Analyzing Consumer Behavior in LendingClub Data

Jia Lin Mei | Report 4

April 28, 2020

Introduction

LendingClub is a peer-to-peer service that matches borrowers to lenders while charging fees to those that use their online platform. One of the first to pioneer this method, it has become a very reputable company in the financial services industry that helps borrowers attain loans with lower interest rates and investors gain competitive returns. A crucial part of this service is screening borrowers and only allowing those who have the highest chances of paying back their loans to be part of the company's clientele. Thus, the goal of my project is to identify the qualities of borrowers or their cases that leads them to successfully pay back their loan. In my data from Kaggle.com, I am looking at LendingClub's quarterly loan data from 2007-2015. This is qualtitative data set. I am seeking to understand a relationship between loan status (current, late, or fully paid) and the 75 possible variables. The file is a matrix of about 890 thousand observations.

First Report

1.1 - Picking my X's

Before uploading my dataset onto R, I first removed variables and columns that I was not interested in, making a dataset that could be loaded more quickly onto in R. I picked 4 variables or X's that I believe could explain Y would be annual income, grade, employment length, and interest rate.

```r
setwd('/Users/Personal/Documents/1. Spring 2020/2. Econ 490/Homework')
loan <-read.csv("loan3.csv", header = T)
str(loan)
```

```
## 'data.frame':    1048575 obs. of  7 variables:
##  $ loan_status: chr  "Current" "Current" "Current" "Current" ...
##  $ annual_inc : num  55000 90000 59280 92000 57250 ...
##  $ grade      : chr  "C" "D" "D" "D" ...
##  $ emp_length : chr  "10+ years" "10+ years" "6 years" "10+ years" ...
##  $ int_rate   : num  13.6 18.9 18 18.9 16.1 ...
##  $ funded_amnt: int  2500 30000 5000 4000 30000 5550 2000 6000 5000 6000 ...
##  $ installment: num  84.9 777.2 180.7 146.5 731.8 ...
```

All the variables are in the right type to be used in my analysis. Finally, to make things smoother later on, I notice that I have "n/a" values in my dataset. I will use subsetting to turn those into NA and remove those observations from my dataset.

```r
loan[loan == "n/a"] <- NA
loan <- na.omit(loan)
```

1.2 - Plots of Variables

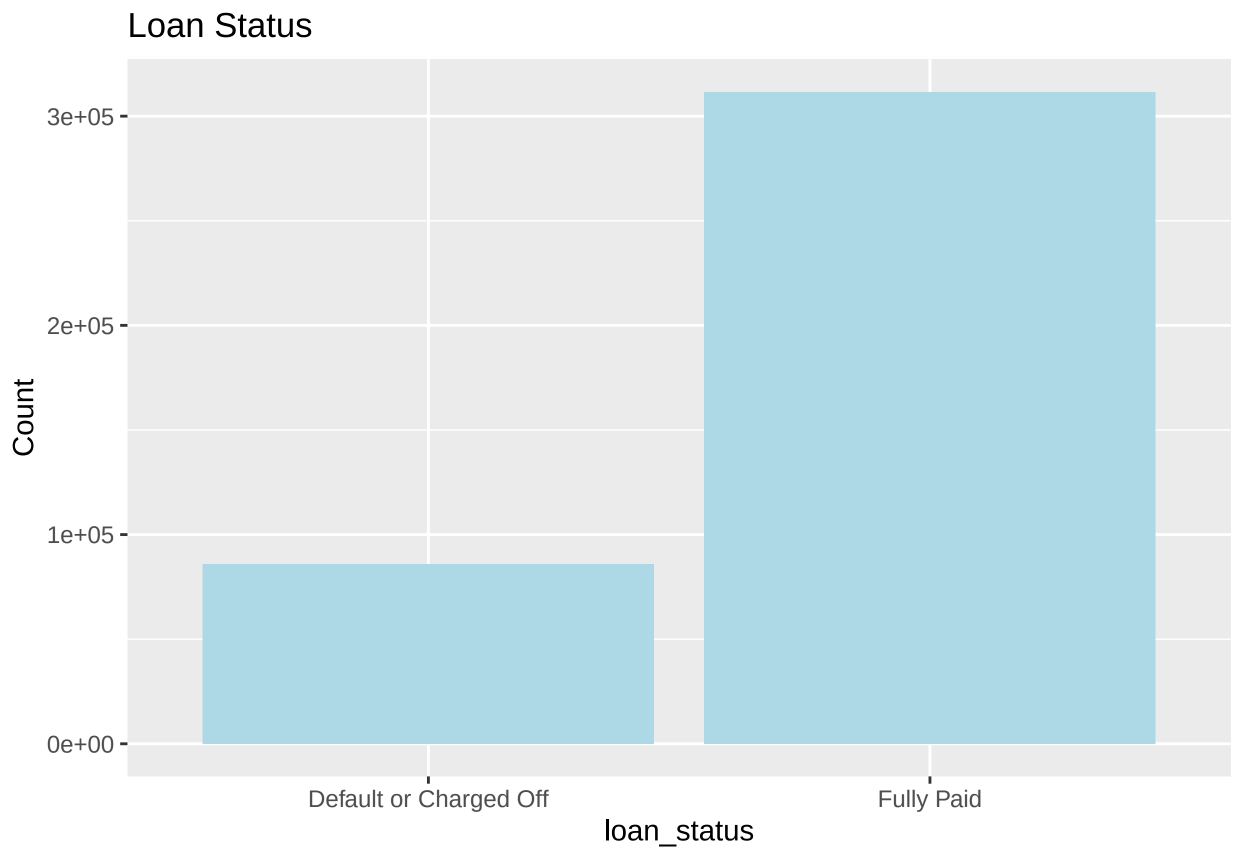
Below is a histogram of my dependent, categorical variable of interest, loan status. Originally, the variable actually had several possible levels such as “Grace Period” and “Late (16-30 Days).” However, because the logit model in R limits me to only binary variables (having only two values), I had to eliminate some of the levels or reassign their observations to other levels. Upon checking that very few people had actually defaulted, I decided to include “Charged Off” since those accounts are assumed to never be fully paid off. I will change the level name of that variable to “Default or Charged Off” to refelct this change.

```r
loan[loan$loan_status == "Charged Off",]$loan_status <- "Default"
loan <- loan[!loan$loan_status == "Late (16-30 days)",]
loan <- loan[!loan$loan_status == "In Grace Period",]
loan <- loan[!loan$loan_status == "Late (31-120 days)",]
loan <- loan[!loan$loan_status == "Current",]
```

```r
loan$loan_status <- factor(loan$loan_status)
loan$loan_status <- as.character(loan$loan_status)
loan[loan$loan_status == "Default",]$loan_status <- "Default or Charged Off"
loan$loan_status <- factor(loan$loan_status)
```

In the future, I hope to improve on this by adding a “Current” category as there were a lot of observations with this status, and possibly use a multinomial logit model (if it’s introduced in class).

```r
ggplot(data = loan, mapping = aes(x = loan_status)) +
  geom_bar(fill="lightblue") +
  labs(title = "Loan Status", y = "Count")
```



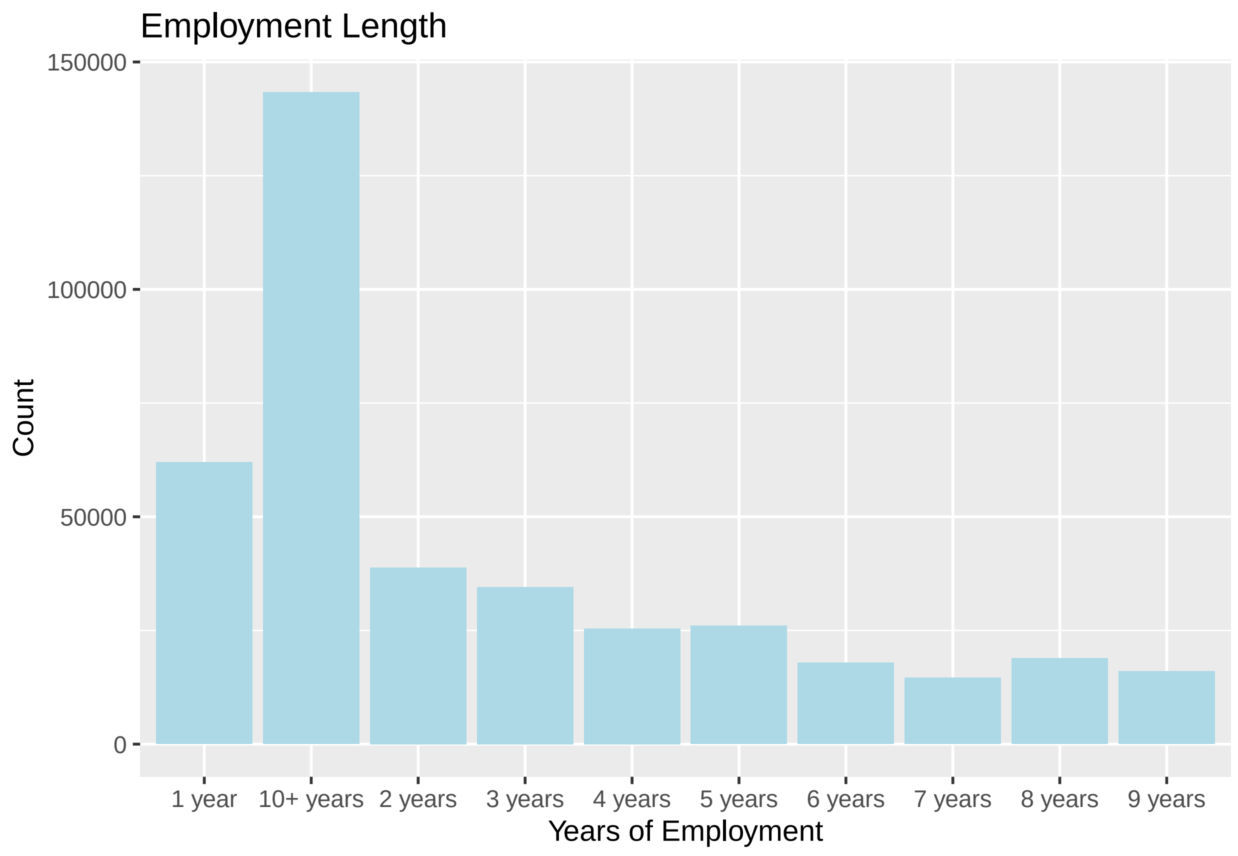
```
table(loan$loan_status)
```

```
##
## Default or Charged Off            Fully Paid
##                  85903                311623
```

As the table shows, a greater fraction of borrowers paid off their loan than deafulted. Now, I will look at the graphs of my regressors to see if there's any casual relationship. Befire I do that however, I can see that emp_length needs some cleaning because there are two different levels that basically mean one year.

