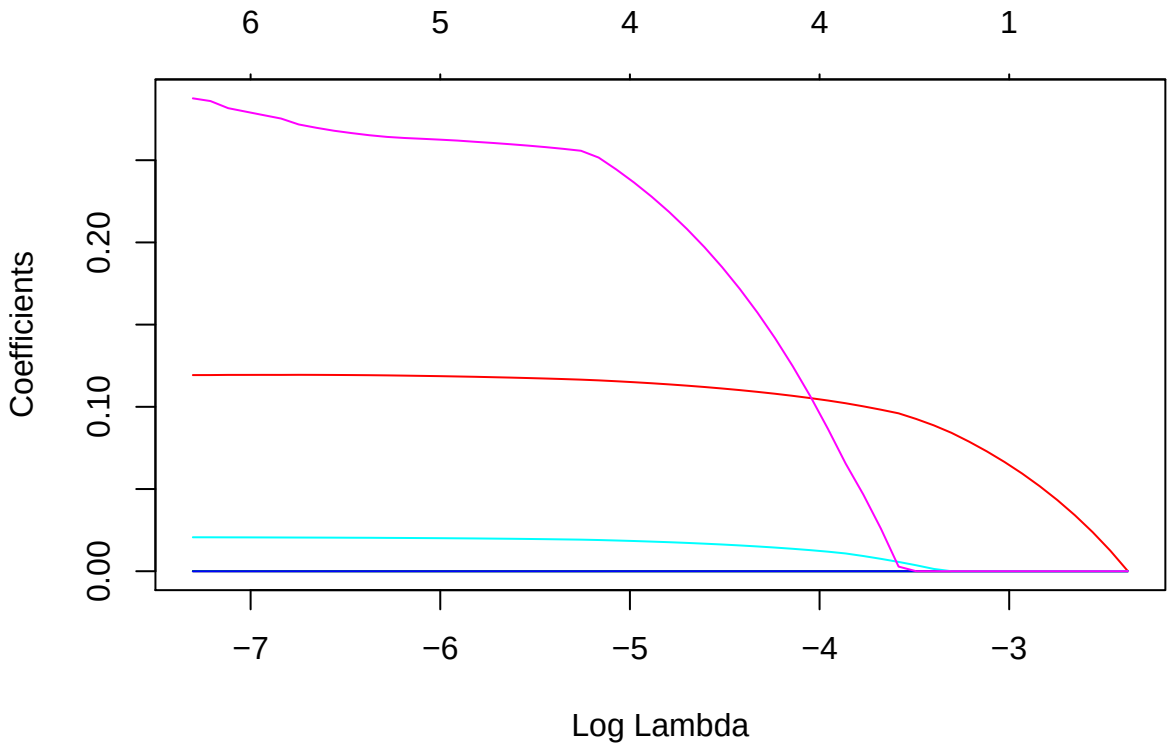
The λ within 1 standard error of the minimum λ is 0.0052.

```
plot(lasso_cv.out)
```



The graph shows the relationship between the log of λ which is selected from the cross-validation error. The two dashed lines the minimum of λ, and the λ which is within 1 standard error away from it.

```
out_lasso = glmnet(X_train, as.factor(Y_train), family = "binomial", alpha = 1)
plot(out_lasso, xvar = "lambda")
```



The graph above shows how the different coefficients shrink. They all shrink to zero when the log of lambda is around -2.5.

```
predict(out_lasso, type = "coefficients", s = lasso_bestlam)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -3.338977e+00
## loan_amnt    3.932950e-07
## int_rate     1.165321e-01
## installment  .
## annual_inc  -3.731480e-06
## dti          1.920561e-02
## term         2.557383e-01
```

The coefficients are all relatively small with installment being shrunk to 0. Term and Interest rate are the largest coefficients which make them more important when predicting loan status.

```
# Lasso Predictions & Error Rate
lasso_pred = predict(out_lasso, s = lasso_bestlam, newx = X_test, type = "response")

error_rate_new = function(mod_pred){
  temp_vec = mod_pred
  temp_vec[temp_vec >= .2] = 1 # Default
  temp_vec[temp_vec < .2] = 0 # Non-default
```

