Assessing Loan Performance in Peer-to-Peer (P2P) Lending with Machine Learning Modeling

Jia Lin Mei | Econ 490 - Final Report

Updated on July 8, 2020

Index

A. Introduction

Peer-to-peer (P2P) lending is a service that directly connects a borrower to a lender typically through an online platform, charging a fee for those who use it. While these transactions are not backed by a bank or other financial institutions, they have numerous advantages in our increasingly digital world and economy. LendingClub, one of the largest P2P lending companies in the U.S, is a clear sign of this service's growing popularity. In 2014, LendingClub issued 3.5 billion dollars in loans, which is more than double the amount of what they loaned the year before [3]. While this is a small fraction of the 3.3 trillion US consumer debt (report by the US Federal Reserve System), it is still a substantial amount that warrants special attention.

A crucial part of this service is screening borrowers and only allowing those who have the highest chances of paying back their loans be part of the company's clientele. Traditionally, P2P companies will look into the borrower's level of income, housing situation, or other factors in order to make a proper judgement. Additionally, a letter grade is assigned to the loan based on the borrower's FICO score and other third party financial information [3]. Loans that are at a higher risk of defaulting are assigned a lower grade and higher interest rates to help offset potential losses. The goal of this paper is to asses if such factors and characteristics are true determinants of loan default. What are the qualities of a loan that investors and lenders should look out for? To answer this question, I analyzed LendingClub's quarterly loan data from 2007-2015, which contained about 890 thousand observations or loans. Using machine learning methods I learned in Econ 490, I created models that established a relationship between loan status (current, late, or fully paid) and a selection of determinants from the 75 variables in the dataset. Machine learning is applicable to this situation because training and testing sets can be made from the dataset and be used to create and improve models. This strategy is central to machine learning.

To begin, I picked four determinants or x-variables that I believed could explain the loan status, my y-variable. They were income, grade of the loan, employment length, and interest rate. While FICO and credit scores was not included in this data set, I can presume it was used to calculate the loan grade. Originally, loan status in the data set had several possible levels such as “Grace Period” and “Late (16-30 Days).” However, many of the models I used in R limited me to only binary y-variables (variables with only two values), so in this case, I only cared about loans that were defaulted or fully paid. I had to eliminate or reassign the extra levels and the loans with those statuses. Upon finding that very few people defaulted in the data set, I decided to include loans that were “Charged Off” with the defaulted status since those transactions are assumed to never be fully paid off. The code below reflects this change. Finally, I removed any NA values in the data set and variables that I was not interested in analyzing.

```
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",]
```

```
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)
```

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

```
##
##    1 year 10+ years   2 years   3 years   4 years   5 years   6 years   7 years
##     61997    143354     38813     34538     25401     26017     17892     14616
##   8 years   9 years
##     18871     16027
```

After running my models, I found that grade, interest rate, level of income, installments paid, and funded amount were important in explaining the status of the loan; grade and interest rate were especially significant. However, because this paper is more a survey of different modeling techniques, more in-depth work and analysis can still be done.

B. Literature Review

As stated before, P2P lending’s growth in popularity is due to its benefits over traditional lending. The biggest benefit by far is bypassing intermediation costs that banks and financial institutions charge to oversee the transaction between the lender and borrower [3]. By eliminating this, interest rates may decrease for the borrower and the lender may receive higher revenues. Other benefits include avoiding capital and liability requirements set by banks. While the majority of P2P loans aren’t large, profit can still be gained because of the high turnover rate that comes with a large volume of small loans. In addition, by using an online platform, LendingClub automates and expedites this process substantially.

Additionally, the advent of P2P lending can help circulate and broaden the reach of credit to those that financial institutions may be hesitant to issue loans to. One typical example of a weakly valued assest is a community development loan. Standardized, predictable, and conventional assests are valued the most in the credit market since they can be easily evaluated, slowly pooled together, and be sold to different investors to spread risk [2]. Loans to community development projects often have none of these qualties, so they are usually lacking in capital and credit to start up. Online P2P platforms can fill this niche by being an efficient combination of finance and social networking that directly connects interested investors to communities. P2P lending can also ease the problem of credit rationing during times of economic difficulty, when not enough loans are made even when borrowers are willing to face large interest rates. Its versatility is why platforms such as LendingClub deserve attention.

However, there are risks to this service. Without the protection of an established financial institution, the risk of a borrower defaulting is passed onto the investor. This is further exacerbated by information asymmetry [1]. Simply put, no one knows the borrower better than the borrower knows him or herself. Lenders may invest in loans that seem good on paper, but the borrower may withhold crucial information, creating ill-matched loans. In financial institutions, these problems are usually controlled by regular monitoring, portfolio management, and credit agencies. These additional services are hard to conduct when they're not in-person, as in most P2P platforms and LendingClub's case. That's why a crucial part of P2P Lending is screening borrowers and only allowing those who have the highest chances of paying back their loans be part of the platforms's clientele. This problem serves as the basis of this paper; P2P lending holds promise, but adequate regulation is needed to bring about its potential.

Numerous studies on this topic have been made using data from P2P lending platforms such as LendingClub, however not many have used more advanced machine learning methods. In one done by Riza Emekter, only binary logistic regressions were used along with applications from survival analysis. The study found that credit score, FICO score, and debt-to-income ratio played a significant role in explaining loan default [1]. In a similar study done by the University of Zaragoza, a clear relationship was established between grade and default. Using t-tests and binary logistic regressions, 94.4% of A-grade loans were fully paid compared to 61.8% of G-grade loans [3]. Other information can better this relationship, but loan amount and length of employment were not found to make any statistical differences in the results. This study also interestingly stated that using advanced analysis methods such as the ones in this paper made little difference in model accuracy. While this claim was true in some cases, I saw in my paper that it doesn't hold for all.

C. Summary Statistics

Below is a summary table of the cleaned data set I used, loan. Roughly 27.5% of the loans in the dataset defaulted. This was used as a threshold throughout this project in order to classify loans as being fully paid and defaulted or charged off. Using the cofactors function, I saw that R treats 1 as Fully Paid, and 0 as Default or Charged Off.

```
summary(loan)
```

```
##                  loan_status      annual_inc      grade
##  Default or Charged Off: 85903   Min.   :      0   A: 70585
##  Fully Paid            :311623   1st Qu.:  50000   B:119907
##                                  Median :  68314   C:115001
##                                  Mean   :  80743   D: 55820
##                                  3rd Qu.:  95000   E: 25198
##                                  Max.   :9550000   F:  8701
##                                                    G:  2314
##      emp_length        int_rate      funded_amnt     installment
##  10+ years:143354   Min.   : 5.31   Min.   : 1000   Min.   :  14.77
##  1 year   : 61997   1st Qu.: 9.17   1st Qu.: 8000   1st Qu.: 248.45
##  2 years  : 38813   Median :11.99   Median :12300   Median : 379.32
##  3 years  : 34538   Mean   :12.91   Mean   :14755   Mean   : 449.28
##  5 years  : 26017   3rd Qu.:15.59   3rd Qu.:20000   3rd Qu.: 602.30
##  4 years  : 25401   Max.   :30.99   Max.   :40000   Max.   :1587.23
##  (Other)  : 67406
```

```
85903/311623
```