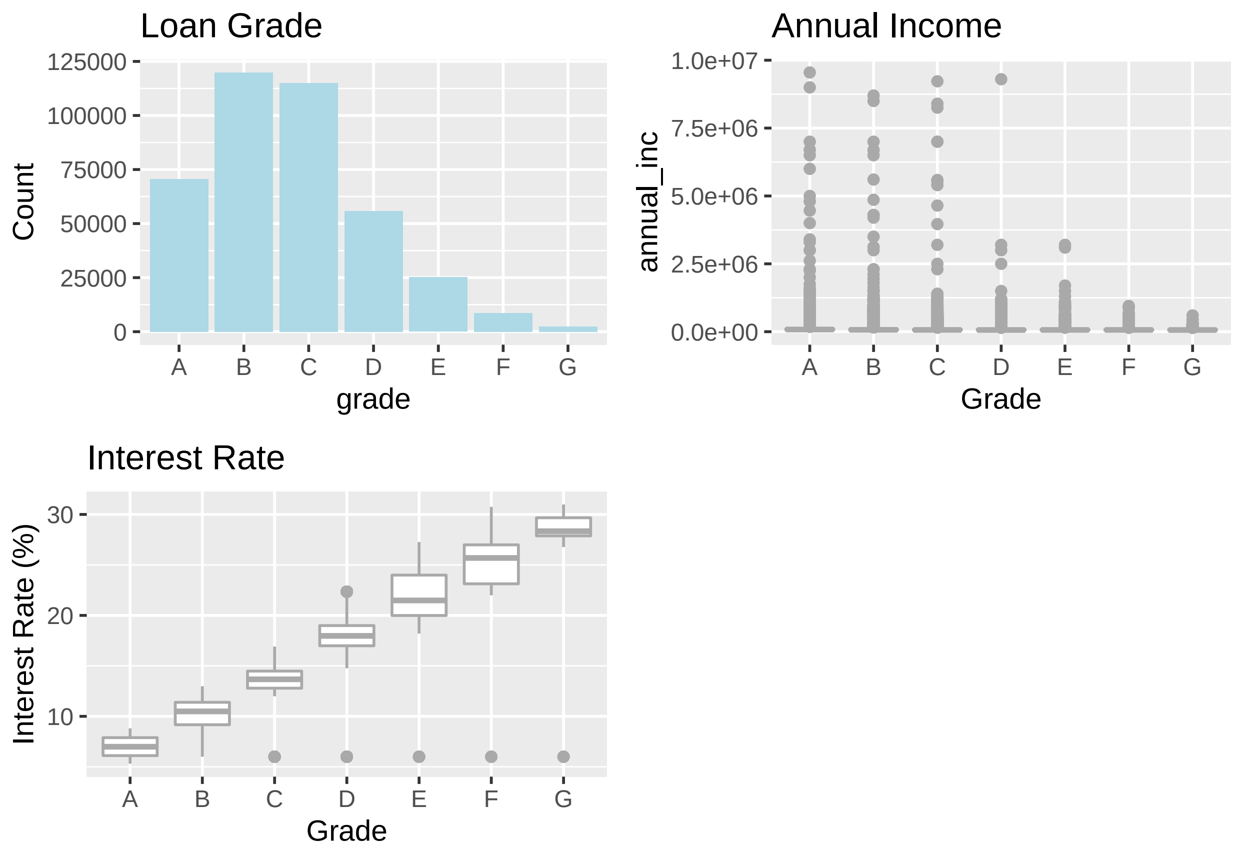
```
loan[loan$emp_length == "< 1 year",]$emp_length <- "1 year"
loan$emp_length <- factor(loan$emp_length)
```

```
p1.1grade <- ggplot(data =loan, mapping = aes(x=grade)) +
  geom_bar(fill = "lightblue") +
  labs(title = "Loan Grade", y = "Count")

p1.2inc <- ggplot(data=loan, mapping = aes(y=annual_inc, x = grade)) + geom_boxplot(color= "darkgray")
  labs(title = "Annual Income", x = "Grade", Y = "Amount in Dollars")

p1.3emp <- ggplot(data = loan, mapping = aes(x=emp_length)) +
  geom_bar(fill = "lightblue") +
  labs(title = "Employment Length", x = "Years of Employment", y = "Count")
p1.3emp
```



```
p1.4ir <- ggplot(data = loan, aes(x=grade, y=int_rate)) +
  geom_boxplot(color = "darkgray") +
  labs(title = "Interest Rate", x = "Grade", y = "Interest Rate (%)")
```

```
grid.arrange(p1.1grade,p1.2inc,p1.4ir,nrow = 2)
```



Looking at the graphs, I believe loan grade, annual income, and interest rate will be significant predictors of my y variable.

Here are plots of the regressors that I believe has an effect on Y. Loan grade represents how volatile the loan issued was, with G being the most volatile and having the most potential for return. For this variable, I had to turn it into a factor class so R could know how to analyze it. As you can see, most of the loans were risk averse. I made box plots for annual income, employment length, and interest rate. As you can see, most people in the data have an income below a million dollars, have been employed for at least a decade, and made loans with interest rates from 10 - 20%.

1.3 - Correlations between Y and X Variables

In this section, I will make plot a correlation between Y and each of my 4 X's. Because linear regression is not appropriate way to fit a model on a categorical variable, I will be using logistic regressions instead. Just to make sure, I need to check whether the probabilities of my logistic models corresponds with fully paid or defaulted/charged off. As the bottom line of code shows, they correspond with the chances of you paying off the loan on your account.

```
contrasts(loan$loan_status)
```

```
##                         Fully Paid
## Default or Charged Off           0
## Fully Paid                       1
```

For each regression, z-statistics are made for each regressor and its factors. I will initially assume that they have no effect on loan status.

```r
model1.1=glm(loan_status~grade,data=loan,family=binomial)
summary(model1.1)
```

```
##
## Call:
## glm(formula = loan_status ~ grade, family = binomial, data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3502   0.3613   0.5657   0.7506   1.3000
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.69640    0.01547  174.29   <2e-16 ***
## gradeB      -0.94515    0.01748  -54.07   <2e-16 ***
## gradeC      -1.57360    0.01692  -93.00   <2e-16 ***
## gradeD      -2.02793    0.01787 -113.49   <2e-16 ***
## gradeE      -2.39702    0.02004 -119.60   <2e-16 ***
## gradeF      -2.77298    0.02645 -104.83   <2e-16 ***
## gradeG      -2.98005    0.04475  -66.59   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 383006  on 397519  degrees of freedom
## AIC: 383020
##
## Number of Fisher Scoring iterations: 5
```

The p-value for all the levels of grade are very low, signifying that it has explanatory power. It seems that as you go up grades, your chances of paying off your account decreases (risk of defaulting increases). This makes sense because the loans become riskier.

```r
model1.2=glm(loan_status~annual_inc,data=loan,family=binomial)
summary(model1.2)
```

```
##
## Call:
## glm(formula = loan_status ~ annual_inc, family = binomial, data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.3306   0.6125   0.6936   0.7187   0.7709
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.061e+00  7.652e-03  138.72   <2e-16 ***
## annual_inc  2.920e-06  8.735e-08   33.42   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 413648  on 397524  degrees of freedom
## AIC: 413652
##
## Number of Fisher Scoring iterations: 4
```

The p-value for annual income is very low, signifying that it has explanatory power. It seems that as your income increases, your chances of paying off your account increases (risk of defaulting decreases). This makes sense because you would greater ability to pay it off.

```r
model1.3=glm(loan_status~emp_length,data=loan,family=binomial)
summary(model1.3)
```

```
##
## Call:
## glm(formula = loan_status ~ emp_length, family = binomial, data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7819   0.6763   0.6763   0.7119   0.7212
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        1.213812   0.009558 126.999  < 2e-16 ***
## emp_length10+ years 0.145101   0.011586  12.523  < 2e-16 ***
## emp_length2 years   0.029600   0.015480   1.912 0.055862 .
## emp_length3 years   0.020138   0.016035   1.256 0.209169
## emp_length4 years   0.049841   0.017903   2.784 0.005370 **
## emp_length5 years   0.044420   0.017732   2.505 0.012241 *
## emp_length6 years   0.093789   0.020613   4.550 5.36e-06 ***
## emp_length7 years   0.084658   0.022305   3.795 0.000147 ***
## emp_length8 years   0.066309   0.020067   3.304 0.000952 ***
## emp_length9 years   0.053287   0.021336   2.497 0.012508 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 414735  on 397516  degrees of freedom
## AIC: 414755
##
## Number of Fisher Scoring iterations: 4
```

The p-value for the levels of employment length seems to decrease as employment length increases. This is interesting; I suppose that as you gain more work experience, it becomes a better indication on your ability to pay off your loan. I'm not sure if it'll be effective to remove the levels that don't have large explanatory power.

```r
model1.4=glm(loan_status~int_rate,data=loan,family=binomial)
summary(model1.4)
```

```
##
## Call:
```

```
## glm(formula = loan_status ~ int_rate, family = binomial, data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2237   0.4202   0.5595   0.6955   1.6018
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.0754306  0.0119997   256.3   <2e-16 ***
## int_rate    -0.1301624  0.0007849  -165.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 385733  on 397524  degrees of freedom
## AIC: 385737
##
## Number of Fisher Scoring iterations: 4
```

The p-value for interest rate is very low, signifying that it has explanatory power. This makes sense, as the interest rate increases, the payments you have to make increase, and you're less likely to pay them off (rate of defaulting increases).

Second Report

2.1 - Fitting my 5 Logistic Regressions

For my second report, I will be making more complex logit regressions using the variables I tested. Looking back at my previous report, I can see that the coefficient for grade had a very low p-value and was shown to have explanatory power. This makes sense, as grade shows how risk averse a borrower is and how he or she may behave in the future. Thus, I will be building my models off of that first. I will pay special attention to interest rate and annual income as those also had low p-values. Emp_length seemed to be insignificant for some of its levels, so I am considering leaving that variable out entirely. While it's true that people who have worked longer might be better at paying off their loans, this data set doesn't distinguish years worked after 10 years, so it may not be a good variable to look at.

```
model2.1=glm(loan_status~grade+annual_inc,data=loan,family=binomial)
summary(model2.1)
```

```
##
## Call:
## glm(formula = loan_status ~ grade + annual_inc, family = binomial,
##     data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.4828   0.3550   0.5682   0.7552   1.3332
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.577e+00  1.723e-02  149.60   <2e-16 ***
## gradeB      -9.286e-01  1.751e-02  -53.03   <2e-16 ***
## gradeC      -1.553e+00  1.697e-02  -91.50   <2e-16 ***
## gradeD      -2.003e+00  1.794e-02 -111.67   <2e-16 ***
## gradeE      -2.373e+00  2.010e-02 -118.07   <2e-16 ***
## gradeF      -2.748e+00  2.650e-02 -103.70   <2e-16 ***
## gradeG      -2.951e+00  4.480e-02  -65.86   <2e-16 ***
## annual_inc   1.292e-06  8.428e-08   15.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 382738  on 397518  degrees of freedom
## AIC: 382754
##
## Number of Fisher Scoring iterations: 5
```

Holding grade constant, the p-value of the coefficient for annual income is still significant and has a positive effect on the chances of a customer paying off their loan.

```r
model2.2=glm(loan_status~grade+emp_length,data=loan,family=binomial)
summary(model2.2)
```

```
##
## Call:
## glm(formula = loan_status ~ grade + emp_length, family = binomial,
##     data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3716   0.3520   0.5692   0.7551   1.3281
##
## Coefficients:
##                       Estimate Std. Error  z value Pr(>|z|)
## (Intercept)            2.63008    0.01801  146.043  < 2e-16 ***
## gradeB                -0.94321    0.01748  -53.955  < 2e-16 ***
## gradeC                -1.57070    0.01692  -92.814  < 2e-16 ***
## gradeD                -2.02527    0.01787 -113.319  < 2e-16 ***
## gradeE                -2.39455    0.02005 -119.457  < 2e-16 ***
## gradeF                -2.77129    0.02646 -104.743  < 2e-16 ***
## gradeG                -2.97760    0.04477  -66.515  < 2e-16 ***
## emp_length10+ years    0.12016    0.01207    9.952  < 2e-16 ***
## emp_length2 years      0.02946    0.01614    1.826   0.0679 .
## emp_length3 years      0.01847    0.01672    1.105   0.2691
## emp_length4 years      0.04786    0.01866    2.565   0.0103 *
## emp_length5 years      0.04203    0.01848    2.274   0.0230 *
## emp_length6 years      0.09524    0.02147    4.435 9.21e-06 ***
## emp_length7 years      0.07084    0.02324    3.049   0.0023 **
## emp_length8 years      0.04951    0.02095    2.364   0.0181 *
## emp_length9 years      0.05118    0.02225    2.301   0.0214 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 382871  on 397510  degrees of freedom
## AIC: 382903
##
## Number of Fisher Scoring iterations: 5
```

Holding grade constant, only two of the factor levels for employment length seem to be insignificant, the rest still have some explanatory power. They don't seem to be as significant as my other regressors, so to make it easier on myself, I'll just leave this variable out.

```
model2.3=glm(loan_status~grade+int_rate,data=loan,family=binomial)
summary(model2.3)
```

```
##
## Call:
## glm(formula = loan_status ~ grade + int_rate, family = binomial,
##     data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3792   0.3488   0.5629   0.7499   1.3473
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.015787   0.024413  123.53   <2e-16 ***
## gradeB      -0.790420   0.019718  -40.09   <2e-16 ***
## gradeC      -1.256911   0.025188  -49.90   <2e-16 ***
## gradeD      -1.508751   0.035441  -42.57   <2e-16 ***
## gradeE      -1.702800   0.045575  -37.36   <2e-16 ***
## gradeF      -1.908032   0.057394  -33.24   <2e-16 ***
## gradeG      -1.969778   0.074468  -26.45   <2e-16 ***
## int_rate    -0.046370   0.002734  -16.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 382718  on 397518  degrees of freedom
## AIC: 382734
##
## Number of Fisher Scoring iterations: 5
```

Holding grade constant, interest rate still exhibits a lot of explanatory power on loan status. What's interesting here is that after separating the effect of the interest rate, grade now has a positive effect on loan repayment. All the variables still have low p-values and are significant.