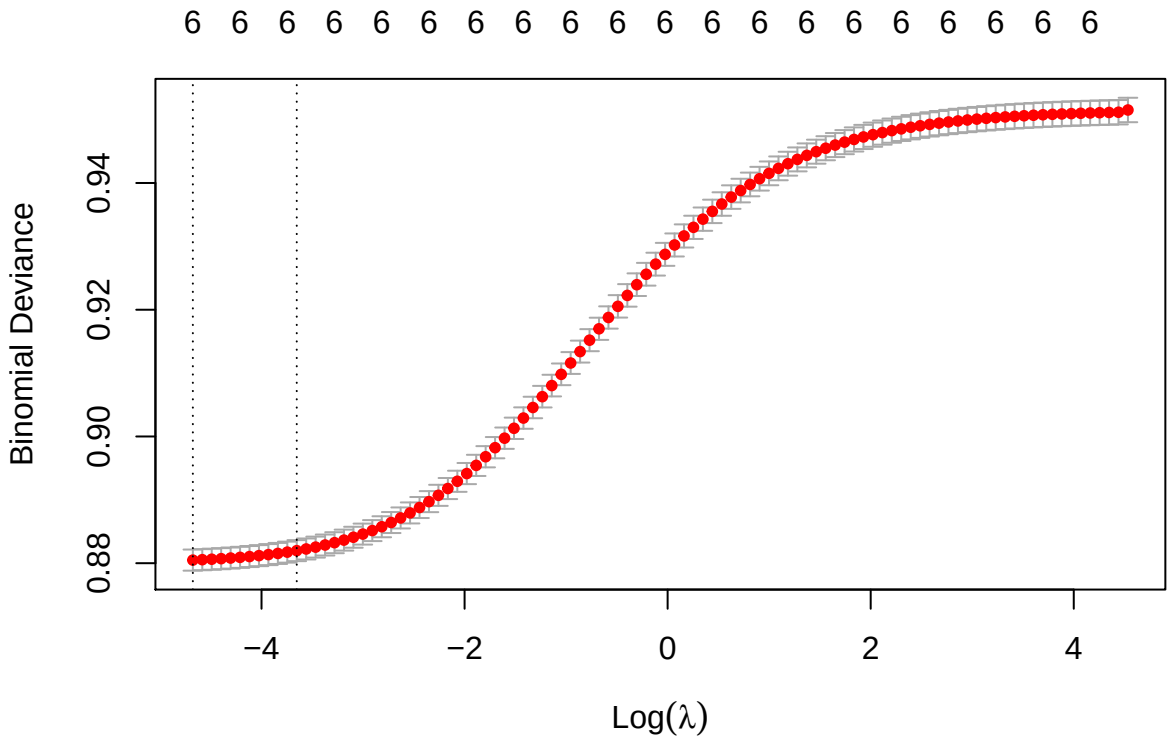
```
  return(mean(temp_vec != Y_test))
}

lasso_mod = error_rate_new(lasso_pred)
error_table = rbind(error_table, lasso_mod)

lasso_mod
```

```
## [1] 0.3163505
```

Above is the lasso mod error rate of 31.63%. This is higher than our previous models.

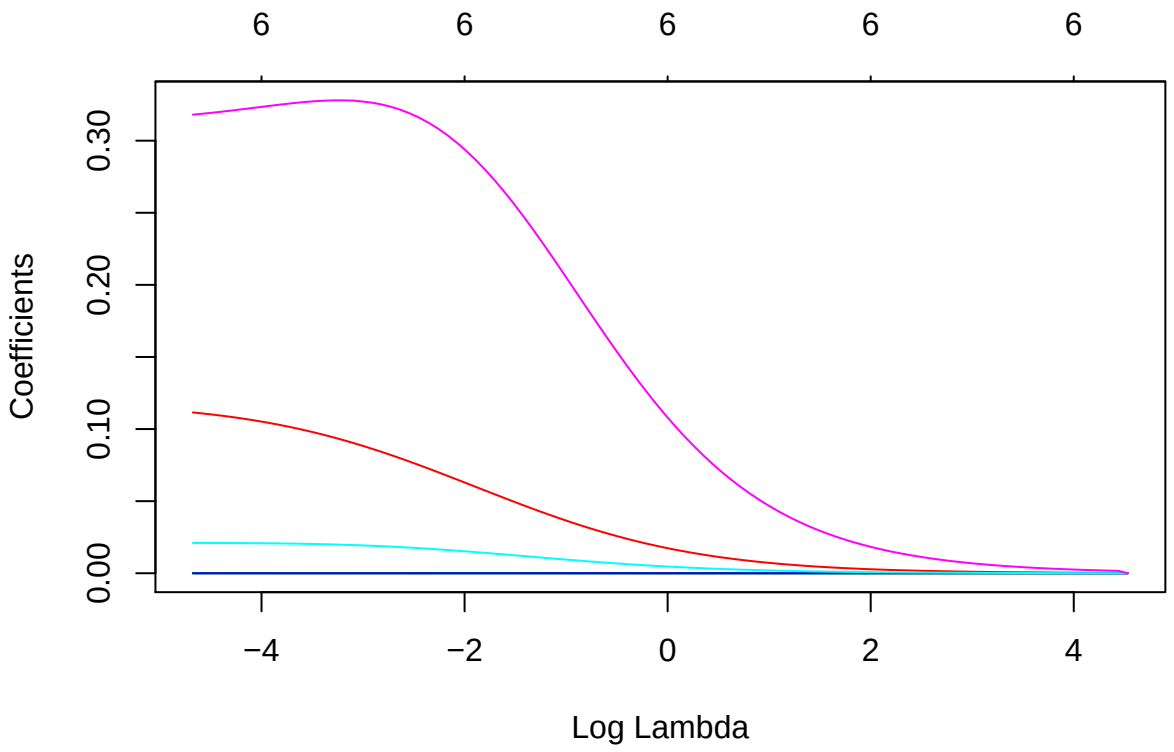
Ridge

```
set.seed(999)
ridge_cv.out = cv.glmnet(X_train, as.factor(Y_train), family = "binomial", alpha = 0)
ridge_bestlam = ridge_cv.out$lambda.1se

plot(ridge_cv.out)
```



The graph shows the relationship between the log of λ which is selected from the cross-validation error. The two dashed lines the minimum of λ, and the λ which is within 1 standard error away from it.

```
out_ridge = glmnet(X_train, as.factor(Y_train), family = "binomial", alpha = 0)
plot(out_ridge, xvar = "lambda")
```



The graph above shows how the different coefficients shrink. They all shrink to zero when the log of lambda is around 2.

```
# Ridge Coefficients
predict(out_ridge, type = "coefficients", s = ridge_bestlam)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -3.191744e+00
## loan_amnt    1.772747e-06
## int_rate     1.004374e-01
## installment  1.323794e-04
## annual_inc  -4.074178e-06
## dti          2.050970e-02
## term         3.262911e-01
```