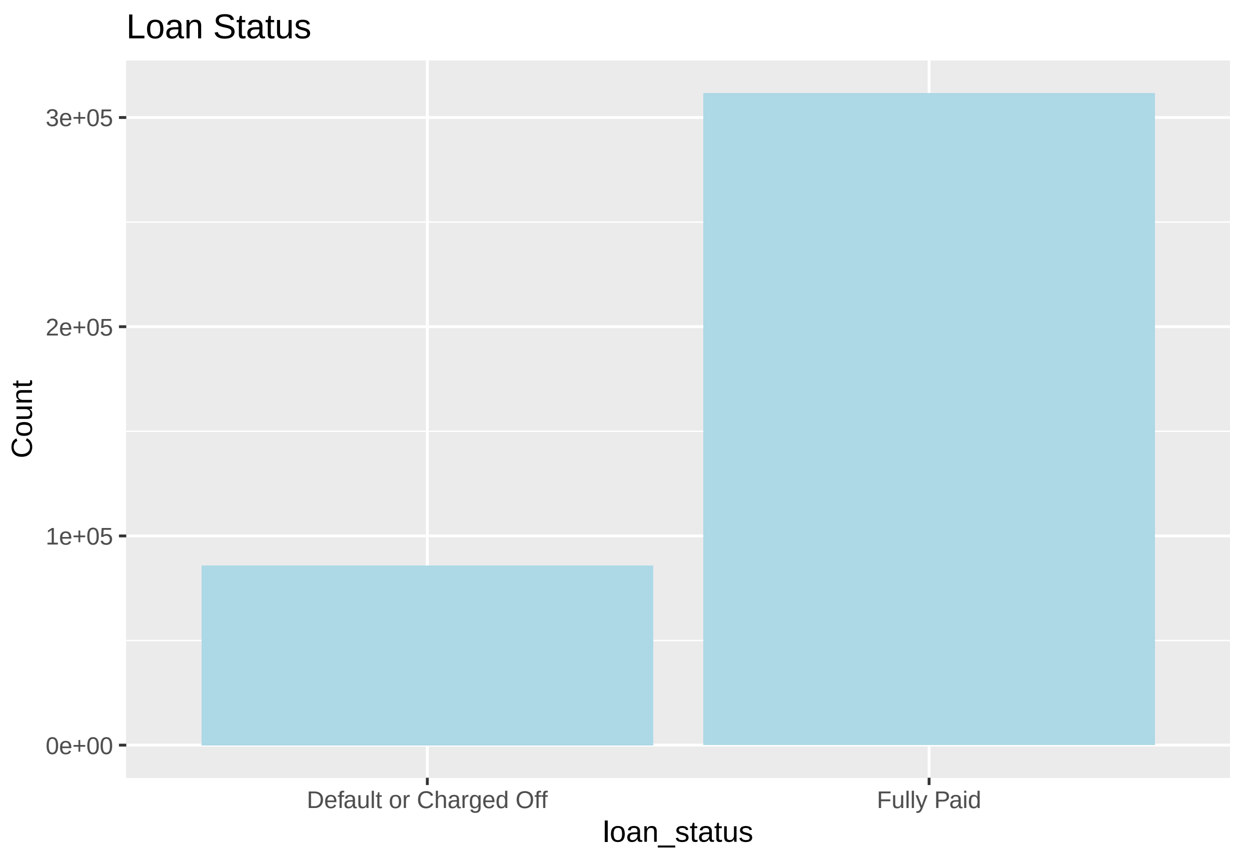
```
## [1] 0.2756632
```

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

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

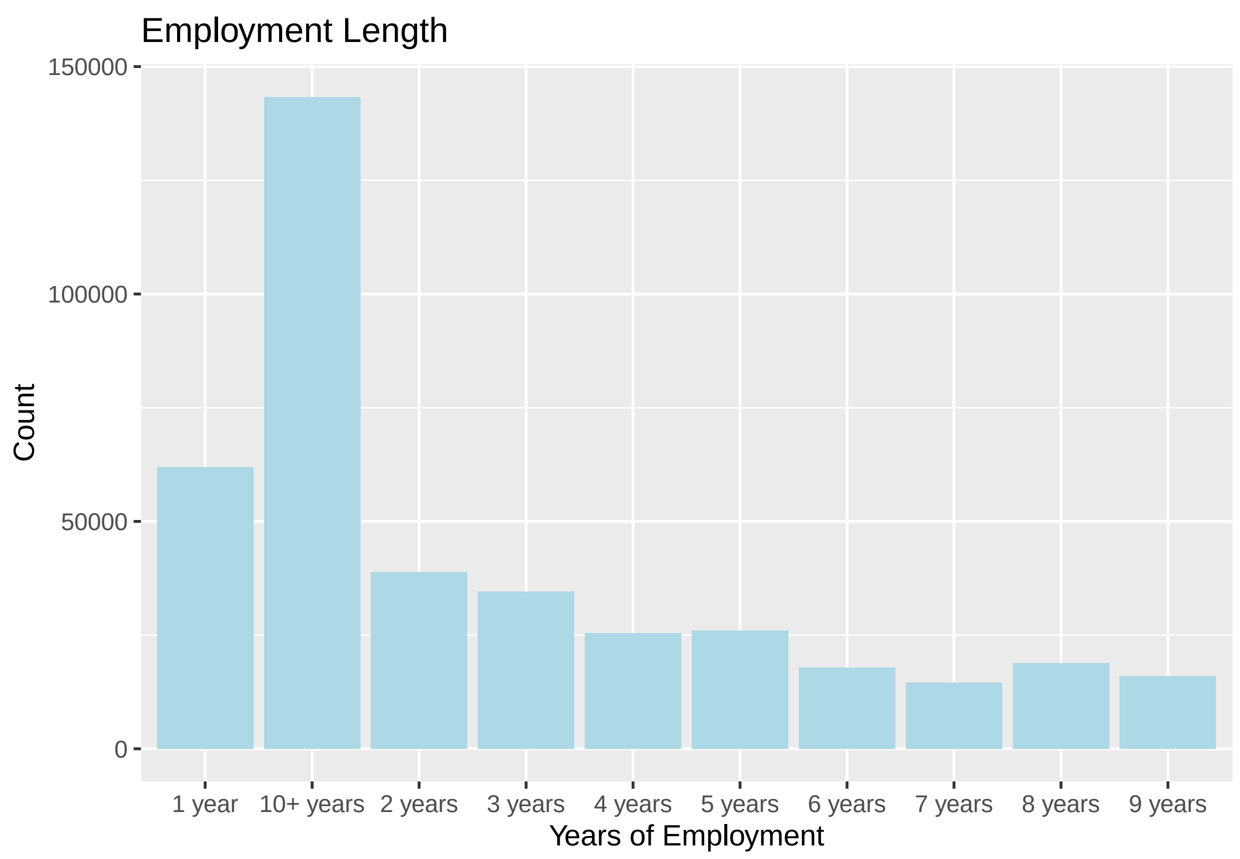
Below is a histogram of my y-variable, loan status.

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



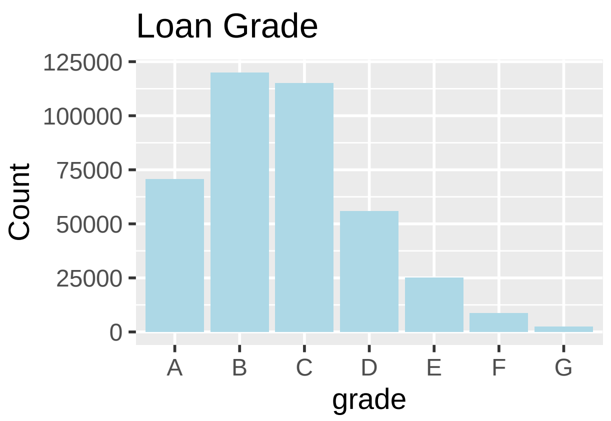
As the table shows, a greater fraction of borrowers paid off their loan than defaulted. Next, I looked at graphs of my x-variables to see if I could identify any casual relationships.

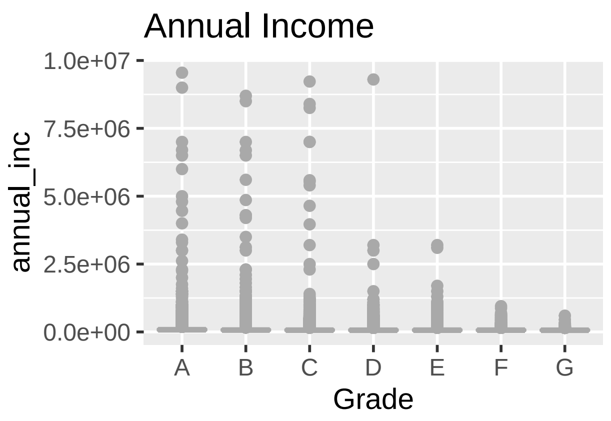
```r
p1.1grade <- ggplot(data =loan, mapping = aes(x=grade)) +
  geom_bar(fill = "lightblue") +
  labs(title = "Loan Grade", y = "Count")

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
```

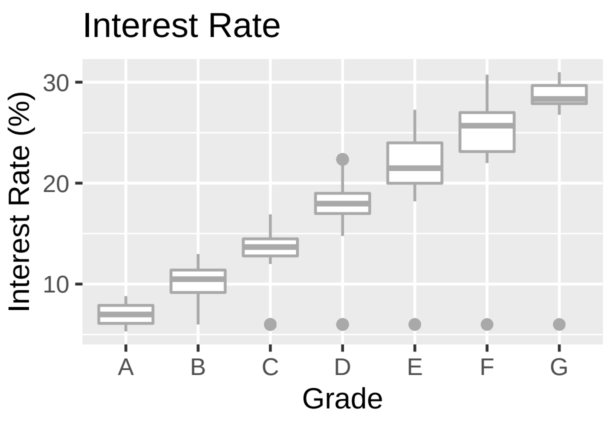


```
p1.4ir <- ggplot(data = loan, aes(x=grade, y=int_rate)) +
  geom_boxplot(color = "darkgray") +
  labs(title = "Interest Rate", x = "Grade", y = "Interest Rate (%)")
```

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







Loan grade represents how volatile or high-risk the loan issued was, with A being the least and G being the most volatile. For this variable, I turned it into a factor class so R could know how to analyze it. Most of the loans were of a higher grade. Next, I made box plots for annual income, employment length, and interest rate. 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%.

D. Logistic/Probit Classifications

Part 1 - One-Variable Logistic Regressions

In this section, I made basic regression plots of Y on each of my four X's. I wanted to see if each X could explain some of the behavior in Y. Because linear regression is not an appropriate way to fit a model on a categorical variable such as loan status, I will be using logistic regressions instead. For each regression, z-statistics are made for each regressor and its factors. I initially assumed that they had 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 were very low, which signified that grade had explanatory power. It seemed as grade went from A to G, the chances of the loan being fully paid decreased (risk of defaulting increases). This made sense because the loans became riskier.

```
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, which signified that it had explanatory power. It seemed that as income increased, the chances of paying off the loan also increased (risk of defaulting decreases). This made sense because the borrower's ability to pay off the loan would have increased with a larger income.

```
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 decreased as employment length increased. This was interesting; I assumed that with work experience, your ability to pay off your loan increased. It didn't seem possible to remove the non-significant levels without removing the whole variable.

```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 was very low, which signified that it had explanatory power. This made sense since monthly payments increased with higher interest rates, making them harder to pay off and possibly increasing the chances of default.

Part 2 - Multiple Variable Logistic Regression

Next, I made more a complex logit regression using more than one x-variable in my model. From the previous models, I saw that the coefficient for loan grade had a very low p-value and was shown to have explanatory power. This made sense, as grade is assigned based on crucial financial information from the borrower such as credit history. The better their history is, the more likely they'll be able to pay off their loan. Using this as a foundation, I built my new logistic regression model by adding more x-variables and seeing if they made a sigificant difference. I paid special attention to interest rate and annual income as those also had low p-values. Employment length seemed to be insignificant for some of its levels, so I was prepared to eliminate it. While it may be true that people who have worked longer wil be better at paying off their loans, the data didn't distinguish years worked after 10 years, which may have had an impact on the explanatory power of this variable. This process of adding variables and testing the model's accuracy is called forward-stepwise selection.

```r
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 after adding it to my model was still significant and had a positive effect on the chances of a borrower paying off their loan.

```
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 seemed to be insignificant, the rest still had some explanatory power. They didn't seem to be as significant as my other regressors, so I made the decision to leave out the variable.

```
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 exhibited a lot of explanatory power on loan status. After separating the effect of the interest rate, grade now had a positive effect on loan repayment. All the variables still had low p-values and were significant.

To make up for not including employment length in my model, I decided to add two more variables into the dataset that I was interested in analyzing: 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.

```r
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
```

Annual income appeared to still have a significant role in explaining defaults.

```
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
```