In order to make up for not including employment length in my model, I decided to add two more variables into the dataset that likely explain the behavior or the loan status variable: funded amount and installments. Funded amount represents how much money was given in each loan, and installments represents how much borrowers paid off their loan each month. Below are my graphs, and I will discuss them more in the following section.

```
model2.4=glm(loan_status~grade+annual_inc+int_rate,data=loan,family=binomial)
summary(model2.4)
```

```
##
```

```
## Call:
## glm(formula = loan_status ~ grade + annual_inc + int_rate, family = binomial,
##     data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.4943   0.3501   0.5646   0.7509   1.3763
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.896e+00  2.559e-02  113.17   <2e-16 ***
## gradeB      -7.747e-01  1.974e-02  -39.24   <2e-16 ***
## gradeC      -1.238e+00  2.521e-02  -49.09   <2e-16 ***
## gradeD      -1.486e+00  3.547e-02  -41.90   <2e-16 ***
## gradeE      -1.682e+00  4.560e-02  -36.88   <2e-16 ***
## gradeF      -1.887e+00  5.742e-02  -32.86   <2e-16 ***
## gradeG      -1.944e+00  7.450e-02  -26.10   <2e-16 ***
## annual_inc   1.287e-06  8.430e-08   15.27   <2e-16 ***
## int_rate    -4.620e-02  2.735e-03  -16.89   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 382452  on 397517  degrees of freedom
## AIC: 382470
##
## Number of Fisher Scoring iterations: 5
```

Regressing on my previous variables, they still appear to be significant.

```r
model2.5=glm(loan_status~grade+annual_inc+int_rate+funded_amnt+installment,data=loan,family=binomial)
summary(model2.5)
```

```
##
## Call:
## glm(formula = loan_status ~ grade + annual_inc + int_rate + funded_amnt +
##     installment, family = binomial, data = loan)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.9478   0.3431   0.5587   0.7417   1.4497
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.989e+00  2.590e-02  115.39   <2e-16 ***
## gradeB      -7.638e-01  1.978e-02  -38.62   <2e-16 ***
## gradeC      -1.197e+00  2.531e-02  -47.30   <2e-16 ***
## gradeD      -1.428e+00  3.557e-02  -40.15   <2e-16 ***
## gradeE      -1.581e+00  4.576e-02  -34.55   <2e-16 ***
## gradeF      -1.771e+00  5.762e-02  -30.73   <2e-16 ***
## gradeG      -1.843e+00  7.466e-02  -24.68   <2e-16 ***
## annual_inc   2.524e-06  1.013e-07   24.93   <2e-16 ***
## int_rate    -4.859e-02  2.752e-03  -17.65   <2e-16 ***
## funded_amnt -3.921e-05  1.384e-06  -28.34   <2e-16 ***
## installment  8.692e-04  4.552e-05   19.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 414953  on 397525  degrees of freedom
## Residual deviance: 381250  on 397515  degrees of freedom
## AIC: 381272
##
## Number of Fisher Scoring iterations: 5
```

2.2 - Choosing my Best Regression

```r
summary(model2.5)$coef
```

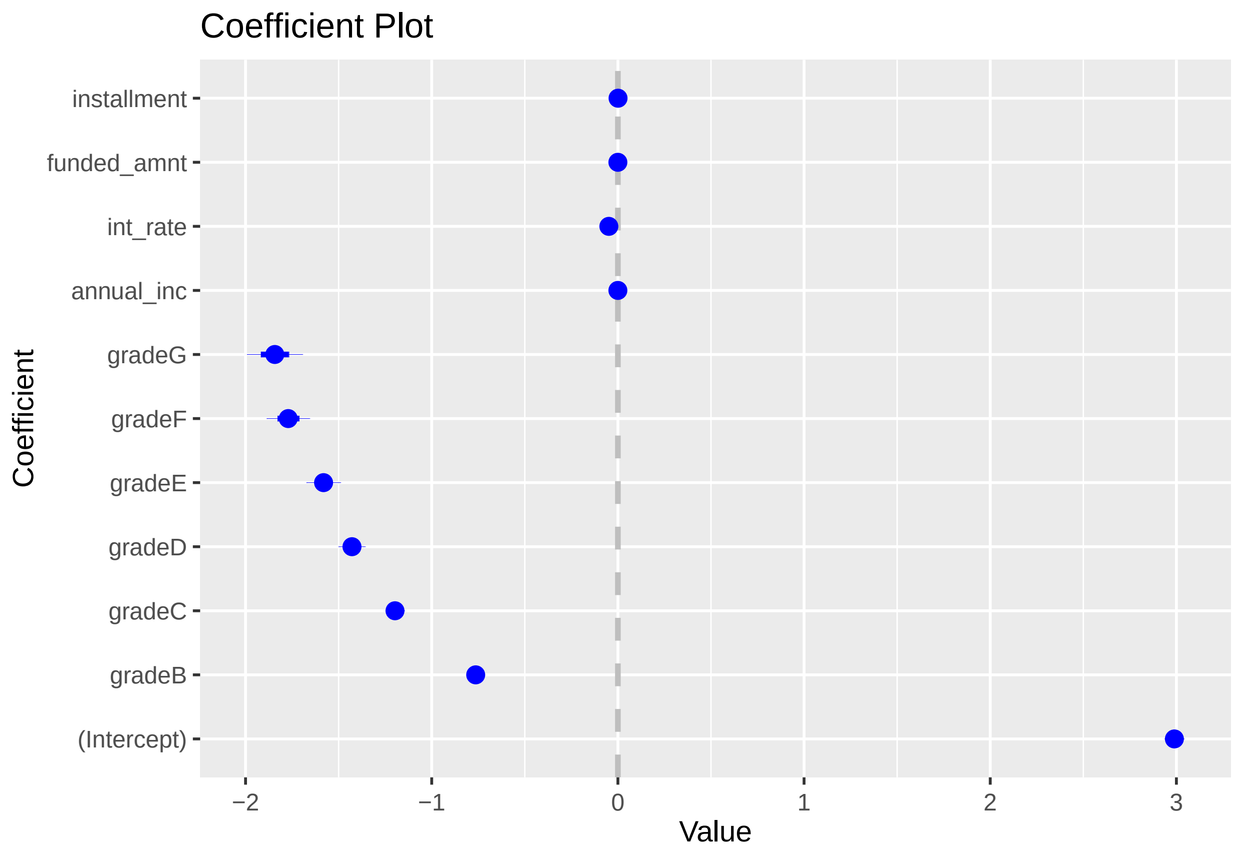
```
##                  Estimate   Std. Error   z value      Pr(>|z|)
## (Intercept)  2.989162e+00 2.590474e-02 115.39054  0.000000e+00
## gradeB      -7.637781e-01 1.977710e-02 -38.61931  0.000000e+00
## gradeC      -1.197177e+00 2.531230e-02 -47.29627  0.000000e+00
## gradeD      -1.428278e+00 3.557221e-02 -40.15152  0.000000e+00
## gradeE      -1.580919e+00 4.576459e-02 -34.54459 1.718626e-261
## gradeF      -1.770640e+00 5.761766e-02 -30.73085 2.204152e-207
## gradeG      -1.842729e+00 7.465799e-02 -24.68227 1.658117e-134
## annual_inc   2.524025e-06 1.012586e-07  24.92651 3.839316e-137
## int_rate    -4.858748e-02 2.752428e-03 -17.65259  9.719728e-70
## funded_amnt -3.921068e-05 1.383833e-06 -28.33484 1.286833e-176
## installment  8.691584e-04 4.551747e-05  19.09505  2.775825e-81
```

My method of adding variables onto grade proved to be fruitful. Doing so allowed me to surely eliminate other variables from my model that didn't seem to have much explanatory power. The best regression I found was model 2.5 where loan status is regressed by grade, annual income, interest rate, funded amount, and installments. Because all these coefficients have very low p-values, I can reject the null and say they play a role in explaining the behavior of loan status.

I can interpret the coefficients as how much your chances of fully paying off your loan or not (defaulting or being very late) increases per unit. As the grade goes up, it has a positive effect on the rate of paying off the loan. It initially had a negative effect, but after I separated the effect of the interest rate, it changed. Same goes with installments; the more you pay off each month, the more likely you are to pay off your loan. The reverse is true for interest rates and funded amounts; the more you're funded might be an indication of how much trouble you're in. Higher interest rates naturally make loans harder to pay off.

2.3 Graph of Coefficients and Standard Error

```r
coefplot(model2.5)
```



Here is a plot showing the coefficients of my models and their standard errors, expressed as a confidence interval. Apart from grade, many of the coefficient values are close to zero, so while they are significant, I havea feeling they may be eliminated in other moders. However, the standard errors appear to be small, which is a good sign.

2.4 - Confusion Matrix and True Positive & False Positive Rates

To calculate these, I will first need to train my model on a portion of the data set, test it on another, and see how the predicted values compared to the orginal values. I will split the dataset into halves.

```
set.seed(2)
train2 = loan %>% sample_frac(0.4701)
test2 = loan %>% setdiff(train2)
```

```
glmtrain2.1=glm(loan_status~grade+int_rate+funded_amnt+installment+annual_inc,data=train2,family=binomi
summary(glmtrain2.1)
```

```
##
## Call:
## glm(formula = loan_status ~ grade + int_rate + funded_amnt +
##     installment + annual_inc, family = binomial, data = train2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.9595   0.3461   0.5595   0.7409   1.4253
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.966e+00  3.766e-02   78.75   <2e-16 ***
## gradeB      -7.522e-01  2.866e-02  -26.24   <2e-16 ***
## gradeC      -1.182e+00  3.675e-02  -32.16   <2e-16 ***
## gradeD      -1.434e+00  5.173e-02  -27.73   <2e-16 ***
## gradeE      -1.568e+00  6.661e-02  -23.54   <2e-16 ***
## gradeF      -1.746e+00  8.373e-02  -20.85   <2e-16 ***
## gradeG      -1.787e+00  1.089e-01  -16.41   <2e-16 ***
## int_rate    -4.810e-02  4.009e-03  -12.00   <2e-16 ***
## funded_amnt -4.070e-05  2.010e-06  -20.25   <2e-16 ***
## installment  9.228e-04  6.614e-05   13.95   <2e-16 ***
## annual_inc   2.537e-06  1.479e-07   17.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 195432  on 186876  degrees of freedom
## Residual deviance: 179651  on 186866  degrees of freedom
## AIC: 179673
##
## Number of Fisher Scoring iterations: 5
```

Now that we have the trained model, we will use the testing dataset in order to predict the loan status for our test data. Afterwards, we can compute the predictions and comapre them to the actual results from the original dataset.

```
glmtest2.2=predict(glmtrain2.1,test2,type="response")
glmtest2.2[1:25]
```

```
##         1         2         3         4         5         6         7         8
## 0.6343810 0.5971922 0.9319040 0.7691295 0.6877595 0.6033781 0.9493062 0.9585378
##         9        10        11        12        13        14        15        16
## 0.9421790 0.9299905 0.8220454 0.8240758 0.7736542 0.7148447 0.8457206 0.8463025
##        17        18        19        20        21        22        23        24
## 0.6631351 0.9422358 0.7687342 0.6543103 0.9311203 0.9296649 0.7435399 0.8770258
##        25
## 0.7227116
```

The probability of each observation being fully paid is actually quite high. This seems to match up with how many fully paid observations there were in the original dataset. Because of this, I will set a stricter threshold for the observations to be treated as fully paid.

```
glmpredict2.3=rep("Default or Charged Off",186630)
glmpredict2.3[glmtest2.2>.8] = "Fully Paid"
table(glmpredict2.3,test2$loan_status)
```

```
##
## glmpredict2.3            Default or Charged Off Fully Paid
##   Default or Charged Off                  32174      69092
##   Fully Paid                              11172      74192
```

```
#True Positive Rate
(74192)/(74192+11172)
```

```
## [1] 0.8691252
```

```
#False Positive Rate
(32174)/(32174+69092)
```

```
## [1] 0.3177177
```

```
1 - (74192+32174)/186630
```

```
## [1] 0.4300702
```

```
logit_error = 1 - mean(glmpredict2.3 == test2$loan_status)
logit_error
```

```
## [1] 0.4300702
```

The TP rate is relatively high, and the FP rate is pretty low. This is normally a good sign, but it appears on average, the logit model isn't really accurate in predicting whether an observation defaults or gets paid off. The error rate is around 43%, which is a little better than guessing.