Compared to Lasso, ridge uses all of the coefficients. Annual income is negatively associated with loan default which stands out to me.

```
# Ridge Predictions & Error Rate
ridge_pred = predict(out_ridge, ridge_bestlam, newx = X_test, type = "response")
```

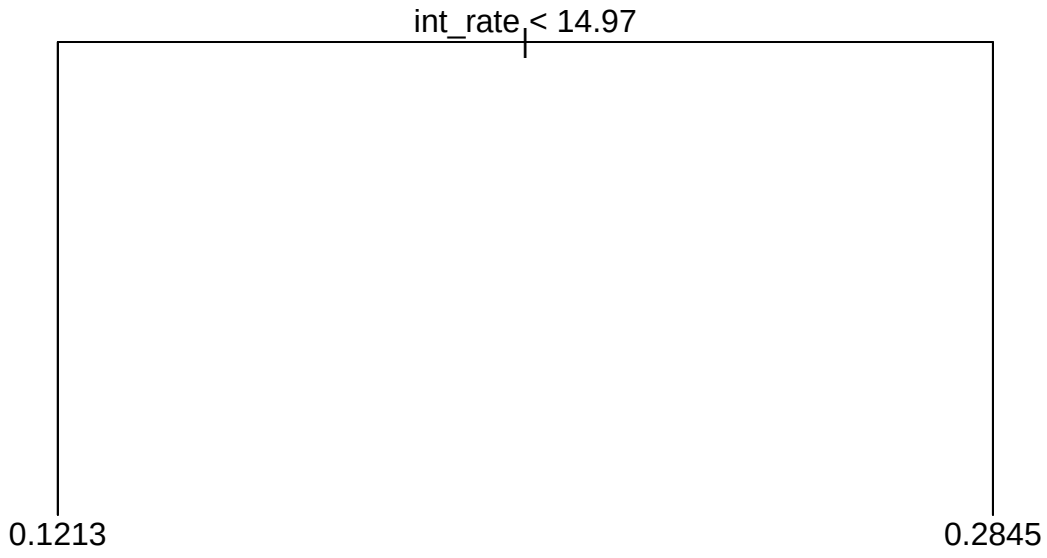
```
ridge_mod = error_rate(ridge_pred)
error_table = rbind(error_table, ridge_mod)
ridge_mod
```

```
## [1] 0.3141148
```

Ridge has an error rate of 31.41% which is only .2% less than the lasso model counterpart.

Decision Tree

```
library(tree)
tree_loan = tree(loan_status ~. , data = loan_train)
plot(tree_loan)
text(tree_loan, pretty = 0)
```



When creating a decision tree, it only takes interest rate as an important split.

```
tree_pred = predict(tree_loan, loan_test)
tree_mod = error_rate(tree_pred)
tree_mod
```

```
## [1] 0.3463276
```

Despite having only one split, it did not do too bad in predicting default's as much as I thought it would. It has the highest of 34.64% error rate but I thought it could be a lot worse.

Bagging

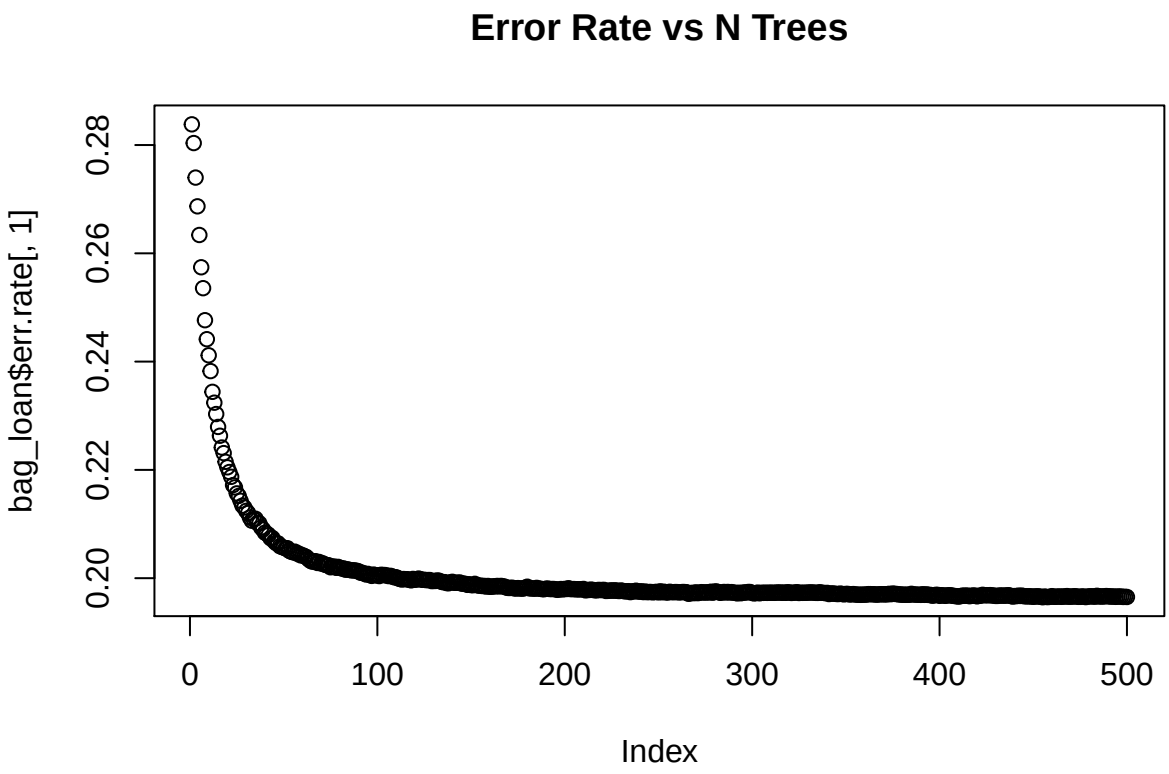
```
library(randomForest)
set.seed(500)
memory.limit(size = 10000)
bag_loan = randomForest(factor(loan_status) ~., data = loan_train,
                        mtry = ncol(loan_train) - 1,
                        importance = TRUE,
                        do.trace = 100)
#saveRDS(bag_loan, file = "bag_loan.rds")
```

```
library(randomForest)
bag_loan = readRDS("bag_loan.rds")

oob_error = bag_loan$err.rate[c(100, 200, 300, 400, 500), ]
oob_error = cbind("n_tree" = c(100,200,300,400,500), oob_error)
rownames(oob_error) = c("","","","","")
oob_error
```

```
##  n_tree       OOB          0         1
##     100 0.2005120 0.04713238 0.8857405
##     200 0.1980151 0.04276067 0.8916198
##     300 0.1972703 0.04116862 0.8946599
##     400 0.1967352 0.04034691 0.8954055
##     500 0.1965254 0.04001952 0.8957210
```