Part 3 - Picking my Best Model

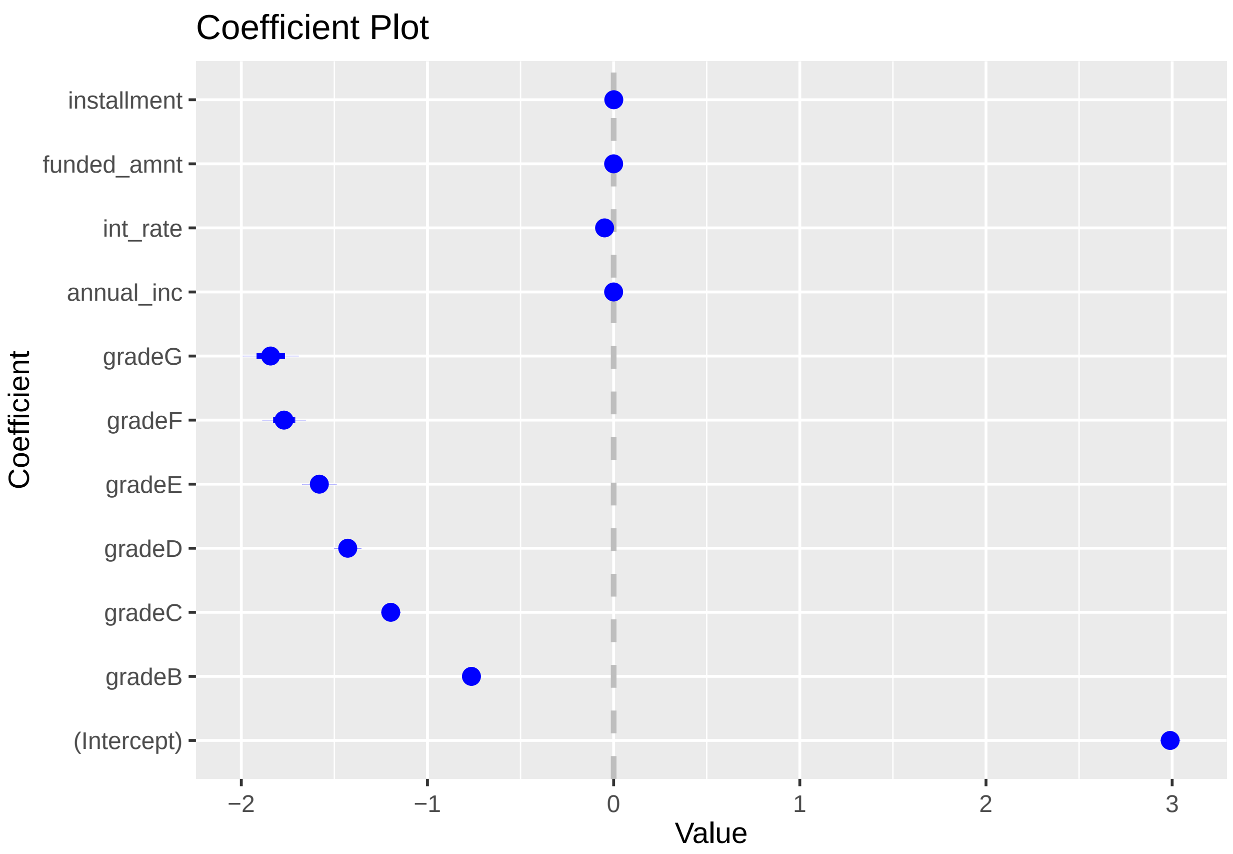
```
summary(model2.5)$coef
```

```
##                  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 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 had very low p-values, I can 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 went up, it had 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. The same goes for installments; the more that was paid off each month, the more likely the loan will get paid off. The reverse is true for interest rates and funded amounts. Higher interest rates naturally made loans harder to pay off.

```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 were close to zero, meaning they didn't have much explantory power. There was reason to believe they may be eliminated in other models that scrutinize coefficient values close to zero. However, the standard errors appeared to be small, which was a good sign.

Part 4 - Confusion Matrix and True Positive & False Positive Rates

As mentioned before, a crucial strategy of machine learning is to set aside a training data set to create models and assess them using a testing set. This is a step that I used frequently throughout this paper to evaluate each machine learning model. To begin, I first splitted the dataset into halves, and remade my best multiple logistic regression model using data from my training set.

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

```
set.seed(2)
glmtrain2.1=glm(loan_status~grade+int_rate+funded_amnt+installment+annual_inc,data=train2,family=binomia
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
```

Next, I ran my trained model using the testing data set instead. Here, I created a function to see how well my trained model could accurately predict when the loans in my testing data defaulted or not. I used the .275 or .725 threshold that I found earlier in my paper; if the loan has a probability larger than .725, the model predicted that it will be fully paid and vice verca.

```
set.seed(2)
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
```

This code chunk shows that the probability of the first 25 loans being fully paid were actually quite high. This matched up with how many fully paid observations there were in the original dataset.

```
set.seed(2)
glmpredict2.3=rep("Default or Charged Off",186630)
glmpredict2.3[glmtest2.2>.725] = "Fully Paid"
table(glmpredict2.3,test2$loan_status)
```

```
##
## glmpredict2.3            Default or Charged Off Fully Paid
##   Default or Charged Off                  20629      34102
##   Fully Paid                              22717     109182
```

```
#True Positive Rate
(109182)/(22717+109182)
```

```
## [1] 0.8277697
```

```
#False Positive Rate
(20629)/(20629+34102)
```

```
## [1] 0.3769162
```

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

```
## [1] 0.3044473
```

The TP rate was relatively high, and the FP rate was rather low. This is normally a good sign as we want a higher percentage of true postives (the model predicted fully paid correctly) than false positives. My logit model also appeared to be fairly accurate at making predictions, only make mistakes 30% of the time. This set a very high bar for the rest of my models.

Part 5 - Probit Classification

Probit classification is a variation of the logit classification that uses a different probability distribution to create its models. The difference in effects may be small, but it is generally worth seeing if there is any.

```
set.seed(2)
glmtrain2.4=glm(loan_status~grade+int_rate+funded_amnt+installment+annual_inc,data=train2,family = binor
```

```
## 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>.725] = "Fully Paid"
table(glmpredict2.6,test2$loan_status)
```

```
##
## glmpredict2.6          Default or Charged Off Fully Paid
##   Default or Very Late                      20560      34310
##   Fully Paid                                22786     108974
```

```
#True Positive Rate
(108974)/(108974+22786)
```

```
## [1] 0.8270644
```

```
#False Positive Rate
(20560)/(20560+34310)
```

```
## [1] 0.3747038
```

```
1 - (20560+108974)/186630
```

```
## [1] 0.3059315
```

Compared to the rates from our logistic model, the TP and TN rates appeared to be the same, meaning the model held the same level of explantory power. However, the error rate showed that the probit model wasn't very accurate, so the logistic model was the better performer of the two.

E. Ridge

In these next two sections, I used two shrinkage methods called ridge and lasso to improve on my original logistic model. The purpose of shrinkage methods is to reduce the amount of variance that is present in our model. High amounts of variance are present in a model when it fits just its training data set very well, but poorly fits testing sets and other data. I used many variables to train my logistic model, so I was concerned this may have been the case. The ridge method corrects this by using a tuning parameter called lambda to lower the influence of any insignificant variables in my model. This best value for lambda is found using cross-validation; R tests various lambdas and the best one is the one with the lowest cross-validation error. The larger lambda is, the more tuning it is needed. The smaller lambda is, the more the new model will resemble the original model.

Like in the last section, I began by splitting my data set into training and testing sets in order to estimate the test errors of my ridge regressions and lasso functions.

```
set.seed(3)
loan2 = loan[,-4]
loan2$grade <- as.factor(loan2$grade)
train3 = loan2 %>% sample_frac(0.415)
test3 = loan2 %>% setdiff(train3)

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

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
```

Below, I used my training data in order to create a ridge regression model. I also found the best value for lambda to reduce the amount of variance present.

```
set.seed(3)
cv.out = cv.glmnet(x_train3, y_train3, alpha = 0,family = "binomial")
```

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

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

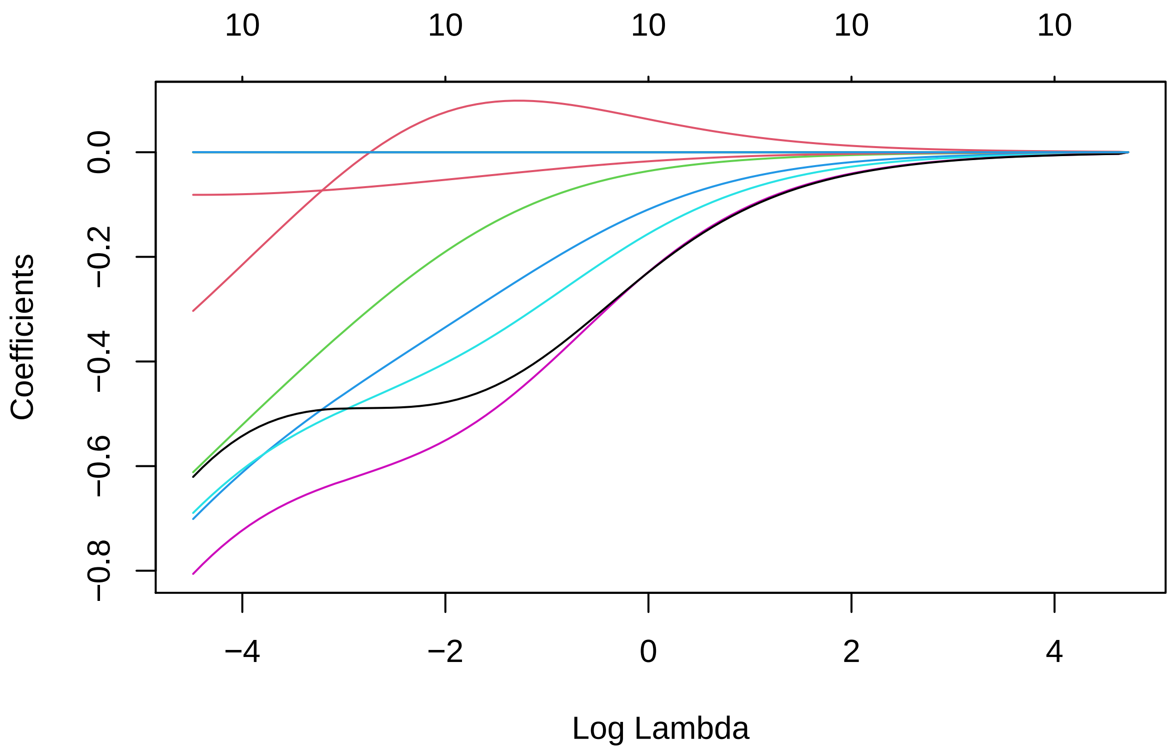
For my ridge model, the lambda that minimized the cross validation error was very small. That meant the original logistic model may not need a lot of tuning i.e. the variance of the original model was already at an optimal level. Next, I extracted the standard error and than added it to the minimum lambda to see the range of values it could be.

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

```
## [1] 0.9759704
```

While adding standard error to lambda increased it (close to 1 now), the difference made is small.

```
#Ridge Model
ridge_train = glmnet(x_train3, y_train3, alpha = 0, family = "binomial")
plot(ridge_train, xvar = "lambda")
```



As expected and to illustrate the concept of ridge, the value of my variable coefficients goes to zero as the value of my tuning parameter lambda increases. However, it is important to note that they cannot actually become zero. It is interesting that the best value of lambda was very small. In these next code chunks, I tested my model on the training set to see how well it can predict the results of my testing set.

```
set.seed(3)
ridge_pred = predict(ridge_train, s = bestlam, newx = x_test3)
ridge_matrix = rep("Default or Charged Off",164046)
ridge_matrix[ridge_pred>.725] = "Fully Paid"
table(ridge_matrix, y_test3)
```

```
##                          y_test3
## ridge_matrix              Default or Charged Off Fully Paid
##   Default or Charged Off                   14456      19901
##   Fully Paid                               27463     102226
```

```
#Ridge Model Error Rate
ridge_error = 1 - (14456+102226)/164046
ridge_error
```

```
## [1] 0.2887239
```

As expected, the ridge model was only slightly more accurate than my logit model, meaning not much tuning was done.