2.5 - Probit Regression

```
glmtrain2.4=glm(loan_status~grade+int_rate+funded_amnt+installment+annual_inc,data=train2,family = bino
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glmtest2.5=predict(glmtrain2.4, test2, type = "response")

glmpredict2.6=rep("Default or Very Late",186630)
glmpredict2.6[glmtest2.5>.8] = "Fully Paid"
table(glmpredict2.6,test2$loan_status)
```

```
##
## glmpredict2.6          Default or Charged Off Fully Paid
##   Default or Very Late                  32464      70421
##   Fully Paid                            10882      72863
```

```
#True Positive Rate
(72863)/(72863+10882)
```

```
## [1] 0.8700579
```

```
#False Positive Rate
(32464)/(32464+70421)
```

```
## [1] 0.3155368
```

```
1 - mean(glmpredict2.6 ==test2$loan_status)
```

```
## [1] 0.6095858
```

Compared to the rates from before, the TP and TN rates appear to be the same, meaning the model holds about the same even when using a function based off of the normal distribution. However, the error rate seems to show that the probit model isn;t very accurate.

Third Report

3.1 - Dividing my Data

As the question asked for, I will start by splitting my data set into a training and testing sets in order to estimate the test errors of my ridge regressions and lasso functions.

```
set.seed(3)
loan2 = loan[,-4]
```

```r
loan2$grade <- as.factor(loan2$grade)
train3 = loan2 %>% sample_frac(0.415)
test3 = loan2 %>% setdiff(train3)

grid = 10^seq(10, -2, length = 100)

x = model.matrix(loan_status~., train3)[,-1]
y = train3$loan_status

x_train3 = model.matrix(loan_status~., train3)[,-1]
y_train3 = train3$loan_status

x_test3 = model.matrix(loan_status~., test3)[,-1]
y_test3 = test3$loan_status
```

3.2 - Ridge and Lasso Regression

a) Using cross-validation to find the flexibly of the model (lambda)

```r
cv.out = cv.glmnet(x_train3, y_train3, alpha = 0,family = "binomial")
```

```r
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.01127969
```

For my ridge model, the lambda that appears to minimizes the cross validation error appears to be a very small decimal. That means the original model may not need a lot of tuning; the variance of the original model was already at an optimal level. Below is my lasso model.

```r
cv.out2 = cv.glmnet(x_train3, y_train3, alpha = 1,family = "binomial")
```

```r
bestlam2 = cv.out2$lambda.min
bestlam2
```

```
## [1] 0.0002430135
```

According to the lasso model, the tuning parameter should be even smaller. Thus, I don't believe shrinkage would help in fine-tuning my model since the results will be so close to the original logit model anyways. However, I will test out both models to see how well they work.

b) Choose the lambda which is within 1 standard error of the minimum lambda

Here, I first extract the standard error and than add it to the minimum lambda. Here are the values for both kinds of graphs.

```r
#Ridge Model
mse.min1 <- min(cv.out$cvm)
mse.min1 + bestlam
```

```
## [1] 0.9759922
```

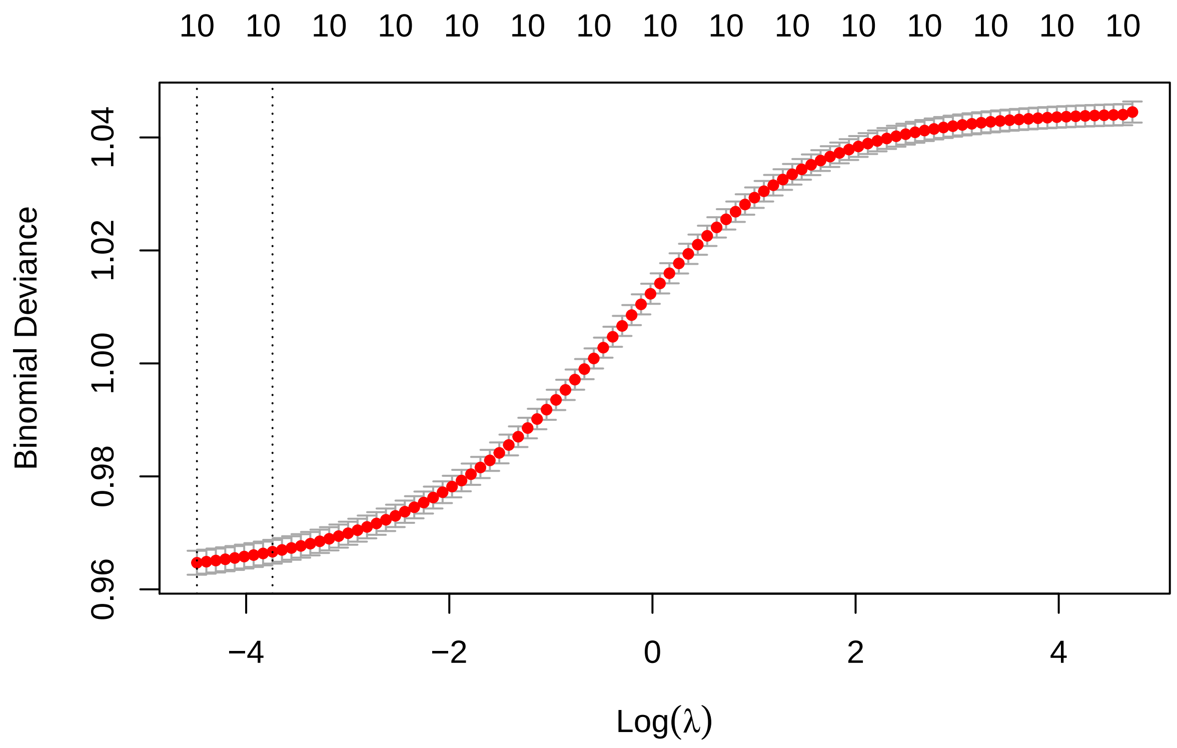
```r
#Lasso Model
mse.min2 <- min(cv.out2$cvm)
mse.min1 + bestlam2
```

```
## [1] 0.9649555
```

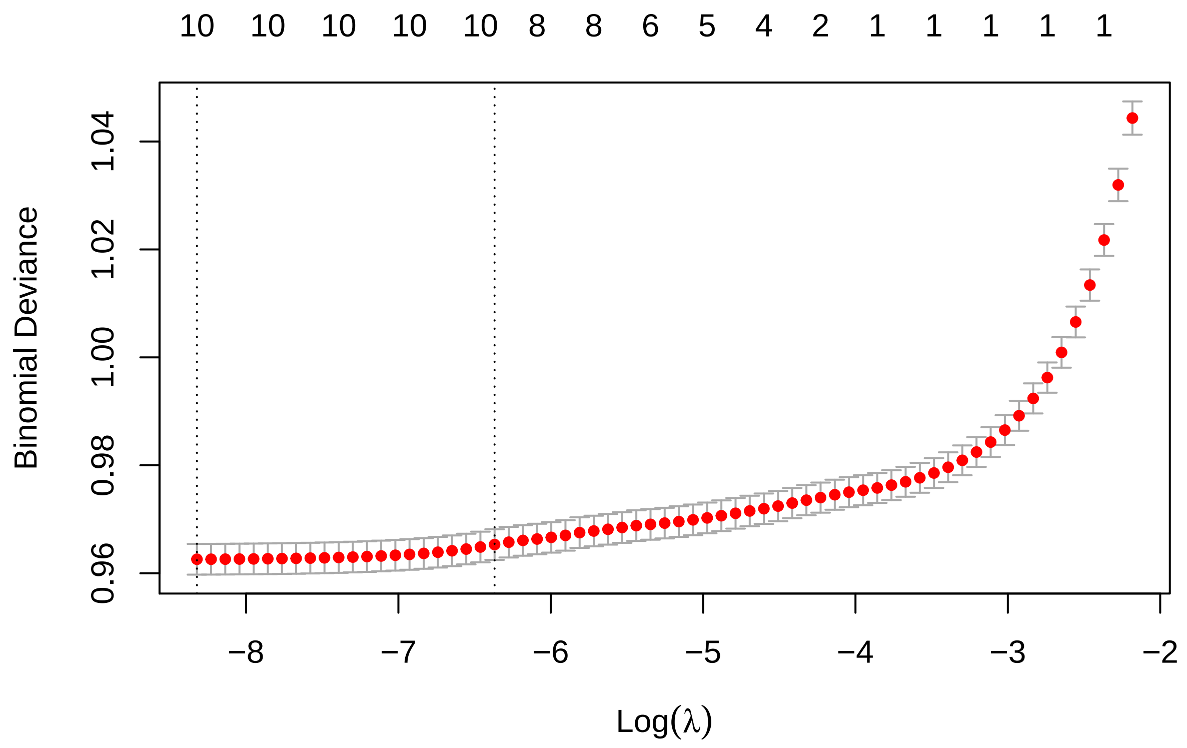
While lambda increased by a bit, at most, we can say lambda is close to 1 now, which wouldn't make much of a difference.

c) Plot of cross-validation error and the chosen lambda

```
#Ridge Model
plot(cv.out)
```