The out of bag error decreases as the number of trees increases. We can also see that visually in the graph below.

```
plot(bag_loan$err.rate[,1]) +
  title(main = "Error Rate vs N Trees")
```




```
## integer(0)
```

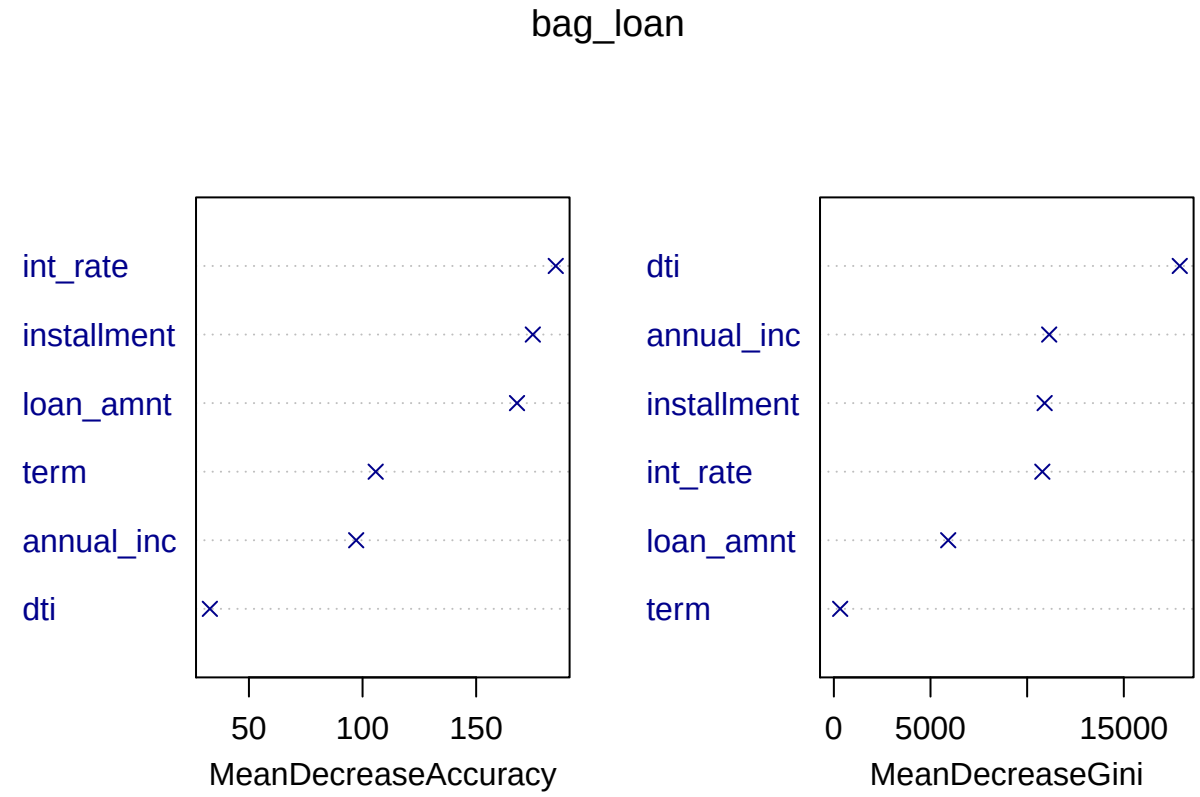
The error rate decreases dramatically at around 50 trees, and slowly decreases as the number of trees reaches 500.

```
predictions = predict(bag_loan, newdata = loan_test, type = "response")
predictions = as.numeric(predictions) - 1

bagging_mod = mean(predictions != loan_test$loan_status)
error_table = rbind(error_table, bagging_mod)
bagging_mod
```

```
## [1] 0.1954027
```

The overall error rate is less than 19.54% which is the lowest we have seen in all our models.

```
varImpPlot(bag_loan, pch = 4, color = "darkblue")
```