F. Lasso

Lasso, an extension of ridge, is more suitable for situations where only a few out of the many possible predictors are known to have any explanatory power. Unlike ridge, the coefficient values of any unnessecary variables can be zero, eliminating its effects. This technique is worth exploring to make my model more simpler and easier to interpret. A tuning parameter, lambda, is also used to better the model. In this section, I used the same training and testing data sets from my ridge model to create my lasso model.

```
set.seed(3)
cv.out2 = cv.glmnet(x_train3, y_train3, alpha = 1,family = "binomial")

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

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

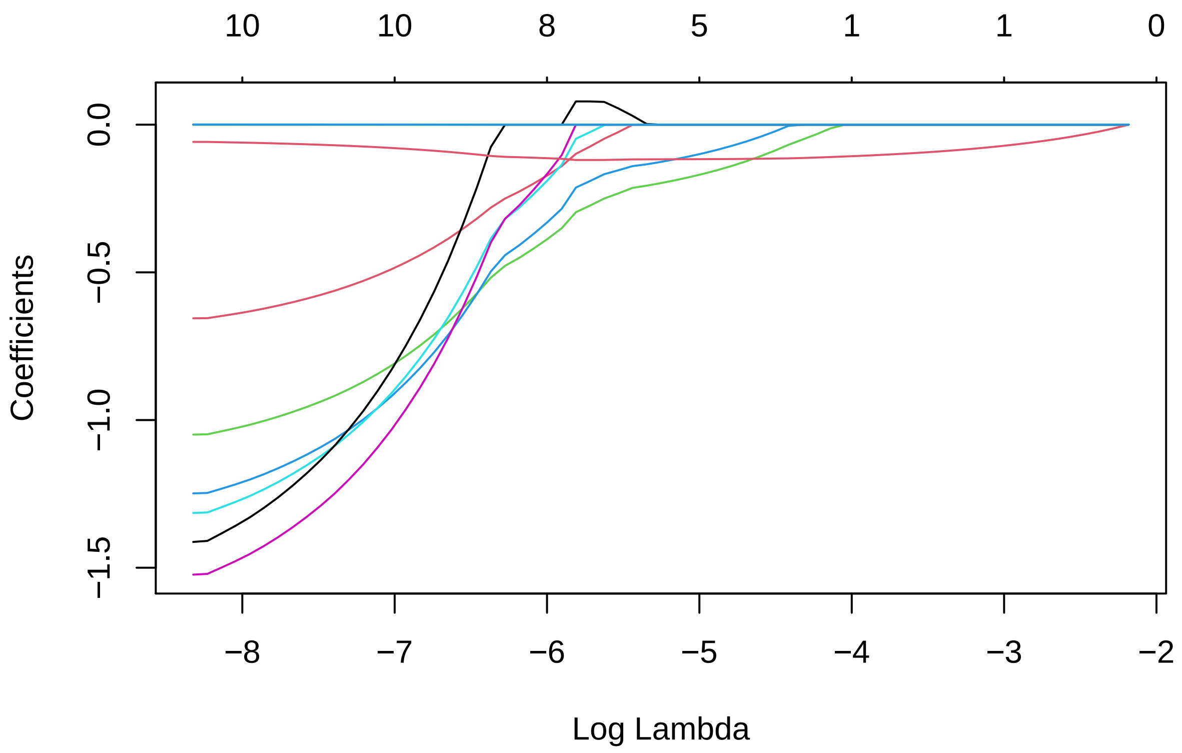
According to the lasso model, the tuning parameter should also be really small. Thus, I didn't believe lasso would help in fine-tuning my original model either since the results would be similar to the original logit model.

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

```
## [1] 0.9628033
```

While adding standard error tolambda increased it (close to 1 now), the difference made is small.

```
lasso_train = glmnet(x_train3, y_train3, alpha = 1, family = "binomial")
plot(lasso_train, xvar = "lambda")
```



As expected and to illustrate the concept of lasso, the value of my variable coefficients became zero as the value of my tuning parameter, lambda, increased. In these next code chunks, I tested my model on the training set to see how well it can predict fully paid loans. Because the value of the best lambda was very small, I expected the lasso model to be no more accurate in predicting defaulted or fully charged loans than the ridge or original logistic model.

```
set.seed(3)
lasso_pred = predict(lasso_train, s = bestlam2, newx = x_test3)
lasso_matrix = rep("Default or Charged Off",164046)
lasso_matrix[lasso_pred>.725] = "Fully Paid"

table(lasso_matrix, y_test3)
```

```
##                         y_test3
## lasso_matrix             Default or Charged Off Fully Paid
##   Default or Charged Off                  15503      21561
##   Fully Paid                              26416     100566
```

```
#Lasso Model Error Rate
lasso_error = 1 - (15503+100566)/164046
lasso_error
```

```
## [1] 0.2924607
```

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

```
##                   [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
```

Compared to my logit model, which had an error rate of 30%, my ridge and lasso models decreased the error rate only a little. This was to be expected, the values of the coefficients were extremely small, and the optimal lambda was very low, signalling that the new models would be very similar to the original logit model.

G. Decision Tree

In contrast to regressions, decision trees are meant to provide more of a visual aid. Starting with a central question (root node), a decision is tree is built by adding "branches" and nodes to show how the question progresses. For example, I can start a tree by asking what's a ice cream shop's favorite flavor, and then separate responses by seeing what season it is. This will serve is the first level, and I can further grow a tree by using more variables and questions to separate the responses. In this way, trees produce a model that's more intuitive to interpret. For this paper, my root node asks which loans were defaulted or charged off. Next, R may use the borrower's income to separate the loans and see if there's a certain threshold in income where borrowers are more likely to fully pay them off, and so on. The order in which variables are chosen to build the tree is based on which ones help to gain the most new information; R automatically does this using measures such as the Gini Index.

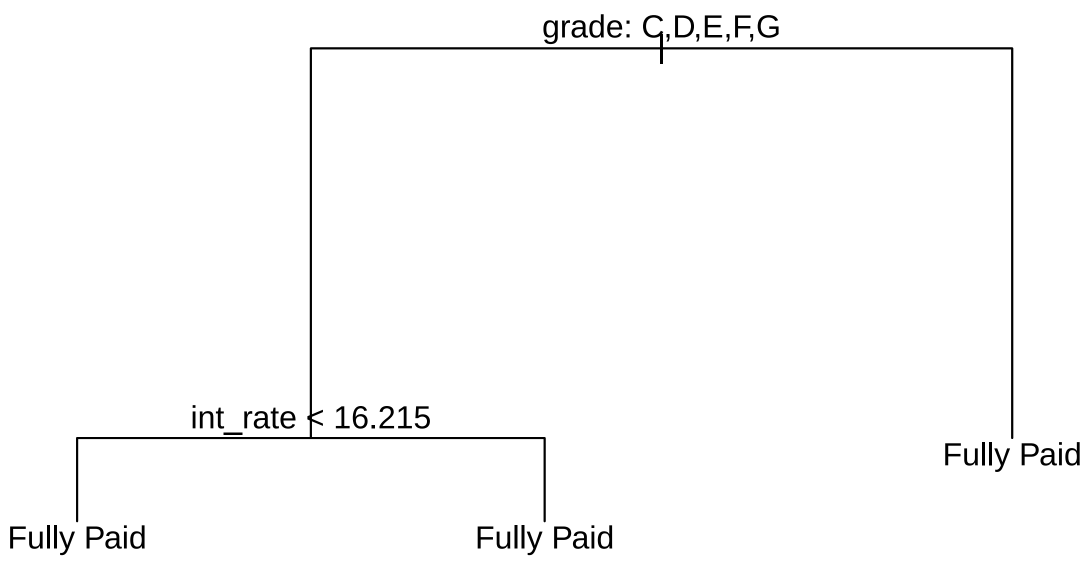
Part 1 - Unpruned Tree

To begin modeling, I first started by subsetting and sampling the appropriate training and testing sets from the original dataset. The resulting tree may be in need of pruning or optimization of the number of levels, nodes, etc.

```r
set.seed(3.5)
train3.5 = loan2 %>% sample_frac(0.415)
test3.5 = loan2 %>% setdiff(train3.5)

tree_loan = tree(factor(loan_status)~.,
                 data = train3.5,
                 control = tree.control(nobs = 186877, mincut=1,minsize=2,mindev=0.01))
summary(tree_loan)
```

```
##
## Classification tree:
## tree(formula = factor(loan_status) ~ ., data = train3.5, control = tree.control(nobs = 186877,
##     mincut = 1, minsize = 2, mindev = 0.01))
## Variables actually used in tree construction:
## [1] "grade"    "int_rate"
## Number of terminal nodes:  3
## Residual mean deviance:  0.9783 = 161400 / 165000
## Misclassification error rate: 0.2164 = 35692 / 164973
```

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



The resulting tree didn't show many levels and wasn't very interesting; only few of my variables were used. Moreover, all separation and branching led to the loans just being fully paid off, which doesn't make much sense.

```r
set.seed(3.5)
tree_pred = predict(tree_loan, test3.5, type = "class")
table(tree_pred, test3.5$loan_status)
```

```
##
## tree_pred                Default or Charged Off Fully Paid
##   Default or Charged Off                      0          0
##   Fully Paid                              41919     122127
```

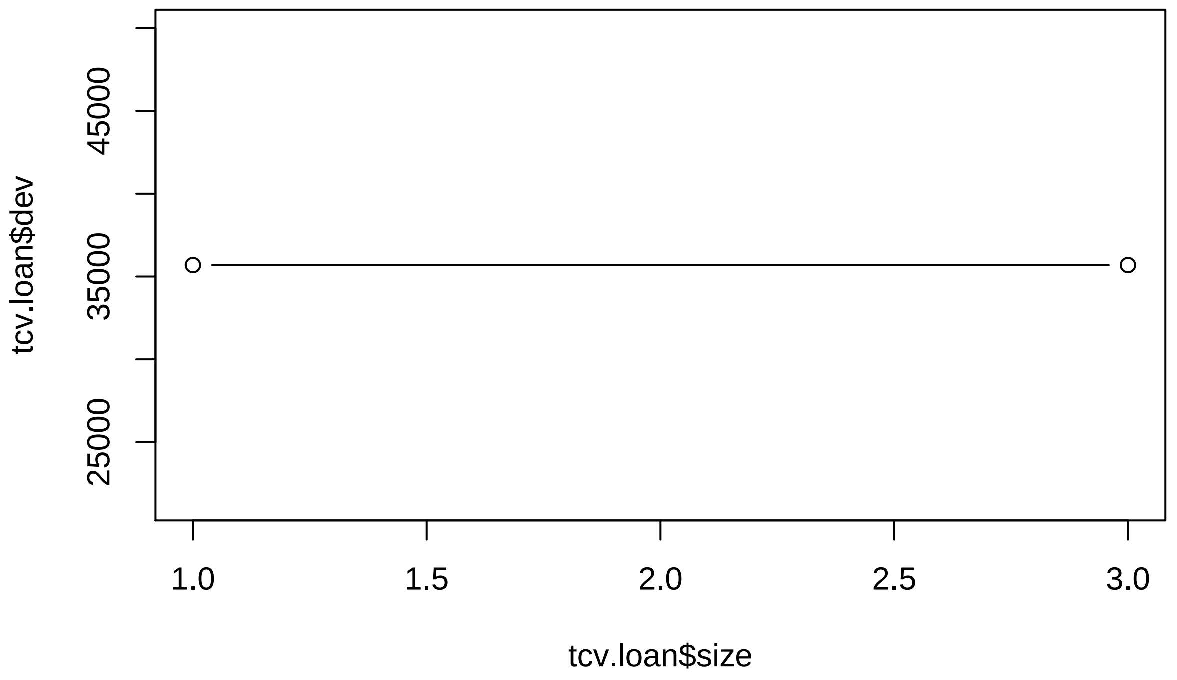
```r
tree_error = 1 - (122127)/(164046)
tree_error
```