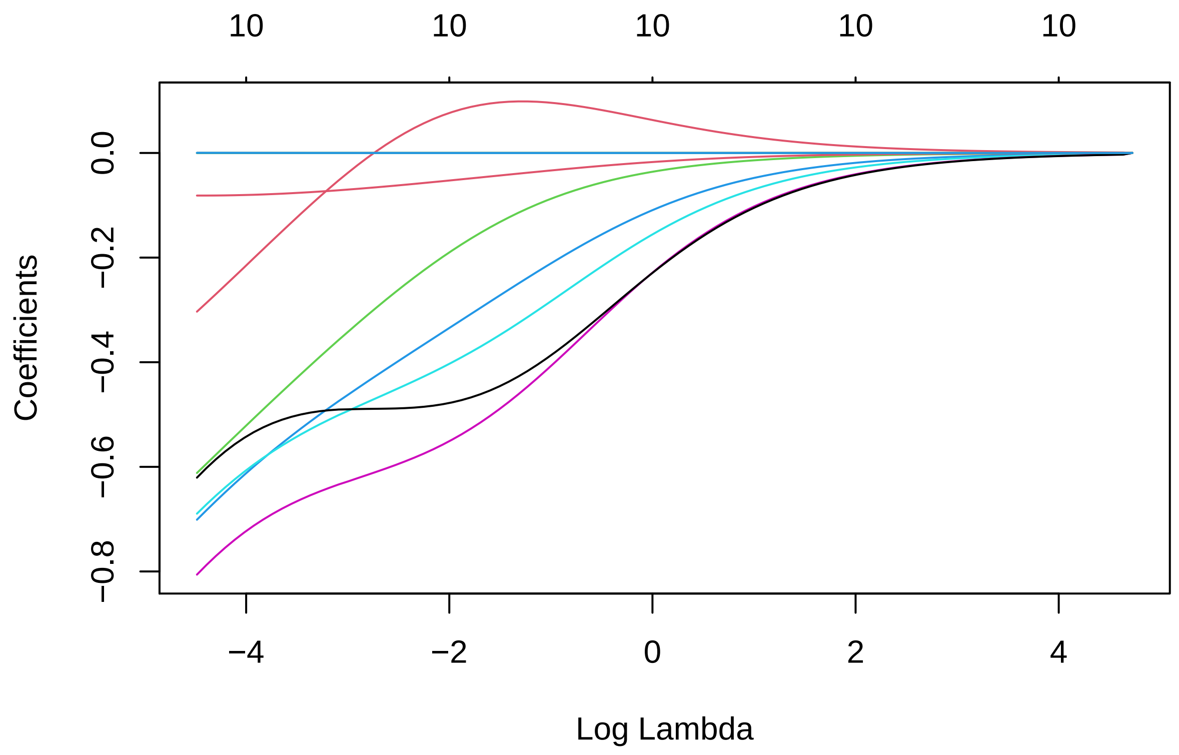
```
#Lasso Model
plot(cv.out2)
```



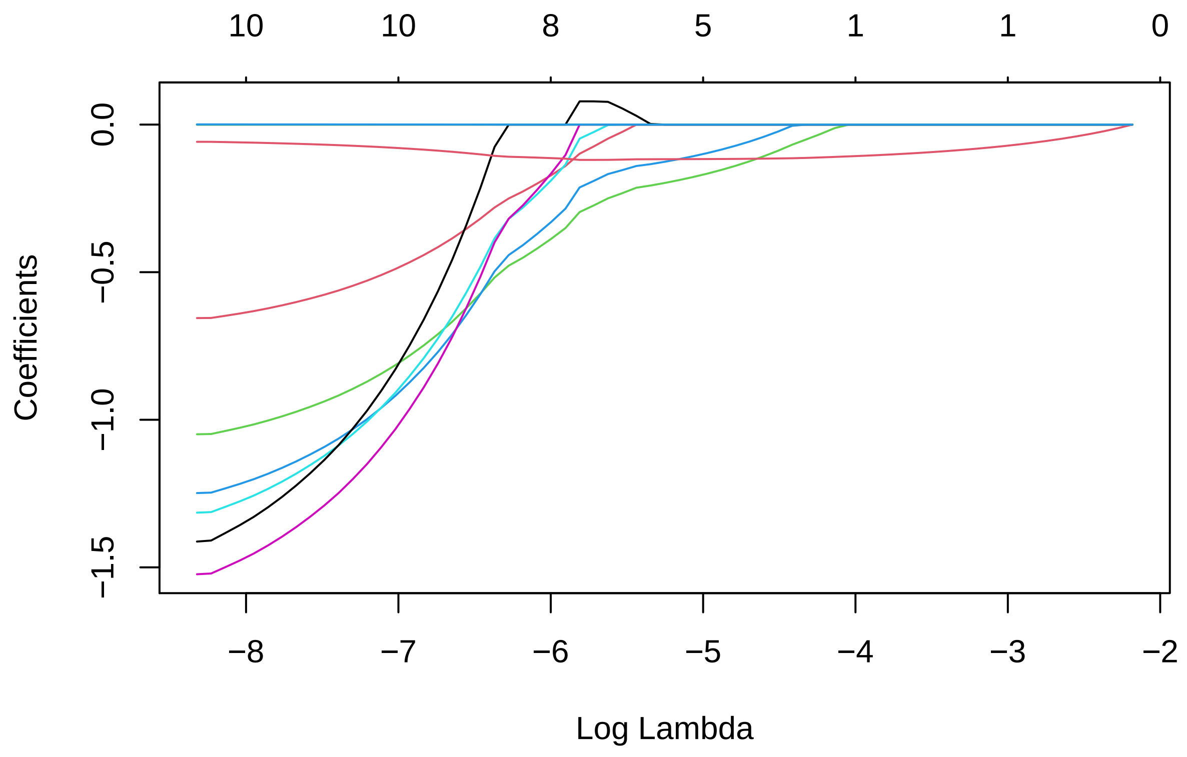
The cross validation errors for all the lambdas in general appears to be very small.

d) Plot of coefficient varaition vs. lambda

```
#Ridge Model
ridge_br = glmnet(x, y, alpha = 0, family = "binomial")
plot(ridge_br, xvar = "lambda")
```



```
#Lasso Model
ridge_bl = glmnet(x, y, alpha = 1, family = "binomial")
plot(ridge_bl, xvar = "lambda")
```



e) Coefficients of chosen lambda

```
predict(ridge_br,type = "coefficients", s=bestlam)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  2.908498e+00
## annual_inc   1.845643e-06
## gradeB      -3.033692e-01
## gradeC      -6.117536e-01
## gradeD      -7.011458e-01
## gradeE      -6.895396e-01
## gradeF      -8.060310e-01
## gradeG      -6.206645e-01
## int_rate    -8.139015e-02
## funded_amnt -1.908890e-05
## installment  2.548399e-04
```

```
predict(ridge_bl,type = "coefficients", s=bestlam2)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  2.994668e+00
## annual_inc   2.228675e-06
## gradeB      -6.555611e-01
## gradeC      -1.049161e+00
## gradeD      -1.248275e+00
## gradeE      -1.314740e+00
## gradeF      -1.523394e+00
## gradeG      -1.412620e+00
## int_rate    -5.829325e-02
## funded_amnt -3.429697e-05
## installment  7.185895e-04
```

f) Error Rate of the test subset

```
ridge_mod = glmnet(x_train3, y_train3, alpha = 0, family = "binomial")
ridge_pred = predict(ridge_mod, s = bestlam, newx = x_test3)
ridge_matrix = rep("Default or Charged Off",164046)
ridge_matrix[ridge_pred>.8] = "Fully Paid"
table(ridge_matrix, y_test3)
```

```
##                         y_test3
## ridge_matrix             Default or Charged Off Fully Paid
##   Default or Charged Off                  16276      23455
##   Fully Paid                              25643      98672
```

```
lasso_mod = glmnet(x_train3, y_train3, alpha = 1, family = "binomial")
lasso_pred = predict(lasso_mod, s = bestlam2, newx = x_test3)
lasso_matrix = rep("Default or Charged Off",164046)
lasso_matrix[lasso_pred>.8] = "Fully Paid"
table(lasso_matrix, y_test3)
```

```
##                         y_test3
## lasso_matrix             Default or Charged Off Fully Paid
##   Default or Charged Off                  17343      25628
##   Fully Paid                              24576      96499
```

```
#Ridge Model Error Rate
ridge_error = 1 - (98672+16276)/164046
ridge_error
```

```
## [1] 0.2992941
```

```
#Lasso Model Error Rate
lasso_error = 1 - (17343+96499)/164046
lasso_error
```

```
## [1] 0.3060361
```

g) Error rate comparison and logit model coeffcients

```
error_table3.1 = rbind(logit_error,ridge_error,lasso_error)
error_table3.1
```

```
##                  [,1]
## logit_error 0.4300702
## ridge_error 0.2992941
## lasso_error 0.3060361
```

```
coefficients(model2.5)
```

```
##   (Intercept)        gradeB        gradeC        gradeD        gradeE
##  2.989162e+00 -7.637781e-01 -1.197177e+00 -1.428278e+00 -1.580919e+00
##        gradeF        gradeG    annual_inc      int_rate   funded_amnt
## -1.770640e+00 -1.842729e+00  2.524025e-06 -4.858748e-02 -3.921068e-05
##   installment
##  8.691584e-04
```

Compared to my logit model, which had an error rate of 43%, my ridge and lasso models has decreased this rate by about 10%. With these techniques, it looks like optimally increasing bias and lowering variance in my models have helped them considerably.

3.3 - Running a Regression or Classification Tree

a) Tree Plot

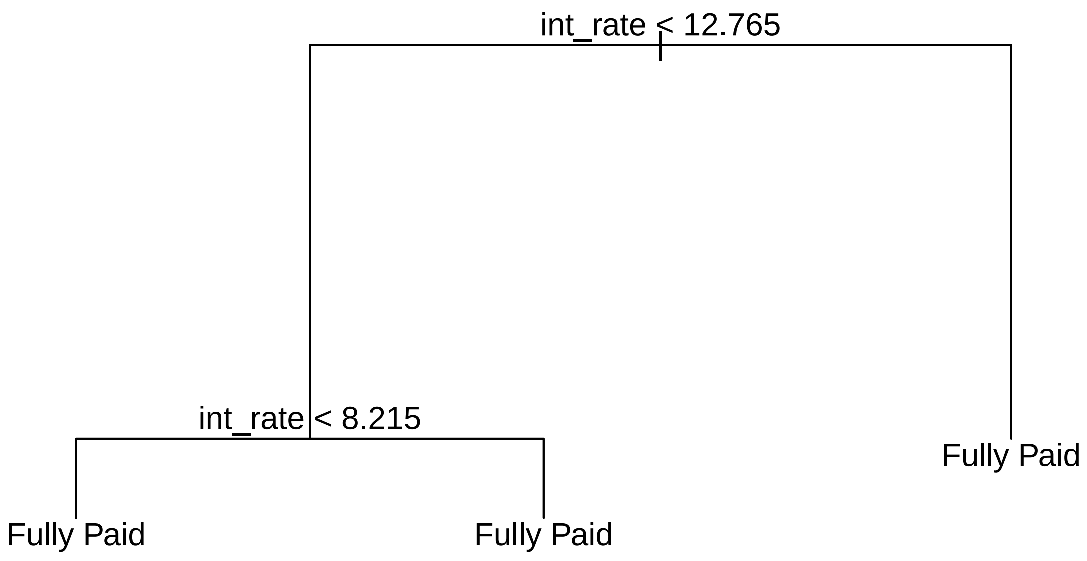
```
set.seed(3.5)
train3.5 = loan %>% sample_frac(0.4701)
test3.5 = loan %>% setdiff(train3.5)

tree_loan = tree(factor(loan_status)~grade+int_rate+funded_amnt+installment+annual_inc, data = train3.5
```

```
## Warning in tree(factor(loan_status) ~ grade + int_rate + funded_amnt +
## installment + : NAs introduced by coercion
```

```
summary(tree_loan)
```

```
##
## Classification tree:
## tree(formula = factor(loan_status) ~ grade + int_rate + funded_amnt +
##     installment + annual_inc, data = train3.5, control = tree.control(nobs = 186877,
##     mincut = 1, minsize = 2, mindev = 0.01))
## Variables actually used in tree construction:
## [1] "int_rate"
## Number of terminal nodes:  3
## Residual mean deviance:  0.9792 = 183000 / 186900
## Misclassification error rate: 0.2163 = 40425 / 186877
```

```
plot(tree_loan)
text(tree_loan, pretty = 0)
```



The resulting tree doesn't show many levels and isn't very interesting. Unfortunatly I don't believe pruning this would do any better.

b) Error rate of the testing set

```
head(predict(tree_loan, test3.5))
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
##   Default or Charged Off Fully Paid
## 1             0.32094003  0.6790600
## 2             0.06491967  0.9350803
## 3             0.16072163  0.8392784
## 4             0.32094003  0.6790600
## 5             0.06491967  0.9350803
## 6             0.06491967  0.9350803
```

```
head(predict(tree_loan, test3.5, type = "class"))
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
## [1] Fully Paid Fully Paid Fully Paid Fully Paid Fully Paid Fully Paid
## Levels: Default or Charged Off Fully Paid
```

```
tree_pred = predict(tree_loan, test3.5, type = "class")
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
table(tree_pred, test3.5$loan_status)
```

```
##
```

```
## tree_pred                Default or Charged Off Fully Paid
##   Default or Charged Off                      0          0
##   Fully Paid                              43427     143294
```

```
1 - (143294)/(186721)
```

```
## [1] 0.2325769
```