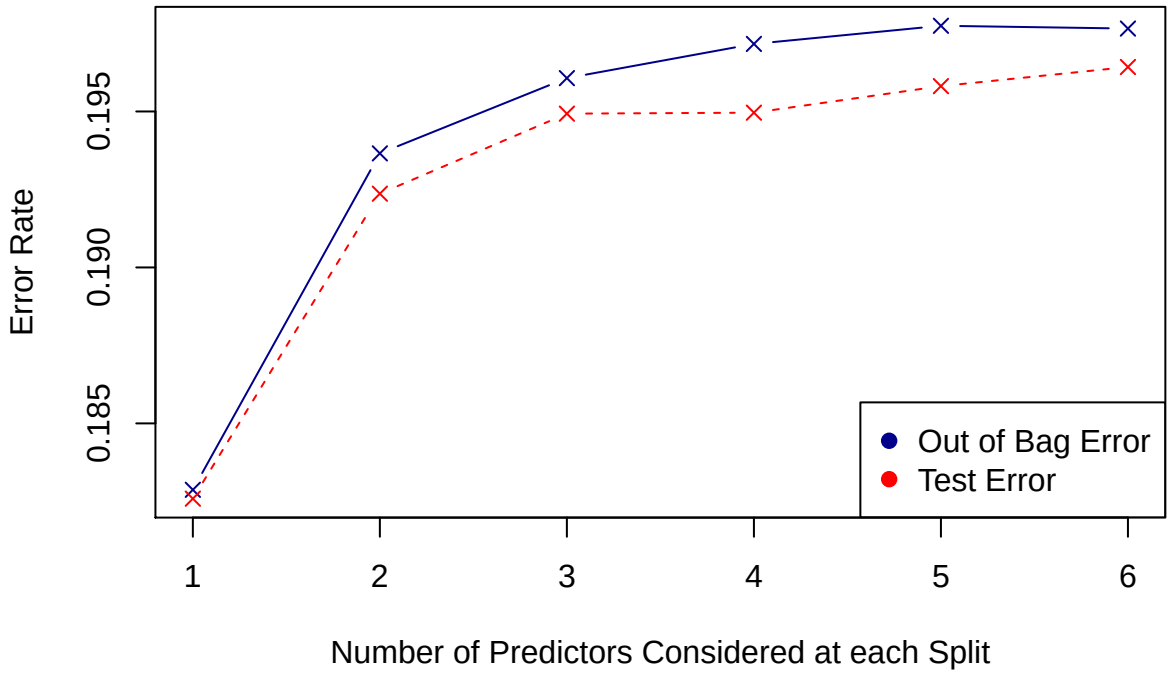
The plot above tells us that the most important predictors are int_rate, installment, and loan_amnt. Getting rid of these variables will greatly increase our error rate. Other variables like dti and annual_inc were important splits in our model.

Random Forest

```
set.seed(500)
oob_err = double(6)
test_err = double(6)

for(mtry in 1:6){
  rf = randomForest(factor(loan_status) ~. ,
                    data = loan_train,
                    mtry = mtry,
                    ntree = 200)
  oob_err[mtry] = rf$err.rate[200]

  pred = predict(rf, loan_test, type = "response")
  pred = as.numeric(pred) - 1 # as test data is coded 0,1 not 1,2

  test_err[mtry] = mean(pred != loan_test$loan_status)
}
```

```
#error_list = list(oob_err = oob_err, test_err = test_err)
#saveRDS(error_list, "rf_error.RDS")
```

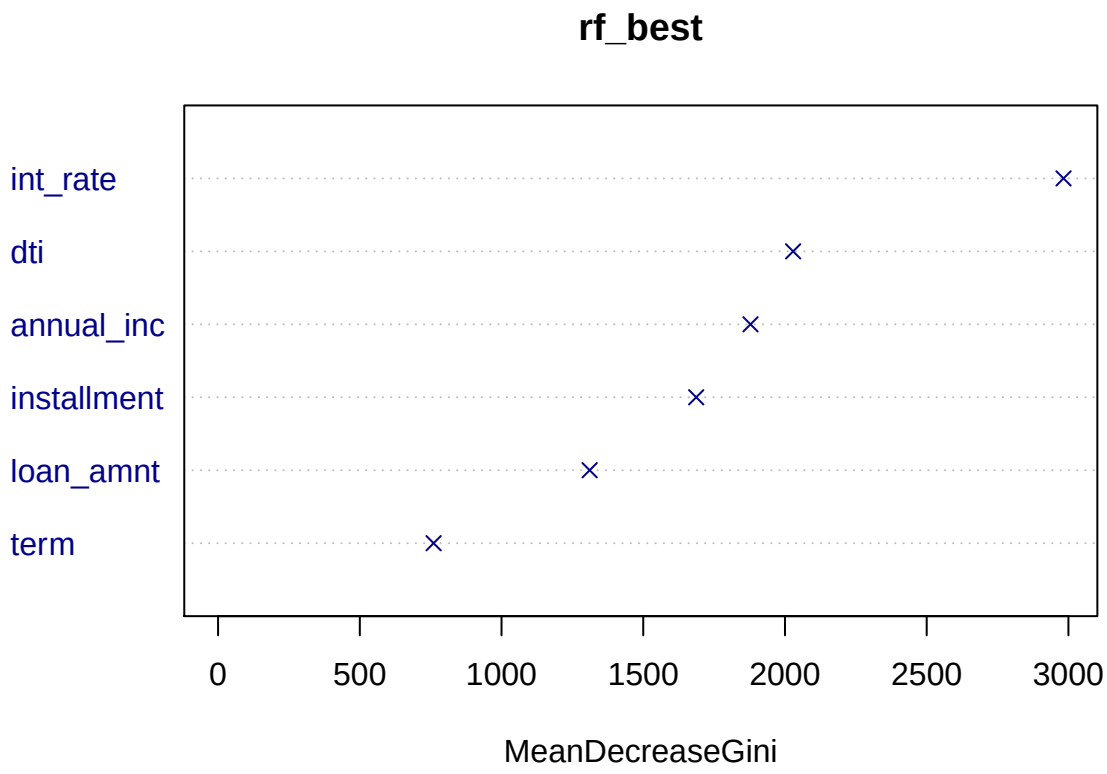
```
error_list = readRDS("rf_error.RDS")
matplot(1:6 ,cbind(error_list$oob_err,error_list$test_err),
        pch=4,col=c("darkblue","red"),
        type="b",ylab="Error Rate",
        xlab="Number of Predictors Considered at each Split")
legend("bottomright", legend=c("Out of Bag Error","Test Error"),
       pch=19, col=c("darkblue","red"))
```