```
## [1] 0.255532
```

This approach led to an error rate of about 25%, which is lower than my previous models. This number may have been low because of the large number of people in my data set who already fully paid their accounts, but I was skeptical of this error rate because of how unsual my tree was. I next attempted pruning it by using cross-validation.

Part 2 - Pruned Tree with cross-validation

```r
set.seed(3.5)
tcv.loan = cv.tree(tree_loan, FUN = prune.misclass)
plot(tcv.loan$size, tcv.loan$dev, type = "b")
```

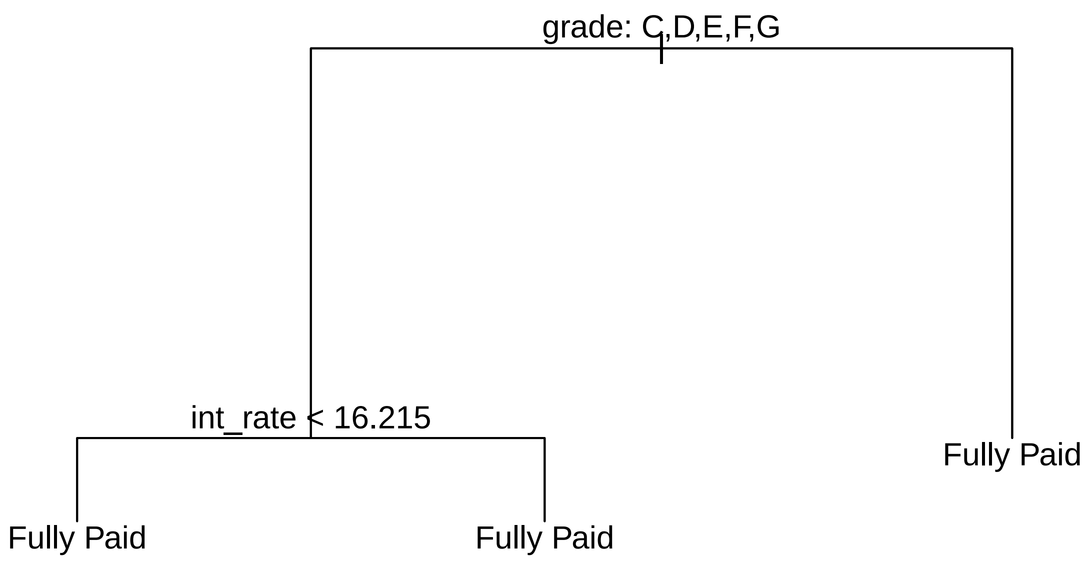


```
tcv.loan
```

```
## $size
## [1] 3 1
##
## $dev
## [1] 35692 35692
##
## $k
## [1] -Inf    0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

It seemed that any number of terminal nodes (1 or 3) led to a similar cross validation error rate, with 5504 errors (my data set is large, so this number is trivial). So to prune my tree, I used 3 terminal nodes.

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



Pruning didn't change my tree at all. In my model, grade was the most significant regressor that explained the behavior of loan status followed by interest rate. However, the final nodes of my tree still made little sense, so I couldn't interpret or use these results.

```
set.seed(3.5)
tree_pred2 = predict(tprune_loan, test3.5, type = "class")
table(tree_pred2, test3.5$loan_status)
```

```
##
## tree_pred2                Default or Charged Off Fully Paid
##   Default or Charged Off                       0          0
##   Fully Paid                               41919     122127
```

Because my tree didn't change, the error rate didn't change either. The decision tree still had a low error rate, but again, I'm not inclined to believe it. I reached out for help during office hours, and it seemed that other students with this data set had also found similar results. Thus, I decided to ignore this modeling techinque.

H. Bagging Model

Bagging classification is meant to improve on the normal tree model by fitting many tree models instead of just one. All the results are aggregated into one tree. Using this, we can reduce the variance of our model. In general, simple decision trees are susceptible to small changes in data that can vastly alter the shape of the tree, so averaging over many trees helps to control this. Another advantage of this method is that there's no need to subset a testing set; bagging automatically sets aside 1/3 of the data to test on the model trained on the other 2/3. These observations set aside are called out-of-bag (OOB) observations, and we can tell how well the trained model predicted the loan status using the OOB error rate. However, a testing set was used anyways for his method for more robust results.

To begin this last half of my paper, I randomly sampled a smaller data set to use for splitting. When I tried using the full dataset on my random forest models, R couldn't knit the chunks into a document because they were too big. So for the remainder of this paper, a smaller data set was randomly sampled and used instead.

```
set.seed(4)
loan_rf = sample_n(loan2,35000)
train4 = loan_rf %>% sample_frac(0.48)
test4 = loan_rf %>% setdiff(train4)
```

```
set.seed(4)
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,
                        nodesize = 3)
```

```
## ntree      OOB      1      2
##   100:  23.99% 80.62%  8.38%
##   200:  23.93% 81.19%  8.14%
##   300:  23.88% 81.61%  7.96%
```

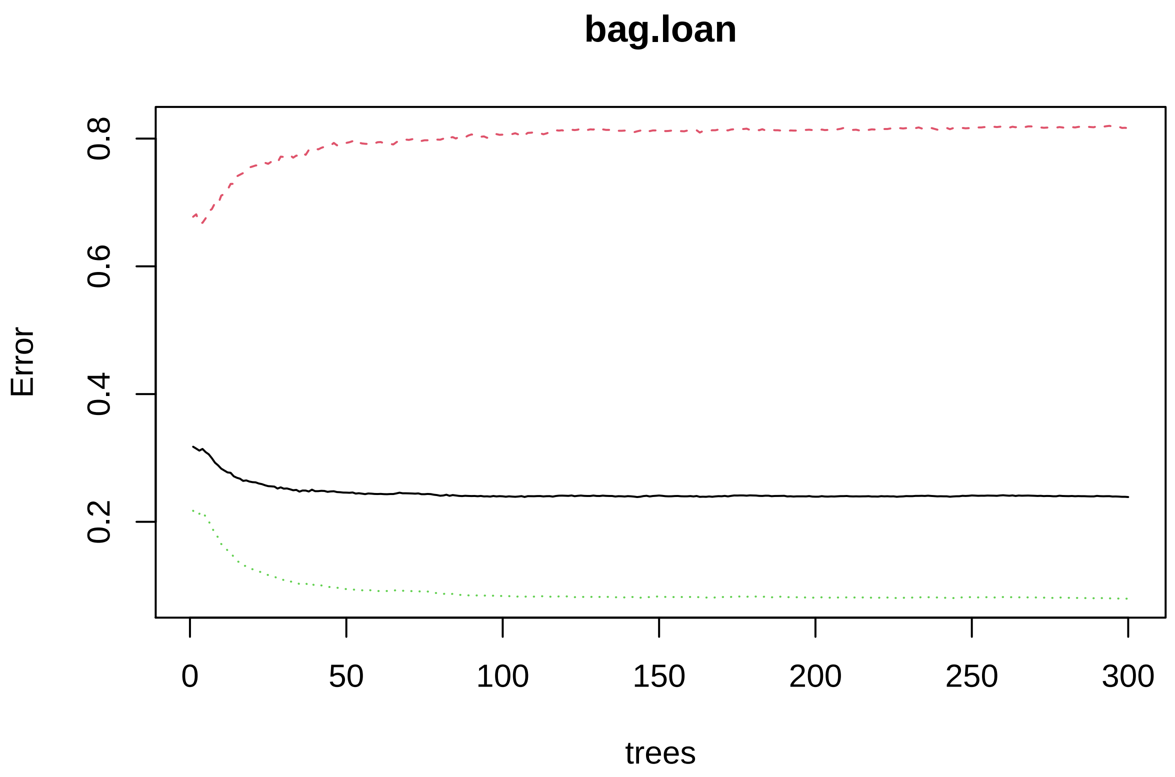
```
bag.loan
```

```
##
## 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: 23.88%
## Confusion matrix:
##                          Default or Charged Off Fully Paid class.error
## Default or Charged Off                      668       2964  0.81607930
## Fully Paid                                 1048      12120  0.07958688
```

Bagging uses all relevant variables for each split node of the tree. I also used the pruned terminal node size in order to tune the model (from Tree Model/Report 3).

The graph below shows that the OOB error rate decreased and stayed consistent after about 50 trees were used to aggregate results. It was also low which was good, and it seemed that using 2 variables per split was ideal.

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



```
set.seed(4)
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                    654       1065
##   Fully Paid                               3278      11835
```

Here is the error rate table that charts the predicted values of my bagging model against the observed values from the testing data set. The bagging model naturally interprated the different possible outcomes, so there was no need to convert or subset percentages like in the previous models.

```
bag_error = 1 - (652+11827)/16832
error_table4.2 = rbind(error_table3.1,bag_error)
error_table4.2
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
```

The bagging regression subtantially improved the error rate of my original logit model. Fitting more trees proved to be more effective than just one.

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

```
##              Default or Charged Off Fully Paid MeanDecreaseAccuracy
## annual_inc                7.161875   13.68217             17.05423
## grade                     6.125227   14.67892             20.07530
## int_rate                  5.861903   16.68580             23.92776
## funded_amnt             -24.186971   40.29074             43.05710
## installment             -29.200035   43.54209             43.21372
##             MeanDecreaseGini
## annual_inc         1243.1228
## grade               271.3289
## int_rate            973.6053
## funded_amnt         765.0583
## installment        1189.2343
```

The importance matrix of the bagging model told that annual income decreased the gini of my model the most i.e. it had the most explanatory power. If left out, the Gini index and impurity would have increased substantially.