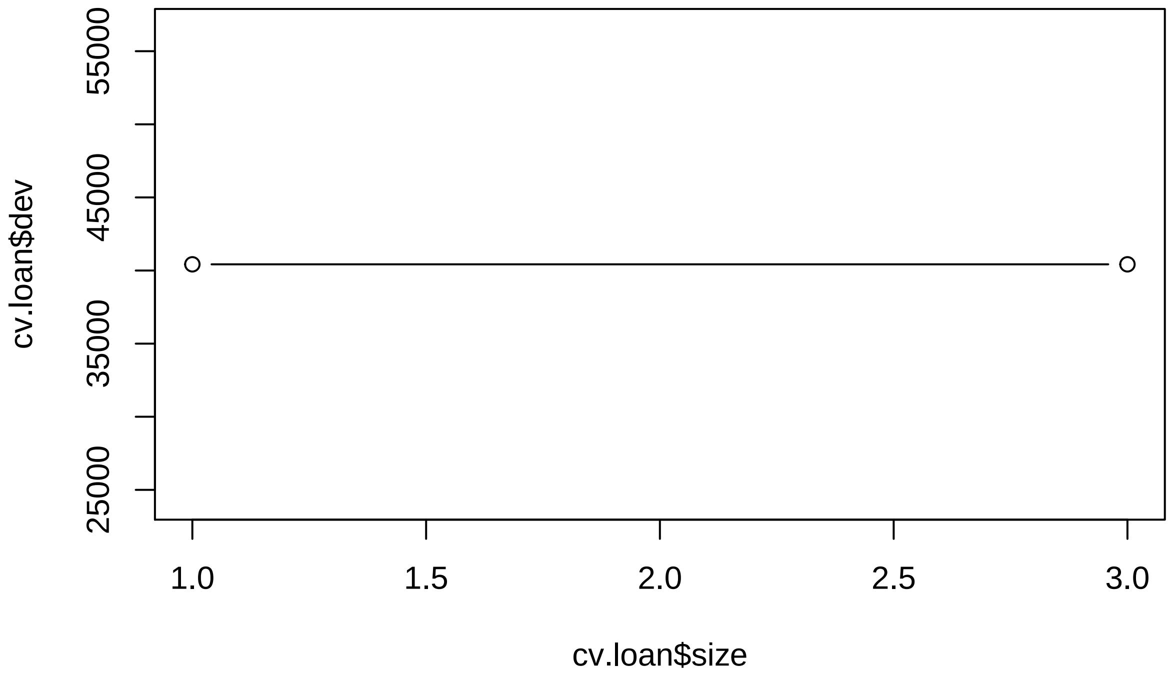
This approach leads to an error rate of about 23%. This number may be high because of the large number of people in my dataset who already fully paid their accounts, but I'm not sure how much I can trust this error rate because of how simplistic my tree is.

c) Tree pruning with cross validation

```
set.seed(5)
cv.loan = cv.tree(tree_loan, FUN = prune.misclass)
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```r
plot(cv.loan$size, cv.loan$dev, type = "b")
```



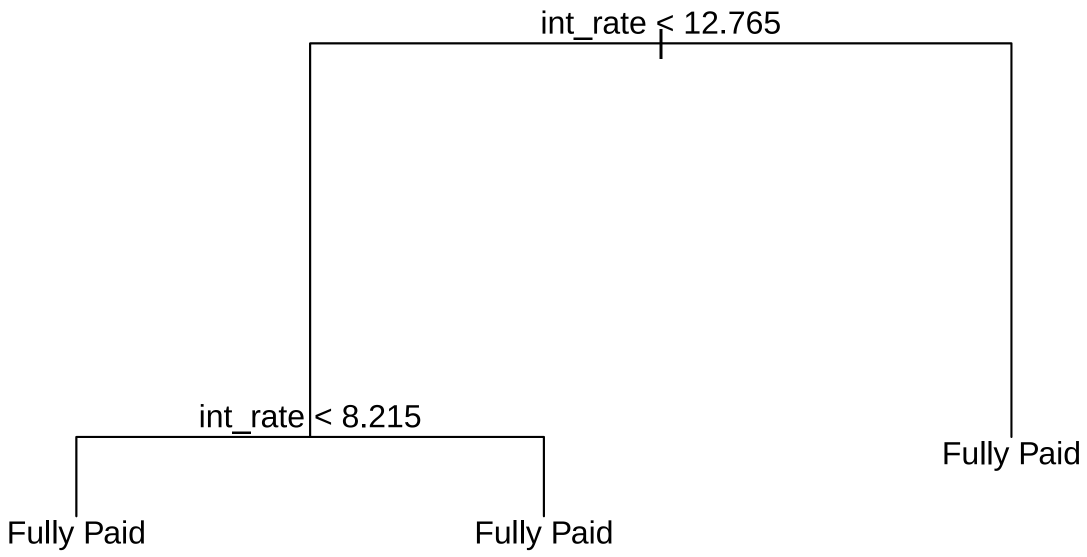
```r
cv.loan
```

```
## $size
## [1] 3 1
##
## $dev
## [1] 40425 40425
##
## $k
## [1] -Inf    0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

It seems that any number of terminal nodes (1 or 3) leads to a similar cross validation error rate, with 5504 errors (my data set is large, so this number is trivial). So for the time being, I will just use 3 terminal nodes.

d) Pruned Tree Model

```r
prune_loan = prune.misclass(tree_loan, best = 3)
plot(prune_loan)
text(prune_loan, pretty = 0)
```



e) Tree Interpretation of splits and selected variables

Pruning didn't change my tree at all. In my model, grade seems to be the most significant regressor that explains the behavior of loan status followed by interest rate. However, the final nodes of my tree doesn't make much sense, so I can't make a good interpretation of this tree.

f) Comparison of Error Rates

```
tree_pred = predict(prune_loan, test3.5, type = "class")
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
table(tree_pred, test3.5$loan_status)
```

```
##
## tree_pred                 Default or Charged Off Fully Paid
##   Default or Charged Off                       0          0
##   Fully Paid                               43427     143294
```

Because my tree didn't change, the error rate shouldn't have changed from the beginning. This model still has a low error rate, but again, I'm not inclined to believe it. I have reached out for help during office hours, and it seems like other students with this data set have also found that this model doesn't fit into a tree model very well. Thus, I will ignore this modeling techinque.

Fourth Report

4.1 - Creating a training and testing subset

```
#For some reason, my grade variable was interpreted as a character, so I had to change its type.
#Also, I randomly sampled a smaller dataset to use for splitting. When I tried using the full dataset o
set.seed(4)
loan_rf = sample_n(loan2,35000)
train4 = loan_rf %>% sample_frac(0.48)
train4$grade <- as.factor(train4$grade)
test4 = loan_rf %>% setdiff(train4)
test4$grade <- as.factor(test4$grade)
nrow(train4)
```

```
## [1] 16800
```

```
nrow(test4)
```

```
## [1] 16832
```

```
view(train4)
```

4.2 - Bagging Classification

Bagging classification is meant to improve on the normal tree models by fitting many tree models and and aggregating all of their values. Using this, we can reduce the variance of our prediction and prevent some overfitting.

a) Bagging Classification Model& Plot of Out-of-Bag Error rate vs. Number of Trees

```
library(MASS)
library(randomForest)
```

```
bag.loan = randomForest(factor(loan_status)~annual_inc+grade+int_rate+funded_amnt+installment,
                        data=train4,
                        mrty=ncol(train4)-1,
                        ntree=300,
                        importance=TRUE,
                        do.trace = 100)
```

```
## ntree      OOB      1      2
##   100:  24.02% 79.46%  8.68%
##   200:  24.49% 80.61%  8.96%
##   300:  24.08% 80.30%  8.52%
```

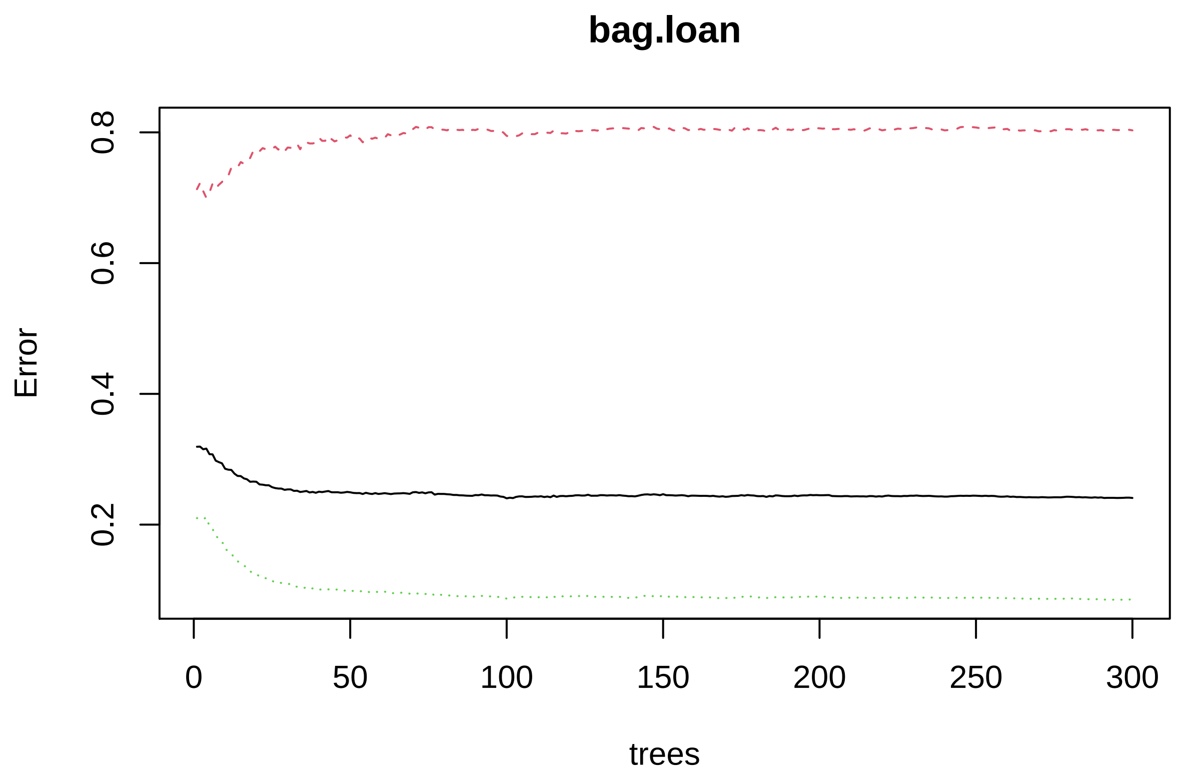
```
bag.loan
```

```
##
## Call:
##  randomForest(formula = factor(loan_status) ~ annual_inc + grade +      int_rate + funded_amnt + ins
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 24.08%
## Confusion matrix:
##                         Default or Charged Off Fully Paid class.error
## Default or Charged Off                     443       1806  0.80302357
## Fully Paid                                 692       7434  0.08515875
```

In this model, I use a bagging classification model in order to regress loan status on all my other variables. This model uses all predictor variables for each split node of the tree, but it improves on a regular tree model

by making many of them and making different samples for each one.
The out-of-bag error rate doesn't seem to decrease after many trees are used. It's very low which is good, and it seems that using 2 variables per split is ideal.