The error rate increases a decent amount from below .185 to above .190 after the model chooses to use 2 splits instead of 1. In this case we prefer to have one predictor considered at each split.

```
set.seed(500)
rf_best = randomForest(factor(loan_status) ~. ,
                  data = loan_train,
                  mtry = 1,
                  ntree = 200)
#saveRDS(rf_best, file = "rf_best.rds")
```

```
rf_best = readRDS("rf_best.rds")
varImpPlot(rf_best, pch = 4, color = "darkblue")
```



After tuning our model for one predictor considered at each split, we see that this time the model prefers to have int_rate and dti as the most useful predictors.

```
pred = predict(rf_best, loan_test, type = "response")
pred = as.numeric(pred) - 1
best_random_forest = mean(pred != loan_test$loan_status)
error_table = rbind(error_table, best_random_forest)
best_random_forest
```

```
## [1] 0.182571
```

Our best random forest model has produced the best error rate of 18.26%. This was roughly 1.3% better than the bagging model.

Boosting

```
library(gbm)
set.seed(500)

boost_loan = gbm(loan_status ~. ,
                 data = loan_train,
                 distribution = "bernoulli",
                 n.trees = 5000,
```

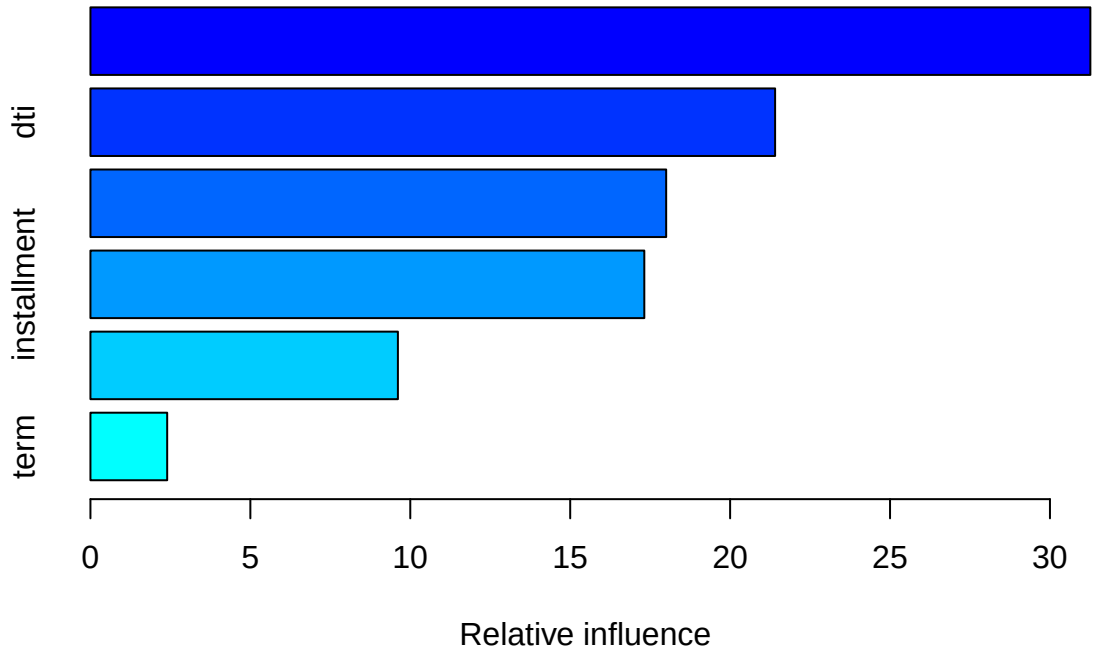
```
                  interaction.depth = 4)
# saveRDS(boost_loan, file = "boost_loan.rds")
```

```
library(gbm)
boost_loan = readRDS("boost_loan.rds")
yhat_boost = predict(boost_loan,
                     newdata = loan_test,
                     n.trees = 5000,
                     type = "response")

error_rate = function(mod_pred){
  temp_vec = mod_pred
  temp_vec[temp_vec >= .3] = 1 # Default
  temp_vec[temp_vec < .3] = 0 # Non-Default
  return(mean(temp_vec != loan_test$loan_status))
}

boost_mod = error_rate(yhat_boost)
error_table = rbind(error_table, boost_mod)
boost_mod
```

```
## [1] 0.2247343
```

Our boosting model has an error rate of 22.4%. This is one of the better error rates we've seen but not the best. It still outperforms our logit/probit/lasso and ridge models.

```
summary(boost_loan)
```




```
##                     var   rel.inf
## int_rate       int_rate 31.263331
## dti                 dti 21.406363
## annual_inc   annual_inc 18.000853
## installment installment 17.316712
## loan_amnt     loan_amnt  9.613146
## term               term  2.399596
```

The boosting model prioritizes int_rate, dti, and annual_inc as the top 3 most important predictors in determining loan default.