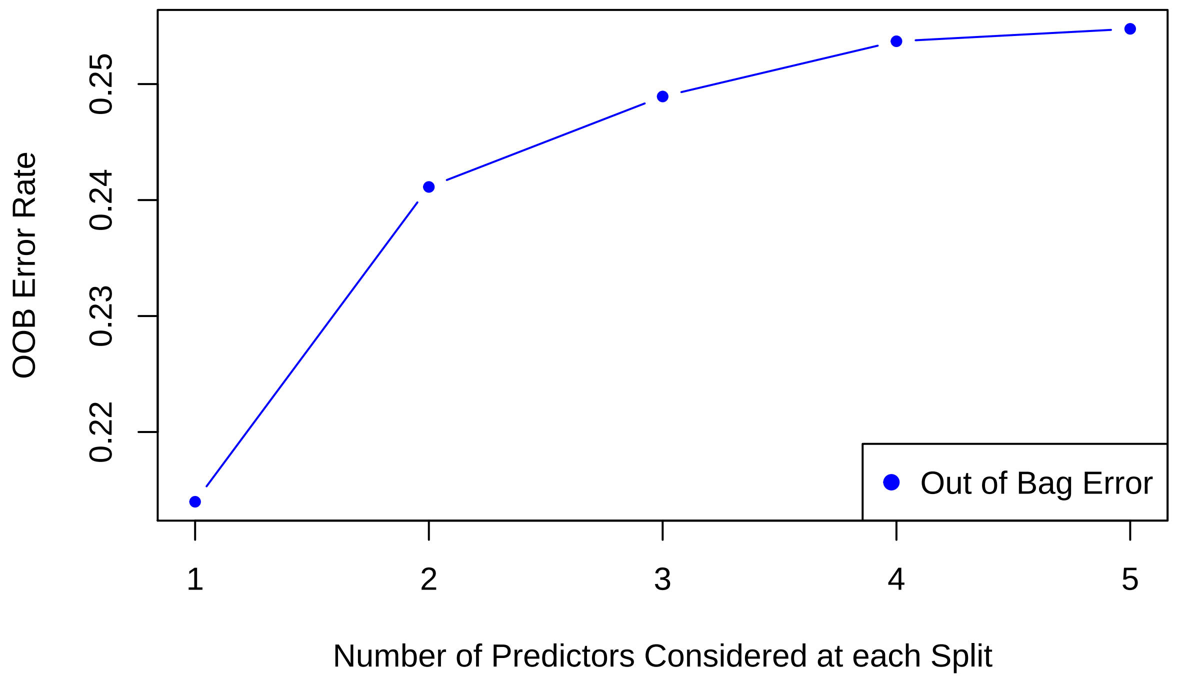
I. Random Forest Classification

Random forest models are meant to improve on bagging by decorrelating trees. Even though bagging aggregates the results of many trees, if there is a variable that consistently helps the model make very accurate predictions, many of the trees will have a similar structure and the results will be skewed (the variable in question will always be used first to build the tree). Random forest takes this into account by varying which variables are used in each level as a tree is being built, thus creating a variety of trees. Like before, the results of these trees are aggregated into one, and the final tree should be one that doesn't overfit the training data at all.

The code used is similar to bagging and an OOB error rate is also automatically given. Here, a loop was made in order to test the optimal amount of variables to use in each split/level.

```r
set.seed(4)
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
}
```

```r
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"))
```



Looking at the OOB error plot of my random forest model, the error rate seemed to increase as more variables were used for each split, using one predictor leads to the smallest error rate. This might have happened because I have few predictors. After tuning the model by using only one variable per split, I tested it using a subsetted testing data set.

```
set.seed(4)
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:  21.54% 91.27%  2.30%
##   200:  21.38% 91.96%  1.91%
##   300:  21.35% 91.82%  1.91%
```

```
set.seed(4)
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                      311        274
##   Fully Paid                                 3621      12626
```

```
rf_error = 1 - (311+12626)/16832
error_table4.3 = rbind(error_table4.2,rf_error)
```

```
error_table4.3
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
```

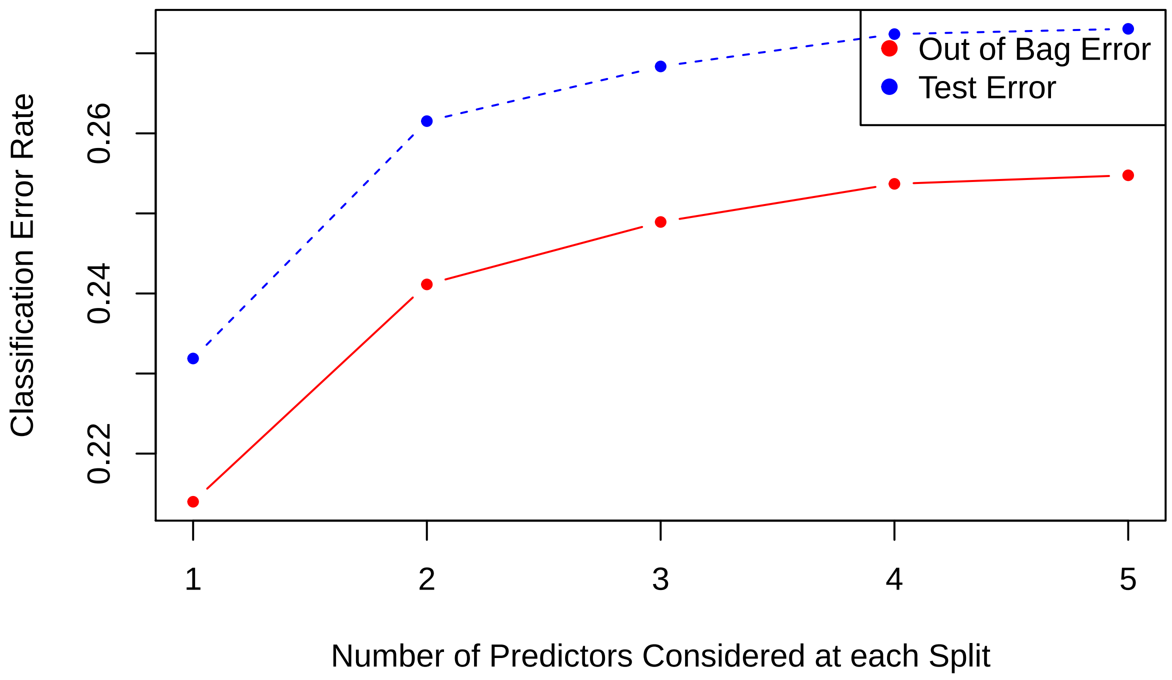
The error rate of my random forest model was 23%, which improved on all my models substantially. It had the lowest error rate by far, which wasn't suprising considering the measures it took to create variation in my data.

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

```
##              Default or Charged Off Fully Paid MeanDecreaseAccuracy
## annual_inc                 2.255044   12.25624             14.17064
## grade                      2.055319   13.92457             18.32954
## int_rate                   5.239833   15.17534             22.05545
## funded_amnt              -18.828510   24.71636             25.30635
## installment              -20.041555   23.80232             23.60038
##              MeanDecreaseGini
## annual_inc           458.4752
## grade                284.7739
## int_rate             544.3362
## funded_amnt          363.0433
## installment          487.4550
```

From the importance matrix, interest rate decreased the gini and impurity of the splits the most i.e. it had the most explantory power as the tree was being built.

```
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 test error followed a similar trend as the OOB error rate. Using one variable per split seemed to reduce the error the most, and the improvement plateaued as more were used.

J - Boosting Classification

Part 1 - Untuned Boosting Model

Boosting model takes a diferent approach than bagging or random forest to build a tree model. Instead of aggregating the results of many different trees, boosting starts by using may small trees to build a large one. With each tree or period, it takes into account the mistakes that were made by the trees before and tries to improve on it. It measures improvement using residuals between the predicted and actual results i.e. how accurate the predictions were. The way it self-improves seemed very interesting to me and held promise.

The boosting xgboost model asked for my y-variable to be a column of 0's and 1's instead of words or phrases. Thus, I made a new dataset that subsets these new values in for the loan status column. Next, I subsetted my testing and training datasets in order to create and assess my model.

```
loan4 = loan_rf

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)
```

```
set.seed(4)
train44 = loan4 %>% sample_frac(0.48)
test44 = loan4 %>% setdiff(train44)
```

```r
set.seed(4)
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)
```

```r
set.seed(4)
boost.pred=predict.gbm(object = boost.loan, newdata = test44, type = "response", n.trees = 1000)
boost.pred2=rep("Default or Very Late",16829)
boost.pred2[boost.pred>.725] = "Fully Paid"
table(boost.pred2,test44$loan_status)
```

```
##
## boost.pred2               0    1
##   Default or Very Late 1974 3264
##   Fully Paid           1972 9619
```

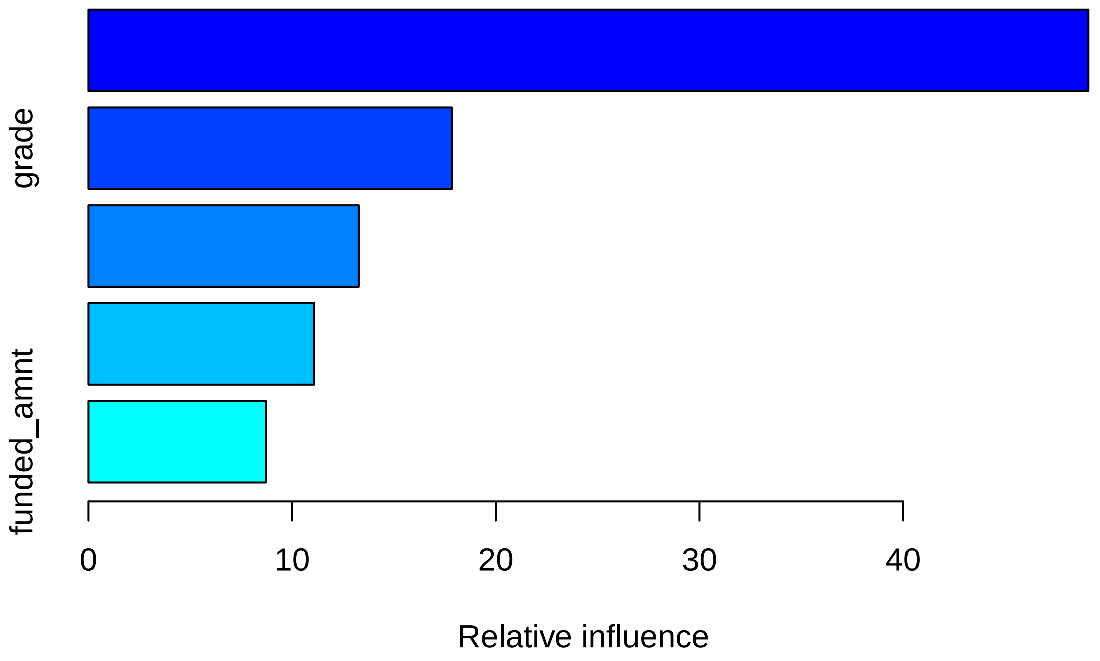
Looking at the error rate table, there seemed to have been more errors made with this model compared to the others. An important note to consider is that there are more arguments and factors used in the boosting model in contrast to the other tree methods. In bagging and random forest, I only paid attention to number of variables used per split, and the number of trees used was negligible (in fact, the more the better). Here, number of trees used is important because using too many can "overgrow" the single tree model and cause it to overfit. Another factor to consider is shrinkage i.e. learning rate or how quickly the model improves on its mistakes. A large learning rate goes through the process faster, but can cause overfitting. There is also interaction depth or the number of splits used per level i.e. how much the tree is allowed to branch off. With all these arguments to consider, this model required more tuning.