```
plot(bag.loan)
```



The plot shows that the out-of-bag (OOB) error rate actually stays pretty stteady, but it dips and plateus after the first 50 trees.

b) Error rate table of predicted vs. observed values

```
bag.pred=predict(bag.loan, newdata = test4, type = "response")
table(bag.pred, test4$loan_status)
```

```
##
## bag.pred                  Default or Charged Off Fully Paid
##   Default or Charged Off                     545        881
##   Fully Paid                                2656       9749
```

Here is my error rate table that charts the predicted values of my bagging model against the observed values from the testing data set. Because the bagging model already gives responses in factors, there's no need for me to convert or subset percentages like I did for my models in previous reports.

c) Comparison of Error Rates

```
bag_error = 1 - (658+11761)/16832
error_table4.2 = rbind(lasso_error,ridge_error,bag_error)
error_table4.2
```

```
##                   [,1]
## lasso_error 0.3060361
## ridge_error 0.2992941
## bag_error   0.2621792
```

The bagging regression only imporved the error rate of my model by very little.

d) Importance Matrix

```
randomForest::importance(bag.loan)
```

```
##             Default or Charged Off Fully Paid MeanDecreaseAccuracy
## annual_inc                3.160408   8.993497             10.25545
## grade                    10.262698   4.698242             13.98043
## int_rate                 28.955339  10.870031             21.36770
## funded_amnt             -20.406179  34.721964             36.03455
## installment             -23.134123  35.489747             35.71076
##             MeanDecreaseGini
## annual_inc          922.0586
## grade               168.3155
## int_rate            744.6915
## funded_amnt         561.0273
## installment         860.2215
```

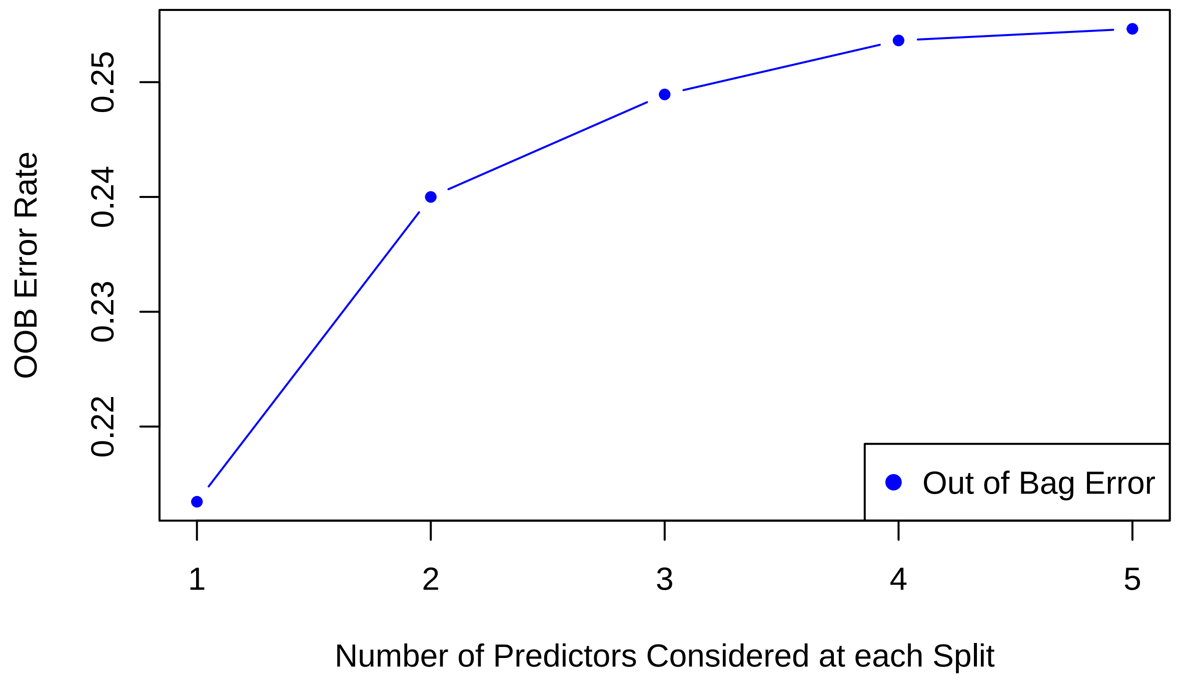
The importance matrix of a bagging model tells that annual income dreases the gini of my model the most, improving the accuracy of my model the most. If we leave it out, the Gini index and impurity will increase substantially.

4.3 - Random Forest Classification

a)Random Forest Model and OOB error rate vs Number of predictors in each split

```
oob.e <-double(5)
test.e <-double(5)

for(mtry in 1:5)
{
  rfa=randomForest(factor(loan_status)~annual_inc+grade+int_rate+funded_amnt+installment,
                   data=train4,
                   mtry=mtry,
                   ntree=300)

  oob.e[mtry] = rfa$err.rate[300] #OOB Error

  rfa.pred=predict(rfa, newdata = test4, type = "response")
  test.e[mtry] = 1 - mean(rfa.pred==test4$loan_status) # Testing Set Error
}
```

```
matplot(1:mtry , oob.e, pch=20 , col="blue",type="b",ylab="OOB Error Rate",xlab="Number of Predictors C

legend("bottomright",legend=c("Out of Bag Error"),pch=19, col=c("blue"))
```



Random forest classification is meant to improve on bagging by varying the number of variables used in each split, thus creating a variety of trees that doesn't overfit the data and making it less sensitive to changes in data.

Looking at the OOB Error plot of my random forest plot, the error rate seems to increase as more variables are used for each split, using one predictor leads to the smallest error rate. This might have happened because I have few predictors.

b) Tuned Random Forest model and OOB Errors

```
rf.loan = randomForest(factor(loan_status)~annual_inc+grade+int_rate+funded_amnt+installment,
                       data = train4,
                       mtry = 1,
                       importance = TRUE,
                       ntree = 300,
                       do.trace = 100)
```

```
## ntree      OOB      1      2
##   100:  21.55% 92.35%  2.02%
##   200:  21.36% 91.91%  1.91%
##   300:  21.37% 91.80%  1.94%
```

After tuning the my model so just one predictor is used per split, the OOB error rate has decreased per 100 trees when compared to the untuned model.

c) Error rate table of predicted vs. observed values

```
rf.pred=predict(rf.loan, newdata = test4, type = "response")
table(rf.pred, test4$loan_status)
```

```
##
## rf.pred                  Default or Charged Off Fully Paid
##   Default or Charged Off                    305        276
##   Fully Paid                               3627      12624
```

d) Comparison of Error Rates

```
rf_error = 1 - (315+12618)/16832
error_table4.3 = rbind(error_table4.2,rf_error)
error_table4.3
```

```
##                  [,1]
## lasso_error 0.3060361
## ridge_error 0.2992941
## bag_error   0.2621792
## rf_error    0.2316421
```

The error rate of my random forest model is 23%, and improves on all my models substantially. It has the lowest error rate by far.

e) Importance Matrix

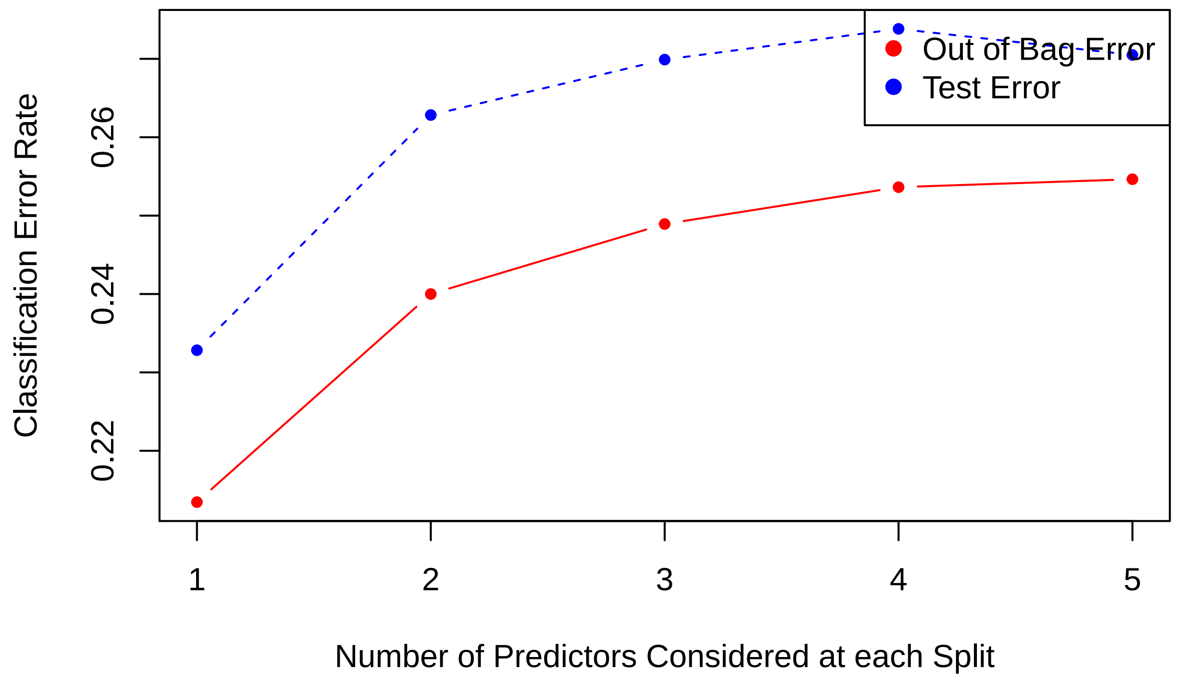
```
randomForest::importance(rf.loan)
```

```
##             Default or Charged Off Fully Paid MeanDecreaseAccuracy
## annual_inc               0.9061136   13.62573             15.12037
## grade                    0.7195733   16.51923             21.40353
## int_rate                 6.5349104   19.16298             26.03406
## funded_amnt            -17.9616871   24.05158             24.78907
## installment            -18.9512797   22.30761             22.06214
##             MeanDecreaseGini
## annual_inc          471.3101
## grade               272.3474
## int_rate            568.2297
## funded_amnt         368.9689
## installment         490.4617
```

Looking at the importance matrix, the annual income variable decreased the gini and impurity of the splits in the random forest model. It seems to be the most important.

f) OOB error rate & Error rate vs Number of predictors in each split

```
matplot(1:mtry , cbind(oob.e,test.e), pch=20,
col=c("red","blue"), type="b",
ylab="Classification Error Rate",
xlab="Number of Predictors Considered at each Split")
legend("topright", legend=c("Out of Bag Error","Test Error"),
pch=19, col=c("red","blue"))
```



The overall test error when running my testing data set on my random forest model follows a similar trend as the OOB error rate, which is reasonable.

4.4 - Boosting Classification

```r
#The boosting xgboost model asks for my y variable to be a column of 0's and 1's. Thus, I will make a n

loan4 = loan2

loan4$loan_status <- factor(loan4$loan_status)
loan4$loan_status <- as.character(loan4$loan_status)
loan4[loan4$loan_status == "Default or Charged Off",]$loan_status <- "0"
loan4[loan4$loan_status == "Fully Paid",]$loan_status <- "1"
loan4$loan_status <- as.numeric(loan4$loan_status)
```

```r
#Using this dataset, I will make a new training and testing dataset that includes the type change for m

set.seed(45)
train44 = loan4 %>% sample_frac(0.415)
train44$grade <- as.factor(train44$grade)
test44 = loan4 %>% setdiff(train44)
test44$grade <- as.factor(test44$grade)
```

a) Boosting classfication model & Error rate table of predicted vs. observed values

```r
library(gbm)
set.seed(44)
boost.loan = gbm(loan_status~annual_inc+grade+int_rate+funded_amnt+installment,
                 data = train44,
                 distribution = "bernoulli",
                 n.trees = 1000,
                 interaction.depth = 4,
                 shrinkage = 0.01,
                 verbose = F)
```

```
boost.pred=predict.gbm(object = boost.loan, newdata = test44, type = "response", n.trees = 1000)
boost.pred2=rep("Default or Very Late",164108)
boost.pred2[boost.pred>.8] = "Fully Paid"
table(boost.pred2,test44$loan_status)
```

```
##
## boost.pred2                0     1
##   Default or Very Late 30370 58242
##   Fully Paid           11557 63939
```

The boosting model is supposed to improve on the other tree-based models because it sequentially builds a model based on the prediction errors of the one right before it. However, looking at the error rate table, there seems to have been more errors made with this model.

b) Comparison of Error Rates

```
boost_error = 1 - (30370+63939)/164108
boost_error
```

```
## [1] 0.4253236
```

```
errortable4.4 = rbind(error_table4.3,boost_error)
errortable4.4
```

```
##                  [,1]
## lasso_error 0.3060361
## ridge_error 0.2992941
## bag_error   0.2621792
## rf_error    0.2316421
## boost_error 0.4253236
```

My suspicions are confirmed, the boosting model performs the worst out of all the models. It is only a little better than guessing. Because of this, this model may require a lot of tuning.

c) Importance Matrix