XGboost

```
library(xgboost)
dtrain = xgb.DMatrix(data = X_train, label = Y_train)
set.seed(500)
xgb_loan = xgboost(data = dtrain,
                   max_depth = 2,
                   eta = .1,
                   nrounds = 40,
                   lambda = 0,
                   print_every_n = 10,
                   objective = "binary:logistic")
```

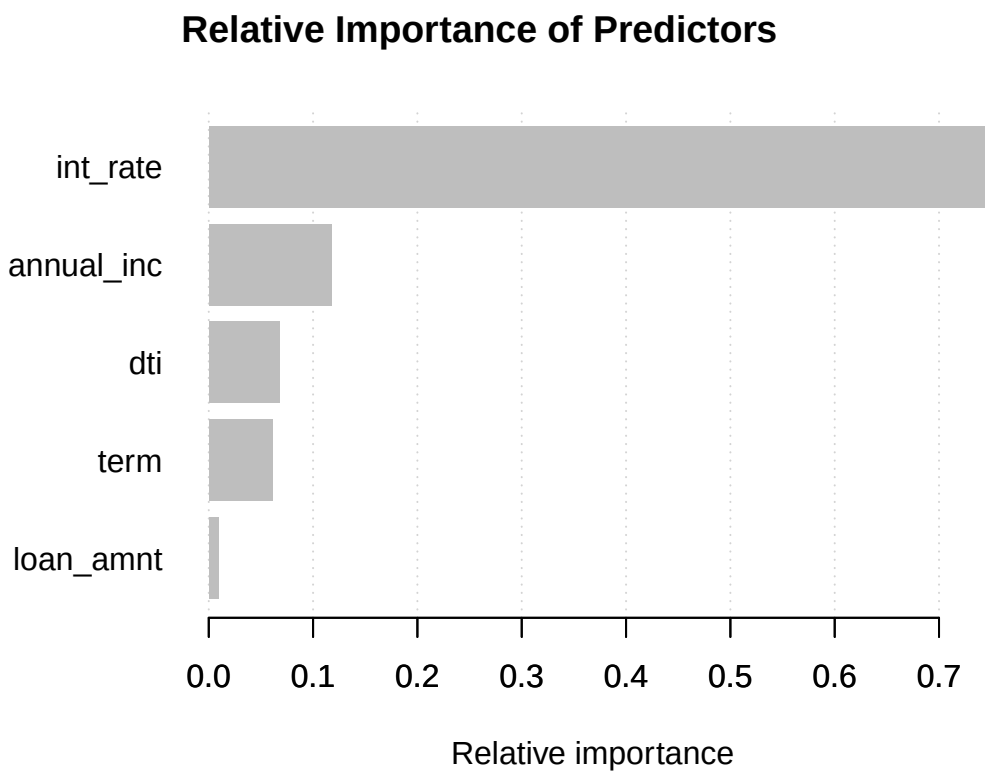
```
# saveRDS(xgb_loan, file = "xgb_loan.rds")
```

```
library(xgboost)
xgb_loan = readRDS("xgb_loan.rds")

yhat_xgb = predict(xgb_loan,
                   newdata = X_test,
                   type = "response")
xgb_boost = error_rate(yhat_xgb)
error_table = rbind(error_table, xgb_boost)
xgb_boost
```

```
## [1] 0.2116665
```

When doing xgbboosting compared to normal boosting, we achieve a slightly lower error rate. It is roughly 1% lower than normal boosting.

```
importance = xgb.importance(colnames(X_train), model = xgb_loan)
xgb.plot.importance(importance, xlab = "Relative importance")
title(main = "Relative Importance of Predictors")
```



The top three most important predictors: int_rate, annual_inc, and dti are also the same as the normal boost method except annual_inc is swapped with dti in this case.

Neural Net

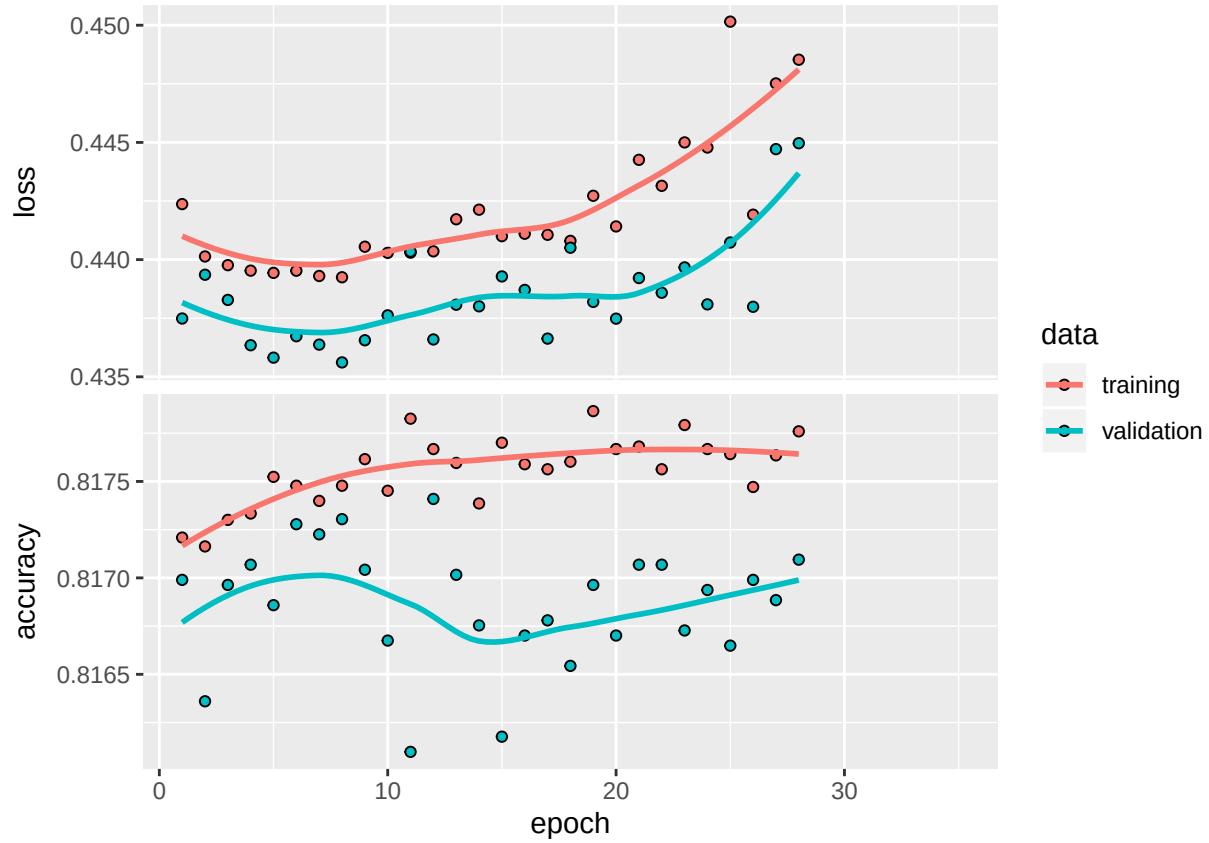
```
library(tensorflow)
library(keras)
set.seed(9999)
```

```
train_data = scale(X_train)
test_data = scale(X_test)
train_labels = to_categorical(Y_train, 2)
test_labels = to_categorical(Y_test, 2)
```

```
nn_model = keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 2, activation = "softmax")


nn_model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