```r
boost_error = 1 - (1974+9619)/16829
error_table4.4 = rbind(error_table4.3,boost_error)
error_table4.4
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
## boost_error 0.3111296
```

My suspicions were confirmed, the initial boosting model performed the worst out of all the models.

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



```
##                      var   rel.inf
## int_rate       int_rate 49.093811
## grade             grade 17.838949
## installment installment 13.270072
## annual_inc   annual_inc 11.083433
## funded_amnt funded_amnt  8.713736
```

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

Part 2 - Tuned Boosting Model

In order to tune for my boost model, I needed to make a hypergrid that contained all the possible values of the parameters that I would like to test. That includeed shrinkage, interaction depth, and optimal number of trees. The method I used to find the optimal values for these arguments was hyper-parameter tuning and through a for-loop.

```
hyper_grid <- expand.grid(
shrinkage = c(0.005, 0.075, 0.01),
interaction.depth = c(1, 5, 10),
optimal_trees = 0,
min_error = 0)
```

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

# train model
gbm.tuned <- gbm(
```

```r
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     0.075                 1          2030 0.9745674
## 2     0.005                 1          2373 0.9760652
## 3     0.010                 1          1283 0.9762239
## 4     0.075                 5            73 0.9765605
## 5     0.005                 5           865 0.9766999
## 6     0.010                10           347 0.9767171
## 7     0.010                 5           423 0.9767772
## 8     0.005                10           712 0.9770260
## 9     0.075                10            44 0.9776084
```

After using hyperparameter tuning, it seemed like the parameters that led to the smallest prediction error rate was a learning rate of 0.075, a depth of 1, and 2030 trees. However, the error rate was rather high, so I wasn't sure if there was an improvement or not.

```r
set.seed(46)
bboost.loan = gbm(loan_status~annual_inc+grade+int_rate+funded_amnt+installment,data =
                    train44, distribution = "bernoulli", n.trees = 2030, interaction.depth
                   = 1, shrinkage = 0.075, verbose = F)
```

```r
set.seed(46)
bboost.pred=predict.gbm(object = bboost.loan, newdata = test44, type = "response", n.trees = 1046)
bboost.pred2=rep("Default or Very Late",16829)
bboost.pred2[bboost.pred>.725] = "Fully Paid"
table(bboost.pred2,test44$loan_status)
```

```
##
## bboost.pred2              0    1
##   Default or Very Late 2007 3350
##   Fully Paid           1939 9533
```

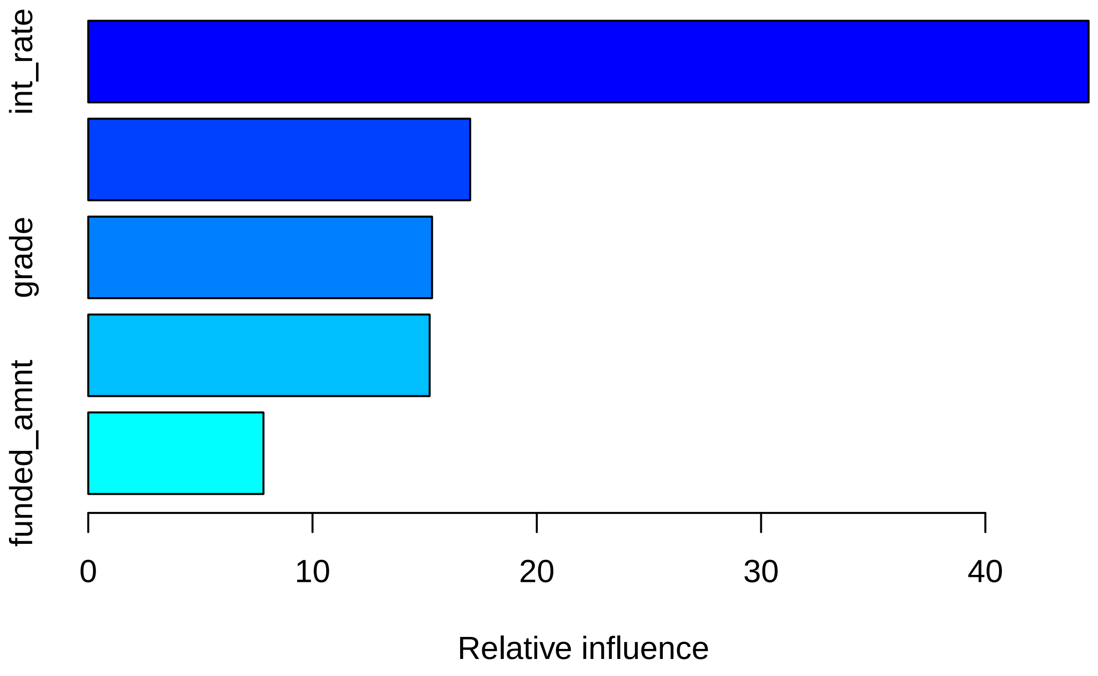
Compared to the other error rate tables I have seen, there are substaintially more errors made with this tuned model still.

```
bb_error = 1-(2007+9533)/16829
error_table4.5 = rbind(error_table4.4, bb_error)
error_table4.5
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
## boost_error 0.3111296
## bb_error    0.3142789
```

My tuned boosted model didn't improve on the original one. An improvement wasn't necessary and it may have been that this modeling method didn't suit the data, but there are other methods apart from a hypergrid to go about tuning a boosting model. These might be worth pursuing in a different project, but since the report is more of a survey of different machine learning methods, I will use these results.

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



```
##                     var   rel.inf
## int_rate       int_rate 44.604652
## installment installment 17.029915
## grade             grade 15.330251
## annual_inc   annual_inc 15.222376
## funded_amnt funded_amnt  7.812806
```

Despite not making any improvements, the tuned boosted model still showed that interest rate seemed to be the most important variable in building the model.

K - XGBoost Classification

Part 1 - Untuned XGBoost Model

The XGboost model, like the boosting model, also trains using previous iterations and updates its model with new information to fix mistakes. However, it uses a different technique to evaluate the how effective each split or level is in building a tree model. XGBoosting has been proven to be very successful, having been used in 17 out of 29 Kaggle ML challenge winning solutions in 2015. The coding for this method is similar to boosting in that it also uses many arguments and parameters to create its model. Otherwise, the steps are straightforward.

```
#Putting data into matrix form with new testing and training sets
set.seed(4)
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(4)
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:0.215238
## [11] train-error:0.215238
## [21] train-error:0.215238
## [31] train-error:0.215238
## [40] train-error:0.215179
```

```
set.seed(4)
xgb.pred=predict(xgb.loan, newdata = dtest4, type = "response")
xgb.pred2=rep("Default or Very Late",16829)
xgb.pred2[xgb.pred>.725] = "Fully Paid"
table(xgb.pred2,test44$loan_status)
```

```
##
## xgb.pred2                 0    1
##   Default or Very Late 1996 3458
##   Fully Paid           1950 9425
```

The results of my table seemed to be similar to my boosting model; a lot more prediction errors were made. Again, this might have been because of the number of parameters to test for in this model. There is max depth or how far the tree can grow, eta or the learning rate of the model, lambda or a regularization term, and other parameters. With all these arguments to consider, this model required more tuning.

```
xgb_error = 1 - (1996+9425)/16829
error_table4.6 = rbind(error_table4.5,xgb_error)
error_table4.6
```

```
##                    [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
## boost_error 0.3111296
## bb_error    0.3142789
## xgb_error   0.3213501
```

As expected, the xgboost didn't perform very well. Some tuning on its parameters was needed.

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

```
##        Feature        Gain       Cover   Frequency
## 1:    int_rate 0.874300086 0.692149546 0.616666667
## 2: funded_amnt 0.035534844 0.085747826 0.100000000
## 3:  annual_inc 0.033512945 0.117349043 0.158333333
## 4: installment 0.026807331 0.076928306 0.091666667
## 5:      gradeC 0.026275317 0.015311727 0.016666667
## 6:      gradeD 0.001935520 0.002000469 0.008333333
## 7:      gradeF 0.001633957 0.010513083 0.008333333
```

Like in my boosting model, interest rate played the largest role in building my model.

Part 2 - Tuned XGBoost Model

In this last part, I tuned my XGBoost model with a similar method I used to tune my boosting model. Now, I am interested in eta, maximum depth, lambda, and the optimal number of trees. The best values for those parameters will bring about the lowest prediction error.

```
set.seed(46)
nrounds = 60
hyper_grid2 <- expand.grid(
  eta = seq(2,30,5)/nrounds,
  max_depth = c(1,2,3,4,5,6),
  lambda = 0,
  optimal_trees = 0,
  min_error = 0
)
```

```
set.seed(46)
for(i in 1:nrow(hyper_grid2)) {
  cv.nround = 200
  cv.nfold = 3
  params = list(
    eta = hyper_grid2$eta[i],
    max_depth = hyper_grid2$max_depth[i],
    lambda = 0
    )
  loan.xgb.cv <- xgb.cv(param=params, data = dtrain4,
                        nfold = cv.nfold,
                        nrounds=cv.nround,
                        early_stopping_rounds = 20,
                        verbos = 0,
                        metrics = {'error'}
                          )
  hyper_grid2$optimal_trees[i] <- loan.xgb.cv$best_iteration
  hyper_grid2$min_error[i] <-
```

```
    loan.xgb.cv$evaluation_log$test_error_mean[loan.xgb.cv$best_iteration]
}

hyper_grid2 %>%
dplyr::arrange(min_error) %>%
head(10)
```

```
##           eta max_depth lambda optimal_trees min_error
## 1  0.28333333         2      0            24 0.2130953
## 2  0.45000000         2      0            44 0.2139287
## 3  0.28333333         3      0            10 0.2141663
## 4  0.11666667         4      0            28 0.2141667
## 5  0.11666667         5      0            19 0.2141667
## 6  0.03333333         5      0            61 0.2141670
## 7  0.28333333         5      0            19 0.2142853
## 8  0.28333333         4      0            12 0.2143453
## 9  0.36666667         1      0            60 0.2143457
## 10 0.36666667         2      0            13 0.2145233
```