```
summary(boost.loan)
```

```
##                      Length Class  Mode
## initF                     1 -none- numeric
## fit                  164973 -none- numeric
## train.error            1000 -none- numeric
## valid.error            1000 -none- numeric
## oobag.improve          1000 -none- numeric
## trees                  1000 -none- list
## c.splits                510 -none- list
## bag.fraction              1 -none- numeric
## distribution              1 -none- list
## interaction.depth         1 -none- numeric
## n.minobsinnode            1 -none- numeric
## num.classes               1 -none- numeric
## n.trees                 1 -none- numeric
## nTrain                  1 -none- numeric
## train.fraction          1 -none- numeric
## response.name           1 -none- character
## shrinkage               1 -none- numeric
## var.levels              5 -none- list
## var.monotone            5 -none- numeric
## var.names               5 -none- character
## var.type                5 -none- numeric
## verbose                 1 -none- logical
## data                    6 -none- list
## Terms                   3 terms  call
## cv.folds                1 -none- numeric
## call                    8 -none- call
## m                       5 -none- call
```

Looking at my importance matrix, interest rate has the largest effect on building my boosting model, followed by grade.

4.5 - XGBoost Classification

a) XGBoost Classification Model & Error rate table of predicted vs. observed values

```
#Putting data into matrix form with new testing and training sets

y_train4 <- train44$loan_status
x_train4 <- model.matrix(loan_status~., train44)[,-1]

#Preparing for xgboost model
dtrain4 <- xgb.DMatrix(data = x_train4, label = y_train4)
dtest4 <- model.matrix(loan_status~.,test44)[,-1]
```

```
#Building xgboost model using xgboost package
set.seed(46)
xgb.loan = xgboost(data=dtrain4, max_depth=2, eta = 0.1, nrounds=40,
                   lambda=0,
                   print_every_n = 10, objective="binary:logistic")
```

```
## [1]  train-error:0.216326
## [11] train-error:0.216326
## [21] train-error:0.216326
## [31] train-error:0.216326
## [40] train-error:0.216326
```

```
xgb.pred=predict(xgb.loan, newdata = dtest4, type = "response")
xgb.pred2=rep("Default or Very Late",164108)
xgb.pred2[xgb.pred>.8] = "Fully Paid"
table(xgb.pred2,test44$loan_status)
```

```
##
## xgb.pred2                  0     1
##   Default or Very Late 32190 65530
##   Fully Paid            9737 56651
```

The XGboost model, while similar to bagging, uses a different technique to evaluate the how effective each split is in building the model.

The results of my table seem to be similar to my boosting model. Again, a lot more errors seemed to have been made with the predictions.

b) Comparison of Error Rates

```r
xgb_error = 1 - (18875+93654)/164108
xgb_error
```

```
## [1] 0.3142991
```

```r
errortable4.5 = rbind(errortable4.4,xgb_error)
errortable4.5
```

```
##                  [,1]
## lasso_error 0.3060361
## ridge_error 0.2992941
## bag_error   0.2621792
## rf_error    0.2316421
## boost_error 0.4253236
## xgb_error   0.3142991
```

Luckily, it seems that XGboost model has a similar error rate as my lasso and ridge models. It might benefit from some tuning.

c) Importance Matrix

```r
importance_xgb <- xgb.importance(colnames(x_train4),model=xgb.loan)
importance_xgb
```

```
##        Feature        Gain       Cover   Frequency
## 1:    int_rate 0.895951752 0.688981728 0.633333333
## 2: funded_amnt 0.032971430 0.087666119 0.125000000
## 3: installment 0.021977168 0.067222714 0.075000000
## 4:  annual_inc 0.018640598 0.089827614 0.091666667
## 5:      gradeC 0.010691584 0.007506380 0.008333333
## 6:      gradeB 0.008413214 0.012479762 0.016666667
## 7:      gradeF 0.008352818 0.032343117 0.033333333
## 8:      gradeD 0.001538779 0.003402421 0.008333333
## 9:      gradeG 0.001462657 0.010570146 0.008333333
```

Like in my boostinf model, interest rate plays the largest role in building my model and it seems to be the most important.

4.6 - (Extra Credit) Tuned Boosting Model

a) Tuned Boosting Model & Error rate table of predicted vs. observed values

```r
hyper_grid <- expand.grid(
shrinkage = c(0.005, 0.08, 0.01),
interaction.depth = c(1, 5, 10),
optimal_trees = 0,
min_error = 0)
```

```r
for(i in 1:nrow(hyper_grid)) {
set.seed(46)

# train model
gbm.tuned <- gbm(
formula = loan_status~annual_inc+grade+int_rate+funded_amnt+installment,
distribution = "bernoulli",
data = train44,
n.trees = 3000,

interaction.depth = hyper_grid$interaction.depth[i],
shrinkage = hyper_grid$shrinkage[i],

train.fraction = 0.75,

verbose = FALSE
)

# add min training error and trees to grid
hyper_grid$optimal_trees[i] <- which.min(gbm.tuned$valid.error)
hyper_grid$min_error[i] <- min(gbm.tuned$valid.error)
}

hyper_grid %>%
dplyr::arrange(min_error) %>%
head(10)
```

```
##   shrinkage interaction.depth optimal_trees min_error
## 1     0.010                 5          2983 0.9529234
## 2     0.005                10          2998 0.9530625
## 3     0.010                10          2159 0.9531281
## 4     0.080                 5           476 0.9532564
## 5     0.005                 5          2986 0.9535449
## 6     0.080                10           295 0.9538155
## 7     0.080                 1          2880 0.9550751
## 8     0.010                 1          2998 0.9559767
## 9     0.005                 1          2998 0.9569171
```

After using hyperparameter tuning, it seems like the parameters that leads to the smallest error rate is a learning rate of 0.01, a depth of 5, and 2983 trees. However, I'm not sure if my error rate was coded correctly.

```
library(gbm)
set.seed(46)
bboost.loan = gbm(loan_status~annual_inc+grade+int_rate+funded_amnt+installment,data =
                    train44, distribution = "bernoulli", n.trees = 2983, interaction.depth
                  = 5, shrinkage = 0.01, verbose = F)
```

```
bboost.pred=predict.gbm(object = bboost.loan, newdata = test44, type = "response", n.trees = 2983)
bboost.pred2=rep("Default or Very Late",164108)
bboost.pred2[bboost.pred>.8] = "Fully Paid"
table(bboost.pred2,test44$loan_status)
```

```
##
## bboost.pred2               0     1
##   Default or Very Late 30395 57777
##   Fully Paid           11532 64404
```

Compared to the other error rate tables I have seen, this one doesn't seem to do a good job. There are substaintially more errors made with the prediction.

b) Comparison of Error Rates

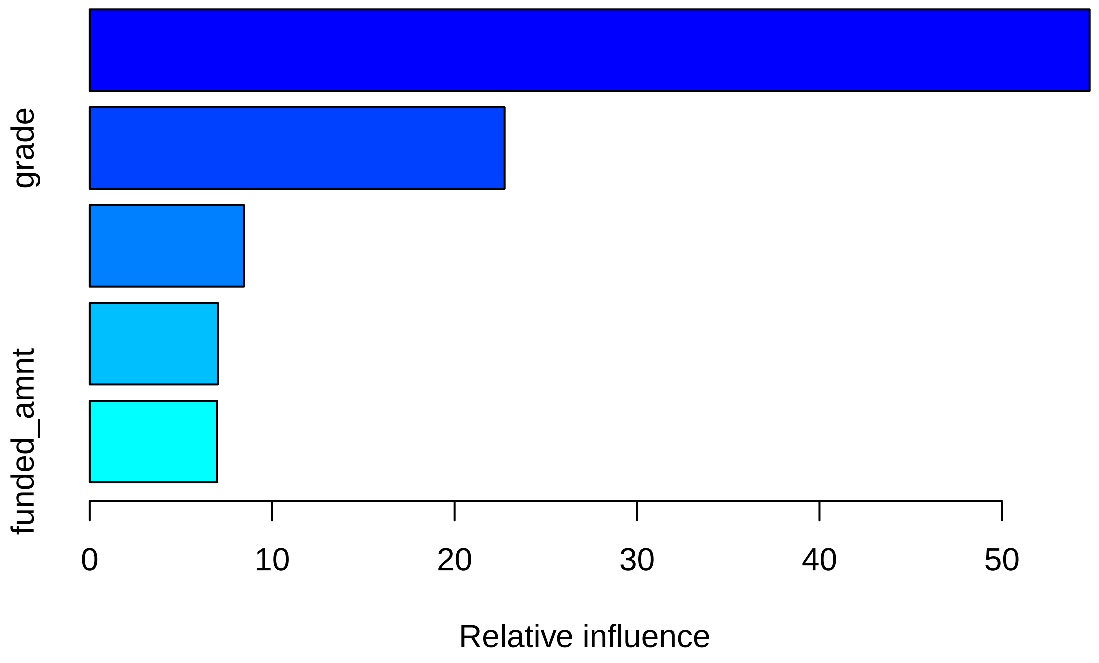
```
bb_error = 1-(64404+30395)/164108
error_table4.6 = rbind(errortable4.5, bb_error)
error_table4.6
```

```
##                   [,1]
## lasso_error 0.3060361
## ridge_error 0.2992941
## bag_error   0.2621792
## rf_error    0.2316421
## boost_error 0.4253236
## xgb_error   0.3142991
## bb_error    0.4223377
```

My tuned boosted model didn't imporve on the original one, so this leads me to believe that I didn't code something correctly in my original tuned model. I revisit this as I believe the model has a lot of potential, and show what work I have so far.

c) Importance Matrix

```
summary(bboost.loan)
```



```
##                     var   rel.inf
## int_rate       int_rate 54.805972
## grade             grade 22.741082
## installment installment  8.452124
## annual_inc   annual_inc  7.022577
## funded_amnt funded_amnt  6.978246
```

Despite note having the best error rate, the tuned boosted model still shows that interest rate seems to be the most important variable in building the model.

Conclusion

After running historgrams and boxplots of my predictor variables in my first report as well as regressing them on loan status, I found that annual income, interest rate, and grade produced models with very low p-values for their corresponding t-tests, meaning they had high explanatory power. It's a shame that employment length was not one of these; the data set had several levels for yeas, and some of them were significant and others were not. For the sake of convenience, I decided to leave it out.

In my second report, I began building more complex logistic regression models by including more than one regressor in each one. I gradually built five models, adding a variable each time and checking the significance of each as I did. My resulting model ended up having loan status being regressed on annual income, interest rate, grade, installment, and funded amount. The last two I added in order to make my model more interesting, and their t-test pvalues showed that they were just as significant as the other variables. However, the error rate of this original logit model was 43%, which is a little better than guessing, but still not ideal. The probit model only increased this error rate.

I was really satisfied to find how much the ridge and lasso model imporved my model error rate in my third report. Both techniques attempt at reducing varaince by penalizing for extra variables that I add in my model. After running them, I found that the error rate went down to 30%, which was a substantial decrease. The ideal lambda that I found for these models were surprisingly small, but it still had a large effect. Unfortunatly, the tree model didn't give aa meanigful result possibly because of how my data was interpreted. Perhaps the many levels of grade were confusing for the model to work with.

Finally, in my fourth model, I used modeling techniques that built off of tree models. The best model from this report, and for my entire project so far, is a random forest model. After creating a for loop to find the optimal number of variables to use in each split (it was one variable), I used that value to model a new random forest model. This constrasts with my bagging model which uses the full number of regressors per split. I was able to get an error rate of 23%, which is very good. My boosting model ended up needing a lot of tuning that I attempted at, but I will revisit again for my final submission.

My report has a decent amount of subsetting and cleaning because of how very few observations in the data set actually defaulted at first. In the future, I would like to find some way to explore the other levels of loan status, such as late or charged off. For now and for the sake of convenience, I made loan status to be a simple, binary variable.