```
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 20)

epochs=35
history_class <- nn_model %>%
  fit(
    train_data,
    train_labels,
    epochs = epochs,
    validation_split = 0.2,
    callbacks = list(early_stop)
)
# saveRDS(history_class, file = "history_class.RDS")
```

```
history_class = readRDS("history_class.rds")
plot(history_class)
```



After training our neural net, we see that we do not need to train the model very long. The accuracy peaks around 10 epoch's and any further than that we increase our loss as the training goes on.

```
test_predictions = nn_model %>%
  predict(test_data)

test_class = nn_model %>%
  predict_classes(test_data)

#nn_model = error_rate_new(test_class)
# saveRDS(nn_model, "nn_error.rds")
nn_model = readRDS("nn_error.rds")
error_table = rbind(error_table, nn_model)
nn_model
```

```
## [1] 0.195938
```

The neural net's error rate is 19.59%, this is slightly higher almost miniscule than our second best model of bagging.It is in the thousandths difference.

```
#nn_table = table("actual" = test_labels[, 2], test_class)
#saveRDS(nn_table, "nn_table.rds")
nn_table = readRDS("nn_table.rds")
nn_table
```

```
##       test_class
```

```
## actual     0     1
##      0 50903  1014
##      1 11431   167
```

In this case, the neural net had a hard time distinguishing defaulted loans and often putting them as non-defaulted 11431 times.

Comparing All Models

```
error_table
```

```
##                     Error_Rate
## model_4              0.2761238
## probit_mod           0.2690860
## lasso_mod            0.3163505
## ridge_mod            0.3141148
## bagging_mod          0.1954027
## best_random_forest   0.1825710
## boost_mod            0.2247343
## xgb_boost            0.2116665
## nn_model             0.1959380
```

After compiling all the models and putting their error rates into a table, we can finally compare and rank the results. Each model has some good and bad to them but the model that performed the best was the random forest tuned to selecting one predictor at a split. It's error rate was 18.25% which means that its accuracy rate was 81.75%. Bagging method was a relatively close in second with a 19.54% error rate and the neural net was only less than .01% higher than that. Both boosting methods were close after that being similiar to each other in the 21-22% error rate. The worse models that performed were in the >30% error rates with the lasso and ridge models.

Conclusion

I used a variety of different models in this paper. Each of them had various pros and cons in creating them and using them. The top performing models were sub 20% in error rate. Those models determined that interest rates, annual income, and debt income ratio (dti) were the three most important predictors in determine loan default. It makes sense why these three are up there. If your annual income is low, you're less likely to be able to pay off the loan. If interest rates are super high, the loan amount is going to keep growing and growing. The dti is also important as someone who has a lot of their income going to other debt obligations (high dti), adding on more debt is just going to end up swamping them before they are forced to default on the loan. These results are very interesting to me as the models were able to hone in on these main three predictors. The full dataset contained 74 variables, around 15-20 of which were filled with NA's. A lot of them were qualitative with groups more than 10 such as employment, employment length, grade of loan, state of loan and purpose of loan. These variables were omitted as I would be left with huge models with lots of dummy variables in them. As a result, I was left with around 10-12 useable numeric variables. The limitations of this paper were my laptops computational power which caused me to pick what I thought were 6 of the most important variables to train my models on. Even then, some of the models took 15-30 minutes to create. I couldn't imagine the time it would take using more than the 6 variables I pre-selected.

The computational power of my laptop showed greatly throughout this project and it would be interesting to repeat this project with more CPU and GPU power. This would allow me to select more variables to

use beforehand, and allow be to have more time to tune my models finer. The model that I tuned the best, which also took the one of the quickest to run, was the model that performed the best. That model was the tuned random forest. Perhaps the boosting and XGBboosting model could have also performed similiarly or ultimately beat the best model if they had more time/power to be able to tune itself. I think the neural net could have also been the best model with a little more time to tune and include more variables within it.