After using hyperparameter tuning, it seemed like the parameters that led to the smallest error rate is an eta of 0.283, a depth of 2, lambda of 0, and 24 trees. These values are really small, but the error rate seemed promising.

```
set.seed(5)
xgb.loant = xgboost(data=dtrain4, max_depth=2, eta = 0.283, nrounds=60,
                    lambda=0,ntrees = 24,
                    print_every_n = 10, objective="binary:logistic")
```

```
## [1]  train-error:0.215238
## [11] train-error:0.215238
## [21] train-error:0.213095
## [31] train-error:0.211429
## [41] train-error:0.211190
## [51] train-error:0.210833
## [60] train-error:0.210893
```

```
set.seed(4)
xgb.pred3=predict(xgb.loant, newdata = dtest4, type = "response")
xgb.pred4=rep("Default or Very Late",16829)
xgb.pred4[xgb.pred3>.725] = "Fully Paid"
table(xgb.pred4,test44$loan_status)
```

```
##
## xgb.pred4                 0    1
##   Default or Very Late 2027 3463
##   Fully Paid           1919 9420
```

```
xgbt_error = 1 - (2027+9420)/16829
error_table4.7 = rbind(error_table4.6,xgbt_error)
error_table4.7
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
## boost_error 0.3111296
## bb_error    0.3142789
## xgb_error   0.3213501
## xgbt_error  0.3198051
```

While the tuning improved my original xgboost model, it was not any more accurate than the majority of the other methods. Like in boosting, there are different ways that XGboosting can be done in R, but for the sake of this paper I will keep with these results.

L. Neural Network

Neural networks are arguably the most advanced modeling technique in this paper. Structurally, they are similar to tree models, but the way they learn is done almost automatically with little prompting. By first taking in the data as an raw input, this modeling technique tries to decipher a pattern within the data by using levels of neurons. Neurons contains bits of information about a certain quality that can tell us about a result. For example, a number recognition machine may have a neuron that detects loops which may help it decipher eight or nines. In practice, it is hard to tell what each neuron and layer does. A neural network continues building its structure by building on levels of neurons and branching them together until it can reach a decision. It can then go back and adjust the weight or influence of each neuron and branch using a process called back-propogation.

To start, I needed my y's to be a column of 0 or 1's. Thus, I used my testing and training sets from my bagging and boosting models.

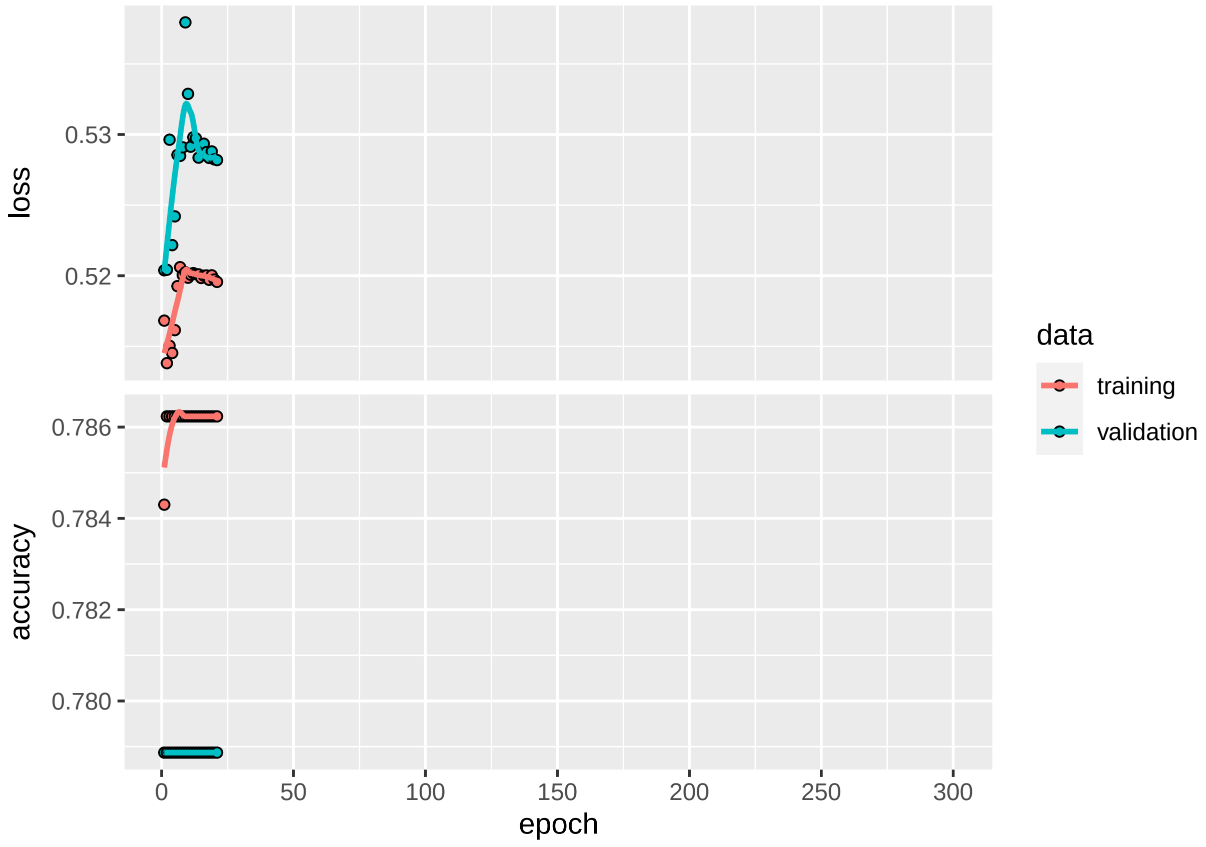
```
set.seed(6)
train_data = model.matrix(loan_status~., train44)[,-1]
train_labels = to_categorical(train44$loan_status)

test_data = model.matrix(loan_status~., test44)[,-1]
test_labels = to_categorical(test44$loan_status)
```

Next, I fitted the model. Because a classification neural network is being used, an activation function (determines the weights of the neurons) for the last level is needed. Instead of reLu, I uses softmax or logistic. Softmax is used to handle more than one level in the y-variable. Below, I created a neural network model that had three levels with 64 neurons in each. The last layer represents the two possible predictions, defaulted or charged off and fully paid. The model should be able to sort the loans in my data set into either category.

```
set.seed(6)
NN_model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "sigmoid",
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dense(units = 2, activation= "sigmoid")

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

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

epochs=300

loan_class <- NN_model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = 0.2,
  callbacks = list(early_stop)
  )
```

```
plot(loan_class)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



For my neural network model, the results proved to be rather unintersting. The accuracy of the model never shifted from about 78%, and the graphs exhibited no interesting trends to speak of. I tried adding more layers and taking away variables, but nothing seemed to improve the model.

```
set.seed(6)
NN_testpred <- predict(object = keras_model_sequential(), x = test_data)
head(NN_testpred)
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7]  [,8]  [,9]  [,10]
## [1,] 110000    0    0    1    0    0    0 17.57 20000 718.75
## [2,]  50000    1    0    0    0    0    0 10.99 20000 654.68
## [3,]  65000    0    0    0    0    0    0  6.99  8000 246.99
## [4,] 106000    0    0    0    0    0    0  5.32  8000 240.92
## [5,]  51000    0    0    0    0    1    0 26.49 24000 725.55
## [6,]  50000    0    0    0    1    0    0 18.99 15000 549.77
```

```
set.seed(6)
NN_values <- NN_testpred[,2]
NN_values[1:10]
```

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

```
test_class <- if_else(NN_values>0.725, true = 1, false = 0)
test_class[1:10]
```

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

Instead of using a predict class function, I used an ifelse loop in order to change the threshold and properly classify for my model.

```
set.seed(6)
table(test_labels[,2], test_class)
```

```
##    test_class
##        0    1
##   0 3152  794
##   1 8740 4143
```

```
nn_error = 1 - 7295/16800
```

```
nn_error
```

```
## [1] 0.5657738
```

The error rate showed that the model performed poorly, only accurately predicting the outcome of a loan about half of the time, which is no better than guessing. Because neural networks are a complicated modeling method, it is difficult to interpret the model and what might have went wrong. Like with boosting and xgboost, other ways to go about coding the neural network should be considered.

M. Conclusion

```
error_table4.8 = rbind(error_table4.7,nn_error)
error_table4.8
```

```
##                  [,1]
## logit_error 0.3044473
## ridge_error 0.2887239
## lasso_error 0.2924607
## bag_error   0.2586145
## rf_error    0.2314045
## boost_error 0.3111296
## bb_error    0.3142789
## xgb_error   0.3213501
## xgbt_error  0.3198051
## nn_error    0.5657738
```

After running histograms and boxplots of my predictor variables as well as building logistic models by regressing them on loan status, I found that annual income, interest rate, and grade produced were statistically significant and had a large role in explaining whether a loan was fully paid or defaulted and charged off. Employment length was not one of these; the data set had several levels for years, and only some were significant. Grade and interest rate deserve special attention because the boosting models consistently showed those variables as being the most helpful in gaining information. This matches with previous studies made on this topic; measures such as grade that are derived from the borrower's financial and credit history hold a lot of information.

My final model ended up having loan status being regressed on annual income, interest rate, grade, installment, and funded amount. The last two were added to make my model more interesting, and their t-test p-values showed that they were just as significant as the other variables. The best modeling technique from this report was the random forest model. After creating a for-loop to find the optimal number of variables to use in each split, I used that value to create a new random forest. This constrasted with my bagging model which used the full number of regressors per split. I obtained an error rate of 23%, which was very good. With the exception of the neural network model, all the models performed roughly similarly, so the study done by the University of Zaragoza was not wrong to point this out.

There are parts of this report that can be further explored. First, loan status originally had several possible outcomes that I had to eliminate because most of the modeling techniques used in this report required a binary variable; it might be interesting to see how these other outcomes could affect the model. Second, I was skeptical to find how little the ridge and lasso model improved the prediction error rate, this meant the original model had variables that all explained loan status very well. Some of the variables might have been correlated to one another because they seemed related, so this can be tested more. Lastly, the boosting, xgboost, and neural network models can be coded in more ways that may produce better results.

N. References

1. Emekter, Riza, et al. "Evaluating credit risk and loan performance in online Peer-to-Peer (P2P) lending." Applied Economics 47.1 (2015): 54-70.
2. Galloway, Ian. "Peer-to-peer lending and community development finance." Community Investments 21.3 (2009): 19-23.
3. Serrano-Cinca, Carlos, Begoña Gutiérrez-Nieto, and Luz López-Palacios. "Determinants of default in P2P lending." PloS one 10.10 (2015).